Putá medzi formálnymi kontextami a ich efektívne vyhľadávanie

Analýza a návrh riešenia

Bc. Marián Kozák

1Im, 2018 - 2019

Abstrakt. Našim cieľom je prispôsobiť a implementovať známe algoritmy pre vyhľadávanie formálnych konceptov pre potreby vyhľadávania pút medzi dvoma formálnymi kontextami. Na začiatok sa zameriame na vyhľadávanie rôznych algoritmov pre vyhľadávanie konceptov jedného kontextu. Tieto algoritmy sa pokúsime prispôsobiť tak, aby namiesto konceptov jedného kontextu vyhľadávali putá dvoch kontextov. Tieto algoritmy budeme po implementácii testovať a porovnávať na základe dosiahnutých výsledkov a časovej zložitosti. Na záver našej práce sa zameriame na preskúmanie niektorých vlastností pút.

Kľúčové slová: kontext, koncept, algoritmus Norris, algoritmus cez backtrack extentov, puto, duálne puto

1 Úvod

Vyhľadávanie spojitostí medzi dvoma kontextami je oblasť, ktorej sa venuje málo ľudí. Ide o zhlukovacie algoritmy, ktoré sú súčasťou nášho každodenného života. Naša práca bude vychádzať z našej bakalárskej práce, v ktorej sme sa venovali nájdeniu algoritmu, ktorý bude vyhľadávať tieto spojenia medzi kontextami. Hoci sme našli algoritmy, ktoré boli rýchlejšie, ako naivný algoritmus, napriek tomu nenašli všetky duálne putá medzi kontextami pri väčšom počte objektov a atribútov. Konkrétne sme sa venovali prerobeniu známych algoritmov Object Intersection na vyhľadávanie konceptov z kontextu a algoritmu Next Neighbor na algoritmus vyhľadávania pút medzi dvoma kontextami. Týmto smerom by sme chceli pokračovať aj teraz, pričom sa viac pozrieme na vlastnosti daných kontextov a pút, prerobíme nové algoritmy, ktoré vyhľadávajú koncepty z kontextov na algoritmy vyhľadávania pút, zlepšíme už navrhnuté algoritmy z bakalárskej práce a zameriame sa aj na zložitosť daných algoritmov. V prvej podkapitole sa budeme zaoberať súčasným stavom, základnými pojmami a algoritmami, ktoré medzi kontextami vyhľadávajú koncepty. Ciele našej práce si uvedieme v podkapitole 1.2.

1.1 Prehľad súčasného stavu

Vysvetlenie základných pojmov a využitie v praxi je dôležité uviesť hneď na začiatku. Pri tom budeme vychádzať z prác niekoľkých autorov, ktorý sa už zaoberali niektorými základnými vlastnosťami kontextov a zároveň navrhli nejaké algoritmy na vyhľadávanie konceptov medzi kontextami. Základným algoritmom, s ktorým sme pracovali aj v našej bakalárskej práci bol Object Intersection, ktorý uvádzajú Carpineto, Romano [1] vo svojej knihe. Takisto sme sa zaoberali algoritmom Next Neighbor, ktorý sa používa na hľadanie susedov, ktorý uvádzajú Bělohlávek, De Baets, Outrata, Vychodil [6]. Týmto algoritmom a ďalším novým algoritmom, ktoré vo svojom článku uvádzajú Kuznetsov a Obiedkov [7] sa budeme zaoberať v podkapitole 1.1.2, a z ktorých budeme neskôr odvodzovať algoritmy na vyhľadávanie pút medzi kontextami. V podkapitole 1.1.3 sa budeme zaoberať tým, ako z dvoch kontextov vznikne puto a načo je dobré takéto spojenia medzi tabuľkami vôbec robiť.

1.1.1 Kontext, derivačný operátor, koncept

Našim prvým a najdôležitejším pojmom, ktorý budeme najčastejšie používať je formálny kontext, alebo zjednodušene kontext. Kontext je trojica (G, M, I) pozostávajúca z dvoch množín a to z množiny G a množiny M a z relácie I medzi G a M. Navyše Valtchev a Jäschke [3] dodávajú, že množiny G a M musia byť konečné a I je binárna relácia. Prvky množiny G nazývame objekty a množiny M atribúty. Vzťah gIm tak znamená, že objektu g je priradený atribút m. Menší kontext tak môžeme reprezentovať ako tabuľku, kde riadky obsahujú názvy objektov a stĺpce obsahujú názvy atribútov [2].

V jednej triede máme piatich žiakov: Marek, Ján, Eva, Peter, Ema, ktorí majú tieto predmety: fyzika, matematika, biológia, geografia, chémia, dejepis. Množinu objektov v tomto prípade tvoria študenti, ktorí sú reprezentovaný ich menami a množinu atribútov tvoria predmety, ktoré sú reprezentované ich názvami. V Tabuľke 1. riadky obsahujú mená študentov danej triedy a stĺpce obsahujú názvy predmetov. „X“ nám reprezentuje reláciu, že študenta daný predmet baví.

