Putá medzi formálnymi kontextami a ich efektívne vyhľadávanie

Bc. Marián Kozák

2Im, 2019 - 2020

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. Do porovnania zahrnieme aj direktný súčin, ktorého extenty sú duálnymi putami dvoch kontextov.

Kľúčové slová: kontext, koncept, direktný súčin, algoritmus object intersection, algoritmus next neighbour, 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. Zároveň medzi algoritmy pridáme aj algoritmus, ktorý vyhľadáva putá z direktného súčinu, keďže extenty direktného súčinu sú duálnymi putami. Takisto si ukážeme aké prepojenie vzniká medzi direktným súčinom a algoritmom Object Intersection. 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]. Týmito algoritmami sa budeme zaoberať v podkapitole 1.1.2, a z nich 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.

Definícia 1. Kontext je trojica (A, B, R) pozostávajúca z dvoch množín a to z množiny A a množiny B a z relácie R medzi A a B.

Navyše Valtchev a Jäschke [3] dodávajú, že množiny A a B musia byť konečné a R je binárna relácia. Prvky množiny A nazývame objekty a množiny B atribúty. Vzťah aRb tak znamená, že objektu a je priradený atribút b. 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.

Definícia 2. Pre množinu objektov β, ktorá je podmnožinou A definujme:

$$\beta' = \{b \in B \mid aRb \; pre \; všetky \; a \in \beta\}. \quad \textbf{(1)}$$

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

$$\gamma' = \{a \in A \mid aRb \; pre \; všetky \; b \in \gamma\}. \quad \textbf{(2)}$$

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

Derivačný operátor pre objekty $a \in A$ môžeme takisto chápať, ako minimálnu množinu takých atribútov $b \in B$, že $\gamma(b) \Rightarrow R(a, b)$:

$$\beta(a) = min_{b \in B}(\gamma(b) \Rightarrow R(a, b)). \quad \textbf{(3)}$$

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

Definícia 3. Koncept kontextu (A, B, R) je pár (β, γ), kde:

$$\beta \subseteq A, \gamma \subseteq B, \beta' \subseteq \gamma, \gamma' \subseteq \beta. \quad \textbf{(4)}$$

Množinu β v tomto prípade nazývame extent a množinu γ nazývame intent konceptu (β, γ). Množinu všetkých konceptov kontextu (A, B, R) označujeme $C(A, B, R)$. Taktiež platí, že množina β, ktorá je podmnožinou A je extentom konceptu, ak platí $\beta'' = \beta$. Podobne je to s množinou γ, ktorá je podmnožinou B, ktorá je intentom konceptu, ak platí $\gamma'' = \gamma$.

Pre príklad si zoberme troch študentov Mareka, Evu a Emu z Tabuľky 1. Títo traja študenti majú radi fyziku a matematiku. Čiže množina β v tomto prípade obsahuje tri mená študentov, pričom β' obsahuje dva názvy predmetov fyziku a matematiku. Takisto množina γ obsahuje názvy vyššie spomenutých predmetov a zároveň γ' obsahuje tri mená a to Mareka, Evu a Emu, ktorí majú radi dané predmety. Takúto dvojicu, ktorá pozostáva z množiny β a množiny γ nazývame koncept, pričom extentom daného konceptu je množina objektov β a intentom toho istého konceptu je množina atribútov γ. V prípade, že množina β obsahuje mená všetkých študentov, tak množina β' 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 objektov *A*. 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: Context (G, M, I)
  Output: Set C of all concepts of (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 is different from intent from C
  6.          C = C ∪ {(Inters', Inters)};
```

V našej diplomovej práci sme naprogramovali aj iné algoritmy na vyhľadávanie konceptov, ktoré 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: Context (G, M, I)
  Output: Set L of all concepts of (G, M, I)

  1. L = Ø;
  2. for each g ϵ G
  3.    Process({g}, g, ({g}'', g));
  4. return L;
program Process
  Input: set of objects A, object g, concept (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 is subset of {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.

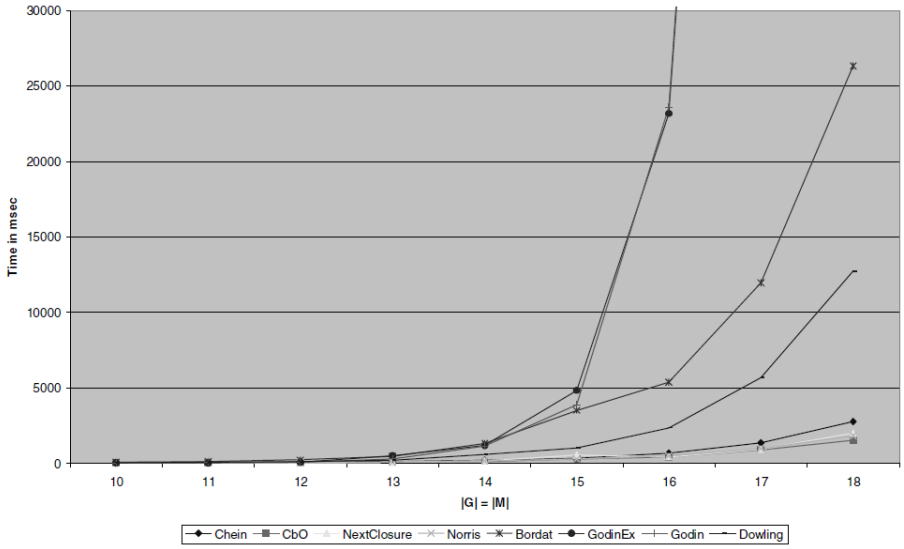
Algoritmus 3. Algoritmus Norris z práce [7]

```
program Norris
  Input: Context (G, M, I)
  Output: Set L of all concepts of (G, M, I)

  1. L = Ø;
  2. for each g ϵ G
  3.    Add(g, L);
  4. return L;
program Add
  Input: object g, set of concepts L
  Output: set of concepts L

  1.for each (A, B)ϵ L
  2.   if B is subset {f}'
  3.      A = A ∪{g};
  4.   else
  5.      D = B ∩ {g}';
  6.      if {h|hϵG\A & h is added & D is subset {h}'}
= Ø;
  7.         L = L ∪{(A∪{g}, D)};
  8. if {h|hϵG & h is added & {g}' is subset of {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. Tento návrh sme mali v pláne ešte dokázať, ale keďže sme sa nakoniec rozhodli, že sa týmto algoritmom nebudeme zaoberať, tak sme dôkaz neurobili. Urobili sme niekoľko úprav v metóde Add, pričom metóda Norris z algoritmu ostala nezmenená. Nižšie uvádzame návrh nášho upraveného algoritmu Norris.

Algoritmus 4. Upravený algoritmus Norris z práce [7]

```
program Norris
  Input: Context (G, M, I)
  Output: Set L of all concepts (G, M, I)

  1. L = ∅;
  2. for each g ϵ G
  3.    Add(g, L);
  4. return L;

program Add
  Input: object g, set of concepts L
  Output: set of concepts L

  1.for each (A, B)ϵ L
  2.   if B is not subset of {f}'
  3.      D = B ∪{g}';
```

```
  4.      if {h|h∈G\A & h is added & D is subset of
{h}'} = ∅;
  5.         if {(A∪{g}, D) is added to L} = ∅;
  6.            L = L ∪{(A∪{g}, D)};
  7. if {({g}, {g}') is added to L} = ∅;
  8.    L = L ∪({g}, {g}');
```

Zo všetkých algoritmov na vyhľadávanie pút sme sa rozhodli, že sa nakoniec budeme zaoberať iba algoritmom Object Intersection. Vybrali sme si ho hlavne kvôli tomu, že hľadá koncepty tak, že robí prienik medzi intentom a derivovaným objektom. Cez prieniky máme možnosť najľahšie získať putá, keďže prienik dvoch duálnych pút je takisto puto.

1.1.3 Puto, duálne puto

Ak chceme spojiť dva kontexty do jednej tabuľky, tak toto spojenie, ktoré spĺňa určité podmienky nazývame puto.

