Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jan Šmid

Priporočilni sistem v grafni podatkovni bazi

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Matjaž Kukar

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge: Priporočilni sistem v grafni podatkovni bazi

Raziščite možnosti za implementacijo in implementirajte priporočilni sistem s sodelovanjem (collaborative filtering) v grafni podatkovni bazi. Posebno pozornost namenite učinkovitosti izvajanja in porabe pomnilnilka. Izvedbo eksperimentalno ovrednotite v primerjavi z eno izmed knjižnic za priporočanje.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ACID	Atomicity, Consistency, Isolation and Durability	atomarnost, konsistentnost, samostojnost/izolacija, trajnost
CRUD	create, read, update and delete	ustvari, preberi, posodobi in izbriši
MAE	mean absolute error	povprečna absolutna napaka
NoSQL	not only structured query language	ne samo strukturirani povpraševalni jezik
OLAP	Online Analytical Processing	analitična obdelava podatkov s povezavo
OLTP	Online Transaction Processing	transakcijska obdelava podatkov s povezavo
RMSE	root mean square error	koren povprečne kvadratne napake
SQL	structured query language	strukturirani povpraševalni jezik

Povzetek

Naslov: Priporočilni sistem v grafni podatkovni bazi

Avtor: Jan Šmid

V diplomskem delu je predstavljen priporočilni sistem, ki je implementiran s pomočjo grafnih podatkovnih baz. Sistem za priporočanje si prizadeva predvideti oceno, ki bi jo uporabnik dal elementu, oziroma predvideti, kateri elementi bi zanimali uporabnika. Poznamo več algoritmov priporočanja. Pomembnejši pristopi so: s sodelovanjem, vsebinsko in hibridno priporočanje. Grafne podatkovne baze so zaradi svojega podatkovnega modela še posebej primerne za takšne sisteme. Njihov najvidnejši predstavnik je Neo4j. Na osnovi sistema Neo4j smo razvili priporočilni sistem za priporočanje filmov (na podlagi podatkov GroupLens) in ga nadgradili s spletno aplikacijo. Uporabili smo priporočanje s sodelovanjem in vsebinsko priporočanje. Rezultate aplikacije smo primerjali z rezultati orodja Surprise in ugotovili, da sta vrednosti MAE in RMSE podobni, če uporabimo enak algoritem.

Ključne besede: grafne podatkovne baze, priporočilni sistem, Neo4j.

Abstract

Title: Recommender system in a graph database

Author: Jan Šmid

The thesis presents a recommender system, which is implemented using graph databases. The recommender system aims to predict the "rating" which the user would give to the element or predict which elements would be of interest to the user. There are several algorithms for recommendations. The more important approaches are: collaborative filtering, content-based filtering, and hybrid recommender systems. Graph databases are particularly suitable for such systems due to their data model. The most prominent representative is Neo4j. Based on the Neo4j system, we developed a recommender system to recommend movies (based on GroupLens data) and supported it with a web application. We used collaborative and content-based filtering. The results of the application were compared with the results of the Surprise tool. We found out that the values of MAE and RMSE are similar if we use the same algorithm.

Keywords: graph databases, recommender system, Neo4j.

Poglavje 1

Uvod

Skoraj vsi smo se že znašli v položaju, ko nismo vedeli, kateri izdelek kupiti, kam iti na počitnice, katera naložba je najboljša, katero šolo ali univerzo izbrati, katero skladbo poslušati. V digitalni dobi nam to izbiro lahko olajšajo priporočilni sistemi. Komercializacija priporočilnih sistemov je bila v veliki meri posledica hitrega razvoja in širitve interneta. Podjetja, ki so želela ponuditi priporočilni sistem, niso smela več gledati samo na to, da je sistem pravilno priporočal, ampak je moral ponuditi izdelke, ki bi lahko sprožili dodatne nakupe. Poleg tega so morala podjetja upoštevati, da se hitrost izvajanja spletne strani ni upočasnila. To je bil povod za nove algoritme, ki bi lahko zmanjšali računski čas. Za dosego tega so se raziskovalci začeli bolj zanimati za zajem uporabniških ocen izdelkov, uporabo umetne inteligence in podatkovnega rudarjenja. [11]

Ključno vlogo pri hitrosti priporočanja ima struktura podatkov o izdelkih in uporabnikih. V zadnjem času so postale priljubljene grafne podatkovne baze in z njimi najvidnejši predstavnik Neo4j. Zaradi kompleksnih poizvedb in "živih" podatkov so grafi odlična izbira za podatkovno bazo. [5]

Cilj diplomskega dela je na praktičnem primeru prikazati delovanje različnih algoritmov priporočanja nad podatki, ki so shranjeni v grafni podatkovni bazi, in ovrednotiti dobljene rezultate.

V 2. poglavju bomo spoznali priporočilne sisteme - kaj sploh so, kje in kdo

jih uporablja. Opisali bomo algoritme, ki bodo uporabljeni v programu. V 3. poglavju bomo na hitro spoznali nerelacijske podatkovne baze in podrobneje, kaj so grafne podatkovne baze. Predstavili bomo tudi najvidnejšega predstavnika grafnih baz - Neo4j.

V 4. poglavju bomo predstavili postopke razvoja priporočilnega sistema z Neo4j, skupaj s tehnologijami in podatki, ki smo jih uporabili.

V 5. poglavju bomo predstavili izdelavo spletne aplikacije in tehnologije, ki smo jih uporabili za razvoj. Priložena sta grafično gradivo končnega videza in opisom funkcij posameznih delov aplikacije.

Da si bomo lažje predstavljali, kako dobro priporoča naš sistem, smo v 6. poglavju rezultate primerjali z rezultati orodja Surprise. To je knjižnica napisana v jeziku Python, namenjena hitremu testiranju različnih algoritmov priporočanja nad poljubnimi podatki.

V 7. poglavju bomo predstavili zaključke, skupno oceno uporabljenega pristopa in možnosti za nadaljnje delo.

Poglavje 2

Priporočilni sistemi

V današnjem času se skoraj vsak dan srečujemo s priporočilnimi sistemi, čeprav se večina ljudi tega ne zaveda. Predstavljajte si, da želite obiskati vašo najljubšo spletno trgovino in kupiti novo televizijo. Takoj ko izberete prvo, se vam nekje na strani pokaže polje z izdelki, zraven pa napis: "Morda bi vas zanimalo tudi", ali pa: "Izbrano za vas". Sistem, ki določi, kateri izdelki naj se prikažejo posameznemu uporabniku, se imenuje priporočilni sistem. Pomembno je poudariti tudi, da se vsakemu uporabniku prikažejo različni seznami priporočenih izdelkov. Prilagojeni so glede na to, katere izdelke je uporabnik kupil, katere si je ogledal, katerim izdelkom je dal oceno in kakšno. Zbiranje teh informacij o uporabnikih imenujemo gradnja uporabniškega modela. [11]

Glede na to, katere informacije uporabimo, ločimo več različnih pristopov priporočanja. Opisali bomo tri: s sodelovanjem, vsebinsko in hibridno priporočanje.

2.1 Priporočanje s sodelovanjem

Glavna ideja priporočanja s sodelovanjem je, da uporabimo informacije o preteklem vedenju ali mnenjih obstoječih uporabnikov, za predvidevanje, kateri izdelki bi bili trenutnemu uporabniku lahko všeč. Takšni sistemi so zelo

razširjeni v industriji, zlasti kot orodja v spletnih trgovinah za povečanje prodaje.

Ta pristop vzame uporabniške ocene (tj. vse ocene, ki so jih uporabniki do sedaj dali izdelkom), kot edini vir podatkov, rezultat pa vrne v numerični obliki, kot mera, koliko je uporabniku izdelek všeč, ali pa kot seznam n priporočenih izdelkov. Takšen seznam ne sme vsebovati izdelkov, ki jih je uporabnik že kupil. [3]

2.1.1 Priporočanje glede na najbolj podobne uporabnike (angl. User-based)

Prvi pristop, ki si ga bomo pogledali, in tudi eden izmed najstarejših se imenuje *priporočanje glede na najbolj podobne uporabnike*. Ta metoda uporabi podatke o uporabniških ocenah, da določi podobnost ostalih uporabnikov s trenutnim. Na podlagi teh podobnosti lahko nato predvidimo, kako bi trenutni uporabnik ocenil še neocenjene izdelke. Metoda predvideva, da bo imel uporabnik v prihodnosti podoben okus, kot ga je imel do sedaj, in da bodo uporabnikove preference ostale stabilne in konstantne. [11]

Poznamo več meril, s katerimi računamo podobnost oziroma različnost podatkov: *kosinusna podobnost* (ki si jo bomo podrobneje pogledali v poglavju 2.1.2), *Spearmanov koeficient korelacije*, *evklidska razdalja*, *Pearsonov korelacijski koeficient* ... Izmed mer podobnosti se najpogosteje uporablja *Pearsonov korelacijski koeficient*, ki se izračuna na naslednji način:

$$sim(a,b) = \frac{\sum_{p \in P}(r_{a,p} - \overline{r_a})(r_{b,p} - \overline{r_b})}{\sqrt{\sum_{p \in P}(r_{a,p} - \overline{r_a})^2}\sqrt{\sum_{p \in P}(r_{b,p} - \overline{r_b})^2}} \tag{2.1}$$

Spremenljivki a in b predstavljata uporabnika, ki ju primerjamo, P je množica ocen, r_{ij} je ocena uporabnika i za j-ti izdelek, $\overline{r_x}$ je povprečna ocena uporabnika x. Rezultat, ki ga ta formula vrne, je na intervalu od $+1$ (ki predstavlja popolno ujemanje) do -1 (predstavlja popolno neujemanje). Da lahko predvidimo, kakšno oceno bi uporabnik dal izdelku, uporabimo podobnost z

drugimi uporabniki, ki smo jo izračunali v formuli 2.1, in upoštevamo, koliko ta vpliva na končno oceno.

$$pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)} \tag{2.2}$$

Ocene uporabnikov običajno shranjujemo v *matriki uporabniških ocen*, kjer vsaka vrstica predstavlja enega uporabnika in vsak stolpec en izdelek. V realnem svetu so te matrike ogromne, zato je pomembno, da pozornost namenimo računski kompleksnosti algoritmov. Poleg tega so te matrike tipično zelo redke, saj uporabniki ocenijo le majhen del vseh izdelkov. Prav tako nastane problem pri tem, kaj priporočiti novim uporabnikom ali kako obravnavati nove izdelke, za katere še ni ocen. Z uporabo primernega merila podobnosti oziroma različnosti lahko naštete težave bolj ali manj uspešno obvladujemo.[3]

Izdelki

Uporabniki	w	x	y	z
a	4	5		
b		4		2
c		3	3	
d	4	3	5	

Slika 2.1: Matrika uporabniških ocen.

Pomembno vlogo pri uspešnosti napovedovanja igra tudi izbira velikosti soseščine (kako merimo uspešnost napovedovanja, je razloženo v poglavju 6).