Tabuľka 1. Obľúbenosť predmetov u študentov

Študent/ Predmet	Fyzika	Matematika	Biológia	Geografia	Chémia	Dejepis
Marek	X	X				X
Ján			X	X	X	
Eva	X	X	X		X	
Peter				X	X	X
Ema	X	X	X	X	X	X

Ďalším pojmom, ktorý vychádza z kontextu je derivačný operátor. Pre množinu objektov A, ktorá je podmnožinou G definujme:

$$A' = \{m \in M \mid gIm \text{ pre všetky } g \in A\} . \qquad (1)$$

Podobne pre množinu atribútov B, ktorá je podmnožinou M definujme:

$$B' = \{g \in G \mid gIm \text{ pre všetky } m \in B\} . \quad (2)$$

Inými slovami, *A'* je množina atribútov spoločných pre objekty v *A*, kým *B'* je množina objektov, ktoré majú všetky atribúty v *B* [1].

Derivačné operátory si priblížime na príklade z Tabuľky 1. Zoberme si napríklad dve mená: Ema a Peter. Títo dvaja študenti, ktorí reprezentujú množinu *A*, majú radi geografiu, chémiu a dejepis. Tieto predmety v tomto prípade pridáme do množiny *A'*. Podobne je to aj napríklad s matematikou a fyzikou, ktoré reprezentujú množinu *B*. Tieto dva predmety majú radi iba Marek, Eva a Ema. Týchto študentov potom pridáme do množiny *B'*.

Koncept kontextu (*G*, *M*, *I*) je pár (*A*, *B*), kde:

$$A \subseteq G, B \subseteq M, A' = B \text{ a } B' = A. \quad (3)$$

Množinu *A* v tomto prípade nazývame extent a množinu *B* nazývame intent konceptu (*A*, *B*). Množinu všetkých konceptov kontextu (*G*, *M*, *I*) označujeme *C*(*G*, *M*, *I*). Taktiež platí, že množina *A*, ktorá je podmnožinou *G* je extentom konceptu, ak platí *A''* = *A*. Podobne je to s množinou *B*, ktorá je podmnožinou *M*, ktorá je intentom konceptu, ak platí *B''* = *B*.

Pre príklad si zoberme troch študentov Mareka, Evu a Emu z Tabuľky 1. Títo traja študenti majú radi fyziku a matematiku. Čiže množina *A* v tomto prípade obsahuje tri mená študentov, pričom *A'* obsahuje dva názvy predmetov fyziku a matematiku. Takisto množina *B* obsahuje názvy vyššie spomenutých predmetov a zároveň *B'* obsahuje tri mená a to Mareka, Evu a Emu, ktorí majú radi dané predmety. Takúto dvojicu, ktorá pozostáva z množiny *A* a množiny *B* nazývame koncept, pričom extentom daného konceptu je množina objektov *A* a intentom toho istého konceptu je množina atribútov *B*. V prípade, že množina *A* obsahuje mená všetkých študentov, tak množina *A'* bude prázdna. Koncept potom bude vyzerať nasledovne: ({Marek, Ján, Eva, Peter, Ema}, Ø).

1.1.2 Algoritmy na vyhľadávanie konceptov

V predchádzajúcej podkapitole sme si uviedli, ako vyzerá jednotlivý koncept daného kontextu. Ak však chceme nájsť všetky koncepty pre jeden kontext, tak použijeme niektorý z algoritmov, ktoré tento problém riešia. Carpineto a Romano [1] uvádzajú vo svojej knihe tri najznámejšie algoritmy: naivný algoritmus, algoritmus Object Intersection a algoritmus Next Closure. Základným algoritmom je naivný, keďže vyhľadáva koncepty kombináciou všetkých možných vytvorených podmnožín množiny objektov a množiny atribútov a zisťuje, či tieto dve podmnožiny tvoria koncept. Tento algoritmus je však veľmi neefektívny. Druhým algoritmom, ktorý lepšou časovou zložitosťou nájde množinu konceptov je algoritmus Next Closure. Ten je založený na usporiadaných podmnožinách množiny *G*. Ďalej sa týmto algoritmom však nebudeme zaoberať, pretože existujú ešte algoritmy, ktoré majú oveľa lepšiu časovú zložitosť.

Algoritmom Object Intersection sme sa zaoberali v našej bakalárskej práci a budeme sa ním zaoberať aj v diplomovej práci, preto si ho rozoberieme podrobnejšie a uvedieme si jeho náčrt kódu. Tento algoritmus je založený na pozorovaní, že každý extent z konceptu je prienikom extentov atribútov a každý intent z konceptu je prienikom intentov objektov. Na generovanie množiny konceptov, stačí vytvoriť všetky možné prieniky medzi množinami objektov pridruženými ku každému atribútu a využiť kontext na nájdenie intentu, ktorý sa zhoduje s každým vygenerovaným extentom konceptu [1].

Na začiatku si vytvoríme množinu *C*, do ktorej si budeme naše koncepty pridávať. Hneď potom do tejto množiny pridáme základný koncept, ktorý pozostáva z množiny atribútov *M* a z množiny *M'*. Ďalej prechádzame objektami z množiny objektov *G* a všetkými konceptami, ktoré sa nachádzajú v množine *C*. Z objektu *g* si vytvoríme *{g}'*. Medzi touto množinou a množinou atribútov *B* z vybraného konceptu urobíme prienik, ktorý si označíme *Inters*. Tento prienik nasledovne porovnáme s intentami konceptov. Ak je *Inters* rôzny od všetkých týchto intentov, tak do množiny konceptov pridáme množinu *Inters* a množinu *Inters'*. Týmto spôsobom nájdeme všetky koncepty daného kontextu (*G*, *M*, *I*). Tento algoritmus sme naprogramovali podľa návrhu kódu, ktorý uvádzajú Carpineto a Romano [1]. Ak tento algoritmus aplikujeme na Tabuľku 1, tak po skončení algoritmu nám vznikne množina konceptov, ktorá má 11 konceptov.