Definícia 4. Nech (A_1, B_1, R_1) a (A_2, B_2, R_2) sú kontexty. Reláciu $R \subseteq A_1 \times B_2$ nazveme putom týchto dvoch kontextov, ak platí, že každý riadok je intentom nejakého konceptu druhého kontextu a každý stĺpec je extentom nejakého konceptu prvého kontextu.

Navyše Kuznetsov, Schmidt [4] dodávajú, že $R \subseteq A_1 \times B_2$ je puto z kontextov (A_1, B_1, R_1) a (A_2, B_2, R_2) vtedy a len vtedy, ak $R_1 \cup R \cup R_2 \cup (A_2 \times B_1)$ je uzavretá relácia sumy kontextov (A_1, B_1, R_1) a (A_2, B_2, R_2). Pre dva kontexty existuje viacero pút. Pre lepšiu predstavivosť si puto môžeme interpretovať ako klasickú tabuľku. V našej práci sa klasickým putom zaoberať nebudeme, skôr sa zameriame na duálne puto, ktoré je definované podobne ako puto z extentov a intentov, ale na to, aby sme ho mohli zadefinovať je dobré, aby sme vedeli aj ako vyzerá puto.

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.

Definícia 5. Nech (A_1, B_1, R_1) a (A_2, B_2, R_2) sú kontexty. Reláciu $R \subseteq A_1 \times A_2$ nazveme duálnym týchto dvoch kontextov, ak platí, že každý riadok je extentom nejakého konceptu druhého kontextu a každý stĺpec je extentom nejakého konceptu druhého 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. V našej práci sa budeme zaoberať hlavne duálnymi putami.

Pre príklad 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 rozbor direktného súčinu dvoch kontextov, vďaka ktorému môžeme nájsť množinu duálnych pút. Druhým bodom je upravený algoritmus Object Intersection, ktorý používa stĺpcové putá kontextov. Nasledujúcim spôsobom, ako nájsť množinu pút je úprava už nami navrhnutého algoritmu Next Neighbor Bond z bakalárskej práce a výpočet jeho časovej zložitosti. Ďalším bodom je algoritmus, ktorý backtrackom cez extenty kontextu nájde množinu pút. Tieto body si postupne vysvetlíme a doplníme v nasledujúcich podkapitolách. Posledným bodom je porovnanie všetkých týchto algoritmov na základe počtu nájdených duálnych pút a na základe ich časovej zložitosti.

Pri každom algoritme sme používali rovnaký príklad, na ktorom sme zistili správnosť daného algoritmu a vypočítali časovú zložitosť navrhnutého algoritmu. Na konci tejto kapitoly všetky algoritmy porovnáme na základe ich časovej zložitosti a správnosti. Kontexty, ktoré používame pozostávajú iba z „1“ a „0“, čo znamená, či medzi daným objektom a atribútom je relácia, alebo nie. Toto zobrazovanie relácii budeme označovať ako „X“. Na miestach, kde sa nachádza „0“ nebudeme dávať nič.

Tabuľka 5. Prvý kontext na porovnanie algoritmov

			X		
			X		
	X			X	X
		X			
	X		X	X	X

Tabuľka 6. Druhý kontext na porovnanie algoritmov

X	X	X	X
	X		

X	X		X
			X
X	X	X	

2.1 Direktný súčin

Prvým spôsobom, ako nájsť množinu pút je pomocou direktného súčinu. Je to najjednoduchší spôsob, ako nájsť putá medzi kontextami. Na začiatok uvedieme zopár definícii, ktoré budeme potrebovať na pochopenie a dôkaz direktného súčinu.

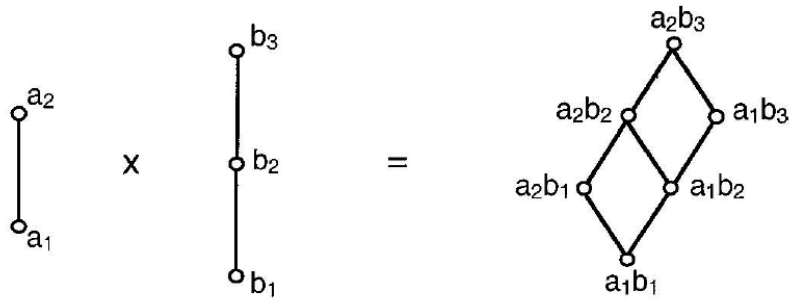
Definícia 6. Direktný súčin dvoch (disjunktných) usporiadaných množín $(P_1, \leq)$ a $(P_2, \leq)$ je usporiadaná množina $(P_1 \times P_2, \leq)$, kde $(P_1 \times P_2)$ je karteziánsky súčin P_1 a P_2, a usporiadaná relácia na súčine je definovaná nasledovne:

$$(x_1, x_2) \leq (y_1, y_2) \leftrightarrow x_1 \leq y_1 \; a \; x_2 \leq y_2.$$

Definíciu direktného súčinu môžeme rozšíriť na ľubovoľný počet množín: ak T je indexová množina a $(P_t, \leq)$, $t \in T$ sú usporiadané množiny, potom

$$\times_{t \in T} (P_t, \leq) = (\times_{t \in T} P_t, \leq) \; s \; (x_t)_{t \in T} \leq (y_t)_{t \in T} \leftrightarrow x_t \leq y_t \; pre \; všetky \; t \in T.$$

Direktný súčin dvoch množín môžeme vidieť na nasledujúcom príklade, ktorý uvádzajú Carpineto a Romano vo svojej knihe [1]:



Obrázok 1. Direktný súčin dvoch usporiadaných množín

Na to, aby sme mohli nájsť množinu duálnych pút potrebujeme urobiť direktný súčin dvoch kontextov.

Definícia 7. Nech $C_1 = \langle A_1, B_1, R_1 \rangle$ a $C_2 = \langle A_2, B_2, R_2 \rangle$ sú kontexty. Direktným súčinom dvoch kontextov nazývame kontext $C_1 \Delta C_2 = \langle A_1 \times A_2, B_1 \times B_2, R \rangle$, kde platí, že

$$R\big((a_1, a_2), (b_1, b_2)\big) = R_1(a_1, b_1)\ OR\ R_2(a_2, b_2).$$

Pre lepšie pochopenie uvedieme príklad na dvoch malých kontextoch.

Tabuľka 7. Prvý kontext na direktný súčin

	b_1^1	b_1^2	b_1^3
a_1^1		X	
a_1^2	X		X

Tabuľka 8. Druhý kontext na direktný súčin

	b_2^1	b_2^2
a_2^1		X
a_2^2	X	
a_2^3	X	X

Tabuľka 9. Direktný súčin dvoch kontextov

	$\langle b_1^1, b_2^1\rangle$	$\langle b_1^1, b_2^2\rangle$	$\langle b_1^2, b_2^1\rangle$	$\langle b_1^2, b_2^2\rangle$	$\langle b_1^3, b_2^1\rangle$	$\langle b_1^3, b_2^2\rangle$
$\langle a_1^1, a_2^1\rangle$		X	X	X		X
$\langle a_1^1, a_2^2\rangle$	X		X	X	X	
$\langle a_1^1, a_2^3\rangle$	X	X	X	X	X	X
$\langle a_1^2, a_2^1\rangle$	X	X		X	X	X
$\langle a_1^2, a_2^2\rangle$	X	X	X		X	X
$\langle a_1^2, a_2^3\rangle$	X	X	X	X	X	X

Duálne puto z direktného súčinu získame tak, že nájdeme množinu konceptov tohto direktného súčinu. Každý extent z konceptov je duálnym putom dvoch kontextov. Predtým, ako dokážeme, že každý extent je duálnym putom, uvedieme niekoľko definícií, potrebných na dôkaz. Nasledujúcu definíciu uvádza Knor vo svojej práci [9]:

Definícia 8. Základné výrokové spojky sú $\neg$ a $\Rightarrow$, pričom ostatné spojky, ako napríklad $\wedge$, $\vee$ a $\Leftrightarrow$ sú odvodené výrokové spojky. Teda:

(a) $A \wedge B$ je iba skrátený zápis formuly $\neg(A \Rightarrow \neg B)$;
(b) $A \vee B$ je skrátený zápis formuly $\neg A \Rightarrow B$;
(c) $A \Leftrightarrow B$ je skrátený zápis $(A \Rightarrow B) \wedge (B \Rightarrow A)$, čiže $\neg((A \Rightarrow B) \Rightarrow \neg(B \Rightarrow A))$

Z tejto definície je pre nás dôležitý iba skrátený zápis disjunkcie, ktorý budeme potrebovať prepísať na implikáciu. Zároveň Knor v knihe uvádza vetu, ktorú dokazuje a ktorú použijeme v dôkaze:

Veta 1. $A \Rightarrow (B \Rightarrow C) \Leftrightarrow (A \wedge B) \Rightarrow C$.