Če bomo v našem izračunu v soseščino vključili vse uporabnike, bo to ne samo podaljšalo čas računanja, ampak tudi negativno vplivalo na uspešnost rezultata. Po drugi strani pa, če v soseščino vključimo premalo uporabnikov, ima šum v podatkih večji vpliv, zato se uspešnost rezultatov zmanjša. Kakšno velikost soseščine bomo izbrali, je odvisno od podatkov. Tipično se uporablja velikost nekje od 20 do 50 sosedov. [11]

2.1.2 Priporočanje glede na najbolj podobne izdelke (angl. Item-based)

Priporočanje glede na najbolj podobne izdelke primerja podobnosti med izdelki namesto uporabniki. Uporablja se predvsem, ko imamo opravka z veliko količino podatkov (več milijonov uporabnikov in izdelkov). V takšni situaciji je nemogoče v realnem času izračunati vse napovedi ocen z uporabo najbolj podobnih uporabnikov. Zato uporabimo predpripravo rezultatov podobnosti izdelkov in na podlagi teh hitro priporočanje. Enak pristop bi lahko uporabili tudi pri metodi priporočanja glede na najbolj podobne uporabnike, vendar je število ocen dveh uporabnikov za enake izdelke sorazmerno majhno. To pomeni, da lahko nekaj dodatnih ocen hitro vpliva na podobnost med uporabniki. V primerjavi s temi uporabniškimi podobnostmi so podobnosti med izdelki veliko bolj stabilne, saj imajo v povprečju izdelki veliko število ocen. Nova ocena uporabnika tako skoraj ne vpliva na razliko v podobnosti ocenjenega izdelka z drugimi izdelki.

Za najbolj uspešno merilo podobnosti se je izkazala *kosinusna podobnost*. Kosinusna podobnost je mera za podobnost med dvema n-dimenzionalnima vektorjema v n-dimenzionalnem prostoru. Ima interval od 0 do 1, kjer 0 predstavlja popolno neujemanje, 1 pa popolno ujemanje.

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|} \tag{2.3}$$

Kosinusna podobnost ne upošteva razlik v navadi povprečnega ocenjevanja uporabnika.

Primer: če imamo uporabnika, ki ima povprečno oceno 3 in oceni nov izdelek z oceno 4, lahko sklepamo, da mu je izdelek nadpovprečno všeč. Za uporabnika s povprečno oceno 5, ki isti izdelek oceni z oceno 4, bi lahko sklepali, da mu izdelek ni tako všeč kot do zdaj ocenjeni izdelki.

Če želimo upoštevati to razliko, uporabimo *prilagojeno kosinusno podobnost*. Interval se spremeni in je enak kot pri *Pearsonovem korelacijskem koeficientu* (od -1 do 1). Izračuna se:

$$sim(a,b) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r_u})(r_{u,b} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,b} - \overline{r_u})^2}} \tag{2.4}$$

Spremenljivki a in b predstavljata izdelka, ki ju primerjamo, U je množica uporabnikov, ki so ocenili oba izdelka, r_{ij} je ocena uporabnika i za j-ti izdelek, $\overline{r_u}$ je povprečna ocena uporabnika u. Ko imamo izračunane podobnosti, lahko napovemo, kakšno oceno bi izdelku dal uporabnik. Predvidena ocena, ki bi jo uporabnik u dal izdelku p, se izračuna:

$$pred(u,p) = \frac{\sum_{i \in ocenjeniIzdelki(u)} sim(i,p) * r_{u,i}}{\sum_{i \in ocenjeniIzdelki(u)} sim(i,p)} \tag{2.5}$$

Tudi tu moramo določiti velikost soseščine, ki jo bomo uporabili za napovedovanje. Podrobneje je metoda opisana v [3].

2.2 Vsebinsko priporočanje

Pri priporočanju s sodelovanjem smo potrebovali samo informacijo o ocenah uporabnikov. Vsebinsko priporočanje uporabi uporabnikove preference. Preprost primer bi bil, da uporabniku priporočimo Beethovnovo Simfonijo št. 5, če vemo, da rad posluša klasično glasbo. Takšno priporočanje se imenuje vsebinsko.

Kot podatke je tej metodi treba posredovati podrobne informacije o izdelkih in uporabniški profil, ki opisuje, kakšna so uporabnikova zanimanja.

Izdelki, ki jih priporočamo uporabniku, so predstavljeni z množico lastnosti, ki jih imenujemo atributi ali značilke. Vzemimo primer priporočanja

filmov. Vsak film je predstavljen z značilkami, kot so: seznam igralcev, režiser, žanr, leto izdaje ... Večino teh podatkov najdemo v besedilni obliki. Tu nastanejo problemi zaradi jezikovne dvoumnosti pri primerjavi nizov, med drugimi: večpomenke (beseda ima lahko več pomenov) in sopomenke (več besed ima lahko isti pomen). Najboljši pristop pri reševanju tega problema je semantična analiza.

2.2.1 Uporabniški model

Uporabniški model je sestavljen iz različnih tipov informacij. Pogledali si bomo dva:

1. Model uporabniških preferenc, ki jih uporabnik nastavi. Izbere, kaj mu je všeč in kaj ne.

2. Zgodovina uporabnikovih interakcij s priporočilnim sistemom. Sem spadajo uporabnikove ocene, ogledi izdelkov, dosedanji nakupi, iskalni nizi ...

Grajenje uporabniškega modela ima nekaj slabosti. Prvič, zahteva dodaten napor uporabnika in ni preprosto dobiti dovolj ljudi, ki so to pripravljeni storiti. Drugič, sistemi za prilagajanje ne ponujajo načina, kako določiti vrstni red predstavitve izdelkov, kar lahko privede do premalo ali preveč rezultatov, ki jih želimo prikazati. Kako gradimo uporabniške modele, je razloženo v [15].

2.2.2 Računanje podobnosti

Če lahko kolaborativno priporočanje opišemo kot "priporoči izdelke, ki so bili všeč podobnim uporabnikom", lahko vsebinsko opišemo s "priporoči izdelke, podobne tem, ki so bili uporabniku všeč v preteklosti".

Tudi pri vsebinskem priporočanju lahko uporabimo metodo najbolj podobnih sosedov. Obstaja veliko različnih meril za računanje podobnosti med

dvema množicama, med drugimi: *Sørensen–Dice koeficient, Jaccardov indeks, Ochiaiev koeficient* ... [4]

Za namene našega priporočilnega sistema se bomo osredotočili na računanje podobnosti med izdelki z *Jaccardovim indeksom* 2.6. Izračunamo ga tako, da delimo velikost preseka dveh množic z velikostjo njune unije. Podobnost je izražena s številko od 0 do 1. Števila, ki so bližje 0, predstavljajo nizko podobnost, medtem ko števila, bližje 1, predstavljajo visoko. [19]

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.6}$$

2.3 Hibridno priporočanje

Na hitro se bomo dotaknili tudi hibridnega priporočanja, ki združuje vsebinsko priporočanje in priporočanje s sodelovanjem. Takšno priporočanje omogoča združitev pozitivnih lastnosti različnih algoritmov in metod z namenom obvladovanja pomanjkljivosti in problemov posameznih pristopov.

Netflix je dober primer hibridnega priporočanja [17]. Sistem priporoča glede na primerjavo, kaj podobni uporabniki gledajo in iščejo, prav tako pa ponudi filme, s podobnimi lastnostmi, kot jih imajo filmi, ki jih je uporabnik dobro ocenil.

Poznamo sedem hibridnih tehnik [6]: utežna (angl. Weighted), preklopna (angl. Switching), mešana (angl. Mixed), kombinacija značilk (angl. Feature Combination), grajenje značilk (angl. Feature Augmentation), kaskadna (angl. Cascade) in zaporedna (angl. Meta-level).

Omenjeni pristop priporočanja bi lahko uporabili tudi v naši aplikaciji. Prvi primer (preklopna): če uporabnik ni podal ocene še nobenemu izdelku, ne moremo uporabiti priporočanja s sodelovanjem. To nas ne ovira, da uporabimo vsebinsko priporočanje. Drugi primer (kaskadno): uporabimo priporočanje s sodelovanjem za izdelavo seznama izdelkov, ki so jih dobro ocenili podobni uporabniki. Ta seznam uporabimo pri vsebinskem priporočanju kot vhodni podatek.

Poglavje 3

Grafne podatkovne baze

Številni besedo graf povezujejo z grafičnim prikazovanjem podatkov (krožni diagram, stolpčni diagram, histogram, graf funkcije ...). Mi jo bomo v tem poglavju uporabili na drugačen način. Graf v pomenu, ki ga bomo uporabili mi, predstavlja matematično strukturo, sestavljeno iz vozlišč in povezav. Matematična smer, ki proučuje njihove lastnosti, se imenuje teorija grafov. Pogledali si bomo tudi nerelacijske podatkovne baze, katerih podatkovni model temelji na grafih.

3.1 Nerelacijske podatkovne baze (NoSQL)

Nerelacijske podatkovne baze zagotavljajo mehanizem za shranjevanje in pridobivanje podatkov, ki ni modeliran po principu tabel, kot smo to navajeni iz relacijskih baz podatkov. Motivacije za ta pristop vključujejo [16]:

1. preprostost zasnove,
2. preprostejše horizontalno skaliranje (kar je problem za relacijske podatkovne baze),
3. boljši nadzor nad razpoložljivostjo,
4. prilagojenost na problem, ki ga morajo rešiti,

5. veliko operacij je hitrejših,
6. podatkovni modeli so bolj prilagodljivi.

Razlogi, zakaj nerelacijske baze niso splošno sprejete:

1. ni standardnega povpraševalnega jezika,
2. zavržemo veliko let izkušenj relacijskih podatkovnih baz,
3. veliko nerelacijskih baz ne podpira transakcij ACID (atomarnost, konsistentnost, samostojnost/izolacija, trajnost).

Razdelimo jih v štiri glavne kategorije. Ena izmed njih je tudi zbirka grafov, ki si jih bomo natančneje pogledali:

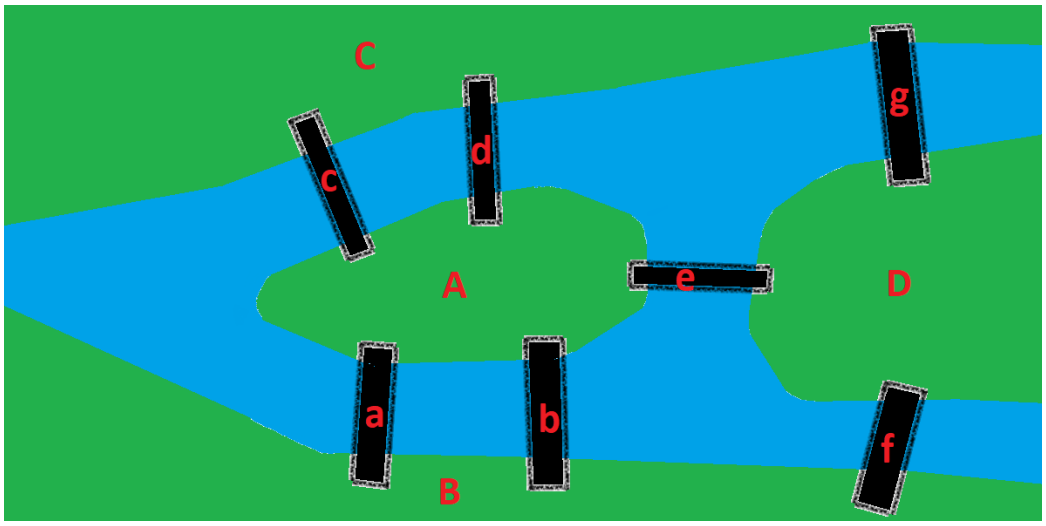
1. **shrambe s ključi** (angl. key-value store),
2. **dokumentne zbirke** (angl. document store),
3. **zbirke grafov** (angl. graph), podrobneje opisane v razdelku 3.4,
4. **stolpično usmerjene zbirke** (angl. column-oriented store).

