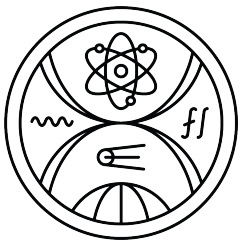
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

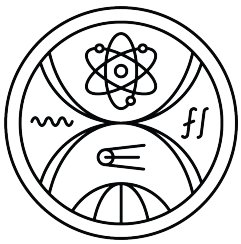


Tester SQL úloh na báze evolučných algoritmov

DIPLOMOVÁ PRÁCA

2024 Bc. Nikola Kulíková

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



Tester SQL úloh na báze evolučných algoritmov

DIPLOMOVÁ PRÁCA

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Vedúci práce:	Ing. Alexander Šimko, PhD.

Bratislava 2024 **Bc. Nikola Kulíková**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Nikola Kulíková
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Tester SQL úloh na báze evolučných algoritmov
SQL exercise tester based on evolution algorithms

Anotácia: Súčasný databázový tester, ktorý sa používa na predmete Databázy (1) testuje SQL úlohy tak, že spustí vzorové a študentské riešenie na databázovom systéme. Následne porovná databázy a odpoveďové tabuľky, ktoré tieto riešenia vrátia a zisťuje rozdiely medzi nimi. Pri porovnávaní tabuliek sa nedokáže vysporiadať so situáciou, kedy boli do databázy vložené tie isté dáta, no databázový systém im priradil iné identifikátory.

Cieľ: Cieľom práce je navrhnúť algoritmus na porovnávanie databáz na báze evolučných algoritmov, ktorý je schopný sa vysporiadať so situáciu, kedy databázy obsahujú rovnaké dáta, no majú pridelené rôzne identifikátory. Z časového hľadiska má algoritmus vrátiť porovnanie obratom. Súčasťou práce bude experimentálne vyhodnotenie, ako je navrhnutý algoritmus efektívny, aké kvalitné riešenia nájde a pod.

Vedúci: Ing. Alexander Šimko, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 30.09.2022

Dátum schválenia: 30.09.2022 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie

Abstrakt

Kľúčové slová:

Abstract

Keywords:

Obsah

Úvod

Kapitola 1

Motivácia

Pre študentov je dôležité aby mali možnosť vyskúšať si, či je ich riešenie správne alebo nie hocikedy. Na toto bol vytvorený databázový tester, kde si študent môže vyskúšať riešiť zadanie na predmete Databázy(1) a získať okamžitú odpoveď či je riešenie správne alebo nie.

Odpovede či je dané riešenie správne alebo nie sa získava za pomoci študentom zadaného SQL dopytu a uloženým správnym dopytom. Každý SQL dopyt pri svojom vykonaní vytvorí databázu. Vytvorená databáza obsahuje tabuľky spolu s riadkami, ktoré boli zadané dopytom. Vytvoria sa dve rôzne databázy, jedna pre študentov dopyt a druhá pre uložené správne riešenie. Na to aby sme boli schopní ohodnotiť správnosť študentového riešenia, je potrebné nájsť mapovanie medzi vytvorenými databázami. Pri hľadaní mapovania medzi databázami hľadáme taktiež mapovanie medzi tabuľkami a ich riadkami. Na riešenie problému mapovania sa momentálne využíva Kuhn-Munkresov algoritmus 3.1. V tomto algoritme ale nastáva problém pri primárnych a cudzích kľúčoch. Viac o tomto probléme si povieme v kapitole 2.

Kapitola 2

Problematika

V tejto kapitole sa budeme podrobnejšie venovať problematike mapovania, ktorú sme si spomenuli v kapitole 1. Uvedieme si daný problém súčastného riešenia a možné riešenie takéhoto optimalizačného problému.

2.1 Ohodnotenie a penalizácia

Ako sme si spomenuli v kapitole 1, pre nájdenie či je dané študentové riešenie správne pracujeme s jeho SQL dopytom a uloženým správnym dopytom.