Vo svojej práci taktiež uvádza komutativitu disjunkcie, ktorú budeme potrebovať pri druhej vetve dôkazu druhej vety:

$$(A \vee B) \Leftrightarrow (B \vee A).$$

Veta 2. $min_{i\in I}(a_i \Rightarrow a) \Leftrightarrow (max_{i\in I}a_i) \Rightarrow a$, ak funkcia je v prvom argumente neklesajúca.

Veta 3. Nech $C_1 = \langle A_1, B_1, R_1\rangle$ a $C_2 = \langle A_2, B_2, R_2\rangle$ sú kontexty a $\langle\beta,\gamma\rangle$ je koncept ich direktného súčinu, čiže $\langle\beta,\gamma\rangle \in CL(C_1 \Delta C_2)$. Potom β je duálnym putom kontextov C_1 a C_2.

Dôkaz: Nech $a_1 \in A_1, a_2 \in A_2, b_1 \in B_1, b_2 \in B_2$.

Potom $\beta(a_1, a_2) = \downarrow(\gamma)(a_1, a_2)$
(podľa definície derivačného operátora)
$= min_{\langle b_1,b_2\rangle\in B_1\times B_2}(\gamma(b_1, b_2) \Rightarrow R((a_1, a_2), (b_1, b_2)))$
(podľa definície derivačného operátora)
$= min_{b_1\in B_1} min_{b_2\in B_2}(\gamma(b_1, b_2) \Rightarrow (R_1(a_1, b_1) \vee R_2(a_2, b_2)))$
(podľa definície direktného súčinu)
$= min_{b_1\in B_1} min_{b_2\in B_2}(\gamma(b_1, b_2) \Rightarrow (\neg R_1(a_1, b_1) \Rightarrow R_2(a_2, b_2)))$
(podľa definície základných a odvodených výrokových spojok v definícii 3.)
$= min_{b_2\in B_2} min_{b_1\in B_1}(\gamma(b_1, b_2) \Rightarrow (\neg R_1(a_1, b_1) \Rightarrow R_2(a_2, b_2)))$
(výmena miním)
$= min_{b_2\in B_2} min_{b_1\in B_1}((\gamma(b_1, b_2) \wedge \neg R_1(a_1, b_1)) \Rightarrow R_2(a_2, b_2))$
(podľa vety 1.)
$= min_{b_2\in B_2}(max_{b_1\in B_1}(\gamma(b_1, b_2) \wedge \neg R_1(a_1, b_1)) \Rightarrow R_2(a_2, b_2))$
(podľa vety 2.)
$= min_{b_2\in B_2}(\phi_{a_1}(b_2) \Rightarrow R_2(a_2, b_2))$
(substitúcia $max_{b_1\in B_1}(\gamma(b_1, b_2) \wedge \neg R_1(a_1, b_1))$ za $\phi_{a_1}(b_2)$)
$=\downarrow_2 (\phi_{a_1})(a_2)$
(podľa definície derivačného operátora)

Dokázali sme, že riadky β sú extentami kontextu C_2. Musíme ešte ukázať, že aj stĺpce β sú extentami kontextu C_1.

$\beta(a_1, a_2) = \downarrow(\gamma)(a_1, a_2)$
(podľa definície derivačného operátora)
$= min_{\langle b_1,b_2\rangle\in B_1\times B_2}(\gamma(b_1, b_2) \Rightarrow R((a_1, a_2), (b_1, b_2)))$
(podľa definície derivačného operátora)
$= min_{b_1\in B_1} min_{b_2\in B_2}(\gamma(b_1, b_2) \Rightarrow (R_1(a_1, b_1) \vee R_2(a_2, b_2)))$
(podľa definície direktného súčinu)
$= min_{b_1\in B_1} min_{b_2\in B_2}(\gamma(b_1, b_2) \Rightarrow (R_2(a_2, b_2) \vee R_1(a_1, b_1)))$
(z komutativity disjunkcie)
$= min_{b_1\in B_1} min_{b_2\in B_2}(\gamma(b_1, b_2) \Rightarrow (\neg R_2(a_2, b_2) \Rightarrow R_1(a_1, b_1)))$
(podľa definície základných a odvodených výrokových spojok v definícii 3.)
$= min_{b_1\in B_1} min_{b_2\in B_2}((\gamma(b_1, b_2) \wedge \neg R_2(a_2, b_2)) \Rightarrow R_1(a_1, b_1))$
(podľa vety 1.)
$= min_{b_1\in B_1}(max_{b_2\in B_2}(\gamma(b_1, b_2) \wedge \neg R_2(a_2, b_2)) \Rightarrow R_1(a_1, b_1))$
(podľa vety 2.)
$= min_{b_1\in B_1}(\psi_{a_2}(b_1) \Rightarrow R_1(a_1, b_1))$

(substitúcia $max_{b_2 \in B_2}\big(\gamma(b_1, b_2) \wedge \neg R_2(a_2, b_2)\big)$ za $\psi_{a_2}(b_1)$)
$=\downarrow_1 (\phi_{a_2})(a_1)$
(podľa definície derivačného operátora)

Dokázali sme, že stĺpce β sú extentami kontextu C_1. To znamená, že β je duálnym putom kontextov C_1 a C_2.

To, že direktný súčin nenájde celú množinu duálnych pút si ukážeme aj na kontextoch z tabuľky 5. a tabuľky 6. Celkový počet pút, ktorý direktný súčin našiel bolo 151 duálnych pút. Je to o dva menej, ako našiel naivný algoritmus. V tabuľke 10. a tabuľke 11. sú zobrazené putá, ktoré direktný súčin nenašiel. Ako môžeme vidieť tieto putá sú dosť podobné. Líšia sa iba v jednom stĺpci.

Tabuľka 10. Prvé duálne puto, ktoré direktný súčin nenašiel

X		X				
X		X				
X		X				X
X		X	X			
X	X	X				X

Tabuľka 11. Druhé duálne puto, ktoré direktný súčin nenašiel

X		X				X
X		X				X
X		X				
X		X	X			
X	X	X				X

Celý algoritmus na získavanie duálnych pút z direktného súčinu je zobrazený nižšie:

Algoritmus 1. Algoritmus Direct Product

```
program directProduct
  Input: Context C1(A1, B1, R1), context C2(A2, B2, R2)
  Output: dualBonds - set of dual bonds

  1. dualBonds = ∅;
  2. dirPro = C1 x C2;
  3. concepts = ObjectIntersectionForConcepts(dirPro);
  4. dualBonds = getExtent(concepts);
  5. return dualBonds;
```

Keďže duálne putá získame tak, že zoberieme extenty direktného súčinu, tak na nájdenie všetkých extentov použijeme algoritmus Object Intersection. To sa prejaví aj v časovej zložitosti celého algoritmu.

Majme dva kontexty $C_1 = \langle A_1, B_1, R_1 \rangle$ a $C_2 = \langle A_2, B_2, R_2 \rangle$. Na vytvorenie direktného súčinu potrebujeme časovú zložitosť $O(|A_1| * |A_2| * |B_1| * |B_2|)$, čo

predstavuje prejdenie všetkými objektmi a atribútmi oboch kontextov. Na nájdenie extentov používame algoritmus Object Intersection, ktorého časová zložitosť je podľa Carpineta, Romana [1] $O(|G| * |C| * |M|)$, kde G je počet objektov, C je počet konceptov a M je počet atribútov. V našom prípade to bude $O(|A_1 \times A_2| * |C| * |B_1 \times B_2|)$. Posledným krokom je získanie extentov z množiny konceptov, ktoré vieme urobiť v čase $O(|C|)$. Ak tieto časové zložitosti zlúčime, tak nám vyjde celková časová zložitosť na nájdenie pút $O(|A_1| * |A_2| * |B_1| * |B_2| + |A_1 \times A_2| * |C| * |B_1 \times B_2| + |C|)$.