Algoritmus 1. Algoritmus Object Intersection z práce [1]

```
program ObjectIntersection
  Input: Kontext (G, M, I)
  Output: Množina C všetkých konceptov (G, M, I)

  1. C = {(M', M)};
  2. for each g ϵ G
  3.    for each (X, Y) ϵ C
  4.       Inters = Y ∩ {g}';
  5.       if Inters je rôzne od všetkých intentov z C
  6.          C = C ∪ {(Inters', Inters)};
```

Niekoľko ďalších algoritmov na vyhľadávanie konceptov pre daný kontext vo svojom článku napísali Kuznetsov a Obiedkov [7]. Zároveň k týmto algoritmom uvádzajú ich časovú zložitosť a podľa nej ich porovnávajú. My sme si vybrali taký algoritmus, ktorý má najlepšiu časovú zložitosť pre rôzne počty objektov a atribútov, a tým je algoritmus Norris.

Časová zložitosť algoritmu Norris je $O(|G|^2*|M|*|L|)$, pričom $|G|$ je počet objektov, $|M|$ je počet atribútov a $|L|$ je počet konceptov. Algoritmus Norris bol prebratý v roku 1978 z algoritmu Close by One, ktorý má časovú zložitosť taktiež $O(|G|^2*|M|*|L|)$ a polynomiálne omeškanie je $O(|G|^3*|M|)$, pričom sa v ňom používa lexikografické usporiadanie. Pre porovnanie týchto dvoch algoritmov si uvedieme obidve kódy, ktoré sú si podobné.

Algoritmus 2. Algoritmus Close by One z práce [7]

```
program Close by One
  Input: Kontext (G, M, I)
  Output: Množina L všetkých konceptov (G, M, I)

  1. L = Ø;
  2. for each g ϵ G
  3.    Process({g}, g, ({g}'', g));
  4. L je množina konceptov

program Process
  Input: množina objektov A, objekt g, koncept (C,D)
  Output:

  1.if {h|hϵC\A & h←g} = Ø;
  2.   L = L ∪{(C, D)};
  3.   for each f ϵ {h|hϵG & g←h}
  4.      Z = C ∪{f};
  5.      Y = D ∩ {f}';
  6.      X = Y'(=Z∪{h|hϵG\Z & Y je podmnožinou {h}'});
  7.      Process(Z, f, (X, Y));
```

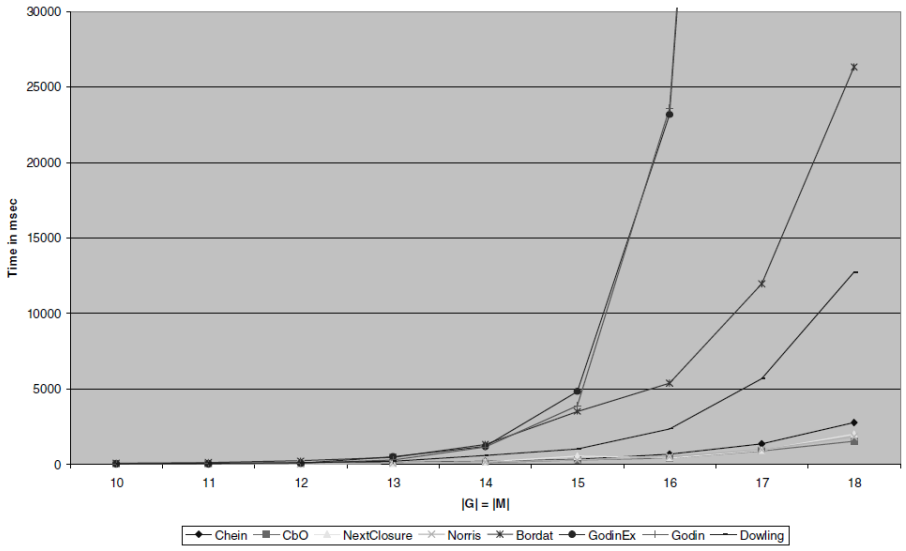
Algoritmus Close by One využíva pomocnú metódu Process, ktorá sa volá rekurzívne a využíva lexikografické usporiadanie definované na podmnožinách objektov. Tento algoritmus taktiež konštruuje diagram, ktorý my nepotrebujeme. Podobným algoritmom je algoritmus Norris, ktorý vychádza z algoritmu Close by One. On nájde množinu konceptov, ale nekonštruuje diagram. Tento algoritmus podobne pozostáva z metódy Norris a z pomocnej metódy Add, ktorá rekurziu nevyužíva.