Následne sa pomocou dopytov vytvoria databázy, medzi ktorými hľadáme mapovanie. Pri hľadaní tohto mapovania medzi databázami sa hľadá mapovanie jednotlivých tabuliek. Tabuľky sa skladajú z riadkov a teda hľadáme mapovanie medzi riadkami. Každý riadok sa skladá z primárneho kľúča, dátových stĺpcov a prípadne cudzích kľúčov. Dátové stĺpce obsahujú zadané dáta rôznych dátových typov. Primárny kľúč je špecifický identifikátor pre daný riadok. Cudzie kľúče sa využívajú na odkazovanie na iný riadok v tabuľke. Tabuľka môže byť rovnaká ako tabuľka riadku, v ktorej sa nachádza ale taktiež môže byť aj iná. Pri hľadaní mapovania medzi riadkami sa počíta penalizácia riadkov a teda ich rozdielom všetkých stĺpcov. Táto penalizácia nám umožňuje určiť nakoľko sú dané riadky podobné. Problém nastáva pri primárnych kľúčoch, kde nám pri mapovaní nezáleží na tom či dané riadky ich majú rovnaké alebo nie. Tento problém sa taktiež objavuje pri cudzích kľúčoch, kde sa do stĺpca doplní daný primárny kľúč. A keďže vieme, že primárne kľúče nemusia byť vždy rovnaké, tak riadok dostáva väčšiu penalizáciu než by mal.

Problém je taktiež pri viacnásobných cudzích kľúčoch, kde riadok z tabuľky sa odkazuje na riadok do inej tabuľky a takto to pokračuje rekurzívne. Takéto prehľadávanie a následné vyhodnocovanie cudzích kľúčov je však časovo náročné.

Pre vrátenie informácie študentovi či jeho riešenie je správne alebo nie, hľadáme také mapovanie, ktoré má najmenšiu možnú penalizáciu. Keď riadok dostane väčšiu penalizáciu pri danom mapovaní, nepovažuje sa toto mapovanie za vhodné aj napriek tomu, že obsah riadkov je zhodný. Naším cieľom je vymyslieť algoritmus na nájdenie vhodného mapovania, ktorý nebude brať do úvahy primárne a cudzie kľúče. Algoritmus sa taktiež bude vedieť vysporiadať so situáciou, kedy sa nachádzajú v databáze viacnásobné cudzie kľúče a bude ich vedieť do istej miery prehľadávať.

2.2 Súčastné riešenie

Namapanovanie riadkov na seba, ak taký už nie je tak na null

Vytvorenie bipartitného grafu

Vyber najlepšieho mapovania cez Hungarian algorithm pridat ref

Najmenšie ohodnotenie

Ohodnotenie vzdialenosti riadkov = rozdiel znakov vo všetkých stĺpcoch

Ak je jeden riadok null tak vráti počet stĺcov

Mapovanie obojsmerne -¿ hodnota hrany je 2 * rozdiel znakov vo všetkých stĺpcoch

Pri rôznych idčkách sa nevie vysporiadať aj keď sú dáta rovnaké

2.3 Evolučný algoritmus

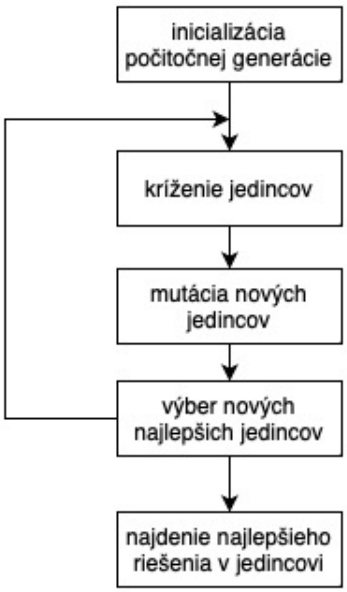
Na problém hľadania mapovania sa používa aj evolučný algoritmus. Tento algoritmus vychádza z biologického základu Darwinovej teórie (asi nejaky odkaz nejakym sposobom na knihu). Darwinova teória reprezentuje vývinový proces na základe 3 hlavných princípoch:

1. prirodzený výber jedinca - silnejší jedinec prežije dlhšie
2. jedince sú navzájom odlišné
3. reprodukčný proces - výmena genetických informácií a lepšie prispôsobenie jedincov

Evolučný algoritmus vytvára na začiatku jedince s určitými chromozómami. Chromozómy si vieme predstaviť ako lineárny reťazec, v ktorom sú zakódované informácie o jedincovi (napríklad vek, druh,...). Na základe týchto informácií je jedincovi vypočítaná hodnota, ktorá hovorí o tom, aký silný je jedinec v porovnaní s ostatnými. Podľa tejto hodnoty sa určí pravdepodobnosť reprodukcie.

Po vytvorení určitého počtu x jedincov nám vznikne generácia, s ktorou pracujeme ďalej. Viac o vytváraní počiatočnej generácii si povieme v sekcii 2.3.1. Pri jednotlivých generáciách v evolučnom algoritme, jednotlivé jedince môžeme vzájomne krížiť a mutovať. Za pomoci týchto dvoch spôsobov dostaneme nové jedince s možnou lepšou reprodukčnou hodnotou. Viac o týchto dvoch spôsoboch zmeny jedincov si vysvetlime v podsekciách 2.3.3 a 2.3.2.