2.2 Upravený algoritmus Object Intersection

Druhý algoritmus, ktorým sa budeme zaoberať v našej práci je algoritmus Object Intersection. Tento algoritmus sa využíva na vyhľadávanie konceptov zadaného kontextu. Viac o tomto algoritme sme písali v kapitole 1.1.2. Pomocou tohto algoritmu sa pokúsime nájsť množinu duálnych pút, pričom budeme používať stĺpcové putá.

Definícia 9. Nech $C_1 = \langle A_1, B_1, R_1 \rangle$ a $C_2 = \langle A_2, B_2, R_2 \rangle$ sú kontexty. Stĺpcovým putom budeme nazývať taký kontext $C = \langle A, B, R \rangle$, ktorého riadky pozostávajú zo stĺpcov kontextu C_2 a stĺpce pozostávajú zo stĺpcov kontextu C_1.

Stĺpcové putá vieme získať iba backtrackom cez stĺpce kontextov, čo nám spôsobí výrazný nárast časovej zložitosti. Pri generovaní stĺpcových pút pridáme do množiny stĺpcov prvého aj druhého kontextu stĺpce, ktoré obsahujú iba hodnotu „TRUE“. Je to kvôli tomu, aby sa vygenerovalo aj stĺpcové puto, ktoré obsahuje iba hodnoty „TRUE“. Celý algoritmus je založený na tom, že s prechádza stĺpcovými putami a novovytvorenými duálnymi putami a medzi nimi sa robí prienik. Náš algoritmus teda pozostáva z dvoch častí: generovanie stĺpcových pút a spustenie algoritmu Object Intersection na stĺpcových putách. Celý algoritmus je zobrazený nižšie:

Algoritmus 2. Algoritmus Object Intersection with column bonds

```
program ObjectIntersectionColumnBonds
  Input: Context C1(A1, B1, R1), context C2(A2, B2, R2)
  Output: dualBonds - set of dual bonds

  1. dualBonds = Ø;
  2. columnBonds = generateColumnBonds(C1, C2);
  3. dualBonds = objectIntersection(columnBonds);
  4. return dualBonds;

program objectIntersection
  Input: Column bonds CB
  Output: DB - set of dual bonds

  1. DB = Ø;
  2. DB.add(trueTable);
  3. for each columnBond from CB
```

```
4.    for each dualBond from DB
5.       intersection = dualBond ∩ columnBond;
6.       if intersection is not in DB
7.          DB.add(intersection);
8. return DB;
```

Po spustení nášho algoritmu na kontextoch z tabuľky 5. a tabuľky 6. sme získali 151 duálnych pút, čo je o 2 menej, ako našiel naivný algoritmus. V nasledujúcich tabuľkách sú zobrazené duálne putá, ktoré navrhnutý algoritmus nenašiel:

Tabuľka 12. Prvé duálne puto, ktoré Object Intersection nenašiel

X		X				
X		X				
X		X				X
X		X	X			
X	X	X				X

Tabuľka 13. Druhé duálne puto, ktoré Object Intersection nenašiel

X		X				X
X		X				X
X		X				
X		X	X			
X	X	X				X

Ako môžeme vidieť, tak sú to presne tie isté putá, ktoré nenašiel ani direktný súčin. Po generovaní 100 párov kontextov s piatimi objektmi a piatimi atribútmi algoritmus Object Intersection vždy nenašiel rovnakú množinu duálnych pút, ako direktný súčin, alebo našiel o niekoľko pút viac. Čiže množina nenájdených pút algoritmu Object Intersection bola podmnožinou nenájdených pút direktného súčinu. Viac sa tomuto porovnaniu budeme venovať v kapitole 2.5.

Časová zložitosť tohto algoritmu je dosť veľká, hlavne kvôli generovaniu stĺpcových pút. Majme dva kontexty $C_1 = \langle A_1, B_1, R_1 \rangle$ a $C_2 = \langle A_2, B_2, R_2 \rangle$. Generovanie stĺpcových pút má časovú zložitosť $O(|A_1| * (|B_1| + 1) * (|B_1| + 1)^{|A_2|})$, pričom $|B_1|$ je počet stĺpcov kontextu $C1$ a keďže ku stĺpcom pridávame aj stĺpec, ktorý má iba hodnoty „TRUE“, tak pripočítame tam ešte jednotku. Veľkosť $|A_1|$ je práca v cykle, kedy sa stĺpec prvého kontextu kopíruje do novovytvoreného duálneho puta. Veľkosť $|A_2|$ je maximálny počet stĺpcov, duálneho puta.

Samotný algoritmus Object Intersection má časovú zložitosť $O(|A_1| * |A_2| + |CB| * |DB| * |A_1| * |A_2|)$. Veľkosť $|A_1| * |A_2|$ predstavuje prácu s tabuľkou, kde na začiatku do množinu duálnych pút pridáme tabuľku s hodnotami „TRUE“ a neskôr, keď robíme prienik medzi stĺpcovým a duálnym putom. Veľkosť $|CB|$ je veľkosť množiny stĺpcových pút a veľkosť $|DB|$ je veľkosť množiny duálnych pút. Množinu duálnych pút teda dokážeme získať v čase $O(|A_1| * (|B_1| + 1) * (|B_1| + 1)^{|A_2|} + |A_1| * |A_2| + |CB| * |DB| * |A_1| * |A_2|)$.

2.3 Úprava algoritmu NextNeighborBond z bakalárskej práce

V bakalárskej práci sme sa snažili prerobiť algoritmus na výpočet horných susedov kontextu, ktorý vo svojom článku uvádzajú Bělohlávek, De Baets, Outrata, Vychodil [6]. Tento algoritmus však nenašiel celú množinu pút, ale bol zo všetkých algoritmov najrýchlejší. Preto sme sa rozhodli, že týmto algoritmom sa budeme aj naďalej zaoberať. Po niekoľkých úpravách náš algoritmus dosahuje lepšie výsledky. Upravený algoritmus pozostáva z niekoľkých metód, ktoré si postupne ukážeme a vysvetlíme.

Algoritmus 5. Metóda Lattice of bonds

```
program latticeOfBonds
  Input: Context C1(G1,M1,I1), context C2(G2,M2,I2)
  Output: F – Set of dual bonds

  1. F = ∅;
  2. beta is table with false values
  3. leftClosure(beta);
  4. rightClosure(beta);
  5. F = F ∪ {beta};
  6. generateFromBond(beta);
```

Metóda Lattice of bonds je prvou základnou metódou celého algoritmu. Na začiatku sa vytvorí globálna premenná F, do ktorej sa budú pridávať duálne putá, počas celého algoritmu. Najprv sa vytvorí tabuľka, ktorá obsahuje iba „FALSE“ hodnoty a uzavrie sa. Toto novovytvorené puto je prvým putom, ktoré je pridané do množiny pút. Je to najmenší sused, čiže obsahuje najmenej „TRUE“ hodnôt. Z tohto puta sa ďalej volá metóda Generate from bond.

Algoritmus 6. Metóda Generate from bond

```
program generateFromBond
  Input: Dual bond beta
  Output:

  1. upper = neighborsBond(beta)
  2. N = upper \ F
  3. all bonds from N add to F
  4. for each gama z N
  5.    generateFromBond(gama);
```

Do metódy Generate from bond ako vstupný parameter prichádza duálne puto beta. Táto metóda vyhľadáva horných susedov duálneho puta. Z týchto susedov odstráni tie tabuľky, ktoré sa už nachádzajú v množine pút. Zvyšné tabuľky potom pridá do množiny F, a znova spustí metódu Generate from bond na všetkými zvyšnými susedmi.

