PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

DIPLOMOVÁ PRÁCE

Výpočet části konceptuálního svazu

2011 Bc. Jiří Rýznar

Anotace

V této práci jsem se snažil o vytvoření efektivních algoritmů pro výpočet části konceptuálního svazu jak v normálním tak fuzzy případě. Vycházel jsem z již stávajících algoritmů pro výpočet celého konceptuálního svazu jakožto jeden z hlavních výstupů Formální konceptuální analýzy (FCA). Fuzzy případ jsem navíc rozšířil o faktorizaci.

Děkuji svému vedoucímu Mgr. Janu Outratovi, PhD. za umožnění pracovat na pěkném tématu a opravy chyb. Také své díky dávám manželce, která musela snášet mé nálady.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

V této práci se budeme zabývat algoritmy na řešení úloh pro výpočet části konceptuálního svazu. Budeme předpokládat, že již máme nějaké znalosti z FCA, klasické (Boolovské) a fuzzy logiky. Pro sjednocení zápisu si v 2. kapitole uvedeme všechny potřebné definice.

Motivací pro nalezení algoritmů bylo, že dosud na tyto úlohy neexistuje řešení. To nám chybí v těch případech, kdy nás zajímá jen nějaká část z celého konceptuálního svazu. Z důvodu absence těchto algoritmů musíme postupovat ve dvou krocích. Nejdříve si musíme vypočítat celý svaz a pak odfiltrovat přebývající koncepty. V praxi tato úloha nastane například, když pomocí FCA budeme analyzovat skupinu lidí v niž budou identifikováni podvodníci s platebními kartami nebo teroristé a nás bude zajímat jejich okolí.

Při vytváření nových algoritmů se pokusíme vyjít z již existujících algoritmů na výpočet celého konceptuálního svazu. Proto si v sekci 3.1. uvedeme některé algoritmy pro výpočet celého konceptuálního svazu a z nich vybereme nejvhodnější, který použijeme pro vytvoření nových algoritmů.

V případech, kdy není jednoznačné, zda objekt má nějakou vlastnost, využíváme škálování nebo fuzzy logiku. Pokud se vydáme cestou škálování, tak zůstáváme stále v crisp případě a pro algoritmy se nic nemění. Při použití fuzzy logiky musíme algoritmy rozšířit o práci s vícehodnotovou logikou. Tyto úpravy se týkaly i nově vytvořených algoritmů, takže se dají použít i na fuzzy kontextech a konceptech.

Ve fuzzy případě máme jednoznačně určený rozklad svazu na faktorový svaz uvedený v [5]. O tuto vlastnost nebudou ochuzeny ani naše algoritmy, které budou jako jeden ze vstupních parametrů brát hodnotu z intervalu $\langle 0, 1\rangle$, jenž určuje stupeň podobnosti. Bohužel jednoznačný rozklad v crisp případu není možný.

2. Základní pojmy

V této kapitole se seznámíme se základními definicemi FCA. Jelikož má FCA silné matematické základy, tak si nejprve uvedeme některé pojmy z lineární algebry. Na závěr si FCA rozšíříme o fuzzy logiku.

2.1. Vybrané definice z univerzální algebry

Definice 2.1. Binární relace $\leq$ na množině P se nazývá *relace uspořádání* na P , jestliže platí pro všechna $x, y, z \in P$:

(a) $x \leq x$ (reflexivita),

(b) $x \leq y$ a $y \leq x$ implikuje $x = y$ (antisymetrie),

(c) $x \leq y$ a $y \leq$ z implikuje $x \leq$ z (tranzitivita).

Nechť P je uspořádaná množina a $x, y \in P$. Řekneme, že x je *pokrýváno* y (nebo y *pokrývá* x), značíme $x \prec y$, když $x < y$ a neexistuje $z \in P$, takové že $x < z < y$.

Pro $x \prec y$ můžeme také říct, že x je *dolním sousedem* y, nebo že y je *horním sousedem* x.

Definice 2.2. (Hassův diagram) Každá konečná uspořádaná množina $(P, \leq)$ může být nakreslena. Malými kruhy reprezentujeme prvky P a úsečkami mezi nimi relaci pokrytí. Prvek x je kreslen pod prvkem y když $x \prec y$ Tím nám vznikne *Hassův diagram.*

Definice 2.3. (Princip duality) Mějme nějakou uspořádanou množinu $(P, \leq)$. $(P, \leq)$ je *duálně* definována jako $(P, \geq)$, kde relace $\geq$ je inverzní relací k relaci $\leq$ (t.j $y \geq x$ platí p.k. $x \leq y$).

Definice 2.4. (maximální, minimální, největší a nejmenší prvek) Nechť P je uspořádaná množina a $Q \subseteq P$. Pak $a \in Q$ se nazývá *maximální* prvek množiny Q, jestliže $a \leq x \in Q$ implikuje $a = x$. Jestliže $x \leq a$ pro všechna $x \in Q$, pak je x *největší* prvek Q. *Minimální* prvek množiny Q a *nejmenší* prvek množiny Q jsou definovány duálně.

Definice 2.5. (horní a dolní závora) Nechť $(P, \leq)$ je uspořádaná množina a $S \subseteq P$. Prvek $x \in P$ se nazývá *horní závora* množiny S, jestliže pro všechna $s \in S$ platí $s \leq x$. *Dolní závora* je definována duálně. Množina všech horních závor S se označuje S^u a množina všech dolních závor S^l.

Definice 2.6. (supremum a infimum) Jestliže existuje nejmenší prvek S^u nazýváme ho *nejmenší horní závora* množiny S (popř. *supremum*). Duálně, největší prvek S^l se nazývá *největší dolní závora* (popř. *infimum*).

Píšeme $x \vee y$ místo $sup\{x, y\}$ a $x \wedge y$ místo $inf\{x, y\}$

Definice 2.7. (svaz a úplný svaz) Uspořádaná množina $(P, \leq)$ se nazývá *svaz*, jestliže pro každou dvojici x a $y \in P$ vždy existuje supremum a infimum. Uspořádaná množina $(P, \leq)$ se nazývá *úplný svaz*, jestliže pro každou $S \subseteq P$ existuje supremum a infimum.

2.2. Vybrané definice z FCA

Definice 2.8. (formální kontext) *Formální kontext* je uspořádaná trojice $\langle X, Y, I\rangle$, kde X a Y jsou neprázdné množiny a I je binární relace mezi X a Y, tj. $I \subseteq X \times Y$

Definice 2.9. Pro formální kontext $\langle X, Y, I\rangle$ jsou definovány operátory $^{\uparrow} : 2^X \to 2^Y$ a $^{\downarrow} : 2^Y \to 2^X$

$$A^{\uparrow} = \{y \in Y \mid \text{pro všechna } x \in A \ : \langle x, y\rangle \in I\},$$
$$B^{\downarrow} = \{x \in X \mid \text{pro všechna } y \in B : \langle x, y\rangle \in I\}.$$

Definice 2.10. (formální koncept) *Formální koncept* v $\langle X, Y, I\rangle$ je pár $\langle A, B\rangle$, kde $A \subseteq X$ a $B \subseteq Y$ takový, že $A^{\uparrow} = B$ a $B^{\downarrow} = A$.

Definice 2.11. (podkoncept a nadkoncept) Pro formální koncepty $\langle A_1, B_1\rangle$ a $\langle A_2, B_2\rangle$ v $\langle X, Y, Z\rangle$ definujeme $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$ p.k. $A_1 \subseteq A_2$ (p.k. $B_2 \subseteq B_1$).

Poznámka. Symbolem $\leq$ označujeme konceptové uspořádání.

Definice 2.12. (konceptuální svaz) Množinu všech formálních konceptů v $\langle X, Y, I\rangle$ označujeme $\mathcal{B}(X, Y, I)$, tj.

$$\mathcal{B}(X, Y, I) = \{\ \langle A, B\rangle \in 2^X \times 2^Y \mid A^{\uparrow} = B, B^{\downarrow} = A\ \}.$$

$\mathcal{B}(X, Y, I)$ obohacenou o konceptové uspořádání $\leq$ nazýváme *konceptuální svaz* a označujeme $\mathcal{L}$.

Poznámka. *Největší koncept* ve svazu můžeme zapisovat jako $\top$ a *nejmenší koncept* ve svazu můžeme zapisovat jako $\bot$.

Věta 2.1. (hlavní věta FCA (první část)) $\mathcal{L}$ *je úplný svaz s infimy a supremy definovanými:*

$$\bigwedge_{j\in J}\langle A_j, B_j\rangle = \langle \bigcap_{j\in J} A_j, (\bigcup_{j\in J} B_j)^{\downarrow\uparrow}\rangle, \bigvee_{j\in J}\langle A_j, B_j\rangle = \langle (\bigcup_{j\in J} A_j)^{\uparrow\downarrow}, (\bigcap_{j\in J} B_j)\rangle$$

2.3. Vybrané definice z fuzzy logiky

Na rozdíl od klasické logiky, kde se používá dvou-hodnotová Boolova algebra, ve fuzzy logice máme více stupňů pravdivosti. Stupně pravdivosti mohou být interval $\langle 0, 1\rangle$ nebo výčet hodnot z tohoto intervalu. Strukturou, nad kterou nejčastěji zavádíme fuzzy logiku, je *úplný reziduovaný svaz*.

Definice 2.13. (úplný reziduovaný svaz) *Úplným reziduovaným svazem* rozumíme algebraickou strukturu $L = \langle L, \wedge, \vee, \otimes, \rightarrow, 0, 1\rangle$, kde:

(a) $\langle L, \wedge, \vee, 0, 1\rangle$ je úplný svaz,

(b) $\langle L, \otimes, 1\rangle$ je komutativní monoid,

(c) $\langle \otimes, \rightarrow\rangle$ je adjungovaný pár (t.j. $a \otimes b \leq c$ p.k. $a \leq b \rightarrow c$).

Pomocí Lukasiewiczovy algebry definujeme následující operace:

(a) $a \rightarrow b = min\{1, 1 - a + b\}$,

(b) $a \otimes b = max\{0, a + b - 1\}$.

Na reziduovaném svazu můžeme zavést odvozenou operaci *Bireziduum*, kterou označujeme $\leftrightarrow$. Definujeme ji jako $y_1 \leftrightarrow y_2 = y_1 \rightarrow y_2 \wedge y_2 \rightarrow y_1$ pro všechna $y_1, y_2 \in L$.