Ďalším krokom evolučného algoritmu je, že nové jedince, vytvorené mutáciou a krížením, spojíme do jednej generácie spolu ich predchodcami a vyberieme tie, ktoré majú, čo najlepšiu pravdepodobnosť reprodukcie , tak aby ich počet bol zas x. Takýto spôsob aplikujeme niekoľkokrát a sledujeme zmeny v chromozómoch a vývin nových vlastností. Celý proces je zobrazený na obrázku 2.1.



Obr. 2.1: Kroky evolučného algoritmu

Pri optimalizačnom probléme ako je aj hľadanie mapovania sa uplatňuje hľadanie globálneho minima. //todo dopisat nieco este nejaky zaver z toho a ze mozeme hladat za pomoci gradientu a backtrackingu a po istom pocte novych generacii najdeme riesenie v najlepsoom jedincovi

2.3.1 Počiatočná generácia

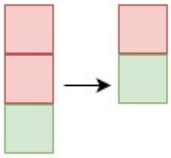
Ako bolo vyššie spomenuté, v evolučnom algoritme sa na začiatku vytvára skupina jedincov. Táto prvotná skupina sa nazýva počiatočná generácia. Vytváranie tejto generácie prebieha za pomoci náhody a teda jednotlivé informácie, chromozómy, o jedincovi sú nastavené náhodne. Napríklad keď jedinec bude mať informáciu o veku v chromozómoch, bude sa na začiatku nastavovať pre každého jedinca náhodne z určitého intervalu, pre náš príklad to bude interval od 0 po 10. Takže každý jedinec bude mat vek v rozmedzí od 0 rokov po 10 rokov.

2.3.2 Mutácia

Mutácia je operácia, ktorá sa deje vrámci jedinca. Pri tejto akcii je na začiatku jedinec skladajúci sa z určitého počtu chromozómov. Ako sme už spomenuli vyššie, chromozómy si vieme predstaviť ako lineárny reťazec, ktorý kóduje informácie o jedincovi. Zmenou týchto chromozómov vieme vylepšovať jedincov a ich schopnosti reprodukcie a prežitia. Pri mutácii si z tohto reťazca náhodne vyberieme jednu pozíciu, ktorú upravíme. Existujú 4 spôsoby ako ju upraviť:

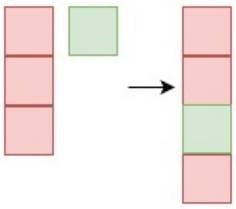
1. vymazanie chromozómu
2. pridanie nového chromozómu
3. inverzia chromozómu
4. nahradenie iným chromozómom

Jednotlivé spôsoby si teraz prejdeme. Ako prvé je vymazanie chromozómu. Tento spôsob je zobrazený na obrázku 2.2. Na ľavej strane môžeme vidieť jedinca obsahujúceho všetky chromozómami s vyznačeným chromozómom, vybratým na vymazanie, zatiaľ čo na pravej strane je už nový jedinec bez jednej časti.



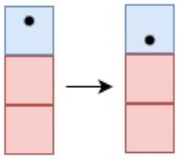
Obr. 2.2: Spôsob mutácie - vymazanie chromozómu

Ďalším spôsobom je pridanie nového chromozómu, ktoré nájdeme na obrázku 2.3. Opäť na ľavej strane máme jedinca a ešte nejaký nový chromozóm, ktorý chceme pridať a na pravej strane je už nový jedinec rozšírený o nový chromozóm.



Obr. 2.3: Spôsob mutácie - pridanie nového chromozómu

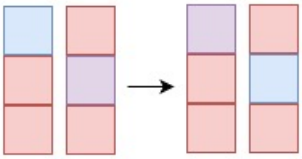
Inverzia chromozómu je na obrázku 2.4 kde na jednej strane máme jedinca a zvýraznený chromozóm, ktorý chceme invertovať a na pravej je už zmenený chromozóm nachádzajúci sa na tom istom mieste ale opačný.



Obr. 2.4: Spôsob mutácie - inverzia chromozómu

Hlavným spôsobom, ktorým sa my v našej práci budeme zaujímať, je nahradenie iným chromozómom. Na obrázku 2.3 môžeme vidieť na jednej strane dva jedince a vyznačené

chromozómy, ktoré sa majú vymeniť. Na ľavej strane už vidíme výsledok, kde sú opäť dva jedince ale s vymenenými chromozómami, ktoré sme si určili.