Algoritmus 7. Metóda Neighbors

```
program neighbors
  Input: Dual bond beta
  Output: Set of all upper neighbors U

  1. U = Ø;
  2. min1 = {b1 ϵ B1 | beta(b1) ≠ B2};
  3. min2 = {b2 ϵ B2 | transformedBeta(b2) ≠ B1};
  4. for each b1ϵ B1: beta(b1) ≠ B2}
  5.    for each gama ϵ moves(beta, b1, row)
  6.       I = Ø;
  7.       for each o1 ϵ (B1\{b1}): beta(o1) ⊂ gama(o1)
  8.          I = I ∪{o1};
  9.       if min1 ∩ I = Ø
  10.          U = U ∪ {gama};
  11.      else
  12.          min1 = min1\{b1};
  13. for each b2ϵ B2: transformedBeta(b2) ≠ B1}
  14.   for each gama ϵ moves(beta, b2, column)
  15.      I = Ø;
  16.      for each o2 ϵ (B2\{b2}): transformedBeta(o2)
⊂ gama(o2)
  17.         I = I ∪{o2};
  18.      if min2 ∩ I = Ø
  19.         U = U ∪ {transformedGama};
  20.      else
  21.         min2 = min2\{b2}};
  22. return U;
```

Táto metóda je najdôležitejšia z celého algoritmu, keďže vyhľadáva horných susedov prichádzajúceho puta a vráti ich ako množinu na výstup. Na začiatku sa vyberú tie riadky, ktoré vo svojom riadku obsahujú aspoň jednu „FALSE“ hodnotu a uložia sa do premennej min1. To isté sa urobí so stĺpcami. Prechádza sa všetkými riadkami puta a tabuľkami gama, ktoré získame z metódy Moves. Do množiny I pridáme tie riadky, ktoré sú rôzne ako riadok, ktorým prechádzame a zároveň riadok puta je podmnožinou riadku gamy na tom istom riadku, ktorý chceme pridať do množiny. Ak prienik množiny I a množiny min1 je prázdna množina, tak gama sa pridá do množiny horných susedov. Ak existuje prienik, tak z množiny min1 odstránime riadok, ktorým prechádzame. Tak isto prechádzame aj stĺpcami duálneho puta a gamou z metódy Moves a robíme ten istý postup, ktorý sme robili s riadkami.

Algoritmus 8. Metóda Moves

```
program moves
  Input: Dual bond beta, row or column b, string
  Output: Set of dual bonds N

  1. un = upper neighbours of beta(b)
  2. N = Ø;
  3. for each g ϵ un
  4.    if string is row
  5.       gama = leftClosure(beta, b, g);
  6.    else
  7.       gama = rightClosure(beta, b, g);
  8.    if gama is dual bond
  9.       N = N ∪{gama};
  10. return N;
```

V metóde Moves hľadáme potenciálne duálne putá, ktoré by mohli byť susedmi duálneho puta beta. Najprv nájdeme horných susedov bety na riadku, alebo na stĺpci s indexom b podľa toho, či na vstup prišiel riadok, alebo stĺpec. Prechádzame týmito susedmi a ak na vstupe prišiel riadok, tak zavoláme metódu Left closure a ak prišiel stĺpec, tak zavoláme metódu Right closure. Z týchto metód získame gamu, ktorá ak je duálnym putom, tak sa pridá do množiny N, ktorú na konci vrátime.

Algoritmus 8. Metódy Left Closure a Right Closure

```
program leftClosure
  Input: Table beta, row b, neighbor g
  Output: Table beta

  1. close columns of beta
  2. close rows of beta
  3. return beta;


program rightClosure
  Input: Table beta, column b, neighbor g
  Output: Table beta

  1. close rows of beta
  2. close columns of beta
  2. return beta;
```

Tieto metódy sú veľmi podobné. Obe dostanú na vstup tabuľku beta, pričom pri Left closure sa riadok na indexe b nahradí riadkom g a pri Right closure sa stĺpec na indexe b nahradí stĺpcom g. V Left closure sa najprv uzavrú stĺpce tabuľky beta a následne sa

uzavrú riadky tohto puta. V Right closure sa najprv uzavrú riadky, až potom stĺpce. Zmenené puto dáme na výstup.

Algoritmus NextNeighborBond nám zo všetkých navrhnutých algoritmov dáva najlepšie výsledky. Zobrali sme dva kontexty z tabuľky 5. a tabuľky 6. Pri týchto kontextoch algoritmus NextNeighborBond našiel všetkých 153 duálnych pút. Čo sa týka jeho úspešnosti pri iných vygenerovaných kontextoch si ukážeme spolu s ostatnými algoritmami v kapitole 2.5.

Časovú zložitosť počítame pre každú jednu metódu samostatne a na konci spočítame celkovú časovú zložitosť algoritmu.

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.

Keďže tento algoritmus sa skladá z viacerých metód, tak si pre každú metódu vypočítame časovú zložitosť osobitne a nakoniec všetky časové zložitosti spojíme. Majme dva kontexty $C_1 = \langle A_1, B_1, R_1 \rangle$ a $C_2 = \langle A_2, B_2, R_2 \rangle$. Začneme od posledných metód Right closure a Left closure. Tie majú rovnakú časovú zložitosť, keďže uzatvárajú stĺpce a riadky tabuľky, len v inom poradí. Ich časová zložitosť je $O(|A_2| * (3|A_1| + 2|A_1||B_1| + |B_1|) + |A_1| * (2|A_2| + 2|A_2||B_2| + |B_2|))$.

Nasledujúcou metódou je metóda Moves. Časová zložitosť je iná, ak sa táto metóda volá nad riadkom puta a keď sa volá nad stĺpcom puta. Pre riadky je časová zložitosť $O(|A_2| * (|A_1| + |ex_2| + |uppN|^2 + |uppN| + |uppNs| + |uppNs| * |A_1|) + |uppNs| * LC)$. Pre stĺpce je časová zložitosť $O(|A_1| * (|A_2| + 1 + |ex_1| + |uppN|^2 + |uppN| + |uppNs| + |uppNs| * |A_2|) + |uppNs| * RC)$. Veľkosti $|uppN|$ a $|uppNs|$ sú veľkosti množín horných susedov. Označenia RC a LC znamenajú časové zložitosti z metód Right closure a Left closure.

Najdôležitejšia metóda Neighbors má časovú zložitosť $O(|A_1| + |A_2| + 2|A_1||A_2| + |A_1| * moves(row) * (|A_1|^2 + 2|A_1||A_2|) + |A_2| * moves(column) * (|A_2|^2 + 5|A_1||A_2|))$. Označenia $moves(row)$ a $moves(column)$ znamenajú časovú zložitosť z metódy Moves pre riadok a stĺpec.

Metóda Generate from sa volá niekoľkokrát. Časová zložitosť bude $O((neighbors * (|DB| * |A_1| * |A_2| + |A_1| * |A_2|)) * |n|!)$. Označenie $neighbors$ znamená časovú zložitosť z metódy Neighbors, veľkosť $|DB|$ znamená veľkosť množiny duálnych pút a veľkosť $|n|$ znamená veľkosť množiny n, ktorá sa v tejto metóde vytvára. Celá táto metóda sa zavolá $|n|!$-krát.

Posledná metóda Lattice of bonds má časovú zložitosť $O(|A_1| * |A_2| + LC + RC + generateFrom)$.

Tento algoritmus nie je veľmi výhodný, čo sa týka časovej zložitosti, ale je to zatiaľ jediný algoritmus, ktorý sa najviac priblížil k správnym výsledkom pri hľadaní duálnych pút.

2.4 Algoritmus Extent backtrack

Posledný algoritmus, ktorému sa budeme venovať sa nazýva Extent backtrack. Nie je to klasický naivný algoritmus, ktorý vygeneruje všetky možné tabuľky a zisťuje, či je daná tabuľka puto, ale pri tomto algoritme máme istotu, že určite nájde celú množinu duálnych pút. Ide taktiež o generovanie tabuliek, ale pri tomto generovaní sa tabuľky získavajú z extentov.

Celý algoritmus pozostáva z troch metód: v prvej sa inicializujú premenné, v druhej sa generujú tabuľky a v tretej sa overuje, či daná tabuľka je duálnym putom.