Ostale kategorije in podkategorije so obširno opisane v [16].

3.2 Kratka zgodovina grafov

Grafe je iznašel oziroma prvič opisal v akademskem članku leta 1741 švicarski matematik Leonhard Euler. Reševal je problem, ki ga poznamo kot "problem sedmih mostov Königsberga".

Mesto, kot je prikazano na sliki 3.1, je bilo razdeljeno na štiri dele (označili smo jih z A, B, C in D). Med seboj so bili povezani s sedmimi mostovi (imenovali jih bomo a, b, c, d, e, f in g). Bistvo problema je najti pot, tako da obiščemo vse štiri dele in prečkamo vseh sedem mostov, ne da bi katerega prečkali dvakrat.



Slika 3.1: Shema sedmih mostov Königsberga.

Euler se je reševanja problema lotil drugače kot ostali. Problem je predstavil z grafom, kjer so deli mesta označeni z vozlišči, mostovi pa s povezavami. S tem, ko je prikazal, da nobeno vozlišče nima sodega števila povezav, je dokazal, da takšna pot ne obstaja.

Pot, kjer prehodimo vsako povezavo natanko enkrat in se vrnemo v izhodišče danes imenujemo *Eulerjev obhod*. Možen je le v grafu, v katerem so vsa vozlišča sode stopnje.

3.3 Teorija grafov

Teorija grafov se je izkazala za fascinantno področje študija, ki je bilo uporabljeno na številnih področjih za reševanje nekaterih najbolj zanimivih vprašanj, s katerimi se sooča človeštvo. Poglejmo si nekaj področij [5]:

1. **Družabna omrežja**: modeliranje interakcij med ljudmi s pomočjo grafov. Primer podjetij na tem področju: Google, Facebook, Twitter, LinkedIn in številna druga.

2. **Biologija**: veliko komponent (beljakovine, molekule, geni ...) je mogoče modelirati in razložiti z grafnimi strukturami.

3. **Računalništvo**: v današnjih aplikacijah so primeri uporabe od oblikovanja čipov, upravljanja omrežja, priporočilnih sistemov, modeliranje UML ...

4. **Problem pretoka**: kako najti najboljšo pot čez neko omrežje. Primere najdemo na področjih, kot so telekomunikacijska omrežja, plinska omrežja, letalska omrežja, omrežja za dostavo paketov ...

5. **Spletni iskalniki**: uporablja se za zagotavljanje boljših rezultatov iskanja.

Teorija grafov formalno definira graf kot množico točk, imenovanih vozlišča, ki so med seboj povezane s povezavami. Te povezave so lahko usmerjene ali neusmerjene. Vsaki povezavi lahko priredimo vrednosti, kar pomeni, da graf razširimo z vpeljavo uteži. Podrobneje je teorija grafov razložena v [10, 18].

3.4 Prednosti grafnih podatkovnih baz

Grafne podatkovne baze so izjemno zmogljive in prilagodljive. Izvajajo lahko kompleksne poizvedbe, ki so sestavljene iz številnih povezovalnih operacij. Takšne operacije so v relacijskih podatkovnih bazah izjemno drage. Treba je namreč računati kartezični produkt indeksov tabel, ki bi ju radi združili, kar privede do pogostih zakasnitev. V grafnih podatkovnih bazah so povezovalne operacije zaradi predstavitve že vnaprej izračunane in eksplicitno prisotne v bazi. Temeljijo na povezavah, ki povezujejo vozlišča. Zato je združevanje podatkov preprosto in primerljivo skoku od enega vozlišča do drugega. [5]

Te operacije pogosto imenujemo *poizvedbe ujemanja grafnih vzorcev* (angl. pattern matching queries). To pomeni, da ko izvedemo neko poizvedbo, moramo le najti začetno vozlišče. Nikoli ni potrebno uporabiti celotnih podatkov, vedno zadošča samo del.

Velika prednost grafnih podatkovnih baz je, da ne potrebujemo vnaprej določene strukture podatkov. Te lahko sproti spreminjamo in prilagajamo našim podatkom in poizvedbam, ki jih želimo izvajati nad bazo.

Razvoj po agilnih metodah postaja dandanes vedno pogostejši. Pomembno je, da izberemo tehnologije, ki takšnega razvoja ne zavirajo in omogočajo preprosto vzdrževanje sistema, zato so tudi za takšne primere grafne baze zelo primerne.

3.5 Neo4j

Obstaja veliko različnih grafnih podatkovnih baz, med katerimi je vredno omeniti: OrientDB, GraphDB, ArangoDB, Microsoft Azure Cosmos DB, Neo4j ... [1] Izbrali smo Neo4j, ki je prvi in najvidnejši predstavnik. Zakaj smo se odločili zanj [5]:

1. visoka zmogljivost branja in pisanja,
2. zelo zanesljiv, tudi za kritične operacije,
3. skalabilen,
4. grafni podatkovni model omogoča preprosto modeliranje podatkov,
5. preprosto učenje in uporaba,
6. velika in aktivna skupnost,
7. podpira transakcije ACID,
8. zagotavlja REST API za številne programske jezike,
9. preprost povpraševalni jezik Cypher.

3.5.1 Štirje temeljni podatkovni konstrukti

Teorija grafov definira veliko različnih tipov grafov. Ti se pojavljajo v številnih oblikah in velikostih, zato Neo4j uporablja zelo specifično vrsto podatkovne strukture, ki je dovolj prilagodljiva za podporo vsestranskosti, ki jo zahtevajo podatki iz realnega sveta. Osnovni podatkovni model Neo4j (angl. *labeled property graph data model*) vsebuje štiri različne temeljne gradnike za strukturiranje in shranjevanje podatkov:

1. **Vozlišča**: uporabljamo jih za shranjevanje informacij o entitetah (tj. objekt iz realnega sveta, ki ga lahko ločimo od ostalih objektov).

2. **Povezave**: se uporabljajo za povezavo vozlišč med seboj. Vsaki povezavi pripadajo začetno vozlišče, končno vozlišče, smer povezave in tip povezave. Povezave lahko kažejo same nase (isto začetno in končno vozlišče), ne morejo pa biti na pol definirane (brez začetnega ali končnega vozlišča).

3. **Značilke (atributi)**: tako vozlišča kot povezave lahko vsebujejo nič ali več značilk. Značilka je podana v paru: ime značilke in vrednost. To je podobno kot v relacijskih podatkovnih bazah, kjer ima vsaka entiteta lahko več atributov (lastnosti). Uporabljajo se za boljšo opredelitev moči oziroma kakovosti razmerja in se lahko uporabijo med poizvedbami za določanje vzorcev, ki jih iščemo.

4. **Oznake**: so bile dodane v Neo4j konec leta 2013 v različici 2.0. Uporabljajo se za hitro in učinkovito ustvarjanje podgrafov. Z dodeljevanjem oznak vozliščem večina podatkovnih modelov postane veliko preprostejša. Vendar pa je v prihodnosti verjetno, da bodo uporabljene za druge namene, kot so dodatna shema, varnost, distribucija grafov in drugo. Če jih primerjamo z relacijsko podatkovno bazo, ena oznaka predstavlja eno tabelo (npr. osebe), medtem ko je vozlišče z eno oznako enakovredno vrstici znotraj tabele (npr. oseba s svojo množico lastnosti). Povezave imajo lahko samo eno oznako.

3.5.2 Izdelan za transakcijsko obdelavo v realnem času

Pri izdelavi priporočilnega sistema je pomembno, da poizvedbe priporočanja, ki se bodo izvajale, ne upočasnijo delovanja spletne strani. Dobra lastnost Neo4j je, da je izdelan za transakcijsko obdelavo v realnem času.

Omenjena lastnost pomaga sistemom, kjer je pomembno, da podatke iz podatkovne baze pošljemo v spletno okolje. To pomeni, da se morajo poizvedbe nad bazo izvesti v časovnem obdobju med spletnim zahtevkom in odgovorom. Z drugimi besedami, izvesti se morajo v milisekundah, ne v sekundah, kaj šele minutah. V svetu relacijskih podatkovnih baz jih imenujemo transakcijski sistemi. Ta lastnost ni potrebna v vseh sistemih za upravljanje baz podatkov. Številni potrebujejo samo odgovor na zahtevo, ki ga pošljemo najprej, odgovor poizvedbe pa lahko tudi več ur kasneje. Te imenujemo analitični sistemi. Poznamo torej dve vrsti: *sistem analitične obdelave* (angl. Online Analytical Processing - OLAP) in *sistem transakcijske obdelave v realnem času* (angl. Online Transaction Processing - OLTP).

V času pisanja diplomskega dela je Neo4j na strani transakcijskih sistemov. To ne pomeni, da z Neo4j ne moremo opraviti analitičnih nalog. Pravzaprav se nekatere analitične naloge iz relacijskega sveta izvajajo bolj učinkovito, vendar Neo4j ni optimiziran za to vrsto opravil. [5]

3.5.3 Povpraševalni jezik Cypher

Ena izmed značilnosti Neo4j je deklarativni povpraševalni jezik Cypher. Izdelan je bil za povpraševanje podatkovnih struktur grafov. Poizvedbe delujejo po principu ujemanja vzorcev, zato je preprost za uporabo. Deklarativnost omogoča, da navedemo, kaj iščemo in določimo vzorec, kako naj se to prikaže. Imperativnemu poizvedovalnemu jeziku je treba povedati, kako naj do podatkov pride in kaj naj z njimi stori.

Ko uporabljamo različne podatkovne baze, velikokrat naletimo na kratico CRUD, ki predstavlja skupino osnovnih operacij: ustvari, preberi, posodobi in izbriši (angl. Create, Read, Update in Delete). To so operacije, ki jih

mora ponujati vsaka podatkovna baza. Poglejmo si, kako so implementirane v Neo4j. [5]

Ustvari podatke (angl. Create)

Pri ustvarjanju podatkov je ključna beseda CREATE. Poglejmo si primer:

```
CREATE (janez:Oseba{ime: "Janez"})
CREATE (simon:Oseba{ime:"Simon"})
CREATE (janez)-[:BRAT]->(simon)
```

V zgornjem primeru smo ustvarili dve vozlišči z oznakama *Oseba* in povezavo BRAT od prvega vozlišča do drugega. Obema vozliščema smo dodali značilko *ime* z vrednostnima: *Janez* in *Simon*. Krajše lahko zgornjo poizvedbo zapišemo takole:

```
CREATE (janez:Oseba{ime: "Janez"}) -[:BRAT]-> (simon:Oseba{ime:
  "Simon"})
```

Vse povezave v Neo4j so usmerjene, zato bi lahko ustvarili dodatno povezavo BRAT od drugega vozlišča do prvega, vendar to ni obvezno. Preverimo namreč lahko, ali obstaja povezava med dvema vozliščema, ne da bi izrecno navedli smer povezave.