Definice 2.14. (fuzzy množina) *Fuzzy množina* A na univerzu U je zobrazení $A : U \rightarrow L$. $A(u)$ pro všechna $u \in U$ představuje stupeň, ve kterém u náleží do A. Pro konečné U zapisujeme $A = \{\ldots, \frac{A(u)}{u}, \ldots\}$

Množinu všech *fuzzy množin* na univerzu X budeme označovat L^X.

Definice 2.15. (fuzzy relace) *Binární fuzzy relace* R mezi univerzy U a V je zobrazení $R : U \times V \rightarrow L$. $R(u, v)$ pro všechna $u \in U$ a $v \in V$ představuje stupeň, ve kterém jsou u a v v R-relaci.

2.4. Vybrané definice z fuzzy FCA

Definice 2.16. (formální fuzzy kontext) *Formální fuzzy kontext* je uspořádaná trojice $\langle X, Y, I\rangle$ kde X a Y jsou neprázdné množiny a I je fuzzy relace mezi X a Y, tj. $I(x, y) \in L$.

Definice 2.17. Pro formální fuzzy kontext $\langle X, Y, I\rangle$ jsou definovány operátory $^{\uparrow}$,$^{\downarrow}$: pro $A \in L^X, B \in L^Y$, definujeme $A^{\uparrow} \in L^Y$ a $B^{\downarrow} \in L^X$

$$A^{\uparrow}(y) = \bigwedge_{x \in X} (A(x) \rightarrow I(x, y))$$

$$B^{\downarrow}(x) = \bigwedge_{y \in Y} (B(y) \rightarrow I(x, y))$$

Definice 2.18. (formální fuzzy koncept) *Formální fuzzy koncept* v $\langle X, Y, I\rangle$ je pár $\langle A, B\rangle$, kde $A \in L^X$ a $B \in L^Y$, takový že $A^{\uparrow} = B$ $B^{\downarrow} = A$.

Definice 2.19. (podkoncept a nadkoncept) (Stejné jako v crisp případě.) Pro formální koncepty $\langle A_1, B_1\rangle$ a $\langle A_2, B_2\rangle$ v $\langle X, Y, Z\rangle$ definujeme $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$ p.k. $A_1 \subseteq A_2$ (p.k. $B_2 \subseteq B_1$).

Poznámka. Symbolem $\leq$ označujeme fuzzy konceptové uspořádání.

Definice 2.20. (fuzzy konceptuální svaz) Množinu všech formálních konceptů v $\langle X, Y, I\rangle$ označujeme $\mathcal{B}(X, Y, I)$, tj.

$$\mathcal{B}(X, Y, I) = \{\ \langle A, B\rangle \in L^X \times L^Y \mid A^{\uparrow} = B, B^{\downarrow} = A\ \}.$$

$\mathcal{B}(X, Y, I)$ obohacenou o konceptové uspořádání $\leq$ nazýváme *fuzzy konceptuální svaz* a označujeme $\mathcal{L}$.

3. Algoritmy pro výpočet konceptuálního svazu

V této kapitole si nejdříve představíme základní myšlenky některých algoritmů pro výpočet celého konceptuálního svazu nebo jen konceptů bez jejich vzájemných vazeb. Následně si vybereme algoritmus, který je pro výpočet části konceptuálního svazu nejvhodnější a důkladně si jej popíšeme jak pro crisp i fuzzy případ.

3.1. Přehled zkoumaných algoritmů

Mezi zkoumané algoritmy jsme zařadili *Naivní algoritmus*, *Next Closure*, *Intersections* a *Next Neighbours*. Při popisu základní myšlenky algoritmu a jeho složitosti budeme uvažovat, že výpočet probíhá přes intenty. Symboly použité v zápisu složitostí mají stejný význam jako v 2. kapitole.

U všech algoritmů se objevují operátory $^{\downarrow}$, $^{\uparrow}$ nebo jejich kombinace. Abychom nemuseli stále opakovat jejich časové složitosti, tak si uveďme, že pro všechny případy je časová složitost rovna $O(|X||Y|)$. Zápisem $|\mathcal{L}|$ budeme označovat počet konceptů ve svazu.

První si uvedeme nejlehčí tj. Naivní algoritmus popsaný v [3]. Vychází z faktu, že $B \subseteq Y$ je intent p.k. $B^{\downarrow\uparrow} = B$. Pro zjištění všech konceptů aplikuje $^{\downarrow\uparrow}$ na všechny prvky potenční množiny Y. Mohutnost potenční množiny je $2^{|\mathcal{L}|}$, proto je celková složitost tohoto algoritmu $O(2^{|\mathcal{L}|}|X||Y|)$. Jelikož se jedná o neefektivní algoritmus, nebudeme se jím dále zabývat.

Algoritmus Next Closure popsaný v [1] a [3] jehož autorem je Bernhard Ganter je založen na lexikografickém uspořádání. Na začátku si určíme libovolné pořadí atributů, které bude neměnné. Pro všechna $B \subseteq Y$ existuje vektor o délce $|Y|$, který má 1 na pozici, která odpovídá pořadí atributu patřícího do B, jinak 0. Všechny intenty konceptů jsou vypočteny v lexikografickém upořádání podle jejich vektorů. Složitost tohoto algoritmu je $O(|\mathcal{L}||X||Y|^2)$. Nicméně tento algoritmus také nepoužijeme a to z důvodu, že koncepty generuje v lexikografickém uspořádání. My totiž potřebuje algoritmus, který generuje koncepty v okolí jiných konceptů a to nám lexikografické uspořádání bohužel neumožňuje. V [3] jde na ukázkovém příkladě vidět, jak chaoticky jsou koncepty generovány vzhledem ke svému svazovému uspořádaní. Navíc nemáme informace o jejich vzájemných vztazích (podkoncept, nadkoncept) a jejich doplnění by nám zvedlo časovou složitost.

Intersections popsaný v [3] vychází z hlavní věty FCA, která nám mimo jiné říká že průnikem intententů dostaneme zase intent. Tento fakt je pak využit ke generování nových intentů konceptů s časovou složitostí $O(|\mathcal{L}||X||Y|)$. Složitost je sice lepší než v minulém případě, ale opět je tu problém s chaotickým generováním konceptů vzhledem k svazovému uspořádání a s chybějícími informacemi o tomto uspořádání. Pro tyto nedostatky algoritmus Intersections také nepoužijeme.

Jako poslední si uvedeme Next Neighbours. Jako jediný generuje koncepty

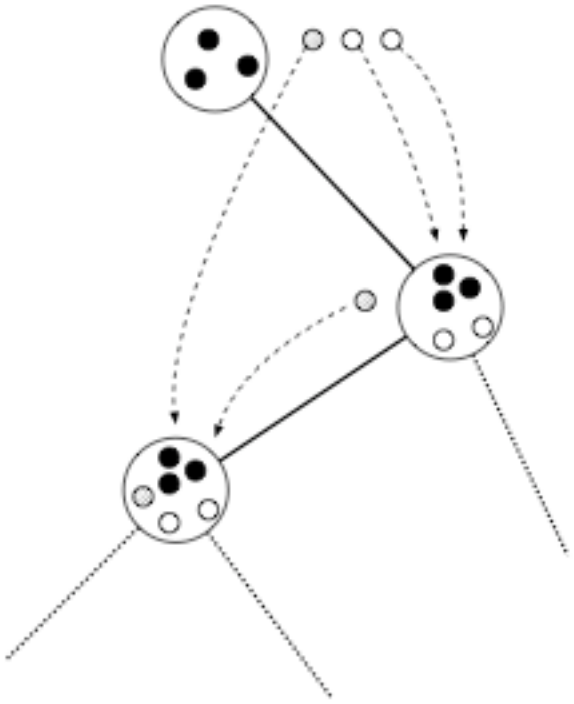
podle svazového uspořádání, proto jsme si jej zvolili jako základ pro algoritmy na výpočet části konceptuálního svazu. Detailní popis nalezneme v následující podkapitole.

3.2. Next Neighbors pro crisp kontext

Algoritmus *Next Neighbors* od Christiana Lindiga popisovaný v [2] a [3] nám vypočítá všechny koncepty pro zadaný kontext a hrany mezi nimi. Algoritmus se skládá ze dvou částí. Jedna část počítá dolní[1] (duálně horní) sousedy k danému konceptu. Druhá část se stará o výběr dalšího konceptu, pro který se budou počítat jeho sousedé, uložení sousedů a jejich hran.

Ještě než přistoupíme k samotnému popisu algoritmu, uvedeme si následující větu jejíž poznatky využijeme pro výpočet sousedů. Obrázek slouží k lepší ilustraci věty a pochopení jejího znění.

Věta 3.1. Nechť $\langle A, B\rangle \in \mathcal{B}(X, Y, I)$ a $\langle A, B\rangle \neq \perp$. Pak $(B \cup \{y\})^{\downarrow\uparrow}$, kde $y \in Y \setminus B$, je intent dolního souseda $\langle A, B\rangle$ p.k. pro všechny $z \in (B \cup \{y\})^{\downarrow\uparrow} \setminus B$ platí následující: $(B \cup \{z\ \})^{\downarrow\uparrow} = (B \cup \{y\})^{\downarrow\uparrow}$.



Obrázek 1. Generování souseda.

[1] V originálu uvedeno naopak.

Na obrázku 1. vidíme, že po přidání atributu k intentu a aplikováním operátorů $^{\downarrow\uparrow}$ sice dostaneme intent, ale nemusí se jednat o souseda. Soused to je v případě, když stejný postup provedeme s novými atributy a dostaneme stejný intent.

Část NEIGHBORS vychází z předchozí věty a pro předaný koncept nám vypočítá jeho sousedy.

```
NEIGHBORS (⟨A, B⟩, ⟨X, Y, I⟩)
 1  min ← Y \ B
 2  neighbors ← ∅
 3  foreach y ∈ Y \ B do
 4     A_1 ← (B ∪ {y})^↓
 5     B_1 ← A_1^↑
 6     if ((min ∩ (B_1 \ B \ {y})) = ∅) then
 7        neighbors ← neigbors ∪ {⟨A_1, B_1⟩}
 8     else
 9        min ← min \{y}
10  return neighbors
```

Když už umíme vypočítat sousedy pro daný koncept, tak nám nic nebrání zkonstruovat celý konceptuální svaz.

```
LATTICE (⟨X, Y, I⟩)
 1  c ← ⟨X, X^↑⟩
 2  insert (c, ℒ)
 3  loop
 4     foreach x in Neighbors (c, ⟨X, Y, I⟩)
 5        try x ← lookup (x, ℒ)
 6        with NotFound → insert (x, ℒ)
 7        x* ← x* ∪ {c}
 8        c_* ← c_* ∪ {x}
 9     try c ← next (c, ℒ)
10     with NotFound → exit
11  return ℒ
```

Na řádku 1 je v originálu uvedeno $\langle \emptyset^{\downarrow}, \emptyset^{\downarrow\uparrow} \rangle$ my jsme použili zápis $\langle X, X^{\uparrow} \rangle$ kvůli větší přehlednosti.