Obr. 2.5: Spôsob mutácie - nahradenie iným chromozómom

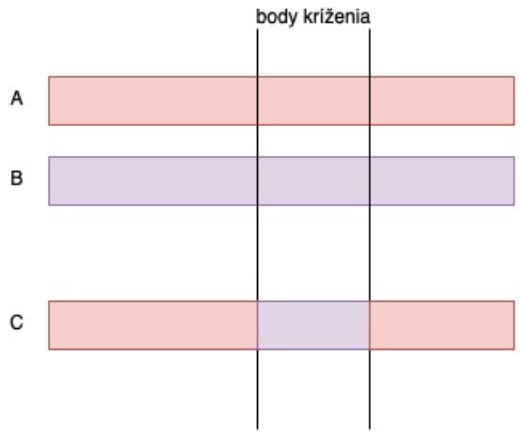
2.3.3 Kríženie

Kríženie je genetická operácia medzi dvomi jedincami, ktorej výsledok je nový jedinec. Nech máme na vstupe do tejto operácie dvoch jedincov - A a B. Oba tieto jedince majú x chromozómov. Chromozómy môžu ale nemusia byť rozličné. Urči sa miesto, v ktorom sa jedince budú krížiť. Potom vytvárame nového jedinca C tak, že po daný bod kríženia vyberieme chromozómy nachádzajúce sa na tých pozíciach z jedinca A, po bode kríženia pridáme chromozómy z rovnakým spôsobom z jedinca B. Celý tento proces môžeme vidieť na obrázku 2.6.

Obr. 2.6: Kríženie dvoch jedincov s jedným bodom kríženia

Bod kríženia si vieme určiť stále rovnaký alebo ho náhodne vyberieme z dĺžky jedinca.

V niektorých prípadoch sa môže použiť aj viac bodov kríženia. Vtedy postupujeme podobne, po prvý bod vyberáme z jedinca A, po druhý bod kríženie vyberieme chromozómy z jedinca B, pri treťom bode zas z jedinca A a tak ďalej až kým nedôjdeme na koniec dĺžky jedincov a počtu bodov kríženia. Kríženie dvoch jedincov s viacerými bodmi kríženia môžeme vidieť na obrázku 2.7.



Obr. 2.7: Kríženie dvoch jedincov s dvomi bodom kríženia

2.3.4 Ohodnocovacia funkcia

Pre výpočet pravdepodobnosti reprodukcie a teda vyhodnotenia aký je jedinec schopný prežitia slúži ohodnocovacia funkcia. Ohodnocovacia funkcia berie do úvahy každý chromozóm, ktorý sa v jedincovi nachádza. Postupne teda prechádza chromozómy a jednotlivo ich ohodnocuje ako moc sú dobré. Tieto hodnoty sa navzájom spočítajú a dostaneme na konci hodnotu ako je daný jedinec schopný prežitia do ďalšej generácie. Hodnotu môžeme daným algoritmom buď sa snažiť minimalizovať alebo maximalizovať.

Napríklad ak má byť ideálny vek jedinca na prežitie je 10 rokov a daný jedinec má v chromozóme hodnotu 5 a ak máme funkciu pre tento chromozóm absolútnu hodnotu rozdielu ideálneho veku a veku jedinca, tak funkcia vráti hodnotu 5.

Kapitola 3

Predchádzajúce riešenia

Na problém hľadania mapovania existujú rôzne riešenia pre ohodnocovaciu funkciu . Tieto prístupy si vysvetlíme v nasledujúcich podkapitolách 3.2, 3.3, 3.4.

Všetky tieto prístupy pracujú s dvomi grafmi na vstupe. Chceme jednotlivé vrcholy a hrany namapovať na seba. Toto mapovanie budeme hľadať na základe susedov vrcholov a referencií susedov na ďalšie vrcholy. Nie vždy musíme nájsť najlepšie mapovanie, hľadáme podobnosť prvkov.

Hodnotu ohodnocovacej funkcie sa pri týchto prístupoch budeme snažiť maximalizovať a teda hľadať, čo najviac podobný vrchol, na ktorý by sme mohli namapovať.

Tieto algoritmy nehľadajú najlepšie mapovanie, ale nájdu všetky mapovania medzi vrcholmi a ich vzájomné podobnosti.

Náš problém sa dá taktiež zapísať ako hľadanie mapovania grafov, nájdenie podobnosti vrcholov. V našom prípade by vrcholy boli riadky tabuliek a hrana medzi nimi bude reprezentovať informáciu ak sa daný riadok odkazuje na druhý riadok. Takto vytvorený graf by reprezentoval danú databázu.