Preberi podatke (angl. Read)

Ključni besedi pri branju podatkov sta MATCH in RETURN. Primer:

```
MATCH (n:Oseba)
WHERE n.ime="Janez"
RETURN n
```

V poizvedbi smo poiskali vozlišče z oznako *Oseba*. Dodali smo tudi pogoj, da mora biti osebi ime *Janez*. Zgornjo poizvedbo je mogoče zapisati tudi brez WHERE:

```
MATCH (n:Oseba{ ime:"Janez"})
RETURN n
```

Posodobi podatke (angl. Update)

Da lahko posodobimo podatke, jih moramo najprej poiskati. To storimo po istem principu kot pri branju podatkov. Nato s pomočjo ključne besede SET spremenljivkam nastavimo nove vrednosti. Recimo, da želimo vsem osebam na koncu leta povečati starost za 1:

```
1 MATCH (n:Oseba)
2 SET n.starost=n.starost +1
3 RETURN n
```

Neo4j omogoča tudi nastavljanje novih vrednosti več značilkam hkrati.

Izbriši podatke (angl. Delete)

Prvo pravilo brisanja vozlišč je, da preverimo, da nimajo nobenih povezav z drugimi vozlišči. Tako kot pri ostalih operacijah moramo vozlišče najprej poiskati, šele nato ga lahko izbrišemo. Če želimo izbrisati *Janeza* iz baze, je poizvedba:

```
1 MATCH (r:Oseba{ime:"Janez"})
2 DELETE r
```

Zgornji primer bi nam vrnil napako, saj ima to vozlišče povezavo BRAT. Če želimo kljub temu izbrisati vozlišče, nam preostaneta dve možnosti:

1. najprej izbrišemo povezavo, nato vozlišče,
2. uporabimo ukaz DETACH DELETE (kaskadno brisanje: avtomatsko izbriše vse povezave vozlišča, ki ga želimo izbrisati, nato pa še vozlišče).

Ostali pomembni ukazi

Kot vsak poizvedovalni jezik ima tudi Cypher nekaj ukazov, s katerimi lahko filtriramo rezultat poizvedbe. Poleg teh si bomo pogledali tudi nekaj ukazov za agregacijo podatkov, ki nam bodo prišli prav pri izdelavi priporočilnega sistema.

Tabela 3.1: Tabela prikazuje ostale pomembne ukaze, ki smo jih uporabili v naši aplikaciji.

Ukaz	Razlaga	Primer
WITH	Prenese rezultat iz dela poizvedbe, ki se izvede pred ukazom WITH, v preostali del poizvedbe. Pogosto se uporablja za agregiranje in filtriranje podatkov med poizvedbo, lahko pa tudi za ločevanje branja in posodabljanja podatkov.	MATCH (n) WITH n ORDER BY n.ime DESC LIMIT 3 RETURN collect(n.ime)
ORDER BY	Uredi seznam rezultatov po izbrani značilki/spremenljivki.	RETURN n ORDER BY n.ime
LIMIT	Omeji število rezultatov, ki jih vrne poizvedba.	RETURN n LIMIT 10
DESC	Uredi seznam rezultatov v obratnem vrstnem redu.	RETURN n ORDER BY n.ime DESC
COLLECT	Ustvari seznam z vrednostmi. Desni primer bi vrnil seznam imen [ime1,ime2,ime3 ...].	RETURN collect(n.ime)
CASE WHEN	Pogojni stavek. Če je pogoj resničen, se izvede prva akcija, če ni, se izvede druga.	CASE WHEN n.starost >= 18 THEN "polnoleten" ELSE "mladoleten"END
SQRT	Kvadratni koren	SQRT(x)
SUM	Vsota	SUM(x)
COUNT	Preštej število vozlišč/povezav	COUNT(x)
AVG	Povprečje	AVG(x)

Sintaktična pravila

Da bomo čim bolj konsistentni, se bomo držali sintaktičnih pravil pisanja v jeziku Cypher [5]:

1. **Imena vozlišč**: začnejo se z veliko začetnico, nato sledijo male tiskane črke. Če je besed več, so zapisane brez presledka, vsaka pa se začne z veliko začetnico. Primer: VečBesed.

2. **Povezave**: so zapisane z velikimi tiskanimi črkami. Če je besed več, so ločene s podčrtajem. Primer: DOLGA_POVEZAVA.

3. **Značilke**: so zapisane z malimi tiskanimi črkami. Če je besed več, so zapisane brez presledka. Vse, razen prve se začnejo z veliko začetnico. Primer: večBesed.

4. **Ključne besede**: so tako kot povezave zapisane z velikimi tiskanimi črkami. Če je besed več, so ločene s presledki. Primer: ORDER BY.

Ko poimenujemo vozlišča povezave in značilke, moramo paziti, da se začnejo s črko angleške abecede (za poimenovanje lahko uporabimo tudi črke drugih abeced in številke). Primer: *1ena* ni dovoljeno, medtem ko je *ena1* dovoljeno.

Vsa poimenovanja razlikujejo med velikimi in malimi črkami. Primer: *Oseba*, *osEba* in *oseba* so tri različne oznake. Izjema so ključne besede, ki jih bo sistem prepoznal ne glede na to, ali jih pišemo z malimi ali velikimi črkami.

Prav tako niso dovoljeni simboli, z izjemo podčrtaja, ki se uporablja namesto presledka pri poimenovanju povezav.

Načeloma bi lahko za poimenovanje vozlišč, povezav in značilk uporabili ključne besede, vendar bi poizvedbe postale nepregledne, zato se to odsvetuje.

Poglavje 4

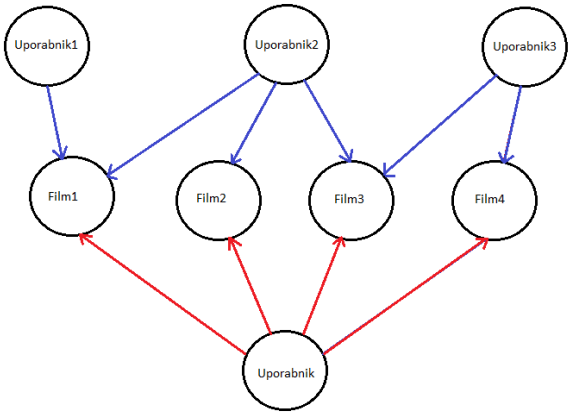
Implementacija priporočilnega sistema z Neo4j

V tem poglavju si bomo najprej pogledali strukturo naše podatkovne baze. Opisana so vsa vozlišča in povezave med njimi, za lažjo predstavo pa je priložena tudi slika. Ker je podatkov veliko, bomo predstavili tudi, postopek vnosa v bazo. Analizirali bomo poizvedbe, ki smo jih izvedli nad bazo, in rezultate našega sistema.

4.1 Struktura podatkovne baze

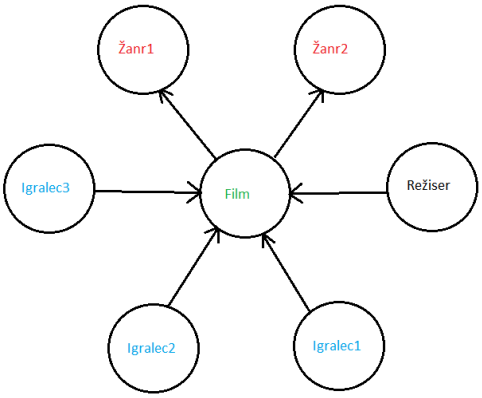
Za čimbolj optimalno poizvedovanje po grafnih podatkovnih bazah, je pomembno, da je načrt baze ustrezen. Odvisen je od poizvedb, ki jih želimo izvajati nad bazo, zato je prvo pravilo načrtovanja, da razmislimo, kaj želimo videti iz podatkov.

Pri priporočanju s sodelovanjem potrebujemo seznam filmov, ki jih je uporabnik že ocenil, in seznam uporabnikov, ki so ocenili vsaj enega od njih. Zato smo se odločili, da bomo uporabniške ocene shranjevali kot povezave, uporabnike in filme pa kot vozlišča. Na sliki 4.1 je primer take poizvedbe. Rdeče povezave predstavljajo seznam filmov, ki jih je uporabnik ocenil, modre povezave pa vodijo do uporabnikov, ki so ocenili vsaj enega od teh filmov.



Slika 4.1: Primer shranjenih ocen.

Za vsebinsko priporočanje potrebujemo zgolj podatke o filmih. Filmi so definirani z nič ali več: igralci, režiserji in žanri. Vsak od njih lahko nastopa v enem ali več filmih. Na podlagi tega smo se odločili, da bomo igralce, režiserje in žanre hranili vsakega v svojem vozlišču, podatke, kot so leto izdaje in povprečna ocena, pa kot značilke v vozlišču filma. Smeri povezav smo določili, kot je prikazano na sliki 4.2. Ker so povezave enako dobro prehodne v obe smeri, bi lahko katero koli povezavo obrnili in to ne bi spremenilo časa izvajanja poizvedb.



Slika 4.2: Primer opisa shranjenega filma v grafu.

V aplikaciji smo uporabili podatke "MovieLens 20M Dataset", ki so bili pridobljeni s spletne strani GroupLens [8]. Bili so v dveh besedilnih datotekah. V prvi datoteki je vsaka vrstica predstavljala en film s pripadajočimi igralci, režiserjem in žanri, v drugi pa oceno uporabnika za posamezni film. Zaradi izjemno velikega števila povezav, ki jih je treba ustvariti, je primerna njihova predstavitev z grafno podatkovno bazo.

4.1.1 Dodajanje podatkov v bazo

V Neo4j je pomembno, da pri dodajanju velikih količin podatkov v bazo te razbijemo na več manjših delov. Priporočeno je, da se najprej ustvarijo vozlišča, nato pa povezave med njimi. Dodajali smo po 10 tisoč vrstic naenkrat z uporabo periodičnega potrjevanja transakcij. V našem primeru je vnos v bazo trajal 8 minut, velikost pa je na koncu znašala 9,5 GB. Vnesli smo podatke o 27 tisoč filmih, 138 tisoč uporabnikih in 20 milijonih ocen.

ime datoteke	velikost
moviesFull.csv	2.841 KB
ratings.csv	520.942 KB

Tabela 4.1: Tabela prikazuje velikosti datotek, ki smo ju vnesli v bazo.