Algoritmus 8. Algoritmus Extent backtrack

```
program extentBacktrack
  Input: context C1(A1, B1, R1), context C2(A2, B2,
R2), extents1, extents2
  Output: Set of dual bonds

  1. dualBonds = ∅;
  2. ex1 = extents1;
  3. ex2 = extents2;
  4. generate(0, context2.length);
  5. return dualBonds;


program generate
  Input: index, count of columns
  Output:

  1. if index == count of columns
  2.    checkDualBond();
  3.    return;
  4. for each ex ϵ ex1
  5.    for each i = 0 to ex.length()
  6.       dualBond[i][index] = ex[i];
  7.    generate(index + 1, count of columns);


program checkDualBond
  Input:
```

```
Output:

for each row ϵ dualBond
   for each ex ϵ ex2
      if row == ex
         columns are equal = true;
   if columns are equal
      is dual bond = true;
      columns are equal = false;
   else
      is dual bond = false;
       break;
if is dual bond
    dualBonds.add(dualBond);
```

Tento algoritmus uvádzame hlavne kvôli preto, lebo ho porovnávame s ostatnými algoritmami na základe správnosti a časovej zložitosti. Tento algoritmus samozrejme stále nájde celú množinu pút, keďže ide o generovanie, kde skúšame rôzne kombinácie extentov v stĺpcoch a zisťujeme, či riadky sú takisto extentami druhého kontextu. Preto aj v našom príklade z tabuľky 5. a tabuľky 6. našiel všetkých 153 duálnych pút. Tento algoritmus preto využívame namiesto naivné algoritmu a výsledky porovnávame s výsledkami ostatných algoritmov.

Problémom je však je časová zložitosť, keďže tento algoritmus využíva backtrack. Najväčším problémom je metóda generuj, ktorá má časovú zložitosť $O((|A_1| * |A_2| + |A_2| * |ex_2| * |A_1|) * |A_1| * |ex_1| * |ex_1|^{|A_2|})$, kde veľkosti $|ex_1|$ a $|ex_2|$ sú veľkosti množín extentov prvého a druhého kontextu. Veľkosť $O(|A_1| * |A_2| + |A_2| * |ex_2| * |A_1|)$ je veľkosť z metódy checkDualBond na overenie duálneho puta a prekopírovanie. Na všetky časové zložitosti a porovnanie výsledkov sa pozrieme v nasledujúcej kapitole.

2.5 Porovnanie algoritmov

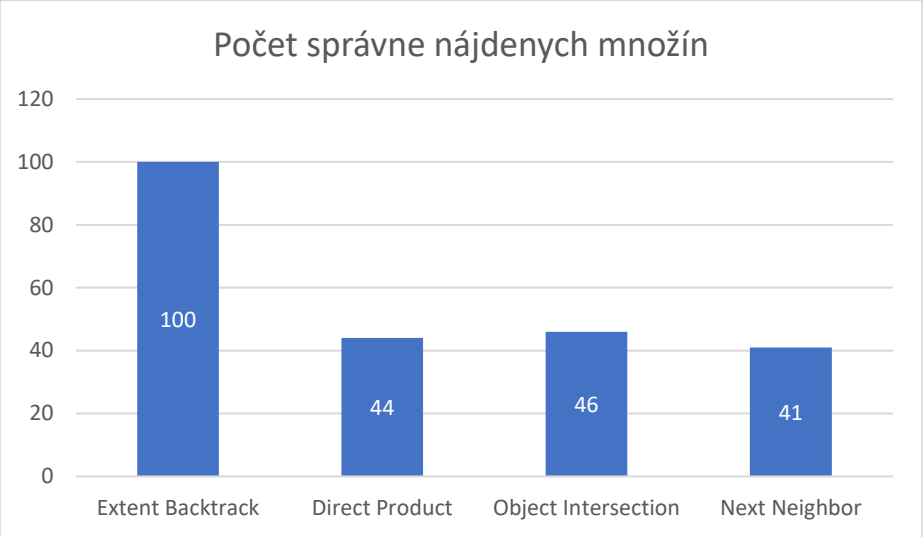
V tejto kapitole porovnáme štyri algoritmy, ktoré sme navrhli: Algoritmus využívajúci direktný súčin, algoritmus Object Intersection využívajúci stĺpcové putá, algoritmus NextNeighborBond a algoritmus ExtentBacktrack. Tieto algoritmy porovnáme na základe ich výsledkov na 100 náhodne vygenerovaných kontextoch s piatimi objektmi a piatimi atribútmi.

Zároveň urobíme krátku analýzu toho, ako spolu súvisia algoritmy Direct Product a Object Intersection, keďže množina nenájdených duálnych pút algoritmu Object Intersection je podmnožinou množiny nenájdených duálnych pút algoritmu Direct Product. Následne porovnáme algoritmy aj na základe ich časovej zložitosti.

Zo všetkých algoritmov algoritmus Extent Backtrack dával najlepšie výsledky, keďže išlo o generovanie tabuliek. Druhý algoritmus, ktorý správne našiel celú množinu duálnych pút bol algoritmus Object Intersection. Zo 100 množín duálnych pút

správne našiel 46 množín. Po ňom nasledoval algoritmus, ktorý využíva direktný súčin. Ten správne našiel 44 množín duálnych pút. A ako posledný, algoritmus Next Neighbor našiel správne 41 množín.

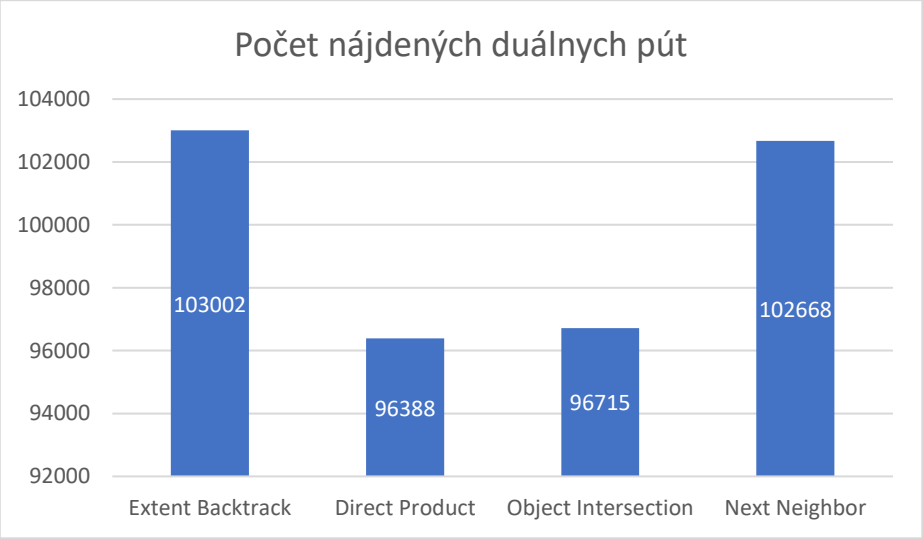
Graf 1. Počet správne nájdených množín duálnych pút pre 100 generovaných párov kontextov



Aj keď Next Neighbor našiel najmenší počet množín správne, jeho rozdiel medzi duálnymi putami, ktoré našiel algoritmus Extent Backtrack bol iba o niekoľko pút. Pri algoritmoch Object Intersection a Direct Product to bolo niekoľko stoviek pút. Preto sme urobili aj porovnanie počtu pút. Algoritmus Extent Backtrack pre 100 generovaných dvojíc našiel 103002 duálnych pút. Na druhom mieste je algoritmus Next Neighbor, ktorý našiel 102668. Algoritmus Object Intersection našiel 96715 duálnych pút. Nakoniec algoritmus Direct Product našiel 96388 pút.