Časová složitost operace $^{\downarrow}$ je v nejhorším případě $O(|X||Y|)$, jelikož pro všechny atributy musíme projít všechny objekty. To platí i pro $^{\uparrow}$, kde pro všechny objekty musíme projít všechny atributy. V části NEIGHBORS v nejhorším případě provádíme operace $^{\downarrow\uparrow}$ pro všechny atributy, proto je její časová složitost $O(|X||Y|^2)$. V části LATTICE cyklus proběhne přesně $|\mathcal{L}|$ krát. Výsledná časová složitost je $O(|\mathcal{L}||X||Y|^2)$.

3.3. Next Neighbors pro fuzzy kontext

Pro fuzzy případ vyjdeme z [4], který si upravíme tak, aby část LATTICE mohla zůstat stejná. To nám dovolí vytvářet algoritmy na obecné úrovni. Budou použitelné jak pro crisp tak pro fuzzy[2] případy. V neposlední řadě nám tato vlastnost velice usnadní implementaci.

Označme $L = \{a_1, \ldots, a_k\}$ tak, aby $a_1 < a_2 < \ldots < a_k$. Když $i < k$ píšeme a_i^+ místo a_{i+1}.

Pro všechna $B \in L^Y$ a $y \in Y$ taková, že $B(y) < 1$, zkracujeme $(B \cup \{^{B(y)^+}/y\})^{\downarrow\uparrow}$ na $[y]_B^{\downarrow\uparrow}$.

Korektnost algoritmů NEIGHBORS a LATTICE je dokázána v [4].

```
NEIGHBORS (⟨A, B⟩, ⟨X, Y, I⟩)
 1  min ← {y ∈ Y | B(y) < 1}
 2  neighbors ← ∅
 3  foreach y ∈ Y such that B(y) < 1 do
 4     D ← [y]_B^{↓↑}
 5     increased ← {z ∈ Y | z ≠ y and B(z) < D(z)}
 6     if min ∩ increased = ∅ then
 7        neighbors ← neigbors ∪ {⟨D^↓, D⟩}
 8     else
 9        min ← min \{y}
10  return neighbors
```

Část NEIGHBORS jsme oproti originálu rozšířili o celý koncept, stačil by pouze intent (duálně extent). Části GENERATEFROM a LATTICE jsme spojili do LATTICE. Jak jde vidět níže, dosáhli jsme toho, že část LATTICE je stejná jak v crisp případě. Tyto úpravy nám umožní bavit se o algoritmech obecně, bez nutnosti rozlišovat crisp a fuzzy případ, čehož také využijeme při implentaci.

```
Lattice (⟨X, Y, I⟩)
 1  c ← ⟨X, X^↑⟩
 2  insert (c, ℒ)
 3  loop
 4     foreach x in Neighbors (c, ⟨X, Y, I⟩)
 5        try x ← lookup (x, ℒ)
 6        with NotFound → insert (x, ℒ)
 7        x* ← x* ∪ {c}
 8        c* ← c* ∪ {x}
 9     try c ← next (c, ℒ)
10     with NotFound → exit
11  return ℒ
```

[2]Bez ohledu na interpretaci spojek a množinových operací.

Při odvozování časové složitosti postupujeme stejně jako v crisp případě. Navíc, ale musíme zahrnout velikost reziduovaného svazu ($|L|$), jelikož soused nemusí mít přidaný atribut hned ve stupni 1, ale může vzniknout řetěz sousedů pro přidaný atribut. Dostáváme tedy časovou složitost jako $O(|\mathcal{L}||L||X||Y|^2)$.

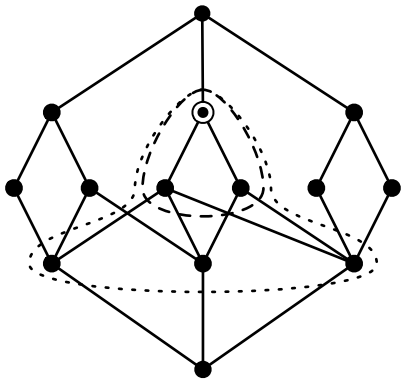
4. Algoritmy pro výpočet části konceptuálního svazu

Všechny algoritmy uvedené v této kapitole nám vrátí části konceptuálního svazu bez nutnosti počítat celý svaz. Výpočet je stejný jak pro crisp či fuzzy části konceptuálního svazu. Algoritmy vycházejí z algoritmu *Next Neighbors*. Část NEIGHBORS zůstává stejná, mění se pouze část LATTICE.

4.1. Horní, dolní kužel

Jak již název napovídá, jedná se o algoritmus na výpočet horního nebo dolního kužele v konceptuálním svazu. Budeme se zabývat dolním kuželem, horní kužel bychom získali duálně.

Úloha bude definována následovně. Máme zadán koncept, výšku kužele a samozřejmě také kontext. Výšku kužele zatím chápejme jako počet hran mezi vrcholem a koncepty v základně kuželu. Problémy spojené s touto definicí budeme zmiňovat následně. Jako výsledek očekáváme kužel, který má vrchol v zadaném konceptu a danou výšku. V ideálním případě bude výsledek vypadat jak na následujícím obrázku. Vrchol je v kroužku a přerušovanou čárou máme vyznačený kužel o výšce 1 a 2.



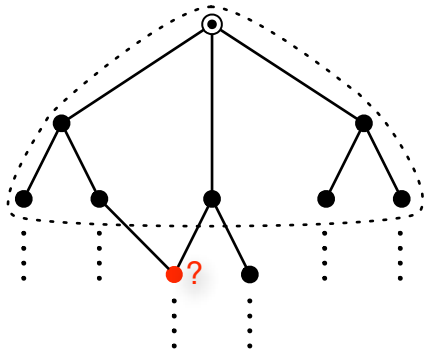
Obrázek 2. Ukázka výsledku algoritmu dolní kužel.

Ideální případ nám může narušit příliš velká výška kužele a nebo nejednoznačnost délky cesty.

První případ nastane, když zadaná výška je větší než počet hran od vrcholu kužele k nejmenšímu prvku svazu. V tomto případě nám kužel zdegeneruje na podsvaz, který má největší prvek koncept, který byl zadaný jako vrchol kužele a nejmenší prvek shodný s nejmenším prvkem svazu.

Nejednoznačnost délky cesty souvisí s faktem, že v FCA není definována vzdálenost mezi koncepty a pojem výška konceptu ve svazu. Což nám vadí u konceptů,

do kterých vede více cest s rozdílným počtem hran od vrcholu kuželu. Mohli bychom si sice stanovit, že budeme brát v potaz maximální nebo minimální počet hran, ale to sebou nese jisté komplikace, které jsou vysvětleny v sekci 4.1.1. Nejlépe je absence tohoto pojmu vidět na následujícím obrázku.

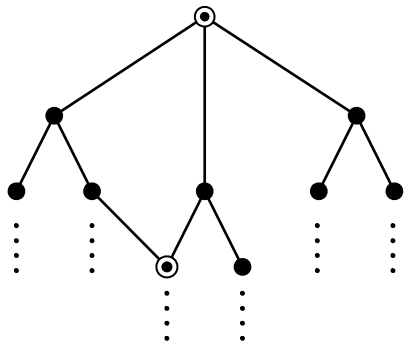


Obrázek 3. Patří koncept označený "?" do kuželu?

4.1.1. Vzdálenost mezi koncepty

Vzdálenost mezi dvěma koncepty se určuje jako počet hran v cestě, která je spojuje. Přitom cestu se snažíme volit tak, abychom šli stále ve stejném směru (shora dolů nebo naopak). Změnu směru používáme pouze v případě, kdy koncepty nejsou vzájemně srovnatelné a to v co nejmenší míře. Jak jsme si již řekli, tato cesta nemusí existovat pouze jedna a tudíž může dojít k nejednoznačnosti v určení vzdálenosti.

Nejednoznačnost bychom mohli odstranit, kdybychom uvažovali jako výslednou vzdálenost minimum nebo maximum ze všech vzdálenosti.



Obrázek 4. Určování vzdálenosti.

Pokud bychom se vydali cestou minima, tak nemusíme do algoritmu nic přidávat a stačí zajistit, aby se nepřepisovala vzdálenost u již vypočtených konceptů. V tomto případě by se při zobrazení některých výsledků s porovnáním s celým svazem mnoho zdát, že jsme vypočetli zbytečně mnoho konceptů. Zvolením maximální délky jako výsledné vzdálenosti sice zamezíme vrácení přebytečných konceptů, ale realizace by byla značně komplikovaná. Museli bychom na výsledku zjistit maximální délku cesty ze všech možných cest, což by nám zhoršilo časovou složitost algoritmů. Navíc při zjištění nesplnění podmínky bychom museli odstraňovat přebytečné koncepty a jejich vazby, což by také ovlivnilo složitost.

Proto se vydáme zlatou střední cestou a při počítání průběžné vzdálenosti nebudeme ošetřovat přepisování vzdáleností na delší. Tím zajistíme, že výsledná vzdálenost se bude pohybovat mezi minimem a maximem ze všech vzdáleností. Bude záležet, kdy k tomuto přepisu dojde a zda z něj ještě nebyli počítáni jeho sousedé. Tím eliminujeme některé zkratky.

Úplně nejlepším řešením této situace by bylo, kdyby byl pojem vzdálenost jednoznačně definován nebo alespoň úroveň konceptu, ve které má být kreslen. Potom bychom za vzdálenost mohli uvažovat rozdíl v úrovních a v případě kruhu počet přestupů mezi úrovněmi.