3.1 Kuhn-Munkresov algoritmus

Tento optimalizačný algoritmus funguje na hľadanie maximálneho mapovania v polynomiálnom čase.

3.2 SimRank

SimRank je algoritmus, ktorý hľadá podobnosť dvoch grafov, jeho vrcholov. Nehľadá jedno najlepšie mapovanie ale nájde podobnosť medzi všetkými vrcholmi. Funguje na princípe dva vrcholy sú si podobné ak odkazujú na rovnaké objekty // TODO cite na clanok. SimRank pracuje s maticou veľkosť n x n, kde n je počet vrcholov v grafe. Hodnoty v matici budú z rozmedzia od 0 po 1, kde 1 je hodnota hovoriaca, že objekty sa maximálne na seba podobajú (napríklad vrchol sám sa seba keby sa namapoval, dostal by v matici hodnotu 1).

Ako prvý krok tohto algoritmu je nainicializovanie matice, kde každý prvok matice bude mať hodnotu 0. Následne túto maticu zmeníme tak, že na diagonálu dáme 1, nakoľko na diagonále máme mať hodnotu ako veľmi sa daný vrchol podobá sám na seba.

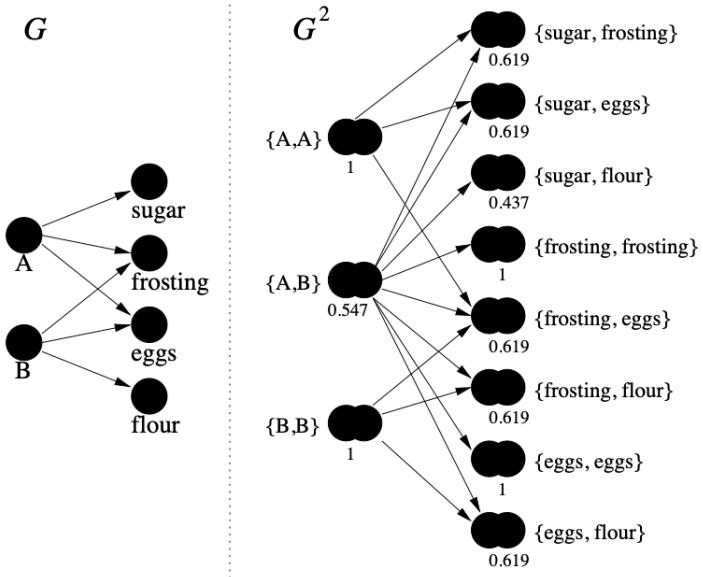
Následne budeme rekurzívne počítať pre skóre vrcholy A a B z matice tak, že ak $A == B$ tak $sA, B = 1$, inak podľa vzorca 3.1, kde $I(A)$ sú všetky susedné vrcholy vrcholu A, C je konštanta z rozmedzia od 0 po 1. Takto budeme rátať skóre pre každú dvojicu vrcholov až kým hodnoty neskonvergujú.

$$s(A,B) = \frac{C}{|I(A)| * |I(B)|} \sum_{i-1}^{|I(A)|} \sum_{j-1}^{|I(B)|} s(I_i(A), I_j(B)) \tag{3.1}$$

Na obrázku 3.1 môžeme vidieť príklad, kde na pravej strane vidíme graf G obsahujúci dve osoby A,B a ich nákupný zoznam. Pre osobu A je to cukor, poleva a vajíčka, pre osobu B je to vajíčka, múka a poleva. Hľadáme mapovanie medzi osobami vzájomne a medzi jednotlivými položkami nákupného zoznamu. Osoby sú si podobné na základe nákupných zoznamov, veci z nákupného zoznamu sú si podobné na základe osôb, ktoré ich kúpili /todo cite na clanok. Na riešenie tohto problému sa môže použiť algoritmus SimRank.

Na pravej strane už vidíme výsledný graf s mapovaním každého prvku so všetkými možnosťami namapovania. Prvky, ktoré sa namapujú samé na seba majú hodnotu 1 nakoľko je to maximálna hodnota, ktorú je možné dostať. Na vyrátania zvyšných podobností sme použili vzorec 3.1, kde konštanta $C = 0.8$. Vypočítalo sa a po pár iteráciach sa dostaneme k výsledku, ktorý môžeme vidieť na obrázku 3.1. Napríklad cukor s vajíčkami majú väčšiu pravdepodobnosť (0.619) než cukor s múkou, nakoľko cukor a vajíčka sa nachádzajú v oboch zoznamoch a teda sú kupované obomi osobami. Zato múka a cukor

sú v rôznych zoznamoch a nemajú spoločných nakupujúcich osôb.