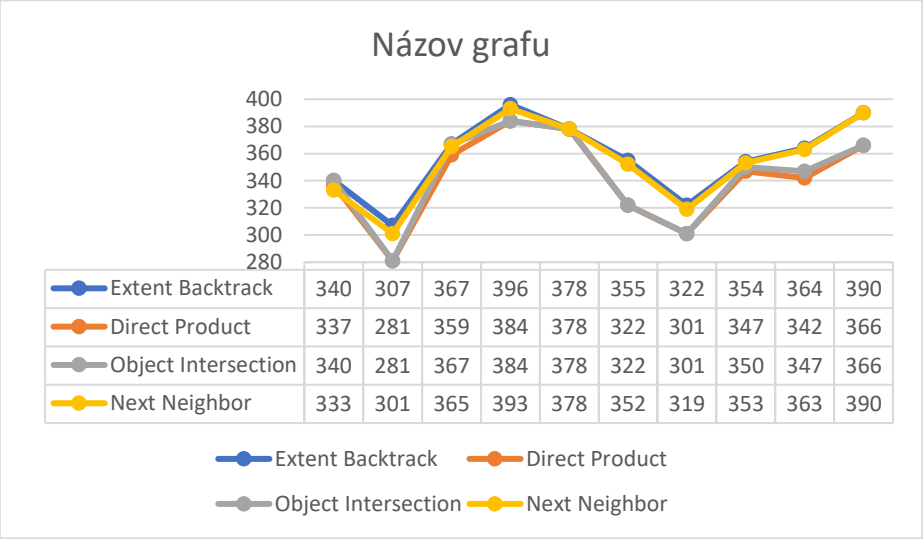
Pri tomto porovnaní môžeme vidieť, že algoritmus Next Neighbor nájde najviac duálnych pút zo všetkých algoritmov. Rozdiel je iba o 334 duálnych pút, pričom Object Intersection a Direct Product ich má o niekoľko tisíc menej. Takisto vidíme, že algoritmus Object Intersection našiel viac duálnych pút ako Direct Product. Každé duálne puto, ktoré Object Intersection nenašiel, nenašiel ho ani algoritmus Direct Product. Celá množina nenájdených pút je teda podmnožinou nenájdených pút algoritmu Direct Product. O tomto porovnávaní budeme ešte písať nižšie v tejto kapitole.

Graf 2. Počet nájdených duálnych pút pre 100 generovaných párov kontextov



To ako naše algoritmy správne nachádzajú duálne putá si ukážeme ešte na nasledujúcom grafe. Ešte raz vygenerujeme 100 párov kontextov, z ktorých si vyberieme desať párov, pri ktorých algoritmus Extent Backtrack nájde množinu pút, ktorá obsahuje od 300 do 400 duálnych pút a pri každom páre si ukážeme, koľko duálnych pút našli jednotlivé algoritmy.

Graf 3. Počet nájdených duálnych pút pre 10 generovaných párov kontextov



Extent Backtrack	340	307	367	396	378	355	322	354	364	390
Direct Product	337	281	359	384	378	322	301	347	342	366
Object Intersection	340	281	367	384	378	322	301	350	347	366
Next Neighbor	333	301	365	393	378	352	319	353	363	390

Hlavnú líniu v tomto grafe tvorí modrá čiara, ktorá predstavuje výsledky algoritmu Extent Backtrack. Pri tomto algoritme máme istotu, že našiel všetky putá, preto budeme

ostatné algoritmy porovnávať s ním. Z tohto grafu môžeme vidieť, že algoritmus Next Neighbor sa zo všetkých navrhnutých algoritmov najviac približoval k správnemu výsledku, pričom pri niektorých prípadoch mal oveľa lepšie výsledky, ako algoritmy Object Intersection a Direct Product. Takisto si môžeme všimnúť, že Algoritmus Object Intersection našiel vždy buď rovnako, alebo viac duálnych pút, ako algoritmus Direct Product. Presné počty duálnych pút, ktoré dané algoritmy v tomto grafe našli si uvádzame v nasledujúcej tabuľke:

Tabuľka 14. Počty nájdených duálnych pút pre 10 generovaných párov kontextov

	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Extent Backtrack	340	307	367	396	378	355	322	354	364	390
Direct Product	337	281	359	384	378	322	301	347	342	366
Object Intersection	340	281	367	384	378	322	301	350	347	366
Next Neighbor	333	301	365	393	378	352	319	353	363	390

V tejto tabuľke môžeme vidieť, že Object Intersection našiel viac pút, ako algoritmus Direct Product. V niektorých prípadoch našli rovnakú množinu pút, ale pri niektorých kontextoch Object Intersection našiel viac duálnych pút. Ukážeme ten prípad, kedy ani algoritmus Object Intersection, ani Direct Product nenájde celú množinu pút, ale Object Intersection ich nájde viac ako Direct Product. Tento príklad ukážeme na nasledujúcich dvoch kontextoch s piatimi objektmi a piatimi atribútmi.

Tabuľka 15. Prvý kontext pre algoritmus Object Intersection a Direct Product

			X	
X		X		
		X	X	
X	X	X	X	
	X		X	

Tabuľka 16. Druhý kontext pre algoritmus Object Intersection a Direct Product

	X	X	X	X
X	X	X	X	X
X				X
X		X	X	
			X	

Z týchto dvoch kontextov algoritmus Extent Backtrack našiel 740 duálnych pút. Algoritmus Object Intersection našiel 739 pút a algoritmus Direct Product našiel 734 duálnych pút. V nasledujúcich tabuľkách si uvedieme puto, ktoré nenašiel ani Direct Product, ani Object Intersection. Ostatné putá, ktoré Direct Product nenašiel uvádzať nebudeme.

Tabuľka 17. Duálne puto, ktoré nenašiel Object Intersection, ani Direct Product

	X		X	
X	X		X	X
X	X		X	
X	X	X	X	X
	X	X	X	

Toto je jedno zo šiestich duálnych pút, ktoré Direct Product nenašiel. Všetky z týchto nenájdených pút mali niečo spoločné. V každom jednom pute bol druhý stĺpec vyplnený hodnotami „TRUE“. Hodnoty v ostatných stĺpcoch sa menili iba zriedkavo. Ostatné nenájde putá boli veľmi podobné tomuto nenájdenému putu v tabuľke 17. Na to, aby sme mohli analyzovať algoritmus Direct Product si v nasledujúcej tabuľke ukážeme direktný súčin dvoch kontextov z tabuľky 15. a z tabuľky 16.:

Tabuľka 18. Direktný súčin kontextov z tabuľky 15. a tabuľky 16.

	X	X	X	X		X	X	X	X		X	X	X	X	X	X	X	X	X		X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X				X	X				X	X				X	X	X	X	X	X	X				X
X		X	X		X		X	X		X		X	X		X	X	X	X	X	X		X	X	
			X					X					X		X	X	X	X	X				X	
X	X	X	X	X		X	X	X	X	X	X	X	X	X		X	X	X	X		X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X				X	X	X	X	X	X	X				X	X				X
X	X	X	X	X	X		X	X		X	X	X	X	X	X		X	X		X		X	X	
X	X	X	X	X				X		X	X	X	X	X				X					X	
	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X				X	X				X	X	X	X	X	X	X	X	X	X	X	X				X
X		X	X		X		X	X		X	X	X	X	X	X	X	X	X	X	X		X	X	
			X					X		X	X	X	X	X	X	X	X	X	X				X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			X	X	
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X	
	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X		X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X				X	X	X	X	X	X	X				X	X	X	X	X	X	X				X
X		X	X		X	X	X	X	X	X		X	X		X	X	X	X	X	X		X	X	
			X		X	X	X	X	X				X		X	X	X	X	X				X	

konceptoch zahrnuté všetky duálne putá. Vieme zatiaľ iba to, že jedna množina nenájdených pút je podmnožinou druhej množiny nenájdených pút. Medzi týmito algoritmami je pravdepodobne spojitosť medzi direktným súčinom a stĺpcovými putami.

To, ako vyzerajú časové zložitosti jednotlivých algoritmov si ukážeme v nasledujúcej tabuľke. Pri algoritme Next Neighbor ukážeme iba časovú zložitosť metódy Lattice of bonds, ktorá volá metódy, pre ktoré časové zložitosti sme rozobrali v kapitole 2.3. Ako dané časové zložitosti vznikli sme opísali v každej kapitole pri daných algoritmoch. V tejto kapitole chceme ukázať všetky algoritmy naraz a porovnať ich časovú zložitosť.