4.1.2. Výsledné řešení

Z poznatků uvedených v sekci 4.1.1. si necháme volnost při rozhodnutí, zda koncept ještě patří do kužele či nikoliv. Tato volnost se projeví v následujícím pseudokódu v podobě přepisující se vzdálenosti aktuálně zpracovávaného konceptu od vrcholu kužele.

Vstup: $\langle A, B\rangle$ je koncept, určující vrchol kužele, výška kužele je označena jako *height* a kontext jako $\langle X, Y, I\rangle$.
Výstup: $\mathcal{L}$ je vypočítaný dolní kužel.
CONE ($\langle A, B\rangle$, height, $\langle X, Y, I\rangle$)

```
 1  c ← ⟨A, B⟩
 2  insert (c, L)
 4  c_d ← 0
 5  loop
 6    foreach x in Neighbors (c, ⟨X, Y, I⟩)
 7      try x ← lookup (x, L)
 8      with NotFound then
 9        insert (x, L)
10        x_d ← c_d + 1
11        if x_d < height then
12          insert (x, Q)
13      x* ← x* ∪ {c}
14      c_* ← c_* ∪ {x}
15    try c ← pop (c, Q)
16    with NotFound → exit
17  return L
```

Část CONE je založena na části LATTICE a její rozšíření je popsáno v následujících bodech:

- Q je fronta konceptů čekajících na zpracování.
- Symbolem $_d$ máme označenou vzdálenost příslušného konceptu od vrcholu kužele. Na řádku 4 a 10 vidíme výchozí nastavení vzdálenosti a její počítání.
- Na řádku 11 a 12 omezujeme kužel. Jakmile dojdeme při výpočtu ke konceptům jejichž vzdálenost od výchozího je rovna nebo větší než *height*, tak je nezařazujeme do fronty konceptů čekajících na zpracování.

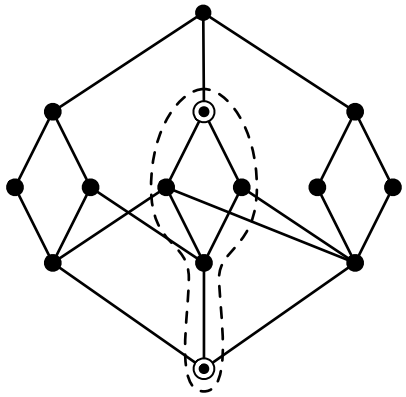
Nyní přistoupíme k určení časové složitosti. Výše máme uvedené odlišnosti oproti výchozímu algoritmu. Přidání a odebrání prvku z fronty má konstantní časovou složitost $O(1)$. Inkrementace čísla a porovnání čísel má také $O(1)$. Všechny tyto operace se opakují tak dlouho, dokud není dosaženo základny. Výsledná časová složitost celého algoritmu je $O(|\mathcal{L}||X||Y|^2)$.

Stejně jako při odvozování časové složitosti pro algoritmus *Next Neighbors* se ve fuzzy případě navíc projeví $|L|$. Dostáváme tedy $O(|\mathcal{L}||L||X||Y|^2)$.

4.2. Podsvaz

V této kapitole se budeme zabývat algoritmem na výpočet podsvazu v konceptuálním svazu. K tomu, abychom tuto úlohu mohli vyřešit, nám stačí zadat

dva koncepty a kontext. Jako výsledek očekáváme podsvaz, ve kterém zadané koncepty tvoří největší a nejmenší prvek. Výsledek je ilustrován na následujícím obrázku. Zadané koncepty jsou označeny v kroužku a výsledný podsvaz přerušovanou čárou.



Obrázek 5. Ukázka výsledku algoritmu podsvaz.

Je důležité uvést, že úloha nemá řešení v případě, kdy zadáme koncepty, které nejsou vzájemně srovnatelné.

Nyní si uvedeme vstup a výstup algoritmu a popíšeme si jej pomocí pseudokódu.

Vstup: dva koncepty $\langle A_1, B_1\rangle, \langle A_2, B_2\rangle$, které určují největší a nejmenší prvek hledaného podsvazu v tomto pořadí a kontext označný jako $\langle X, Y, I\rangle$.
Výstup: $\mathcal{L}$ je nelezený podsvaz.

```
SUBLATTICE (⟨A1, B1⟩, ⟨A2, B2⟩, ⟨X, Y, I⟩)
 1  c ← ⟨A1, B1⟩
 2  insert (c, L)
 3  loop
 4     foreach x in Neighbors (c, ⟨X, Y, I⟩)
 5        if x ≥ ⟨A2, B2⟩ then
 6           try x ← lookup (x, L)
 7           with NotFound then
 8              insert (x, L)
 9              insert (x, Q)
10           x* ← x* ∪ {c}
11           c* ← c* ∪ {x}
12     try c ← pop (c, Q)
13     with NotFound → exit
14  return L
```

Změny oproti LATTICE provedené v SUBLATTICE jsou následující:

- Q je fronta konceptů čekajících na zpracování.
- Na 3. řádku testujeme zda právě vypočtený koncept je nadkonceptem nejmenšího prvku podsvazu. Pokud ano:
  - provážeme vypočtený koncept s jeho sousedem,
  - nebyl-li dosud přidán do generovaného podsvazu (test na 7. řádku), přidáme ho do podsvazu i do fronty nezpracovaných konceptů.

Uvedené odlišnosti oproti výchozímu algoritmu použijeme při odvození časové složitosti. Přidání a odebrání prvku z fronty má konstantní časovou složitost $O(1)$. Test na podkoncept má také $O(1)$ a je více rozebrán v části textu popisující implementaci v sekci 6.1.1. Tudíž výsledná časová složitost celého algoritmu je $O(|\mathcal{L}||X||Y|^2)$.

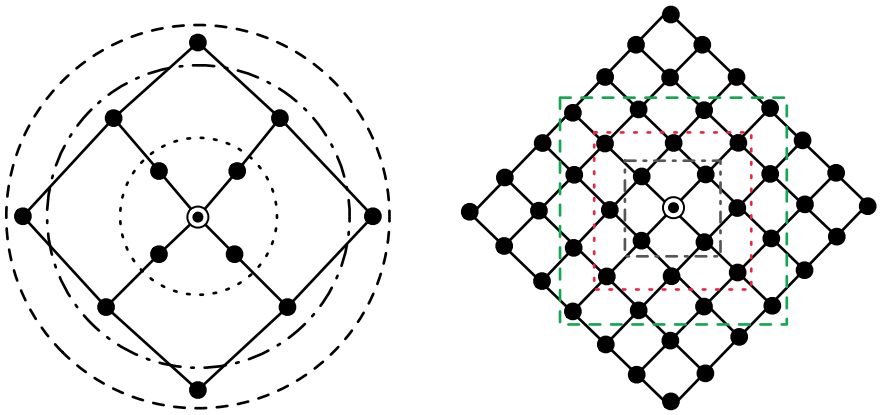
Časová složitost pro fuzzy případ je $O(|\mathcal{L}||L||X||Y|^2)$.

4.3. Kruh

Algoritmus, který bude uveden na následujících řádcích, vygeneruje kolem zadaného konceptu okolí o zadané velikosti. Velikost bude zadána jako číslo a bude mít podobný význam jako v algoritmu pro výpočet kužele. Velikost udává počet hran mezi zadaným konceptem a koncepty na okraji kruhu. Tato úloha se dá připodobnit k vytvoření kruhu, kde zadaný koncept je střed kruhu a velikost je jeho poloměr. Odtud tedy plyne název tohoto algoritmu. Avšak jak jde vidět na obrázku 6., výsledek nemusí mít kruhový tvar a ani ve většině případů mít nebude. Tvar bude záviset na svazovém uspořádání konceptů v okolí zadaného konceptu. Taky se zde projeví nejednoznačnost délky cesty stejně jako u výpočtu kužele a taky na nakreslení Hasseova diagramu svazu.

Když už víme, jak má algoritmus fungovat, popíšeme si jej pomocí pseudokódu. Oproti předcházejícím má nyní dvě části. Hlavní část je Circle, která určí koncept pro aktuální zpracování. Ten se předá části Neighborhood a Neighborhood-dual, která vypočte jejich dolní a horní sousedy. Pokud je vzdálenost právě vypočtených konceptů menší než zadaná velikost kruhu, přidají se do fronty na další zpracování.

Abychom nemuseli neustále předávat jako parametry $\langle X, Y, I\rangle$, $\mathcal{L}$, radius, Q jsou globální vůči oběma pseudokódům.



Obrázek 6. Ukázka výsledků algoritmu kruh.

Vstup: $\langle A, B\rangle$ je koncept, pro který se hledají jeho dolní sousedé
NEIGHBORHOOD ($\langle A, B\rangle$)
1 **foreach** x **in** Neighbors ($\langle A, B\rangle, \langle X, Y, I\rangle$)
2 **try** $x \leftarrow$ lookup $(x, \mathcal{L})$
3 **with** NotFound **then**
4 insert $(x, \mathcal{L})$
5 $x_d \leftarrow \langle A, B\rangle_d + 1$
6 **if** $x_d <$ radius **then**
7 insert (x, Q)
8 $x^* \leftarrow x^* \cup \{\langle A, B\rangle\}$
9 $\langle A, B\rangle_* \leftarrow \langle A, B\rangle_* \cup \{x\}$

Na 5. řádku máme symbolem $_d$ označenou vzdálenost příslušného konceptu od středu. Limitní podmínka, která v případě jejího nesplnění nezahájí další výpočty pro sousedy předaného konceptu, se nachází na 6. řádku.

Vstup: $\langle A, B\rangle$ je koncept, od kterého se mají zjistit všechny koncepty do vzdálenosti radius a kontext $\langle X, Y, I\rangle$.
Výstup: $\mathcal{L}$ hledané kruhové okolí.
CIRCLE ($\langle A, B\rangle$, radius, $\langle X, Y, I\rangle$)

```
1  c ← ⟨A, B⟩
2  insert (c, L)
4  c_d ← 0
5  loop
6     Neighborhood (c)
7     Neighborhood-dual (c)
8     try c ← pop (c, Q)
9     with NotFound → exit
10 return L
```

Na závěr odvodíme časovou složitost. Část NEIGHBORHOOD jednou zavolá NEIGHBORS, o které víme ze sekce 3.2., že má časovou složitost $O(|X||Y|^2)$. Dále se zde zjišťuje, zda jsme daný koncept již uložili do $\mathcal{L}$, uložení konceptu do $\mathcal{L}$, inkrementace a srovnání čísel, vložení konceptu do fronty nezpracovaných konceptů a vytvoření hran mezi koncepty. Všechny tyto operace mají složitost $O(1)$. Tudíž časová složitost části NEIGHBORHOOD je $O(|X||Y|^2)$. Určení časové složitosti pro část CIRCLE je velice jednoduché. Když pomineme volání NEIGHBORHOOD a NEIGHBORHOOD-DUAL, tak všechny operace prováděné v části CIRCLE mají časovou složitost $O(1)$. Cyklus, který začíná na 5. řádku se opakuje tolikrát, kolik je konceptů ve výsledku. Z toho dostáváme, že $O(|\mathcal{L}||X||Y|^2)$ je výsledná časová složitost celého algoritmu.