Algoritmus 3. Algoritmus Norris z práce [7]

```
program Norris
  Input: Kontext (G, M, I)
  Output: Množina L všetkých konceptov (G, M, I)

  1. L = Ø;
  2. for each g ϵ G
  3.    Add(g, L);
  4. L je množina konceptov

program Add
  Input: objekt g, množina konceptov L
  Output: množina konceptov L

  1.for each (A, B)ϵ L
```

```
  2.   if B je podmnožinou {f}'
  3.      A = A ∪{g};
  4.   else
  5.      D = B ∩ {g}';
  6.      if {h|h∈G\A & h už bolo pridané & D je
podmnožinou {h}'} = Ø;
  7.         L = L ∪{(A∪{g}, D)};
  8. if {h|h∈G & h už bolo pridané & {g}' je
podmnožinou {h}'} = Ø;
  9.    L = L ∪({g}, {g}');
```

Algoritmus Norris sme si vybrali preto, že zo všetkých algoritmov na vyhľadávanie konceptov mal najmenšiu časovú zložitosť. Kuznetsov a Obiedkov vo svojom článku [7] uvádzajú aj porovnanie jednotlivých algoritmov na vyhľadávanie konceptov, ktorým na vstup dávajú náhodne vygenerované dáta a zisťujú, ktorý z algoritmov bol najrýchlejší. V nasledujúcom obrázku môžeme vidieť, že algoritmy Norris a Close by One majú skoro rovnakú časovú zložitosť.



Obrázok 1. Porovnanie algoritmov na vyhľadávanie konceptov, kde |G| = |M| v práci [7]

Po implementácii algoritmu Norris nám nenašlo celú množinu konceptov, preto sme sa snažili tento algoritmus pozmeniť a zjednodušiť. Navrhli sme algoritmus, ktorý nám na niekoľkých náhodných vstupoch našiel celú množinu konceptov. Urobili sme niekoľko úprav v metóde Add, pričom metóda Norris z algoritmu ostala nezmenená. Našim ďalším krokom preto bude dokázať správnosť tohto algoritmu. Nižšie uvádzame návrh nášho upraveného algoritmu Norris.

Algoritmus 4. Upravený algoritmus Norris z práce [7]

```
program Upravený Norris
  Input: Kontext (G, M, I)
  Output: Množina L všetkých konceptov (G, M, I)

  1. L = Ø;
  2. for each g ϵ G
  3.    Add(g, L);
  4. L je množina konceptov

program Add
  Input: objekt g, množina konceptov L
  Output: množina konceptov L

  1.for each (A, B)ϵ L
  2.   if B nie je podmnožinou {f}'
  3.      D = B ∪{g}';
  4.      if {h|hϵG\A & h už bolo pridané & D je
podmnožinou {h}'} = Ø;
  5.         if {(A∪{g}, D) už bolo pridané do L} = Ø;
  6.            L = L ∪{(A∪{g}, D)};
  7. if {({g}, {g}') už bolo pridané do L} = Ø;
  8.    L = L ∪({g}, {g}');
```

1.1.3 Puto, duálne puto

Na zjednotenie dvoch kontextov (G, M, I) a (H, N, J) používame putá. Ide o spojenie množiny objektov G z jedného kontextu a množiny atribútov N z druhého kontextu, pričom platí, že relácia $R \subseteq G \times N$ a pre každý riadok daného puta platí, že je intentom konceptu z druhého kontextu a každý stĺpec je extentom konceptu z prvého kontextu. Navyše Kuznetsov, Schmidt [4] dodávajú, že $R \subseteq G \times N$ je puto z kontextov (G, M, I) a (H, N, J) vtedy a len vtedy, ak $I \cup R \cup J \cup (H \times M)$ je uzavretá relácia sumy kontextov (G, M, J) a (H, N, J). Pre dva kontexty existuje viacero pút. Puto môžeme interpretovať ako tabuľku.

Na vysvetlenie pojmu si zhotovíme ďalší kontext.

Tabuľka 2. Uprednostňované predmety na vysokoškolských odboroch

Odbor/ Predmet	Chémia	Biológia	Fyzika	Matematika	Geografia	Dejepis
Medicína	X	X	X			
Farmácia	X	X	X			
Technika			X	X		
Informatika				X		
Paleontológia				X	X	X

Keďže chceme vytvoriť puto, potrebujeme extenty z jedného kontextu a intenty z druhého kontextu. My sme sa rozhodli zobrať intenty z prvého kontextu (Tabuľka 1) a extenty z druhého kontextu (Tabuľka 2). Pút z dvoch kontextov môže vzniknúť niekoľko. My si vytvoríme jedno puto, v ktorom stĺpce obsahujú názvy predmetov z prvého kontextu a riadky obsahujú názvy odborov na vysokej škole z druhého kontextu. Z prvého kontextu si zoberieme intenty {Fyzika, Matematika, Biológia, Chémia}, {Geografia, Chémia, Dejepis}, {Dejepis}. Z druhého kontextu si zoberieme tieto extenty: {Medicína, Farmácia}, {Medicína, Farmácia, Technika}, {Technika}, {Technika, Informatika, Paleontológia}. V novovytvorenom pute platí, že extenty prvého kontextu tvoria riadky a intenty druhého kontextu tvoria stĺpce puta. Vytvorené puto interpretujeme ako tabuľku.