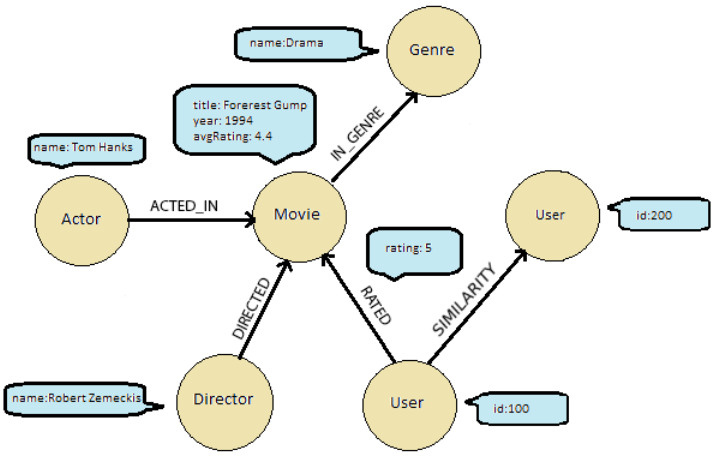
Razdelili smo jih na 5 tipov vozlišč s pripadajočimi značilkami:

- **Movie**: naslov filma, leto izdaje, povprečna ocena,
- **Actor**: ime igralca,
- **Director**: ime režiserja,
- **User**: identifikator,
- **Genre**: ime žanra.

Vozlišča med seboj povezujejo 4 različne povezave:

- **ACTED_IN**,
- **DIRECTED**,
- **IN_GENRE**,
- **RATED**: ocena, časovni žig.

Pri priporočanju s sodelovanjem 4.3 v našo bazo dodamo nov tip povezave: SIMILARITY. Ta med seboj povezuje uporabnike in vsebuje značilko z vrednostjo, koliko sta si uporabnika podobna.



Slika 4.3: Primer dejanske izvedbe grafa za en film. Zaradi preglednosti niso prikazana vsa vozlišča.

4.2 Vsebinsko priporočanje

Bistvo vsebinskega priporočanja je, da na podlagi podanih lastnosti filma, poišče čim bolj podobne. Lastnosti definiramo sami. V našem primeru lahko izberemo eno ali več izmed: žanri, igralci in režiserji.

Da lahko primerjamo, koliko sta si dva filma podobna, potrebujemo merilo. Mi bomo uporabili Jaccardov indeks, ki smo ga opisali v poglavju 2.2.2.

Poglejmo si, kako je videti poizvedba v Neo4j. Po potrebi dodamo oziroma odstranimo oznake povezav: IN_GENRE, ACTED_IN in DIRECTED.

```
1  // v spremenljivko m shranimo vsa vozlisca (filme), ki smo jih
     ze ocenili
2  MATCH (m:Movie)-[r:RATED]-(u:User {id:"id trenutnega uporab."})
3  // v spremenljivko ostala shranimo vozlisca, do katerih obstaja
     vsaj ena pot iz m
4  // v t shranimo presek vsakega para vozlisc m in ostala
5  MATCH (m)-[:IN_GENRE|:ACTED_IN|:DIRECTED]-(t)-[:IN_GENRE|:
     ACTED_IN|:DIRECTED]-(ostala:Movie)
6  // v spremenljivko presek shraninmo stevilo lastnosti v preseku
     mnozic.
7  WITH m,r, ostala, COUNT(t) AS presek
8  // V spremenljivki A in B loceno shraninmo last. obeh mnozic.
9  MATCH (m)-[:IN_GENRE|:ACTED_IN|:DIRECTED]-(mt)
10 WITH m,r,ostala, presek, COLLECT(mt.name) AS A
11 MATCH (ostala)-[:IN_GENRE|:ACTED_IN|:DIRECTED]-(ot)
12 WITH m,r,ostala,presek, A, COLLECT(ot.name) AS B
13 // v unija shranimo, unijo mnozic A in B
14 WITH m,r,ostala,presek,A+filter(x IN B WHERE NOT x IN A) AS
     unija, A, B
15 // uporabimo formulo za racunanje Jaccardovega indeksa
16 RETURN ostala,((1.0*presek)/SIZE(unija)) AS jaccard ORDER BY
     jaccard DESC LIMIT 100
```

V primeru, da želimo upoštevati še ocene, ki smo jih dali filmom, spremenimo zadnjo (17.) vrstico takole:

```
1  RETURN ostala,
2  ((presek*r.rating/5)/SIZE(unija)) AS jaccard ORDER BY jaccard
     DESC LIMIT 100
```

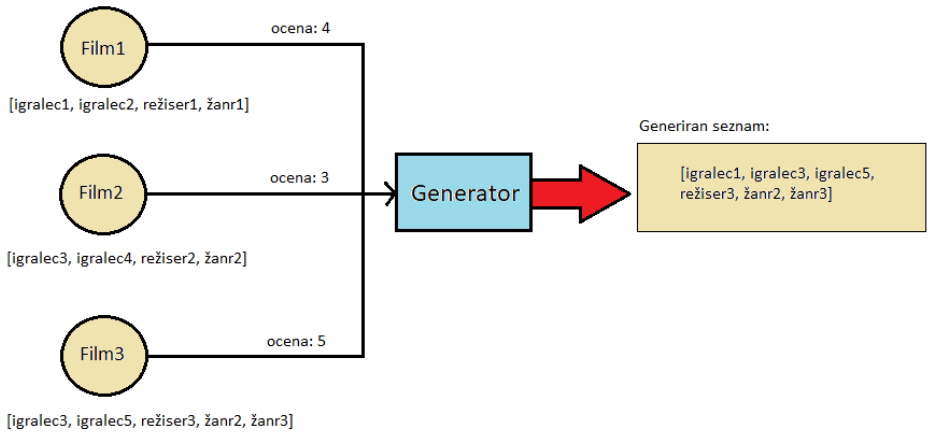
Tabela 4.2: Tabela prikazuje, koliko časa posamezna poizvedba potrebuje za izvedbo.

izbrane lastnosti	3 ocene	10 ocen	30 ocen	60 ocen
DIRECTED	3 ms	6 ms	7 ms	11 ms
ACTED_IN	22 ms	53 ms	116 ms	286 ms
ACTED_IN/DIRECTED	24 ms	58 ms	133 ms	306 ms
IN_GENRE	140 ms	407 ms	660 ms	1721 ms
IN_GENRE/DIRECTED	166 ms	461 ms	804 ms	1779 ms
IN_GENRE/ACTED_IN	364 ms	516 ms	871 ms	1820 ms
IN_GENRE/ACTED_IN/DIRECTED	375 ms	584 ms	952 ms	1875 ms

Rezultati iz tabel 4.2 in 4.3 so povprečje desetih izpeljanih poizkusov. Iz tabele je razvidno, da poizvedba največ časa porabi pri primerjanju po žanru. Razlog za to je, ker imajo nekateri žanri preko 4 tisoč povezav do drugih filmov. V realnem svetu bi bilo priporočanje s to poizvedbo, primerno samo z uporabo množic igralske zasedbe in režiserja.

Pomembno je omeniti tudi, da ima v tej poizvedbi vsak že ocenjen film svojo množico z lastnostmi (igralci, režiser in žanri). Pri vsebinskem priporočanju želi sistem prepoznati, kakšne so preference uporabnika, iz njegovih ocen, ali pa jih ta eksplicitno nastavi.

V programskem vmesniku je mogoče izbrati funkcijo, ki priporočilnemu sistemu omogoči ustvarjanje množice lastnosti filmov, ki so na podlagi ocen uporabnika najbolj ustrezne. Preklapljanje med načinoma je možno s potrditvenim poljem, ki ga najdemo v nastavitvah naše spletne aplikacije (prikaz na sliki 5.2). Poizvedba, ki smo jo opisali na začetku poglavja 4.2, ne vsebuje ustvarjanja seznama. Pri primerjanju dveh filmov v množici skupnih lastnosti (presek množic) prevladujejo žanri. V seznamu priporočenih filmov, so bili na prvih mestih filmi, ki so imeli veliko število žanrov (nekateri filmi imajo tudi do 10 žanrov). Igralci in režiserji pa so se redko pojavljali v preseku množic in tako niso imeli velikega vpliva. Z ustvarjanjem skupnega seznama smo omejili vpliv posamezne lastnosti, obenem pa povečali vpliv tistim, ki



Slika 4.4: Grafični prikaz ustvarjanja seznama lastnosti filmov.

se večkrat ponovijo. Časovno nismo opazili velike razlike, priporočila pa so boljša.

Najprej iz baze preberemo katere filme je uporabnik že ocenil in kakšno oceno jim je dal. Nato primerno utežimo lastnosti teh filmov na podlagi njegovih ocen. Na koncu v poizvedbi vsebinskega priporočanja spremenimo 9. in 10. vrstico takole:

```
1  WITH m, ostala, presek, ["mnozica lastnosti uporabnika"]  AS A
```

Na sliki 4.4 so na levi strani filmi, ki jih je uporabnik že ocenil. Pod vsakim so izpisane njegove lastnosti. Na povezavi, ki gre iz filma, so ocene uporabnika za pripadajoči film. Na sredi je generator, ki na podlagi ocen uporabnika primerno vrednoti te lastnosti. Na desni strani je končni seznam.

4.3 Priporočanje s sodelovanjem

V naši podatkovni bazi so tudi podatki o ocenah ostalih uporabnikov. Pri priporočanju s sodelovanjem uporabimo prav te podatke. Želimo poiskati uporabnike, ki so čim bolj podobno ocenjevali filme. Za dosego tega, podobno kot pri vsebinskim priporočanju, moramo določiti merilo. Jaccardov indeks je primeren za primerjavo dveh množic med sabo, pri uporabniških ocenah

pa moramo upoštevati, da ima vsaka ocena svojo težo. Primerno merilo za računanje takšnih podobnosti je kosinusna podobnost, ki je podrobneje opisana v poglavju 2.1.2.

S pomočjo formule 2.3 lahko oblikujemo poizvedbo v Neo4j, s katero ustvarimo povezave SIMILARITY med uporabniki.

```
1  -- v x in y shranimo vektorja ocen za uporabnika z id="nov" in
     tistim s komer ga primerjamo
2  MATCH (p1:User {id: "nov"})-[x:RATED]->(m:Movie)<-[y:RATED]-(p2
     :User)
3  -- xyDotProduct je stevec iz formule 2.3 (mnozenje vektorja
     ocen up. A z vektorjem ocen up. B)
4  WITH COUNT(m) AS stFilmov, SUM(x.rating * y.rating) AS
     xyDotProduct,
5  -- v xLength in yLength shranimo dolzine vektorjev x in y.
     Racunanje imenovalca v formuli 2.3. REDUCE sluzi kot zanka,
     ki iterira cez ocene uporabnika. Rezultat sharnjujemo v xDot/
     yDot, ki ga na koncu korenimo.
6  SQRT(REDUCE(xDot = 0.0, a IN COLLECT(x.rating) | xDot + a^2))
     AS xLength,
7  SQRT(REDUCE(yDot = 0.0, b IN COLLECT(y.rating) | yDot + b^2))
     AS yLength, p1, p2
8  -- ustvarimo povezave SIMILARITY v grafu
9  MERGE (p1)-[s:SIMILARITY]-(p2)
10 SET s.similarity = xyDotProduct / (xLength * yLength)
```

Ta poizvedba je zelo zamudna in je odvisna od števila ocen uporabnika. Za uporabnika s 25 ocenami traja 2 sekundi, medtem ko za uporabnika s 450 ocenami traja slabih 30 sekund. Razlog za dolgotrajno izvajanje je, da Neo4j ni primeren za uporabo tako kompleksnih agregiranih funkcij nad velikim številom vozlišč. Kako izvedemo računanje podobnosti za vse uporabnike, je razloženo v poglavju 6.3, kjer najprej spoznamo deljenje podatkov na učno in testno množico. Če želimo vrednosti povezave SIMILARITY posodobiti (uporabnik oceni nov film), ponovimo zgornjo poizvedbo. Zaradi ukaza MERGE se ne ustvarijo nove povezave, ampak se samo posodobijo vrednosti.