Časová složitost pro fuzzy případ je $O(|\mathcal{L}||L||X||Y|^2)$.

4.4. Oblast

Myšlenka, která nás motivuje k vytvoření tohoto algoritmu je, že ze zadaných konceptů chceme vytvořit takovou část konceptuálního svazu, která obsahuje všechny tyto koncepty a koncepty, které se nachází mezi nimi.

4.4.1. Průběh řešení

Algoritmus pro výpočet oblasti je jednoznačně nejkomplikovanějším ze všech uvedených algoritmů. Než se dostal do podoby, ve které si jej představíme v sekci 4.4.2. prošel několika zásadními změnami. Tyto změny nám mohou posloužit jako varování kudy se už nevydávat nebo jako podnět pro jiné uplatnění. Nastíníme si je nyní.

1. Jako první jsme se pokoušeli popsat oblast pomocí čtyřech konceptů, které by tvořili vrcholy obdélníku. Řešení mělo být takové, že na levé straně zjistíme počet hran h mezi zadanými koncepty a budeme se snažit postupovat

směrem doprava a to tak, abychom oblast postupně rozšiřovali, dokud v ní nebudou obsaženy koncepty na pravé straně. Přitom dbáme na to, aby se generovaná oblast neroztahovala do výšky. Tyto podmínky bychom se snažili splnit tak, že pro každý koncept si určíme do jaké vzdálenosti může zjišťovat sousedy pro směr nahoru a dolů. Má-li v daném směru vzdálenost větší než 0, vypočítá nejbližší sousedy, kterým předá vzdálenost ve směru, ve kterém se vydal o jedna menší a v druhém o jedna větší. Výpočet se pak postupně přesune do jednotlivých nejbližších sousedů. Jako výchozí koncept bude zvolen levý dolní roh, který má vzdálenost směrem nahoru h a dolů 0.

Problém nastane v případě, kdy mineme některý z pravých zadaných konceptů. Výpočet pak pokračuje tak dlouho, dokud nenarazí na pravý okraj svazu. Vycházíme totiž z faktu, že neznáme celý svaz, tudíž nevíme kolik hran se mezi jednotlivými koncepty nachází. Kdybychom počet hran znali, tak by stejně tento problém nastal při přechodu mezi různými úrovněmi faktorizace.

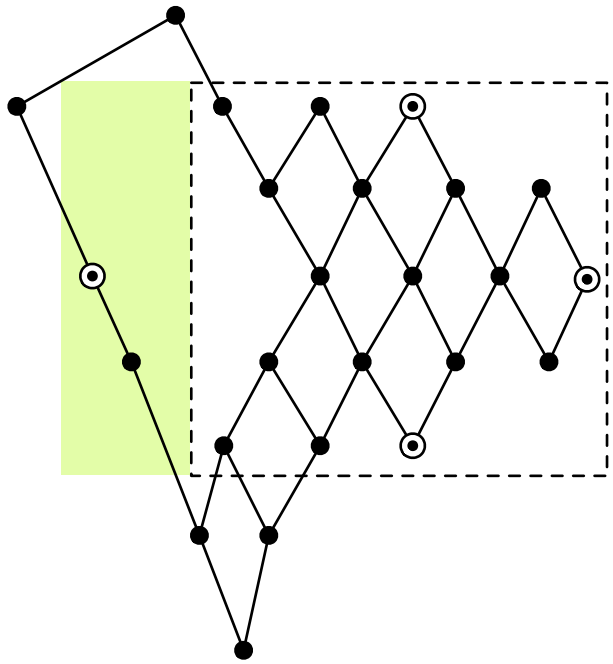
Mohli bychom použít zrcadlový postup zprava doleva a vybrat třeba horní koncept jako výchozí, ale řešili bychom stejné problémy.

2. Další varianta vycházela z předchozí, ale koncepty neurčovali vrcholy obdélníku. Nyní určovali hranice (levá, pravá, horní a dolní) pro oblast. Jako výchozí si můžeme vybrat levý nebo pravý koncept a zjistit jeho maximální vzdálenosti směrem nahoru a dolů. Zde nastává první problém: co když cesta není v přímém směru? (Tato situace muže nastat i v předchozí variantě!) To můžeme vyřešit tím, že pokud při hledání cesty měníme směr, tak odečteme z výsledné vzdálenosti počet projitých hran v opačném směru.

   U tohoto řešení se může stát, že výpočet skončí dříve než dosáhneme posledního konceptu. Na 7. obrázku máme zobrazenou část svazu a v ní čárkovaně označenu vypočtenou oblast, když budeme postupovat zprava doleva. Oblast by ale měla obsahovat i podbarvenou část, ve které jsou dva koncepty, ale mohlo by jich tam být i více. Tyto koncepty nebyly zahrnuty do výsledné oblasti, jelikož k nim neexistovala cesta, aniž by se porušily hranice oblasti.

3. Oblast si necháme zadat, tak jako v prvním případě, ale postupovat budeme jinak. Najdeme cesty mezi zadanými koncepty a spočítáme jejich vzdálenosti. Z nich vybereme minimální, maximální nebo průměrnou vzdálenost. Volbu bychom mohli ovlivnit pomocí parametru. Z každého rohu bychom se snažili počítat koncepty oblasti do zvolené vzdálenosti. Snažili bychom se postupovat v úhlu 90 °. Sjednocením dílčích oblastí bychom dostali výslednou oblast.

   V případě, kdy nějaká cesta nebo všechny cesty jsou mimo oblast, tak nám



Obrázek 7.

mohou negativně ovlivnit výslednou vzdálenost. Fakt, že leží mimo, v této fázi nevíme, protože nevíme, jak výsledná oblast vypadá, z toho důvodu se nemůžeme omezovat a navíc cesta v ní nemusí existovat. Výsledná vzdálenost pak může být příliš malá a výsledné oblasti bude chybět její střed.

V porovnání s výsledným algoritmem na výpočet oblasti může negativní výsledek v tomto řešení způsobit i jedna cesta. Pro způsobení chybějící středové části výsledného algoritmu, by museli všechny cesty existovat mimo oblast a být kratší než cesty vedoucí přes oblast. Navíc určování správného směru v tomto řešení by bylo komplikované.

4. V této verzi algoritmu jsme se snažili odstranit všechny špatné stránky z předchozích verzí:

   (a) zadání oblasti bez potřeby znát část svazu

   (b) zadání oblasti bez potřeby určení jak má daný koncept oblast ohraničovat

   (c) určování vzdáleností

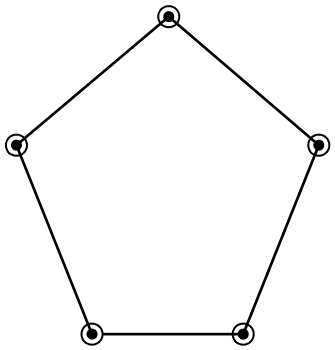
   (d) určování směrů

   Proto na vstupní koncepty nebudeme klást žádné podmínky. Budeme je chápat jako množinu, ke které chceme získat její nejmenší konvexní obal.

Ten získáme pomocí algoritmu na kruhovou oblast, který upravíme tak, aby místo vstupního parametru na poloměr očekával množinu bodů, které až pohltí, tak se přestane zvětšovat. To znamená, že si určíme jeden koncept jako střed a ostatní určují konec. Tento postup aplikujeme na všechny koncepty. Výsledná oblast je průnik kruhových okolí. Tento algoritmus se stal základem pro výsledný algoritmus na výpočet oblasti, který si popíšeme v následující sekci.

4.4.2. Výsledné řešení

Úlohu nalezení oblasti si můžeme připodobnit k úloze, jak nalézt konvexní obal, který obsahuje dané prvky a je nejmenší této vlastnosti. Ideálně vypočtenou oblast znázorňuje obrázek 8.

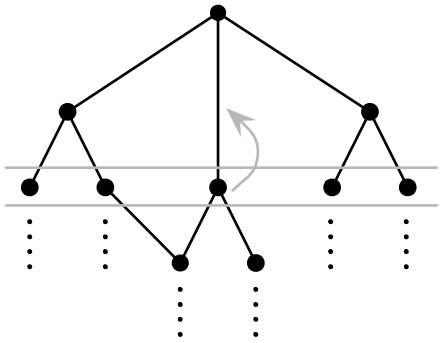


Obrázek 8. Nejmenší konvexní obal.

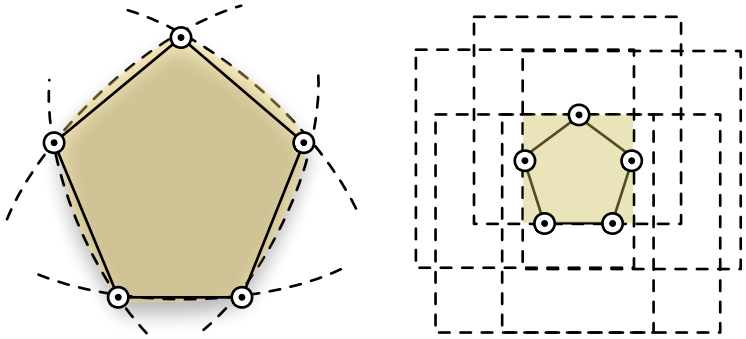
V FCA není definován pojem konvexní obal ani výška konceptu ve svazu. Což nám u některých konceptů dává volnost při rozhodnutí, zda patří do oblasti či nikoliv. Nejlépe je absence těchto pojmů vidět na 9. obrázku, kde máme zvýrazněnu hranici oblasti, která pokračuje směrem dolů. Zde se nám projevuje problém určení výšky konceptu. Máme prostřední koncept ve vyznačené linii přidat do oblasti či nikoliv? Odpověď není jednoznačná.