Tabuľka 3. Puto vytvorené z objektov druhého kontextu a atribútov z prvého kontextu

Odbor/ Predmet	Fyzika	Matematika	Biológia	Geografia	Chémia	Dejepis
Medicína	X	X	X		X	
Farmácia	X	X	X		X	
Technika				X	X	X
Informatika						X
Paleontológia						X

Duálne puto je definované podobne ako puto medzi kontextami (G, M, I) a (H, N, J). Je to relácia $R \subseteq G \times H$, pričom pre každý riadok duálneho puta platí, že je extentom druhého kontextu a pre každý stĺpec platí, že je extentom prvého kontextu [5]. Ide vlastne o spojenie množín objektov dvoch vybraných kontextov. Duálnych pút pre dva kontexty existuje niekoľko. V našom prípade z prvého kontextu (Tabuľka 1) a z druhého kontextu (Tabuľka 2) vieme nájsť 875 duálnych pút. My si vytvoríme jedno duálne puto. Z prvého kontextu si zoberieme tieto extenty: {Marek, Eva, Ema}, {Peter, Ema}, {Ján, Eva, Peter, Ema}. Takisto si z druhého kontextu zoberieme tieto extenty: {Medicína, Farmácia}, {Medicína, Farmácia, Technika}, {Technika}, {Technika, Informatika, Paleontológia}, {Medicína, Farmácia, Technika, Informatika, Paleontológia}. Z nich si vieme vytvoriť takéto duálne puto:

Tabuľka 4. Duálne puto vytvorené z objektov prvého a druhého kontextu

Odbor/ Predmet	Medicína	Farmácia	Technika	Informatika	Paleontológia
Marek	X	X			
Ján			X		
Eva	X	X	X		
Peter			X	X	X
Ema	X	X	X	X	X

1.2 Ciele práce

Našim prvým cieľom je naštudovanie nových algoritmov na vyhľadávanie konceptov a výber takého algoritmu, ktorý má najlepšiu časovú zložitosť. Tento algoritmus následne prispôsobiť a implementovať pre potreby vyhľadávania pút medzi dvoma formálnymi kontextami. Ďalším cieľom je testovanie a porovnávanie jednotlivých prístupov, zistenie časovej zložitosti a dokazovanie daného algoritmu. Posledným cieľom je skúmanie niektorých vlastností pút.

2 Návrh riešenia

Nájdenie vhodného algoritmu pre vyhľadávanie pút medzi dvoma kontextami bude pozostávať z viacerých bodov, ktoré si v nasledujúcich podkapitolách podrobnejšie rozoberieme. Prvým bodom je úprava už nami navrhnutého algoritmu NextNeighborBond z bakalárskej práce a výpočet jeho časovej zložitosti. Druhým bodom je rozbor nami navrhnutých algoritmov, pričom jeden z nich používa pri vyhľadávaní stĺpcové putá a druhý používa backtrack. Tretím bodom je dokázať upravený algoritmus Norris na vyhľadávanie konceptov pomocou Hoareovej metódy dokazovania správnosti programu a prerobenie tohto algoritmu na algoritmus vyhľadávania pút medzi kontextami a taktiež výpočet jeho časovej zložitosti a posledným štvrtým bodom je porovnanie všetkých algoritmov na základe ich časovej zložitosti a správnosti, pričom na dokazovanie správnosti algoritmu budeme používať nami vytvorenú náhodnú tréningovú množinu, ktorú budeme aplikovať na všetky algoritmy. Tieto body si postupne vysvetlíme a doplníme v nasledujúcich podkapitolách, pričom do poslednej kapitoly zahrnieme aj náš posledný cieľ, ktorým je skúmanie niektorých vlastností pút.

2.1 Úprava algoritmov z bakalárskej práce

V bakalárskej práci sme sa snažili nájsť algoritmus, ktorý namiesto vyhľadávania konceptov z kontextu bude vyhľadávať putá medzi dvoma kontextami a nájde túto množinu pút za kratší čas ako naivý algoritmus, ktorý cez backtrack vygeneruje všetky možné tabuľky a zisťuje, či každý riadok je extentom jedného kontextu a každý stĺpec je extentom druhého kontextu.

Najprv sme sa snažili prerobiť už známy algoritmus Object Intersection na vyhľadávanie pút (Algoritmus 1.). Tento pokus bol neúspešný, keďže hoci našiel putá za kratší čas, nenašiel ich všetky. Druhý algoritmus, ktorý sme navrhli sme prerobili z algoritmu na výpočet horných susedov kontextu, ktorý vo svojom článku uvádzajú Bělohlávek, De Baets, Outrata, Vychodil [6]. Upravený algoritmus pozostáva z niekoľkých metód, ktoré si ukážeme.