Obr. 3.1: Príklad grafu a nájdených hodnôt podobnosti

3.3 MatchSim

MatchSim je algoritmus na hľadanie mapovania. Tento algoritmus bol vytvorený na hľadanie podobnosti webových stránok. MatchSim funguje rekurzívne a definuje podobnosť medzi dvomi webovými stránkami na základe priemeru podobnosti najviac podobných susedov /todo cite. Pracuje teda podobne ako SimRanku /todo cite alebo ref ale vynecháva najlepšie možné mapovanie a získava tým lepšie výsledky. Na nájdenie najlepšieho mapovania využíva Kuhn-Munkresov algoritmus. Viac o tomto algoritme nájdete v podsekcií **??**

Algoritmus opäť pracuje s maticou, ako SimRank /todo ref cite alebo co. Rovnako tak, na začiatku sa matica nainicializuje na 0 v každom prvku , na diagonálu sa nastavia 1. Následne sa prepočítavajú všetky prvky matice na základe vzorca 3.2, kde $\widehat{W}$ je hodnota maximálneho skóre z Kuhn-Munkresovho algoritmu **??** a $I(A)$ sú susedia vrcholu A.

$$sim(A, B) = \frac{\widehat{W}}{max(I(A), I(B))} \tag{3.2}$$

3.4 Similarity Flooding

Similarity flooding algortimus taktiež pracuje s dvomi grafmi na vstupe a hľadá mapovania medzi korešpondujúci vrcholmi.

Tento algoritmus pracuje v 3 krokoch, ktoré si následne vysvetlíme:

1. zmena na graf párovej konektivity
2. indukčný propagačný graf
3. hodnoty pevného bodu

Zmena vstupného grafu na graf párovej konektivity je definovaná nasledovne:

$$((x,y),p,(x',y') \in PCG(A,B) \iff (x,p,x') \in A \ \& \ (y,p,y') \in B \tag{3.3}$$

A teda ak x ma suseda x' a hrana medzi nimi je p, a y ma suseda y' s hranou p , tak vytvorí nový vrchol (x, y) s hranou p do (x', y'). Následne sa hrany rekurzívne pospájajú.

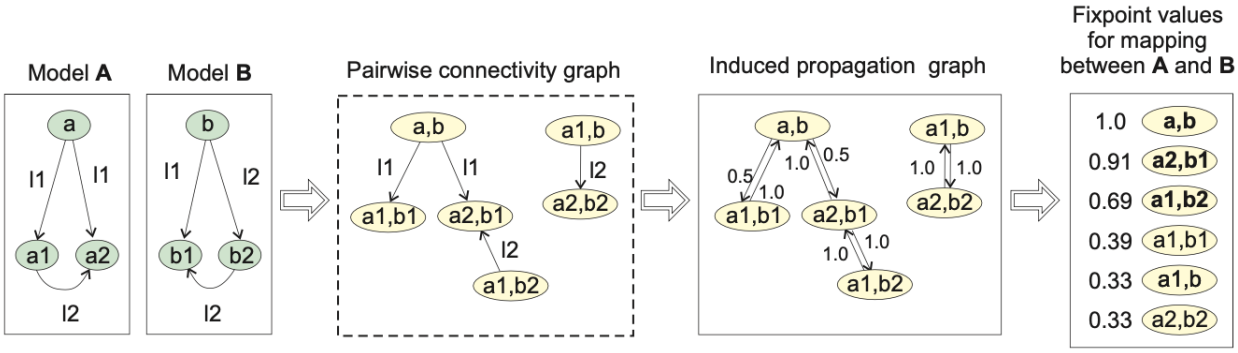
Indukčný propagačný graf pridá k zadaným hranám aj k nim opačné a namiesto zadaných váh na hranách dá pravdepodobnosti nasledujúcim spôsobom: ak z daného vrcholu vychádzajú viaceré hrany, hodnota 1 sa rovnomerne rozdelí medzi nich. Napríklad ak z vrcholu A idú dve hrany, vytvoria sa do vrcholu A taktiež dve hrany. Keďže hrany sú dve, každá z nich bude mať váhu 0.5 v smere od vrchola A a naspäť bude mať hodnotu 1.