Kljub temu je algoritem za priporočanje s sodelovanjem veliko bolj primeren v realnem svetu kot vsebinsko priporočanje. Računanje podobnosti med uporabniki je le prvi korak in ni treba, da ga izvajamo neprestano. Posodabljanje se ne izvaja pri odjemalcu, tako da ga uporabnik niti ne opazi.

V drugem koraku iz množice uporabnikov izberemo k najbolj podobnih. Z uporabo njihovih ocen filmov lahko predvidimo, kakšno oceno bi tem filmom dali uporabniki, ki jim priporočamo. Poizvedba je videti takole:

```
1  MATCH (b:User)-[s:SIMILARITY]-(a:User {id:'nov'})
2  -- z ukazom WITH filtriramo samo k najbolj podobnih
3  WITH b,s,a ORDER BY s.similarity DESC LIMIT 40
4  MATCH (b)-[r:RATED]->(m:Movie)
5  --izberemo samo filme, ki jih uporabnik se ni ocenil
6  WHERE NOT((a)-[:RATED]->(m))
7  WITH b,r,m,a
8  -- kako pridobimo averageRating je razlozeno v naslednji
     poizvedbi
9  ORDER BY r.rating DESC,m.averageRating DESC
10 -- Z DISTINCT preprecimo ponavljanje vrstic (istih filmov) v
     rezultatu
11 RETURN DISTINCT m.title,m.id LIMIT 100
```

Hitrost izvedbe ni odvisna od števila ocen, ki jih je uporabnik dal, ampak od števila najbolj podobnih uporabnikov. Razlog za to je, ker v poizvedbi iščemo samo filme, ki so jih ocenili podobni uporabniki, obenem pa omejimo, da do njih ne sme obstajati povezava od trenutnega uporabnika. To je tudi vidno v tabeli 4.3. Večji kot je k, dalj časa se izvaja poizvedba, večje število ocen pa ne vpliva na čas izvajanja. Kljub temu ni bistvene razlike v času izvajanja, saj skoraj nobena poizvedba ne preseže 200 milisekund.

V naši poizvedbi potrebujemo povprečno oceno za vsak film. Da po nepotrebnem ne upočasnjujemo izvajanja poizvedbe, smo izračune povprečnih ocen pripravili pred priporočanjem in jih shranili v bazo (značilko *averageRating* vozlišča Movie).

```
1 MATCH (m:Movie)
2 SET m.averageRating=(round(100 * m.averageRating)/100)
```

Tabela 4.3: Tabela prikazuje, koliko časa posamezna poizvedba potrebuje za izvedbo z uporabo priporočanja s sodelovanjem.

število k sosedov	20 ocen	50 ocen	100 ocen
k = 10	27 ms	52 ms	44 ms
k = 25	142 ms	123 ms	125 ms
k = 50	205 ms	203 ms	193 ms

Poglavje 5

Spletna aplikacija

Eden izmed ciljev našega dela je bila funkcionalna implementacija priporočilnega sistema. Uporabnik lahko oceni poljubno količino filmov, sistem pa mu glede na njegove ocene priporoči, katere filme naj si pogleda v prihodnje. V tem poglavju so opisane tehnologije, ki smo jih uporabili za izdelavo naše spletne aplikacije. Opisane so funkcije, ki jih aplikacija ponuja, opisano pa je tudi, kje se nahajajo. Za lažjo predstavo smo priložili slikovna gradiva.

5.1 Predpriprava na izdelavo aplikacije

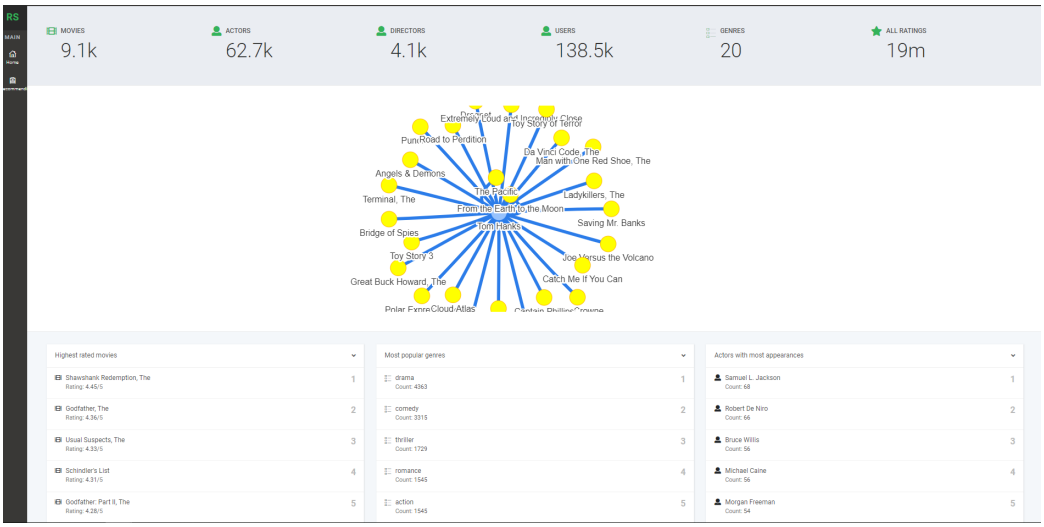
Preden smo začeli z izdelavo aplikacije, smo izdelali načrt poteka gradnje. Definirali smo, katere funkcije bo aplikacija vsebovala in katere podatke bomo uporabili za priporočilni sistem. Na podlagi teh smo pripravili skico, ki nam je bila v pomoč pri izdelavi. Videz aplikacije smo želeli narediti čim bolj pregleden in preprost za uporabo. Na koncu smo se odločili, katere spletne tehnologije bomo uporabili za razvoj.

5.2 Splošno o aplikaciji

Strežniški del vmesnika je izdelan v programskem jeziku Node.js [14], odjemalčev pa v HTML, CSS in Angular. Za komunikacijo s podatkovno bazo

smo uporabili knjižnico "neo4j-javascript-driver", ki je uradni gonilnik za programski jezik JavaScript. [13]

Vmesnik smo razdelili na dva dela. V prvem je pregled nad podatkovno bazo, kjer lahko vidimo število vseh vozlišč (filmi, igralci, režiserji, uporabniki, žanri) in število ocen. Uporabnik lahko nad bazo izvaja tudi preproste poizvedbe na primer: "Izpiši vse filme, v katerih je igral igralec X". Rezultat poizvedbe se prikaže v obliki grafa in je izdelan s pomočjo odprtokodne knjižnice Neovis.js.

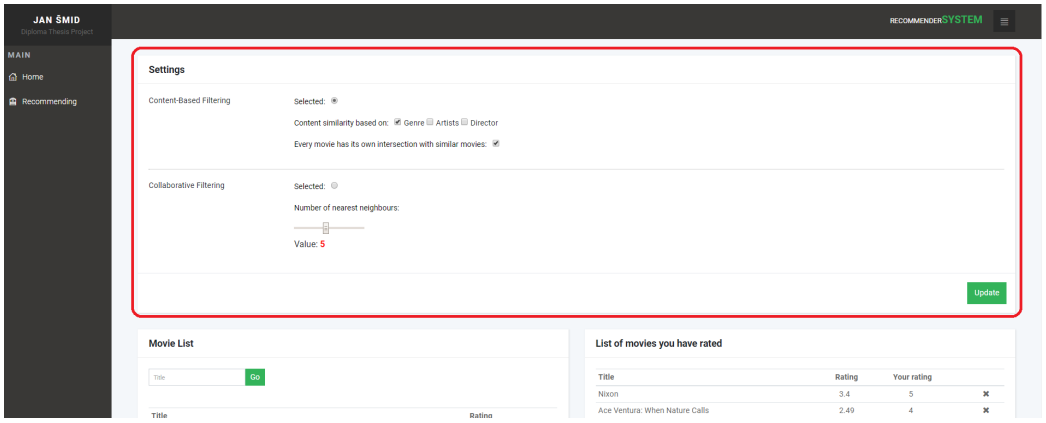


Slika 5.1: Prvi del naše spletne aplikacije.

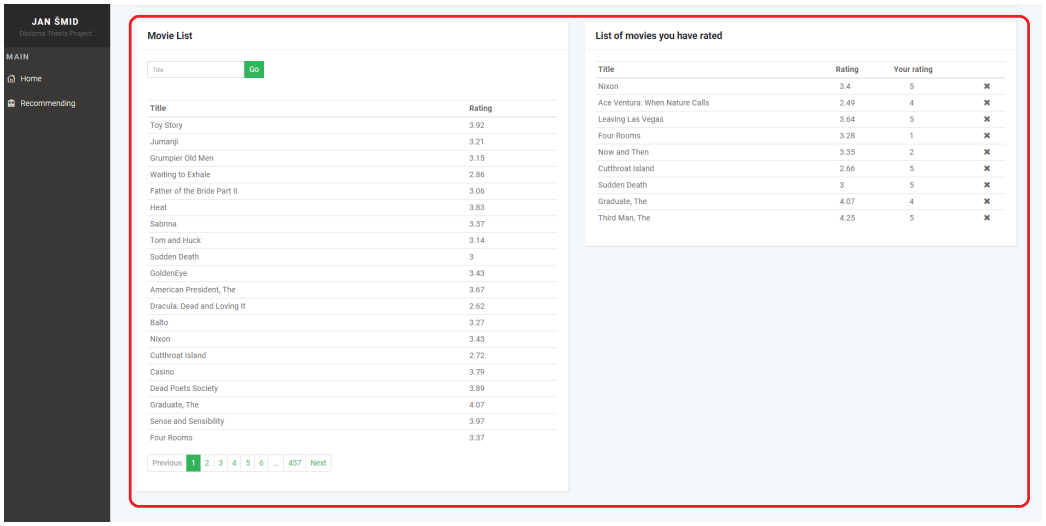
V drugem delu je priporočilni sistem. Ta je razdeljen na tri sekcije. V 1. sekciji ima uporabnik možnost izbire, kateri algoritem naj se uporabi. Uporabnik lahko izbira med dvema algoritmoma za priporočanje, ki sta: s sodelovanjem in vsebinsko. Pri prvem ima možnost nastavitve števila najbolj podobnih sosedov, pri drugem pa, kateri podatki naj se uporabijo za grajenje uporabniškega modela.