Algoritmus 5. Algoritmus NextNeighborBond z práce [7]

```
program latticeOfBonds
  Input: Kontext C1(G1,M1,I1) a kontext C2(G2,M2,I2)
  Output: Množina všetkých duálnych pút F

  1. F = Ø;
  2. beta je prázdna tabuľka
  3. leftClosure(beta);
  4. rightClosure(beta);
  5. F = F ∪ {beta};
  6. generateFromBond(beta);

program generateFromBond
  Input: Duálne puto beta
  Output:

  1. upper = neighborsBond(beta)
  2. N = upper \ F
  3. všetky putá z N pridáme do F
  4. for each gama z N
  5.    generateFromBond(gama);


program generateFromBond
  Input: Duálne puto beta
  Output: Množina všetkých horných susedov U

  1. U = Ø;
  2. min1 = {b1 ϵ B1 | beta(b1) ≠ B2}
  3. min2 = {b2 ϵ B2 | transformovanaBeta(b2) ≠ B1}
  4. for each b1ϵ B1: beta(b1) ≠ B2}
  5.    for each gama ϵ moves1(beta, b1)
  6.       I = Ø;
  7.       for each o1 ϵ (B1\{b1}): beta(o1) ⊂ gama(o1)
  8.          I = I ∪{o1};
  9.       if min1 ∩ I = Ø then U = U ∪ {gama};
  10.      else min1 = min1\{b1};
  11. for each b2ϵ B2: transformovanaBeta(b2) ≠ B1}
  12.   for each gama ϵ moves2(beta, b2)
  13.      I = Ø;
  14.      for each o2 ϵ (B2\{b2}):
transformovanaBeta(o2) ⊂ gama(o2)
  15.         I = I ∪{o2};
  16.      if min2 ∩ I = Ø then U = U ∪
{transformovanaGama};
  17.      else min2 = min2\{b2}
```

```
program moves1
  Input: Riadok b1
  Output: Množina duálnych pút N

  1. un = horný susedia beta(b1)
  2. N = Ø;
  3. for each g ϵ un
  4.    gama = leftClosure(m1(beta, b1, g));
  5.    if gama je duálnym putom
  6.       N = N ∪{gama};

program moves2
  Input: Stĺpec b2
  Output: Množina duálnych pút N

  1. un = horný susedia beta(b2)
  2. N = Ø;
  3. for each g ϵ un
  4.    gama = rightClosure(m2(beta, b2, g));
  5.    if transformovanaGama je duálnym putom
  6.       N = N ∪{gama};

program leftClosure
  Input: Tabuľka gama
  Output: Tabuľka gama

  1. uzavretie stĺpcov gamy
  2. uzavretie riadkov gamy

program rightClosure
  Input: Tabuľka gama
  Output: Tabuľka gama

  1. uzavretie riadkov gamy
  2. uzavretie stĺpcov gamy
```

Algoritmus NextNeighborBond pozostáva z viacerých metód, pričom prvou základnou a spúšťajúcou metódou je metóda latticeOfBonds. Zo všetkých navrhnutých algoritmov tento algoritmus dával najlepšie výsledky. Hoci, čo sa týka času bol tento algoritmus pomalší ako predošlé navrhnuté algoritmy, ale stále bol o dosť rýchlejší ako naivný algoritmus. Výsledky tohto algoritmu boli oveľa lepšie ako výsledky predchádzajúcich algoritmov. Pri malých, aj veľkých tabuľkách našiel celú množinu duálnych pút. Jediným problém sú tabuľky, pri ktorých vzniká veľký počet pút. Pri tabuľke s piatimi objektmi a šiestimi atribútmi a druhej tabuľke so štyrmi objektmi a piatimi atribútmi nám naivný algoritmus vygeneroval 9361 duálnych pút. Náš navrhnutý algoritmus vygeneroval 9355 duálnych pút, čo je iba o 6 pút menej. V príklade so študentmi a predmetmi našiel všetkých 875 duálnych pút za 1,935

sekundy, čo je oveľa menej, ako čas naivného algoritmu, ktorý túto množinu pút našiel za 2591,768 sekúnd.

Tomuto algoritmu sa chceme naďalej venovať pretože čas, za ktorý našiel množinu pút bol rýchly a od celkovej množiny pút mu chýbalo iba zopár tabuliek. Taktiež sa budeme zaoberať aj časovou zložitosťou, pri ktorej tento algoritmus porovnáme spolu s ostatnými navrhnutými algoritmami.

2.2 Nové algoritmy na vyhľadávanie pút

Stĺpcové puto je veľmi podobné ako duálne puto. Stĺpcové puto medzi kontextami (G, M, I) a (H, N, J) je relácia R, ktorá je podmnožinou $H \times G$, pričom pre každý riadok stĺpcového puta platí, že je stĺpcom prvého kontextu a pre každý stĺpec platí, že je stĺpcom druhého kontextu. Naprogramovali sme metódu generateColumnBonds, ktorá nám generovaním všetkých možností vráti množinu všetkých stĺpcových pút dvoch daných kontextov. Tieto putá nám v našom navrhnutom algoritme slúžia ako vstupný parameter.

Algoritmus 6. Algoritmus dualBondsFromColumnBonds

```
program dualBondsFromColumnBonds
  Input: Stĺpcové putá S, extenty E1, extenty E2
  Output: Množina všetkých duálnych pút F

  1. F pridá plnú tabuľku
  2. for each A ϵ S
  3.    for each B ϵ F
  4.       C = A ∩ B;
  5.       if C je duálne puto
  6.          F = F ∪{C};
  7.       D = A ∪ B;
  8.       if D je duálne puto
  9.          F = F ∪{D};
```

Tento algoritmus je založený na robení prienikov a zjednotení medzi stĺpcovými a duálnymi putami. Hoci je rýchlejší ako naivný algoritmus, nenájde nám celú množinu duálnych pút.