V poslednom kroku sa vyrátajú už len hodnoty pevného bodu z propagačného grafu. Rátanie sa uskutočňuje v iteráciach podľa vzorca a následne sa normalizujú podľa vzorca /todo ref

(3.4)

Daný postup si teraz uvedieme na príklade, ktorý môžeme vidieť na obrázku 3.2. Máme dva modely A a B. V prvom kroku, vytvoríme graf párovej konektivity a teda vytvoríme nové vrcholy, ktoré sú si vzájomne podobné. Napríklad vytvoríme nový vrchol *a,b*, čo značí podobnosť vrcholov a a b nakoľko z oboch vedie hrana l1. Keďže model A mal dve hrany vedúce z vrcholu a , aj nový graf bude mať dve hrany s váhou l1 a dva nové vrcholy *a1,b1* a *a2,b1* nakoľko pre ne platí definicia 3.3. Takto postupne sa vytvorí celý graf. Nasledujúci krok je propagačný graf , kde hranám pridáme aj spätné hrany

medzi vrcholmi a dopočítame váhy. Opäť napríklad z vrcholu *a,b* vedú dve hrany a to do *a1,b1* a *a2,b1*. Tieto hrany budú mať váhu 0.5 keďže hodnotu 1 delíme medzi dve hrany. v opačnom smere z *a1,b1* do *a,b* bude mať hodnotu 1 nakoľko iná hrana v danom smere nie je. Takýmto postupom sa to zopakuje pre všetky hrany a vrcholy. Následne už len v iteráciach vypočítame hodnoty pevného bodu, ktoré znormalizujeme podľa vzorcov /todo ref. Vo výsledku môžeme vidieť hodnoty všetkých mapovaní podľa vrcholov.



Obr. 3.2: Príklad grafu a nájdených hodnôt podobnosti

Kapitola 4

Návrh modelu

Kapitola 5

Implementacia

konkretna moja implementácia

Kapitola 6

Výskum

Vyhodnotenie nášho algoritmu sme sa rozhodli otestovať pomocou testov, kde na vstupe budú dve databázy obsahujúce jedna správne riešenie a druhá nie vždy správne. V druhej databáze budeme regulovať počet nesprávne vložených riadkov a teda budeme tým simulovať situáciu kedy študent vloží nesprávne riadky a čaká na vyhodnotenie.

Na prvý test využijeme databázu obsahujúcu 3 tabuľky:

1. zamestnanci
2. oddelenia
3. pracovné pozície

Tabuľka zamestnancov obsahuje informácie o zamestnancoch ako sú meno, priezvisko, oddelenie odkazujúce sa na tabuľku oddelení a pracovnú pozíciu z tabuľky pozícií. Tabuľka pracovných pozícií obsahuje len názvy jednotlivých pozícií. Posledná tabuľka oddelení obsahuje názov oddelenia a odkazuje do tabuľky zamestnancov na vedúceho oddelenia.

V počte riadkov vložených do jednotlivých tabuliek nebudeme v našom prvom teste pracovať na väčších dátach. Tabuľka zamestnancov bude mať 90 riadkov, oddelení bude mať 10 a rovnako tak aj pracovných pozícii.

Takéto tabuľky budú obsahovať obe databázy na vstupe. Študentovú a teda potenciálne nesprávnu databázu budeme vytvárať za pomoci správnej avšak pri jednotlivých spusteniach budeme meniť riadky v tabuľke zamestnancov. Budeme do nich vkladať nesprávne riadky postupne a teda najskôr budú všetky správne, potom zmeníme dva na nesprávne, neskôr 4, 6, 8, 10. Maximálny počet iterácii nastavíme na 500.

Zaznamenávať budeme, v ktorej iterácii našiel náš algoritmus správne mapovanie, s čo najlepšou penalizáciou. Keďže algoritmus pracuje s náhodou, tieto pustenia zopakujeme niekoľkokrát a priemerne výsledky zaznamenáme do tabuľky. Výsledky môžeme vidieť v tabuľke 6.1.

//TODO nejaky opis vysledkov

počet nesprávnych vložení	počet iterácií na nájdenie riešenia
0	453
2	461
4	433
6	490
8	396
10	500

Tabuľka 6.1: Iteračná zložitosť na základe počtu nesprávne vložených riadkov na menších dátach

Druhé testovanie sme uskutočnili na väčších databázach. Pri tomto testovaní budeme vytvárať nasledovné tabuľky:

1. zamestnanci
2. oddelenia
3. pracovné pozície
4. história pracovných pozícii
5. regióny
6. krajiny
7. lokácie