V 2. sekciji sta seznam vseh filmov in seznam že ocenjenih filmov. Zaradi preglednosti sta v seznamu izpisana samo podatka o naslovu in povprečni oceni filma. Ob kliku na posamezni film se prikažejo podrobne informacije

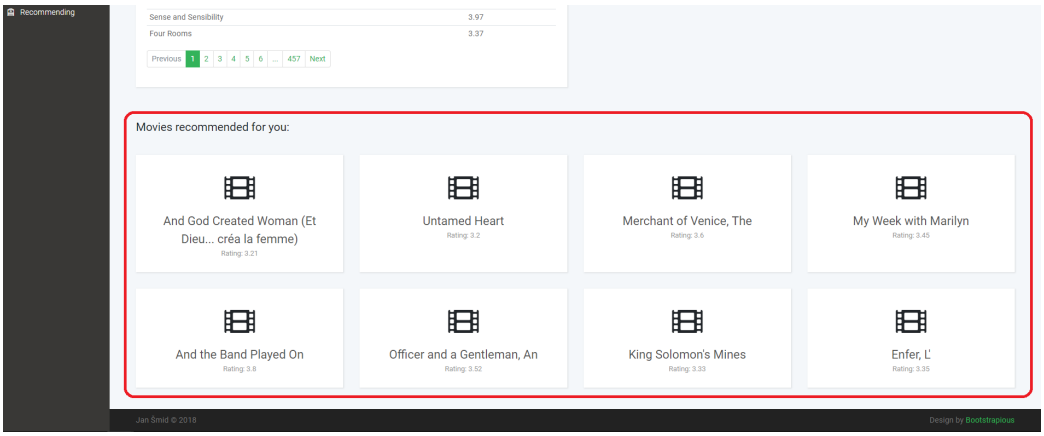
(igralci, režiser, žanri, število vseh ocen in leto izida), uporabniku pa je tudi ponujena možnost, da ga oceni. V 3. sekciji pa je seznam filmov, ki jih sistem priporoča uporabniku.



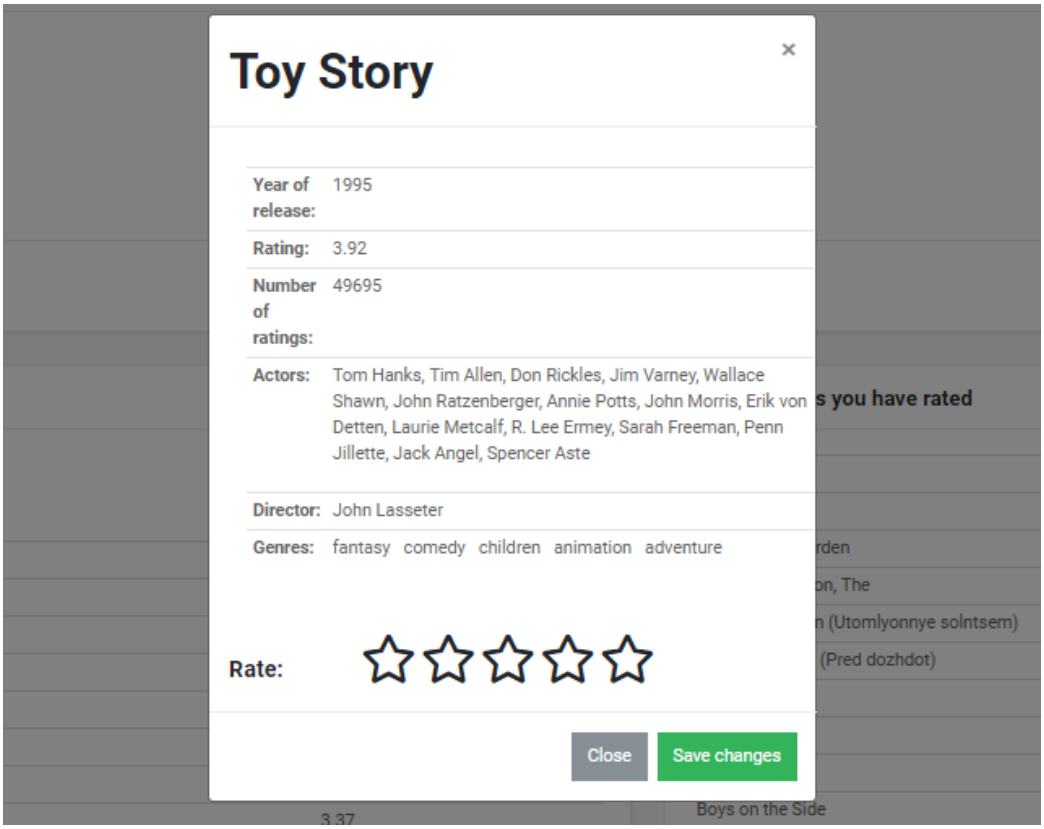
Slika 5.2: Z rdečo označena prva sekcija, v kateri je mogoče izbrati različne nastavitve priporočanja.



Slika 5.3: Z rdečo označena druga sekcija, v kateri je na levi strani seznam vseh filmov, na desni pa seznam ocenjenih filmov.



Slika 5.4: Z rdečo označena tretja sekcija, v kateri je seznam filmov, ki jih sistem priporoča uporabniku.



Slika 5.5: Primer modalnega okna ob kliku na film. Okno prikazuje lastnosti filma (leto izida, povprečna ocena, število ocen, igralci, režiser, in žanri). Film je mogoče tudi oceniti (število zvezdic: od 1 do 5).

Poglavje 6

Primerjava rezultatov

V 4. poglavju smo si pogledali implementacijo našega priporočilnega sistema. Da lahko ocenimo, kateri algoritem najbolje priporoča izdelke uporabnikom, primerjamo velikost napake, ki jo naredi pri napovedovanju ocene. Rezultate našega sistema bomo primerjali z že obstoječim orodjem Surprise, ki je napisan v Pythonu. Primerjali bomo razliko v času izvajanja in velikost napake. Najprej pa si bomo pogledali, kako izračunamo napako pri napovedovanju.

6.1 Merili RMSE in MAE

Povprečna absolutna napaka (MAE) in *koren povprečne kvadratne napake* (RMSE) sta dve najpogostejši meritvi, ki se uporabljata za merjenje uspešnosti napovedovanja.

Povprečna absolutna napaka (MAE): MAE meri povprečno velikost napak v nizu napovedi. Uporablja enako merilo kot podatki, ki jih merimo, zato ga ne moremo uporabiti za primerjavo nizov, ki imajo različna merila. MAE je torej povprečje absolutnih razlik med napovedanimi in dejanskimi ocenami.

$$MAE = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y_j}| \tag{6.1}$$

Koren povprečne kvadratne napake (RMSE): RMSE je kvadratno pravilo točkovanja, ki meri povprečno magnitudo napake. Je kvadratni koren povprečja kvadratnih razlik med napovedanimi in dejanskimi ocenami.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y_j})^2} \tag{6.2}$$

Rezultat je pri obeh merilih na intervalu od 0 do neskončno. Manjši kot je, boljše je napovedovanje. Med njima je tudi nekaj razlik. Ker so napake pri RMSE kvadrirane, preden jih povprečimo, imajo velike napake večji vpliv na končni rezultat. To pomeni, da je RMSE še posebej uporaben, ko so velike napake nezaželene. Katero merilo izbrati, je razloženo v [7]. Mi smo za primerjavo uporabili obe merili.

6.2 Orodje Surprise

Orodje Surprise je knjižnica, napisana v programskem jeziku Python. [9] Namenjena je za preprosto uporabo različnih algoritmov priporočanja nad poljubnimi podatki. Izbrali smo 3 algoritme: SVD razcep, priporočanje glede na najbolj podobne uporabnike in priporočanje glede na najbolj podobne izdelke. SVD razcepa do zdaj še nismo omenili in smo ga izbrali, ker nad podatki MovieLens ustvari najboljše rezultate (testirano z vsemi algoritmi orodja Surprise). Pri vseh algoritmih smo uporabili 5-kratno prečno preverjanje. To pomeni, da smo podatke razdelili na 5 enako velikih množic, nato pa vsako enkrat uporabili za testno množico (angl. test set), ostale štiri pa za učno množico (angl. training set). Pri priporočanju glede na najbolj podobne uporabnike in izdelke smo nastavili število najbližjih sosedov na 40. Za primerjavo rezultatov smo uporabili merili RMSE in MAE. V spodnjih tabelah okrajšava "Mn."pomeni množica, "Povp."povprečje in "Odkl.štandardni odklon.

Tabela 6.1: Tabela prikazuje rezultate orodja Surprise z uporabo algoritma SVD nad našimi podatki.

	Mn. 1	Mn. 2	Mn. 3	Mn. 4	Mn. 5	Povp.	Odkl.
MAE	0.6051	0.6053	0.6060	0.6050	0.6055	0.6054	0.0004
RMSE	0.7964	0.7969	0.7958	0.7951	0.7961	0.7960	0.0007
Čas priprave	1202.60	1234.74	1216.18	1216.81	1226.50	1219.37	12.09
Čas izvajanja	103.75	117.86	111.64	101.64	121.43	111.26	8.60

Tabela 6.2: Tabela prikazuje rezultate orodja Surprise z uporabo priporočanja glede na najbolj podobne uporabnike nad našimi podatki.

	Mn. 1	Mn. 2	Mn. 3	Mn. 4	Mn. 5	Povp.	Odkl.
MAE	0.6437	0.6429	0.6433	0.6441	0.6435	0.6435	0.0005
RMSE	0.8256	0.8255	0.8263	0.8259	0.8254	0.8257	0.0004
Čas priprave	1880.69	1935.36	1902.09	1925.50	1893.84	1907.50	22.55
Čas izvajanja	170.90	161.08	180.39	167.94	177.72	171.60	7.73

Tabela 6.3: Tabela prikazuje rezultate orodja Surprise z uporabo priporočanja glede na najbolj podobne izdelke nad našimi podatki.

	Mn. 1	Mn. 2	Mn. 3	Mn. 4	Mn. 5	Povp.	Odkl.
MAE	0.6073	0.6068	0.6065	0.6067	0.6065	0.6068	0.0003
RMSE	0.7981	0.7980	0.7975	0.7981	0.7983	0.7980	0.0003
Čas priprave	1232.54	1201.11	1213.87	1208.86	1221.43	1215.56	12.03
Čas izvajanja	99.54	107.22	117.69	104.38	119.47	109.66	8.62

Iz preglednic je razvidno, da se je najbolje izkazal algoritem SVD s skupnim časom izvajanja 111 minut in najnižjima vrednostima RMSE in MAE. Takoj za njim pa je k-nn glede na najbližje izdelke, ki je bil sicer 1 minuto hitrejši, vendar ima nekoliko slabši vrednosti RMSE in MAE.

6.3 Primerjava z našim sistemom

Poglejmo si še naš sistem. Uporabili bomo kolaborativno priporočanje, velikost soseščine pa bomo nastavili na 40. Podatke bomo razdelili enako kot z orodjem Surprise, torej na 5 podmnožic.