Ďalším navrhnutým algoritmom je algoritmus extentBacktrack. Tento algoritmus vo svojej metóde používa backtrack. Rozdiel medzi ním a naivným algoritmom je ten, že naivný algoritmus generuje všetky možné tabuľky a zisťuje, či sú stĺpce extentami jedného kontextu a riadky extentami druhého kontextu a algoritmus extentBacktrack generuje všetky možné tabuľky, pričom tieto tabuľky generuje z extentov jedného kontextu, ktoré sú stĺpcami tabuľky a zisťuje, či riadky sú extentami druhého kontextu. Kód tohto algoritmu si uvádzať nebudeme, keďže ide o klasický backtrack cez extenty kontextu. Rýchlosť, za ktorú nájde všetky putá je oveľa lepšia, ako rýchlosť naivného algoritmu, dokonca je lepšia ako rýchlosť navrhnutého algoritmu dualBondsFromColumnBonds, keďže aj on využíva backtrack na vyhľadávanie

stĺpcových pút a vieme, že nájde všetky duálne putá. Tento algoritmus je zatiaľ najlepší správny algoritmus, ktorý sme našli. Ale keďže jeho časová zložitosť je stále dosť veľká, tak hľadáme algoritmus, ktorý nebude mať exponenciálnu časovú zložitosť.

2.3 Algoritmus Norris

Algoritmus na vyhľadávanie pút sme navrhovali tak, že sme si vybrali algoritmus na vyhľadávanie konceptov, ktorý sme sa snažili prerobiť. Snažili sme sa nájsť taký algoritmus, ktorý bol pre nás najjednoduchší. Tentokrát však chceme vybrať taký algoritmus, ktorý bude mať najlepšiu časovú zložitosť zo všetkých algoritmov na vyhľadávanie konceptov a z neho navrhnúť algoritmus na vyhľadávanie pút. Zo všetkých algoritmov, ktoré uvádzajú Kuznetsov, Obiedkov vo svojom článku [7] sme si vybrali algoritmus Norris (Algoritmus 3.) s najlepšou časovou zložitosťou $O(|G|^2*|M|*|L|)$. Keďže po implementácii tento algoritmus nenašiel celú množinu konceptov, upravili sme ho a zjednodušili, pričom tento upravený algoritmus nájde všetky koncepty daného kontextu. Návrh tohto algoritmu sme uviedli v podkapitole 1.1.2 (Algoritmus 4.).

Keďže nie je isté, či je tento navrhnutý algoritmus správny (Algoritmus 4.), rozhodli sme sa, že tento algoritmus dokážeme. Rozhodli sme sa ho dokázať pomocou Hoareovej metódy induktívnych podmienok. Na vyjadrenie sémantiky budeme používať tzv. induktívne výrazy, ktoré Hoare zaviedol, ktorý je dôležitý pri dokazovaní čiastočnej správnosti programu. To, že je program π čiastočne správny vzhľadom na vstupný predikát P a výstupný predikát Q teda znamená odvodiť induktívny výraz:

$$\{P\}\pi\{Q\} \ . \qquad \textbf{(4)}$$

Po dokázaní čiastočnej správnosti programu, chceme dokázať aj totálnu správnosť programu a to tak, že ak by sme význam induktívneho výrazu so špecifikáciami zmenili tak, že v každom príkaze budeme dokazovať okrem čiastočnej správnosti aj to, že program končí, tak tým by sme mohli dokázať totálnu správnosť programu. Teda induktívny príkaz (**4**) bude vyjadrovať nasledovné: Ak P platí bezprostredne pred vykonaním π, potom výpočet podľa π skončí a po skončení platí Q.

Po dôkaze tohto navrhnutého programu sa chceme zamerať na prerobenie tohto algoritmu na vyhľadávanie konceptov daného kontextu na algoritmus vyhľadávania duálnych pút medzi dvoma kontextami. Základným princípom upraveného algoritmu Norris je prechádzanie cez všetky objekty a volanie metódy add. Táto metóda prechádza následne cez všetky koncepty a robí zjednotenie medzi daným objektom a extentom daného kontextu, ktoré sa do množiny spolu s jeho deriváciou pridá, ak splní nejaké podmienky. Zároveň sa do množiny konceptov pridá aj objekt s jeho deriváciou, ak sa už v množine konceptov nenachádza. Rovnakým spôsobom chceme prerobiť tento algoritmus, pričom budeme prechádzať cez extenty kontextu, na ktoré budeme volať metódu add. V tejto metóde budeme prechádzať cez duálne putá a robiť zjednotenie medzi duálnym putom a tabuľkou, ktorá vznikne z extentov. Takisto sa na základe nejakých podmienok táto tabuľka pridá do množiny pút, ak splní nejaké podmienky.

2.4 Porovnanie algoritmov a vlastnosti pút

V bakalárskej práci sme sa venovali algoritmu na vyhľadávanie konceptov Object Intersection a algoritmu na vyhľadávanie horných susedov NextNeighbor z práce [6]. Tieto algoritmy sme následne prerábali na algoritmy vyhľadávania duálnych pút medzi kontextami.

V diplomovej práci chceme pokračovať v skúmaní už navrhnutých algoritmov na vyhľadávanie pút, ktorými sme sa zaoberali a následne ich prerobiť tak, aby našli celú množinu duálnych pút. Zároveň sa chceme pozrieť aj na ich časovú zložitosť a podľa nej tieto algoritmy porovnať.