Proto jsme se rozhodli, nevytvářet nejmenší konvexní obal (což není ani po teoretické stránce možné), ale mírně nafouknutý oproti nejmenšímu. Na 10. obrázku vidíme, jak může nafouknutí vypadat. Jeho tvar vychází z uspořádání konceptů v oblasti a z toho, že využíváme algoritmus pro výpočet kruhového okolí. Jakmile si vysvětlíme fungování algoritmu, bude nám hned jasné, proč se konvexní obal takto nafukuje.

I když námi vypočtená oblast není ta nejmenší, může se při jejím zobrazování stát, že v některých případech budeme mít pocit, jako by něco chybělo. To je ale způsobeno pouze algoritmem pro vizualizaci. Z toho plyne závěr, že musíme vědět, co chceme za výsledek a ne se upínat na zobrazení.



Obrázek 9. Patří koncept do obalu?



Obrázek 10. Mírně nafouknutý konvexní obal určený zadanými prvky.

Algoritmus pro výpočet oblasti se skládá z následujících kroků. Ze vstupních konceptů vybereme jeden, od kterého generujeme kruhové okolí tak dlouho, dokud nepohltíme všechny vstupní koncepty. Tento krok opakujeme postupně se všemi vstupními koncepty, ale již procházíme jen oblast z předchozího kroku a u každého konceptu si pamatujeme, kolikrát jsme jej navštívili. Výstupem jsou ty koncepty, které mají počet návštěv roven počtu vstupních konceptů.

Nyní budou následovat pseudokódy pro výpočet oblasti. Proměnná $\mathcal{L}$ je globální. Proměnné $\langle X, Y, I \rangle$, visit, radius, concepts, Q jsou globální vůči všem částem algoritmu kromě Area.

TestArea rozhodne, zda se má generovaná oblast dále rozšiřovat. Pokud nejsou všechny vstupní koncepty v právě vygenerované oblasti, tak pokračujeme v rozšiřování. Pokud jsou, tak vrátíme výsledek testu, zda vzdálenost testovaného od výchozího konceptu je menší než vzdálenost posledního pohlceného konceptu od výchozího.

Vstup: $\langle A, B\rangle$ je testovaný koncept.
Výstup: pravdivostní hodnota určující, zda má být testovaný koncept přidán do oblasti.

```
TestArea (⟨A, B⟩)
1  if concepts ≠ ∅ then
2     concepts ← concepts \ ⟨A, B⟩
3     if concepts ≠ ∅ then
4        return true
5     else
6        radius ← ⟨A, B⟩_d
7        return false
8  else return ⟨A, B⟩_d < radius
```

Z proměnné concepts se postupně odebírají ty koncepty, které jsou již v oblasti obsaženy. Jakmile je prázdná, tak víme, že v oblasti máme všechny vstupní koncepty.

Část Neighborhood2 se od Neighborhood použité při řešení úlohy na zjištění kruhového okolí v sekci 4.3. liší pouze na 6. řádku. Neprovádí se zde test na rádius, ale volá se TestArea.

Vstup: $\langle A, B\rangle$ je koncept, pro který se hledají jeho dolní sousedé.

```
Neighborhood2 (⟨A, B⟩)
1  foreach x in Neighbors (⟨A, B⟩, ⟨X, Y, I⟩)
2     try x ← lookup (x, ℒ)
3     with NotFound then
4        insert (x, ℒ)
5        x_d ← ⟨A, B⟩_d + 1
6        if TestArea (x) then
7           insert (x, Q)
8     x* ← x* ∪ {⟨A, B⟩}
9     ⟨A, B⟩_* ← ⟨A, B⟩_* ∪ {x}
```

Zda-li se mají pro koncept zjišťovat jeho sousedé nám řekne TestConcept. Tento test je důležitý pro určení hranice oblasti. Může se totiž stát, že některé koncepty, které čekají na zpracování, přeskočíme z následujícího důvodu. Koncepty totiž přidáváme do fronty nezpracovaných, pokud prošly testem TestArea. Po přidání posledního konceptu a určení maximální povolené vzdálenosti od výchozího se nám může stát, že některé koncepty čekající na zpracování jsou dále než je maximální povolená vzdálenost a proto je musíme přeskočit.

Vstup: $\langle A, B\rangle$ je testovaný koncept.
Výstup: pravdivostní hodnota určující, zda testovaný koncept patří do oblasti.

```
TestConcept (⟨A, B⟩)
1  if concepts ≠ ∅ then
2     return true
3  else return ⟨A, B⟩_d < radius
```

NextConcept z fronty nezpracovaných vrátí první koncept splňující podmínku TestConcept. Koncepty, které byly před ním se dále nezpracovávají.

Výstup: proměnná c, ve které je uložen koncept pro další zpracování, není-li takový, je proměnná prázdná.

```
NextConcept ()
1  c ← null
2  loop do
3     c ← pop (c, Q)
4     while c ≠ null and not TestConcept (c)
5  return c
```

Pro vstupní koncept $\langle A, B\rangle$ část Circle vygeneruje nejmenší kruhové okolí, které obsahuje všechny koncepty z concepts.

Vstup: koncept $\langle A, B\rangle$, množina konceptů concepts.
Výstup: $\mathcal{L}$ je výsledné kruhové okolí se středem v $\langle A, B\rangle$ obsahující všechny koncepty z concepts.

```
Circle (⟨A, B⟩, concepts)
1  c ← ⟨A, B⟩
2  insert (c, L)
3  c_d ← 0
4  loop
5     Neighborhood2 (c)
6     Neighborhood2-dual (c)
7     try c ← NextConcept ()
8     while NotFound → exit
9  return L
```

Neighborhood3 na již existující části svazu prochází koncepty, které splňují TestArea a zvyšuje jim návštěvnost.

Pro vstupní koncept se Neighborhood3 po již vypočtené části posune na dolní sousedy, kteří byli navštíveni tolikrát, kolik bylo určeno ve vstupním parametru visit v CircleBrowse. Pokud sousedé vyhovují TestArea přidají se do fronty nezpracovaných

Vstup: $\langle A, B\rangle$ je koncept, přes který se posouváme na jeho sousedy.
NEIGHBORHOOD3 ($\langle A, B\rangle$)
1 **foreach** x **in** $\langle A, B\rangle_*$
2 **if** $x_v =$ visit **then**
3 $x_v \leftarrow x_v + 1$
4 $x_d \leftarrow x_d + 1$
5 **if** TestArea (x) **then**
6 insert (x, Q)

CIRCLEBROWSE na již existující části svazu prochází koncepty, které splňují TESTAREA a zvyšuje jim návštěvnost. Část CIRCLE prochází okolí $\langle A, B\rangle$ v postupně se zvětšujících kruzích. Ignoruje koncepty, které nemají návštěvnost rovnu vstupnímu parametru visit. Zastaví se, až pohltí všechny koncepty z concepts.

Vstup: koncept $\langle A, B\rangle$, množina konceptů concepts a návštěvnost visit.
CIRCLEBROWSE ($\langle A, B\rangle$, concepts, visit)
1 $c \leftarrow \langle A, B\rangle$
2 $c_d \leftarrow 0$
3 $c_v \leftarrow$ visit $+1$
4 **loop**
5 Neighborhood3 (c)
6 Neighborhood3-dual (c)
7 **try** $c \leftarrow$ NextConcept ()
8 **while** NotFound $\rightarrow$ **exit**

Nyní přejdeme ke vstupní části AREA.

1. Ze vstupního parametru concepts se pomocí části CIRCLE vygeneruje nejmenší kruhové okolí se středem v prvním konceptu z concepts, které obsahuje všechny koncepty z concepts.

2. Pro všechny koncepty x z concepts kromě prvního provádíme následující:

   (a) procházíme výsledek z prvního kroku pomocí CIRCLEBROWSE se středem v x,

   (b) inkrementujeme počítadlo návštěv.

3. Odstraníme koncepty, které mají menší návštěvnost než je aktuální. Ve zbývajících konceptech odebereme hrany na odstraněné koncepty.

Vstup: concepts je množina konceptů určujících hledanou oblast,
$\langle X, Y, I \rangle$ je kontext, nad kterým probíhá výpočet.
Výstup: $\mathcal{L}$ je vypočtená oblast splňující vstupní parametry.

```
AREA (concepts, ⟨X, Y, I⟩)
1  c ←first(concepts)
2  others ←concepts \c
3  L ←Circle (c, others)
4  v ← 0
5  foreach x in others
6     CircleBrowse (x, concepts \x, v)
7     v  ← v  + 1
8  RemoveAll (L, v + 1)
9  return L
```

V části TestArea se neprovádějí žádné složité operace, proto je její časová složitost rovna $O(1)$. Část Neighborhood2 je až na 6. řádek stejná jako část Neighborhood, tudíž její časová složitost je $O(|X||Y|^2)$. Časová složitost části TestConcept je $O(1)$. V části NextConcept v nejhorším případě $|Q|$-krát voláme část TestConcept, což můžeme zanedbat, proto je časová složitost části NextConcept rovna $O(1)$. Část Circle $|\mathcal{L}|$-krát volá část Neighborhood2, Neighborhood2-dual a NextConcept. Časová složitost části Circle je $O(|\mathcal{L}||X||Y|^2)$. V části Neighborhood3 pouze procházíme již vypočtenou část svazu, a proto je její časová složitost rovna $O(1)$. Časová složitost části CircleBrowse je také rovna $O(|\mathcal{L}|)$, jelikož se maximálně $|L|$-krát zavolají části s časovou složitostí $O(1)$. Odstranění přebytečných konceptů pomocí *RemoveAll* proběhne s časovou složitostí $|\mathcal{L}|$. Časová složitost části Area a tedy i výsledná složitost celého algoritmu je $O(|\mathcal{L}||X||Y|^2)$.

Časová složitost pro fuzzy případ je $O(|\mathcal{L}||L||X||Y|^2)$.

5. Faktorizace fuzzy konceptuálního svazu dle podobnosti

V této kapitole si ukážeme způsob, jak vypočítat faktorový konceptuální svaz ve fuzzy případě. Tohoto můžeme využít v situaci, kdy je svaz příliš velký a nevadí nám pracovat s jeho faktory.

Faktory budou tvořeny na základě podobnosti intentů [5], duálně bychom mohli definovat pro extenty.