Najprej podatke razdelimo na dva dela (testna in učna množica). Ker je vseh uporabnikov 138 tisoč, za testno množico vzamemo 110 tisoč uporabnikov, preostale pa uporabimo za učno množico. Nato izračunamo podobnosti med uporabniki v učni množici in predvidene ocene, s pomočjo formule 2.2. Predvidene ocene nato primerjamo z ocenami v testni množici, po merilih MAE in RMSE. To ponovimo petkrat tako, da za testno množico vedno izberemo drugo petino uporabnikov.

```
1 MATCH (p1:User)-[x:RATED]->(m:Movie)<-[y:RATED]-(p2:User)
2 WHERE p1.idINT<110000 AND p2.idINT<110000
3 WITH COUNT(m) AS numbermovies, SUM(x.rating * y.rating) AS
    xyDotProduct,
4 SQRT(REDUCE(xDot = 0.0, a IN COLLECT(x.rating) | xDot + a^2))
    AS xLength,
5 SQRT(REDUCE(yDot = 0.0, b IN COLLECT(y.rating) | yDot + b^2))
    AS yLength, p1, p2
6 WITH p1,p2,xyDotProduct / (xLength * yLength) AS sim
7 MERGE (p1)-[s:SIMILARITY {similarity:sim}]-(p2)
```

Tabela 6.4: Tabela prikazuje rezultate našega sistema, z uporabo priporočanja glede na najpolj podobne uporabnike.

	Mn. 1	Mn. 2	Mn. 3	Mn. 4	Mn. 5	Povp.	Odkl.
MAE	0.6652	0.6529	0.6675	0.6692	0.6582	0.6626	0.0031
RMSE	0.8295	0.8364	0.8314	0.8277	0.8342	0.8318	0.0016
Čas priprave	6511.32	5788.21	5966.70	6304.16	6271.83	6168.44	128.83
Čas izvajanja	135.14	157.03	143.11	124.89	153.94	142.82	5.94

Vrednosti MAE in RMSE sta nekoliko slabši od rezultatov orodja Sur-

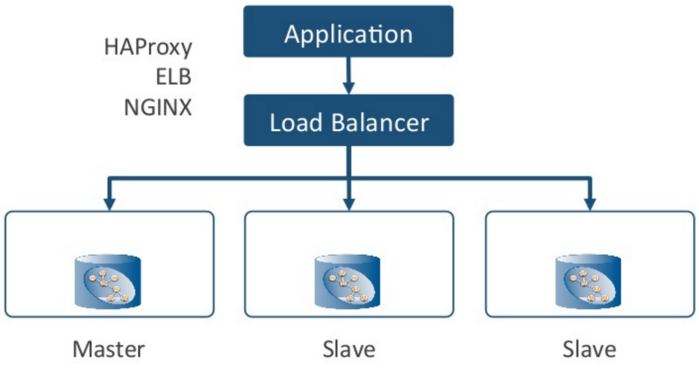
prise, če uporabimo priporočanje s sodelovanjem glede na najbližje uporabnike. Skupni čas trajanja je 525 minut. Čas priprave je občutno daljši kot pri orodju Surprise, čas izvajanja pa je podoben. Rezultate napovedovanja bi morda lahko izboljšali tako, da bi uporabili *Pearsonov korelacijski koeficient*. Iz rezultatov orodja Surprise lahko sklepamo, da bi še boljši rezultat verjetno dobili, če bi uporabili primerjanje izdelkov namesto primerjanja uporabnikov.

Orodje Surprise za računanje nad našimi podatki porabi od 9 do 10 GB pomnilnika, odvisno od algoritma, ki ga uporabimo. Če bi želeli pospešiti izvajanje, bi to lahko dosegli le z uporabo hitrejšega procesorja in pomnilnika ali z spreminjanjem programske kode orodja. Pri Neo4j pa lahko uporabimo tako *vertikalno* kot *horizontalno* sklairanje. [12]

Vertikalno skaliranje dosežemo tako, da strežnik, ki ni dovolj zmogljiv, preprosto zamenjamo z boljšim (podobno kot pri pohitritvah orodja Surprise: hitrejši procesor in pomnilnik). Pomembno je omeniti tudi, da Neo4j lahko porabi veliko več pomnilnika za izvajanje poizvedb, kot orodje Surprise za računanje. Mi smo za izvajanje namenili 10 GB pomnilnika (Neo4j je porabil ves njemu namenjen pomnilnik), da ni prišlo do motenj pri delovanju operacijskega sistema (skupna količina pomnilnika na našem sistemu znaša 16 GB). Česar Neo4j ne more shraniti v pomnilnik, sproti prenaša z diska. To občutno upočasni izvajanje. Iz tega sklepamo, da bi dodajanje pomnilnika, znatno pospešilo izvajanje.

Horizontalno skaliranje se od vertikalnega razlikuje v tem, da ne zamenjamo obstoječega strežnika, ampak mu dodamo novega. Poglejmo si, kako je implementirano v Neo4j. Eden izmed strežnikov je vedno glavni (angl. master) in samo na njem se lahko izvajajo operacije pisanja. Razlog, zakaj je za pisanje namenjen samo en strežnik, je, ker imajo v večini primerov poizvedbe nizko obremenitev pisanja (sem sodijo tudi poizvedbe priporočanja). Vsi ostali strežniki (angl. slave) so namenjeni za izvajanje operacij branja. Zmogljivost branja se povečuje z vsakim dodanim strežnikom. Primer: če imamo 3 strežnike, ki zmorejo na sekundo obdelati 300 zahtev, bi dodajanje novega strežnika povečalo obdelavo zahtev na 400 zahtev na sekundo. Stre-

žniki so med seboj neprestano sinhronizirani; ko glavni strežnik izvede nov zapis, ostali strežniki posodobijo svojo bazo v nekaj milisekundah (čas je odvisen od nastavitve uporabnika). Dodatni strežniki služijo tudi kot varnostna kopija, saj vsak vsebuje kopijo celotne podatkovne baze. Ta pristop ima tudi negativne strani: višji stroški z vsakim novim strežnikom, več vzdrževanja ... Z uporabo horizontalnega skaliranja, aplikacija ne komunicira več z bazo preko Neo4j gonilnika, ampak z uporabo orodja za izenačevanje obremenitve (angl. load balancer). Nekaj bolj poznanih: HAProxy, NGINX in Elastic Load Balancer. Ker v veliko primerih ni mogoče shraniti celotne baze v pomnilnik, je velika prednost orodja za izenačevanja obremenitve, da lahko usmerja, na katerem strežniku naj se določena poizvedba izvede (na strežnik, kjer je večina podatkov, ki jih poizvedba potrebuje, že shranjeno v pomnilniku; angleški izraz: Cache Sharding). V našem primeru lahko usmerjamo glede na identifikator uporabnika ali naslov filma. Primer bi bil, da uporabnike z identifikatorjem od 1 do 10 000 usmerjamo na strežnik 1, od 10001 do 20000 na strežnik 2 itd. Pomembno je, da za usmerjanje izberemo takšno lastnost, ki naredi podgrafe dovolj majhne, da je večina podatkov shranjena v pomnilniku. Prednosti horizontalnega skaliranja bi se poznale predvsem pri več vzporednih zahtevkih, saj bi z ustreznim usmerjanjem lahko dosegli sočasno izvajanje zahtevkov. [12, 2]



Slika 6.1: Shema glavnega strežnika skupaj z novo dodanimi strežniki [2].

Poglavje 7

Sklepne ugotovitve

Že na začetku smo omenili, da ima ključno vlogo pri hitrosti in uspešnosti napovedovanja priporočilnih sistemov struktura podatkov. V diplomskem delu smo preverili, koliko so grafne podatkovne baze primerne za implementacijo takšnih sistemov. Pogledali smo si tri različne vrste priporočilnih sistemov: priporočanje s sodelovanjem, vsebinsko in hibridno priporočanje.

Da smo lahko lažje preverili prednosti grafnih podatkovnih baz, smo izdelali aplikacijo, ki je imela podatke shranjene v bazi Neo4j. Večina funkcionalnosti priporočanja se je nahajala v bazi (poizvedbe). Osredotočili smo se predvsem na priporočanje s sodelovanjem, ki smo ga na koncu primerjali z orodjem Surprise in ugotovili, da sta vrednosti MAE in RMSE podobni, če uporabimo enak algoritem. S programsko opremo in količino pomnilnika, ki smo ga dodelili, je bil čas izvajanja daljši, vendar bi ga lahko s pomočjo horizontalnega in vertikalnega skaliranja znatno pospešili.

7.1 Možnosti za nadaljne delo

Aplikacijo bi lahko še izboljšali. Dodali bi lahko hibridno priporočanje, kolaborativno priporočanje glede na najbolj podobne izdelke, možnost izbire mere podobnosti: Pearsonov korelacijski koeficient, prilagojeno kosinusno podobnost in Sørensen–Dice koeficient (omenjeni v 2.1.1 in 2.2.2) za praktični prikaz

različnega obvladovanja težav pri računanju podobnosti. Uporabno bi bilo tudi, da bi rezultate meril RMSA in MAE vključili v naš grafični vmesnik. Tako bi uporabnik vedel, kako dobra so priporočila.

Pri izdelavi diplomskega dela smo se naučili veliko novega, tako s področja grafnih podatkovnih baz kot priporočilnih sistemov. Usvojili smo tudi osnove razvoja v programskem jeziku Node.js in ogrodju AngularJS ter spoznali orodje Surprise.

Literatura

[1] DB-ENGINES: Ranking of graph DBMS. `https://db-engines.com/en/ranking/graph+dbms`. Dostop: 28. 8. 2018.

[2] Load balancer proxy scale. `https://neo4j.com/blog/graphs-to-production-at-scale/`. Dostop: 27. 8. 2018.

[3] C.C. Aggarwal. *Recommender Systems: The Textbook*. Springer International Publishing, 2016.

[4] Meyer Andréia da Silva, Antonio Garcia, Anete De Souza, and Cláudio Junior. Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (zea mays l). 27, 03 2004.

[5] J. Baton and R. Van Bruggen. *Learning Neo4j 3. X - Second Edition*. Packt Publishing, 2017.

[6] Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[7] Tianfeng Chai and Roland R. Draxler. Root mean square error (RMSE) or mean absolute error (MAE)? Arguments against avoiding RMSE in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

[8] GroupLens. Movielens 20M dataset. `https://grouplens.org/datasets/movielens/20m`. Dostop: 6. 7. 2018.

[9] Nicolas Hug. Surprise, a Python library for recommender systems. `http://surpriselib.com`, 2017.

[10] Wilson Robin James and Watkins John Jaeger. *Uvod v teorijo grafov*. DMFA, Slovenija, 1997.

[11] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, 2010.

[12] David Montag. Understanding Neo4j scalability. *White Paper, Neotechnology*, 2013.

[13] Neo4j. Neo4j Driver for JavaScript. `https://github.com/neo4j/neo4j-javascript-driver`.

[14] Node.js. JavaScript runtime built on Chrome's V8 JavaScript engine. `https://nodejs.org/en/`.

[15] Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[16] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. NoSQL databases. *Lecture Notes, Stuttgart Media University*, 20, 2011.

[17] Andreas Töscher, Michael Jahrer, and Robert M. Bell. The Bigchaos solution to the Netflix grand prize. *Netflix prize documentation*, pages 1–52, 2009.

[18] W.T. Tutte, W.T. Tutte, and C.S.J.A. Nash-Williams. *Graph Theory*. Cambridge Mathematical Library. Cambridge University Press, 2001.

[19] I. E. Vorontsov, I. V. Kulakovskiy, and V. J. Makeev. Jaccard index based similarity measure to compare transcription factor binding site models. *Algorithms for Molecular Biology*, 8(1):23, 2013.