Tabuľka zamestnancov obsahuje informácie o zamestnancoch a to meno, priezvisko, e-mail, telefónne číslo, dátum nástupu, plat, commission pct, oddelenie odkazujúce sa na tabuľku oddelení, pracovnú pozíciu z tabuľky pozícií a jeho nadriadeného z tabuľky zamestnancov. Tabuľka pracovných pozícií obsahuje názvy jednotlivých pozícií, minimálny

plat a maximálny plat. Tabuľka história pracovných pozícii obsahuje odkaz na zamestnanca, ktorého sa údaje týkajú, začiatočný a konečný dátum, odkaz na oddelenie a pracovnú pozíciu. Tabuľka oddelení obsahuje názov oddelenia a odkazuje do tabuľky zamestnancov na vedúceho oddelenia a lokácie z tabuľky lokácii. Tabuľka regiónov obsahuje iba názvy regiónov (napríklad Európa). Ďalšia je tabuľka krajín ktorá obsahuje informácie o názve krajiny a odkaz na región, do ktorého patrí. Posledná je tabuľka lokácií, ktorá obsahuje názov ulice, poštové smerovacie číslo, mesto, názov provincie a odkazuje na krajinu z tabuľky krajín.

V jednom teste na takejto databáze budeme rovnako ako v prvom teste, vkladať nesprávne riadky do tabuľky zamestnancov. Vkladať budeme rovnakým spôsobom a teda najskôr 0 nesprávnych riadkov, potom 2, 4, 6, 8, 10, 15, 20, 25. Merania opäť niekoľkokrát zopakujeme a zaznamenáme priemerný výsledok. Výsledky môžeme vidieť v tabuľke 6.2
//TODO nejaky opis vysledkov

V ďalšom teste sme sa rozhodli vkladať nesprávne riadky do rôznych tabuliek. A teda najskôr vyskúšame vložiť nesprávne riadky do tabuľky zamestnancov a do ostatných nie, potom do tabuľky oddelení a do ostatných nie a takto to vyskúšame na všetkých tabuľkách. Tento test robíme preto, aby sme zistili či súvisí rýchlosť nájdeného riešenia s odkazmi v tabuľkách, niektoré obsahujú viac cudzích klúčov. Opäť tieto testy niekoľkokrát zopakujeme aby sme dostali priemerný výsledok, v ktorej iterácií sme našli riešenie. Výsledky môžeme vidieť v tabuľke 6.3.

//TODO nejaky opis vysledkov

Keďže naše výsledky neboli nájdené v čase ako sme si na začiatku určili rozhodli sme sa vyskúšať urobiť zmeny vo vytváraní počiatočnej generácie.

počet nesprávných vložení	počet iterácií na nájdenie riešenia
0	572
2	539
4	524
6	476
8	426
10	519
15	483
20	489
25	490

Tabuľka 6.2: Iteračná zložitosť na základe počtu nesprávne vložených riadkov v tabuľke zamestnancov na dátach z predmetu Databázy(1)

zamestnanci	oddelenia	prac. pozície	regióny	krajiny	lokácie	his. prac. pozícií	iterácia
5	0	0	0	0	0	0	501
0	5	0	0	0	0	0	544
0	0	5	0	0	0	0	514
0	0	0	5	0	0	0	582
0	0	0	0	5	0	0	562
0	0	0	0	0	5	0	551
0	0	0	0	0	0	5	497

Tabuľka 6.3: Iteračná zložitosť na základe počtu nesprávne vložených riadkov

Kapitola 7

Vysledky

Záver

Literatúra

[1] Glen Jeh and Jennifer Widom. Simrank: A measure of structural-context similarity. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2002.

[2] V. Kvasnička, J. Pospíchal, and P. Tiňo. *Evolučné algoritmy*. Edícia vysokoškolskỳch učebníc / Slovenská technická univerzita. Slovenská technická univerzita, 2000.

[3] Zuqing Li, Bernard Chen, and Dongsheng Che. Solving the subgraph isomorphism problem using simulated annealing and evolutionary algorithms, 2016.

[4] Zhenjiang Lin, Michael Lyu, and Irwin King. Matchsim: A novel similarity measure based on maximum neighborhood matching. *Knowledge and Information Systems*, 32, 11 2009.

[5] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. pages 117 – 128, 02 2002.

[6] Akshitha Puri, Gunveen Batra, and K. B. Ajitha Shenoy. Performance comparison of randomized local search, (1+1)-evolutionary algorithm and genetic algorithm for graph isomorphism problem using permutation matrix search space. *Engineering Letters*, 30(2), May 2022. Publisher Copyright: © 2022, International Association of Engineers. All rights reserved.

[7] Wensi Xi, Edward Fox, Weiguo Fan, Benyu Zhang, Zheng Chen, Jun Yan, and Dong Zhuang. Simfusion: measuring similarity using unified relationship matrix. 08 2005.

Zoznam obrázkov

Zoznam tabuliek