Definice 5.1. (stupeň podobnosti pro fuzzy množiny) *Stupeň podobnosti* pro $B_1, B_2 \in L^Y$ určíme jako

$$B_1 \approx B_2 = \bigwedge_{y \in Y} B_1(y) \leftrightarrow B_2(y) = \bigwedge_{y \in Y} B_1(y) \rightarrow B_2(y) \wedge B_2(y) \rightarrow B_1(y)$$

Ve všech algoritmech jsme používali operátory $^{\downarrow}$ a $^{\uparrow}$. Dvojici těchto operátorů můžeme nahradit obecným fuzzy uzávěrovým operátorem C, který si nyní nadefinujeme.

Definice 5.2. (fuzzy uzávěrový operátor) *Fuzzy uzávěrový operátor* na množině X je zobrazení $C : L^X \rightarrow L^X$ splňující pro všechna $A, A_1, A_2 \subseteq L^X$ podmínky:

(a) $A \subseteq C(A)$,

(b) $A_1 \subseteq A_2 \Rightarrow C(A_1) \subseteq C(A_2)$,

(c) $C(A) = C(C(A))$.

Definice 5.3. (fixní body fuzzy uzávěrových operátorů) Pro fuzzy uzávěrový operátor $C : L^X \rightarrow L^X$ se množina

$$fix(C) = \{A \subseteq L^X \mid C(A) = A\}$$

nazývá *množina fixních bodů* C.

Nyní si podle [5] uvedeme fuzzy uzávěrový operátor C_a. Tím docílíme toho, že po jeho dosazení do algoritmů na výpočet celého fuzzy konceptuálního svazu dostaneme pouze jeho faktorový svaz. Po dosazení do algoritmů na výpočet části fuzzy konceptuálního svazu dostaneme pouze část faktorového svazu. Zdůrazněme, že tak provedeme bez nutnosti generování celého svazu.

Definice 5.4. (podobnost ve stupni) *Podobnost ve stupni alespoň a* definujeme jako

$$B_1 \approx^a B_2 \text{ pk. } B_1 \approx B_2 \geq a$$

Blok je maximální podmnožina, ve které jsou si každé dva koncepty podobné alespoň ve stupni a. Množinu bloků (spolu s uspořádáním bloků $\prec$ definovaným pomocí uspořádání infim $\langle A, B\rangle_a$ resp. suprem $(\langle A, B\rangle_a)^a$ bloku) označujeme faktorový svaz $(\mathcal{B}(X, Y, I)/\approx^a, \prec) = \mathcal{L}/\approx^a$.

Lemma 5.1. ($\approx^a$-bloky) *$\approx^a$-bloky* jsou intervaly z $\mathcal{B}(X, Y, I)$ definovány jako

$$\mathcal{B}(X, Y, I)/\approx^a = \{[\langle A, B\rangle_a, (\langle A, B\rangle_a)^a] \mid \langle A, B\rangle \in \mathcal{B}(X, Y, I)\}$$

Pomocí následující věty dostaneme z množiny všech infim faktorů obsahujících B intent nejpodobnějšího infima k $B^{\downarrow\uparrow}$. Množinu všech takových to intentů označujeme IIB. Duálně z množiny všech suprem faktorů obsahujících A dostaneme extent nejpodobnějšího suprema k $A^{\uparrow\downarrow}$. Množinu všech takových extentů označujeme ESB.

Věta 5.1. Pro daná vstupní data $\langle X, Y, I\rangle$ a práh podobnosti $a \in L$ je zobrazení C_a, které každou fuzzy množinu $B \in Y$ zobrazí do fuzzy množiny $a \to (a \otimes B)^{\downarrow\uparrow} \in Y$, fuzzy uzávěrový operátor v Y, pro který platí $\mathrm{fix}(C_a) = \mathrm{IIB}(a)$.

Na základě volby parametru a dostáváme následující tvrzení:

(a) Je-li $a = 1$ dostaneme celý svaz. Stejné jako bychom dělali $^{\downarrow\uparrow}$.

(b) Je-li $a = 0$ svaz nám zdegeneruje na jeden koncept.

(c) Je-li $a \in (0, 1)$ dostaneme netriviální faktorový svaz.

Poznamenejme, že případy 1 a 2 jsou triviální faktorové svazy.

Lemma 5.2. Pro daná vstupní data $\langle A, B\rangle \in \langle X, Y, I\rangle$ platí

(a) $\langle A, B\rangle_a = \langle (a \otimes A)^{\uparrow\downarrow}, a \ \to B\rangle$

(b) $\langle A, B\rangle^a = \langle (a \to A), (a \otimes B)^{\downarrow\uparrow}\rangle$

Předcházející lemma nám říká, jak k danému konceptu zjistit nejpodobnější infimum a supremum faktoru obsahujícího daný koncept. Toho využijeme v algoritmech, které střídavě generují horní a dolní sousedy, jelikož je potřeba z infima faktoru přejít k supremu faktoru a naopak. Nutnost přechodu vyplývá z věty 5.1.

Následující pseudokódy jsou jednoduchým a praktickým využitím předchozího lemmatu.

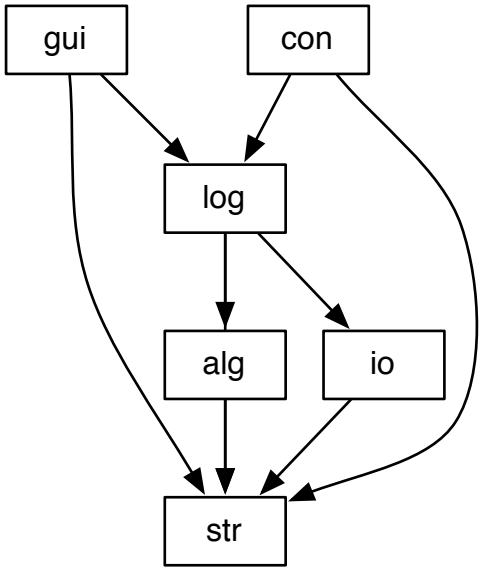
```
LowerRepresentative (⟨A1, B1⟩)
1  return ⟨(a ⊗ A)↑↓, a → B⟩
```

```
UpperRepresentative (⟨A1, B1⟩)
1  return ⟨(a → A), (a ⊗ B)↓↑⟩
```

6. Implementace

Tato práce byla implementována v programovacím jazyce *Lisp* v prostředí *LispWorks Personal Edition 5.1.1*. Pro vytvoření spustitelného souboru v prostředí Windows byla použita verze *LispWorks Professional Edition 5.1.1* pro Windows. Pokud bychom chtěli spustitelný soubor pro Mac OS X nebo GNU/Linux, tak nejjednodušší cesta je získání verze professional pro příslušný operační systém. Tato verze je ovšem placená. Zdarma je pouze verze personal, ve které sice nevytvoříme spustitelný soubor, ale aplikaci v ní spustíme a můžeme používat bez jakéhokoliv omezení. Výsledkem implementace jsou dvě aplikace, které mají stejné výpočetní jádro a balíček pro vstup a výstup. Rozdílné jsou v tom, že jedna se ovládá pomocí příkazové řádky a druhá má jednoduché grafické uživatelské rozhraní.

Implementace je rozdělena do několika balíčků. Název vždy začíná `cpocl`, což je zkratka „Computing part of concept lattice“, pokračuje pomlčkou a zkratkou, k čemu je balíček určen. Pro implementaci struktur je použita zkratka `str`, pro algoritmy `alg`, pro vstup a výstup `io`, pro logiku `log`, pro grafickou nadstavbu `gui` a pro konzoli `con`. Jednotlivé balíčky si blíže představíme v následujících podkapitolách. Než k tomu přistoupíme, tak si uvedeme schéma balíčků, ve kterém máme znázorněny jejich vzájemné vazby a naznačeno jejich využití v aplikacích. Jestliže některý balíček používá jiný, ukazuje na něj šipkou.



Obrázek 11. Schéma balíčků

6.1. Jádro

V jádře jsou naimplementovány nové poznatky této práce. Umožňuje nám vypočítat část konceptuálního svazu, faktorový konceptuální svaz i celý konceptuální svaz. Jádro se skládá z balíčků `cpocl-str`, `cpocl-alg` a `cpocl-log`.

6.1.1. Implementace kontextu a konceptu

V balíčku `cpocl-str` jsou pomocí tříd definovány základní struktury jako je *kontext* a *koncept*. *Kontext* pro crisp případ je reprezentován třídou `context-class`, která má kontext uložen v poli, které obsahuje bitové vektory. Tyto vektory odpovídají řádkům v kontextové tabulce. Bitový vektor je reprezentace objektu, který když má některý bit nastaven na 1, tak to znamená, že má atribut na této pozici.

	atribut 1	atribut 2	atribut 3
objekt 1	x		
objekt 2		x	
objekt 3	x		x

≡

```
#1A(*#100
    *#010
    *#101)
```

Tabulka 1. Vztah kontextové tabulky a uložení kontextu pro crisp případ.

Koncept pro crisp reprezentuje třída `concept-class`. *Intent* je uložen jako vektorové pole o délce rovné počtu atributů v kontextu. Jednotlivé bity jsou nastaveny na 1 pokud *intent* obsahuje atribut na této pozici, jinak 0. *Extent* je uložen jako vektorové pole o délce rovné počtu objektů v kontextu. Jednotlivé bity jsou nastaveny na 1 pokud *extent* obsahuje objekt na této pozici, jinak 0.

To, že máme koncept uložen pomocí bitových polí, nám umožňuje efektivně implementovat operace $^{\uparrow}$,$^{\downarrow}$ a $\leq$. Využíváme bitových operací and a or. Pro $B^{\downarrow}$ uděláme bitový and s takovými sloupci tabulky, které odpovídají atributům z B. Duálně pro $A^{\uparrow}$ kde použijeme odpovídající řádky. Z definice podkonceptu víme, že $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$ p.k. $A_1 \subseteq A_2$ (p.k. $B_2 \subseteq B_1$). Pro $\leq$ nám tedy stačí implementovat $\subseteq$. $C \subseteq D$ p.k. $C = (C$ bit-and $D)$, C a D jsou množiny atributů nebo objektů stejného kontextu.

Pro fuzzy případ jsou použity třídy `fuzzy-context-class` a `fuzzy-concept-class`. Data nejsou uložena v poli bitových vektorů, ale v seznamu seznamů obsahujících čísla z intervalu $\langle 0, 1\rangle$, které nám vyjadřují stupně příslušnosti. Velikosti seznamů a význam jejich pozic je stejný jako v crisp případě, proto jej nebudeme znovu uvádět, pouze ukážeme jejich vztah v 2. tabulce.