Tabuľka 19. Časové zložitosti jednotlivých algoritmov

Extent Backtrack	$O(\left(\lvert A_1\rvert\lvert A_2\rvert * (1 + \lvert ex_2\rvert)\right) * \lvert A_1\rvert * \lvert ex_1\rvert^{\lvert A_2\rvert+1})$
Direct Product	$O(\lvert A_1\rvert\lvert A_2\rvert * \lvert B_1\rvert\lvert B_2\rvert + \lvert A_1 \times A_2\rvert * \lvert C\rvert * \lvert B_1 \times B_2\rvert + \lvert C\rvert)$
Object Intersection	$O(\lvert A_1\rvert * (\lvert B_1\rvert + 1)^{\lvert A_2\rvert+1} + \lvert A_1\rvert\lvert A_2\rvert + \lvert CB\rvert * \lvert DB\rvert * \lvert A_1\rvert\lvert A_2\rvert)$
Next Neighbor	$O(\lvert A_1\rvert\lvert A_2\rvert + LC + RC + generateFrom)$

Pri algoritme Next Neighbor uvádzame časovú zložitosť prvej metódy Lattice of bonds, ktorá volá ďalšie metódy, pre ktoré sú časové zložitosti vypočítané samostatne. Pre porozumenie vysvetlíme významy jednotlivých označení. Máme dva kontexty $C_1 = \langle A_1, B_1, R_1\rangle$ a $C_2 = \langle A_2, B_2, R_2\rangle$. Veľkosti $\lvert A_1\rvert$ a $\lvert A_2\rvert$ sú počty objektov jednotlivých kontextov a $\lvert B_1\rvert$ a $\lvert B_2\rvert$ sú počty atribútov. Veľkosti $\lvert ex_1\rvert$ a $\lvert ex_2\rvert$ označujú veľkosti množín extentov jednotlivých kontextov. Veľkosť $\lvert A_1 \times A_2\rvert$ predstavuje veľkosť objektov, $\lvert B_1 \times B_2\rvert$ predstavuje veľkosť atribútov direktného súčinu a $\lvert C\rvert$ označuje počet konceptov tohto direktného súčinu. Veľkosť $\lvert CB\rvert$ je veľkosť množiny stĺpcových pút a $\lvert DB\rvert$ je veľkosť množiny duálnych pút. Označenia LC a RC sú označeniami metód Left Closure a Right Closure, ktorých časovú zložitosť nájdeme v kapitole 2.3. Označenie generateFrom je označením metódy Generate from, ktorého časovú zložitosť nájdeme takisto v kapitole 2.3.

Z pohľadu časovej zložitosti sa najrýchlejším algoritmom zdá byť algoritmus Direct Product. Ako druhým najrýchlejším sa zdá byť algoritmus Object Intersection, po ktorom nasleduje algoritmus Extent Backtrack. Najpomalším algoritmom je algoritmus Next Neighbor, ktorý vo svojej časovej zložitosti má faktoriál, keďže viackrát volá metódu Generate From. Hoci algoritmus Next Neighbor dáva najlepšie výsledky, čo sa týka vyhľadávania duálnych, je najpomalším algoritmom kvôli vysokej časovej zložitosti. Aj napriek tomu ho považujeme za najlepší z navrhnutých algoritmov na vyhľadávanie duálnych pút.

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. Niektoré z týchto algoritmov budeme ďalej upravovať a zároveň sa budeme venovať ďalším novým algoritmom

Z bakalárskej práce prevezmeme algoritmus Next Neighbor, ktorý budeme ďalej upravovať, aby našiel celú množinu duálnych pút. Ide hlavne o úpravu metódy Moves, ktorú získame spojením dvoch podobných metód. Po niekoľkých úpravách sme dosiahli lepšie výsledky, ako v bakalárskej práci. Zároveň sa budeme z bakalárskej práce venovať aj algoritmu Object Intersection, ktorý využíva generovanie stĺpcových pút.

Celú prácu sme si rozdelili na niekoľko bodov. Prvým bodom je vysvetlenie základných pojmov, ktoré používame v našej práci a nájdenie pre nás vhodné algoritmu na vyhľadávanie konceptov, ktorý by sme vedeli prerobiť na algoritmus vyhľadávania duálnych pút. Jeden algoritmus preberáme z bakalárskej práce a zároveň sa snažíme nájsť iné, ktoré by sme mohli využiť a z nich vyberieme ten, ktorý je pre nás najvýhodnejší. Venovali sme sa viacerým algoritmom, ktoré nie sú spomenuté v našej práci, ale pre ich časovú zložitosť sme sa nimi prestali zaoberať.

Druhým bodom je návrh rôznych algoritmov na vyhľadávanie pút. Prvým je algoritmus využívajúci direktný súčin, ktorého extenty sú duálnymi putami. Druhý je algoritmus Object Intersection, ktorý používame na vyhľadávanie konceptov a ktorý prerobíme na algoritmus vyhľadávania pút pomocou prieniku medzi stĺpcovými putami a novovytvorenými duálnymi putami. Nasleduje prerábanie algoritmu Next Neighbor z bakalárskej práce, aby našiel celú množinu duálnych pút. Posledným algoritmom je algoritmus Extent Backtrack, ktorý určite nájde celú množinu duálnych pút, ale je neefektívny.

Posledným bodom je porovnanie daných algoritmov na základe ich časovej zložitosti a správnosti a porovnanie a skúmanie nenájdených pút z direktného súčinu a algoritmu Object Intersection.

Začali sme implementáciou direktného súčinu a dokázali sme, že extenty tohto direktného súčinu sú duálnymi putami dvoch kontextov. Avšak tento algoritmus hoci má najlepšiu časovú zložitosť, tak zo všetkých algoritmov nájde najmenšiu množinu duálnych pút. Direktný súčin je ale veľmi zaujímavá štruktúra, ktorej sa chceme v budúcnosti naďalej venovať. Je to spôsobom ako nájsť niektoré duálne putá, treba len zistiť, prečo ich nechce nájsť všetky.

Ďalej sme upravili algoritmus Next Neighbor z bakalárskej práce, ktorý po úpravách dáva lepšie výsledky, najlepšie zo všetkých navrhnutých algoritmov. Upravili sme hlavne metódu Moves, pričom sme dve metódy skĺbili do jednej. Hoci má tento algoritmus najhoršiu časovú zložitosť, je zatiaľ najlepším algoritmom, ktorý sa nám podarilo navrhnúť.

Následne sme naprogramovali algoritmus extentBacktrack, ktorý je rýchlejší ako naivný algoritmus, ale stále má veľkú časovú zložitosť. Ide o klasický algoritmus, ktorý používa backtrack a generuje všetky možnosti, pričom zisťuje, či daná tabuľka je, alebo nie je putom. Tento algoritmus sme používali na porovnávanie s ostatnými algoritmami, keďže sme mali istotu, že nájde všetky duálne putá.

Z algoritmov na vyhľadávanie konceptov sme chceli vybrať algoritmus, ktorý má najlepšiu časovú zložitosť. Tým algoritmom je Norris. Avšak nakoniec sme sa rozhodli pre algoritmus Object Intersection. Vybrali sme ho hlavne kvôli tomu, že pri hľadaní konceptov používa prieniky a keďže prienik dvoch duálnych pút je takisto duálne puto, tak je pre nás výhodnejší tento algoritmus. Pomocou neho sa snažíme robiť prieniky medzi stĺpcovými putami a novovytvorenými duálnymi putami a takto nájsť celú množinu duálnych pút. Hoci tento algoritmus zatiaľ celú množinu nenašiel, zistili sme, že medzi ním a algoritmom, ktorý využíva direktný súčin existuje prepojenie, pretože nenájdené putá algoritmu Object Intersection sú podmnožinou nenájdených pút algoritmu Direct Product. Z časovej zložitosti sme zistili, že najefektívnejším algoritmom je algoritmus Direct Product, po ktorom nasleduje Object Intersection.

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)

9. Knor, M.: Úvod do matematickej logiky. Vydavateľstvo STU, Bratislava (2016) https://www.fiit.stuba.sk/buxus/docs/edicna_cinnost/ucebne_texty/uvod-do-matematickej-logiky_text.pdf