Z článku od Kuznetsova a Obiedkova [7] sme si na základe časovej zložitosti vybrali algoritmus na vyhľadávanie konceptov, ktorý chceme prerobiť na algoritmus vyhľadávania pút. Tieto algoritmy na vyhľadávanie konceptov boli na základe časovej zložitosti a rýchlosti v tomto článku porovnané na konceptoch s rôznym počtom objektov a atribútov, preto my sa už týmto porovnaním nebudeme viac zaoberať.

Našim posledným cieľom je skúmanie niektorých vlastností pút. Pre dva rôzne kontexty sme našli množinu všetkých duálnych pút pomocou naivného algoritmu. Ak sme chceli zmeniť jednu hodnotu v jednom kontexte z true na false, tak počet duálnych pút sa rapídne zvýšil. Preto sme sa rozhodli, že sa pozrieme na to, ako jedna hodnota môže spôsobiť zvýšenie počtu extentov a následne zvýšenie počtu pút. Ďalším bodom, ktorým by sme sa chceli zaoberať by bolo, či potrebujeme nájsť celú množinu duálnych pút, alebo stačí nájsť iba niektoré, s ktorými budeme ďalej pracovať.

3 Záver

Naivným algoritmom dokážeme nájsť putá medzi formálnymi kontextami, ale je to časovo zložité, preto sa pokúšame nájsť algoritmus, ktorý tieto putá vyhľadá rýchlejšie. Budeme pokračovať v bakalárskej práci, v ktorej sme navrhli niekoľko algoritmov, ktoré boli rýchlejšie, ale nenašli celú množinu duálnych pút. Celú prácu sme si rozdelili na niekoľko bodov. Prvým je prerábanie algoritmov z bakalárskej práce, aby našli celú množinu duálnych pút. Ďalším bodom je nájdenie najrýchlejšieho algoritmu na vyhľadávanie konceptov a následne jeho transformácia na algoritmus vyhľadávania pút. Nasleduje porovnanie daných algoritmov na základe ich časovej zložitosti a správnosti. Posledným bodom je skúmanie niektorých vlastností pút.

Začali sme implementáciou niektorých transformácií algoritmov z bakalárskej práce. Následne sme naprogramovali algoritmus extentBacktrack, ktorý je rýchlejší ako naivný algoritmus, ale stále má veľkú časovú zložitosť. Z algoritmov na vyhľadávanie konceptov sme vybrali taký algoritmus, ktorý má najlepšiu časovú zložitosť a snažíme sa ho prerobiť na algoritmus, ktorý bude vyhľadávať duálne putá medzi dvoma kontextami a bude mať lepšiu časovú zložitosť, ako doteraz navrhnuté algoritmy.

Literatúra

1. Carpineto, C., Romano, G.: Concept data analysis: theory and applications. John Wiley & Sons, Inc., 11 (2004)
2. Ganter, B., Wille, R.: Formal concept analysis: mathematical foundations. Springer-Verlag Berlin Heidelberg, (1999)
3. Valtchey, P., Jäschke, R.: Formal concept analysis: 9th International Conference. Sprinfer-Verlag Berlin Heidelberg, 82 (2011) https://books.google.sk/books?id=C6Eq2-Gn_70C&pg=PA81&dq=formal+context&hl=sk&sa=X&ved=0ahUKEwje2dmlsozaAhWF3KQKHee5DcM4ChDoAQhEMAQ#v=onepage&q=formal%20context&f=false
4. Kuznetsov, S., Schmidt S.: Formal concept analysis: 5th International Conference. Sprinfer-Verlag Berlin Heidelberg, 10 (2007) https://books.google.sk/books?id=boiv34jzhnAC&pg=PA8&dq=how+find+bonds+in+context&hl=sk&sa=X&ved=0ahUKEwjfqqTLnvDYAhVMXSwKHfgWBwQQ6AEIJzAA#v=onepage&q=how%20find%20bonds%20in%20context&f=false
5. Missaoui, R., Schmid, J.: Formal concept analysis: 4th International Conference. Sprinfer-Verlag Berlin Heidelberg, 91 (2006) https://books.google.sk/books?id=QDmOwDqaZNUC&pg=PA91&lpg=PA91&dq=dual+bonds&source=bl&ots=PCBRZBTLJT&sig=7AvMRmzm6f96htSlRDBN2QyKHmE&hl=sk&sa=X&ved=0ahUKEwjLhNWfj-TWAhWlJ5oKHZ1xC0IQ6AEIaDAI#v=onepage&q=dual%20bonds&f=false
6. Bělohlávek, R., De Baets, B., Outrata, J., Vychodil, V.: Computing the lattice of all fixpoints of a fuzzy closure operator. IEEE Transactions On Fuzzy Systems, vol. 18, no.3, (2010) https://pdfs.semanticscholar.org/d869/4a0ac9131b5cdd67c5dff89a454b5d6cd122.pdf
7. Kuznetsov, S., Obiedkov, S.: Comparing performance of algorithms for generating concept lattices. Taylor & Francis Group., (2002)
8. Almeida, J. B., Frade, M. J., Pinto, J. S., De Sousa, S. M.: Rigorous Software Development: An Introduction to Program Verification. Sprinfer-Verlag Berlin Heidelberg, (2011)