Třídy pro reprezentaci kontextů obsahují metody pro výpočet *extentu* pro množinu atributů, duálně pro *intent* a také metody reprezentující uzávěrový ope-

	atribut 1	atribut 2	atribut 3
objekt 1	2/5		
objekt 2		3/5	
objekt 3	4/5		1/5

$\equiv$

```
((2/5 0 0)
 (0 3/5 0)
 (4/5 0 1/5))
```

Tabulka 2. Vztah kontextové tabulky a uložení kontextu pro fuzzy případ.

rátor, jak na atributech, tak i objektech. Ve fuzzy případě jsme použili Lukasiewiczovu algebru pro implementaci spojek $\otimes$ a $\rightarrow$.

Abstraktní třídy `abstract-context-class` a `abstract-concept-class` jsou předci pro crisp i fuzzy kontexty a koncepty. Jejich význam bude jasný po vysvětlení implementace algoritmů.

6.1.2. Implementace algoritmů

Algoritmy jsou naimplementovány v balíčku `cpocl-alg`. Výpočty probíhají tak, jak jsme si je popsali v kapitole 4. na straně 17. Velkou výhodou je to, že vstupní argumenty jsou abstraktní kontext a koncept, což nám umožňuje mít z velké části jedinou implementaci pro crisp i fuzzy případ. Pouze části Neighbors a Neighbors-dual jsou implementovány zvlášť pro crisp i fuzzy případ. Výstupem všech algoritmů je hešovací tabulka, ve které máme uloženy vypočtené koncepty spolu s jejich konceptovým uspořádáním.

Pro úplnost poznamenejme, že algoritmus na výpočet podsvazu navíc testuje, zda byly koncepty zadány ve správném pořadí. Pokud ne, tak je prohodí. Pokud zjistí, že jsou vzájemně nesrovnatelné, tak aniž by výpočet začal je rovnou ukončen.

6.1.3. Logika

Kód, který je společný pro řízení i uchování dat pro konzolovou aplikaci a aplikaci s grafickým uživatelským prostředím, je umístěn v balíčku `cpocl-log`.

6.2. Práce se vstupem a výstupem

Ošetření vstupů a výstupů se nachází ve třídě `io-data-class` v balíčku `cpocl-io`.

Crisp kontexty se načítají v textovém formátu „dat“, který je popsán v [6]. Stejného zápisu využíváme i při načítání konceptů jako vstupních parametrů pro algoritmy. U konceptů jsou očekávány pouze jejich intenty, každý je uveden na samostatném řádku, který obsahuje výčet atributů v jeho intentu. Pro fuzzy případ jsme si tento formát rozšířili tak, že za číslem atributu následuje „/“ a jeho stupeň pravdivosti jako celé číslo.

Pro uložení výsledku používáme formátu xml. Ukázku crisp výstupu můžeme vidět níže. Fuzzy případ si opět rozšíříme o „/“ a stupeň pravdivosti stejně jako u vstupu. Tento formát je zvolen proto, aby se výsledek mohl zobrazit v programu jLatVis [7]. Bohužel vývoj aplikací probíhal současně a jLatVis neumí načíst fuzzy koncepty. V ukázce vidíme svaz, který obsahuje dva koncepty a jejich vzájemný vztah.

```
<LatVis version="2.0">
  <Lattices>
    <Lattice>
      <Name>null</Name>
      <Elements>
        <Element id="1">
          <Tag name=fca:concept:extent>3 5 </Tag>
          <Tag name=fca:concept:intent>1 2 9 11 12 </Tag>
        </Element>
        <Element id="2">
          <Tag name=fca:concept:extent>3 5 8 </Tag>
          <Tag name=fca:concept:intent>9 11 12 </Tag>
        </Element>
      </Elements>
      <Order>
        <Pair greater="2" less="1"/>
      </Order>
    </Lattice>
  </Lattices>
</LatVis>
```

6.3. Konzolová aplikace

Pomocí příkazové řádky sdělíme aplikaci, jaký má být použit algoritmus, na jakých datech a kam má být výsledek uložen. O pochopení těchto příkazů se stará balíček `cpocl-con`. Na ostatní se používá jádro a `cpocl-io`.

Abychom aplikaci mohli dobře využívat, musíme znát přepínače a jejich parametry, které jsou očekávány. Uvedeme si tedy dvě tabulky, ve kterých si představíme, jak přepínače vyžadující parametr, tak přepínače bezparametrické.

Při zadávání cesty, která obsahuje mezery, ji nesmíme zapomenout dát do uvozovek. Došlo by k jejímu rozdělení, což by vedlo k nenalezení vstupního souboru nebo špatnému uložení výsledku. Bezparametrické přepínače budeme používat pro volbu algoritmu a pro vyvolání nápovědy.

přepínač	parametr
-t	cesta k souboru obsahující kontextovou tabulku
-k	cesta k souboru obsahující koncepty, které slouží jako parametry pro zvolený algoritmus
-v	cesta pro uložení výsledku
-p	přirozené číslo, které je nutné zadat pro algoritmus kruh, kde udává jeho velikost a pro algoritmus kužel, kde udává jeho výšku
-s	stupeň podobnosti jako číslo z intervalu $\langle 0, 1\rangle$ (pouze pro fuzzy)

Tabulka 3. Parametrické přepínače.

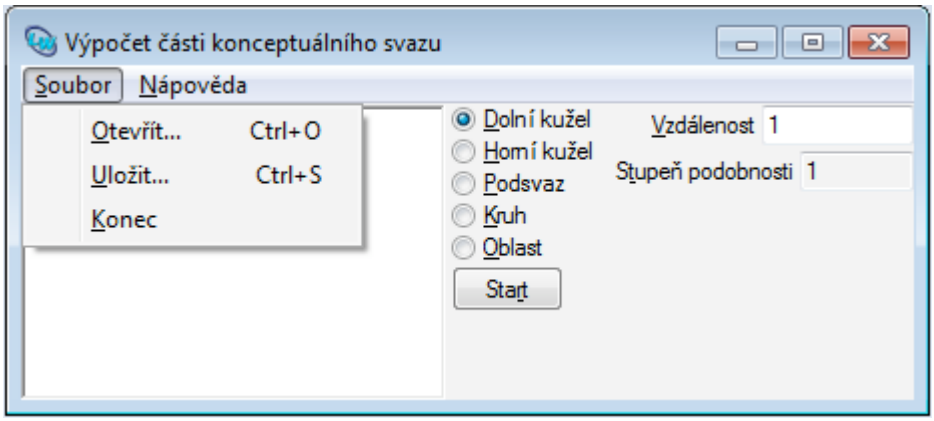
přepínač	význam
-dk	spustí se algoritmus dolní kužel
-hk	spustí se algoritmus horní kužel
-po	spustí se algoritmus podsvaz
-kr	spustí se algoritmus kruh
-ob	spustí se algoritmus oblast
-?	zobrazí nápovědu

Tabulka 4. Bezparametrické přepínače.

6.4. GUI aplikace

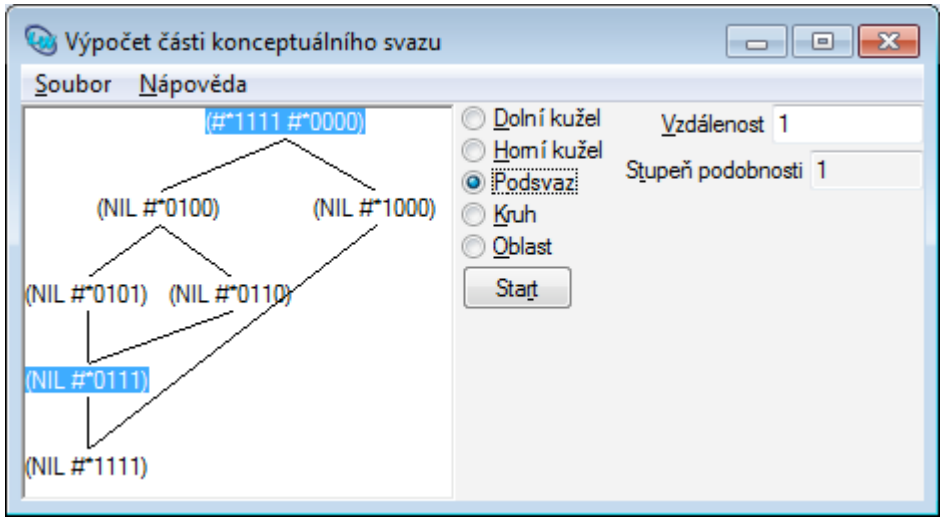
Jedná se grafickou nadstavbu jádra aplikace a `cpocl-io`. Byla vytvořena pro rychlé otestovaní našich algoritmů. Jelikož cílem práce nebylo vytvořit dokonalý zobrazovač svazů, je zobrazení velice primitivní. To nám stačí pro použití na menších kontextech, jinak bychom museli neustále ukládat výsledky do souboru a zobrazovat je pomocí softwaru třetích stran. Pro vykreslení Hassova diagramu byl použit panel `graph-pane` z GUI toolkitu `CAPI`. Díky GUI aplikaci můžeme pohodlně načíst kontext a zobrazit jej jako úplný nebo faktorový svaz. Na zobrazeném svazu lze demonstrovat výsledky algoritmů s možností nastavení jejich vstupních parametrů. Vypočtený svaz je možno uložit pro zobrazení v aplikaci jLatVis [7]. Grafická nadstavba je obsažena v balíčku `cpocl-gui` a velice primitivně nám zobrazuje vypočtené koncepty.

Nyní si na jednoduchém příkladu ukážeme její použití. Při spuštění se nám zobrazí hlavní okno a v menu vybereme Soubor -> Otevřít nebo použije klávesovou zkratku Ctrl+O, jak je znázorněno na obrázku 12. V dialogu na otevření souboru vybereme soubor s uloženým kontextem ve formátu dat [6] nebo jeho rozšíření pro fuzzy kontexty popisované v sekci 6.2.. Pro fuzzy případ se zobrazí dialogové okno k nastavení stupně podobnosti faktorizace. Na zvoleném kontextu se vypočítá konceptuální svaz a zobrazí se v levé části okna.



Obrázek 12.

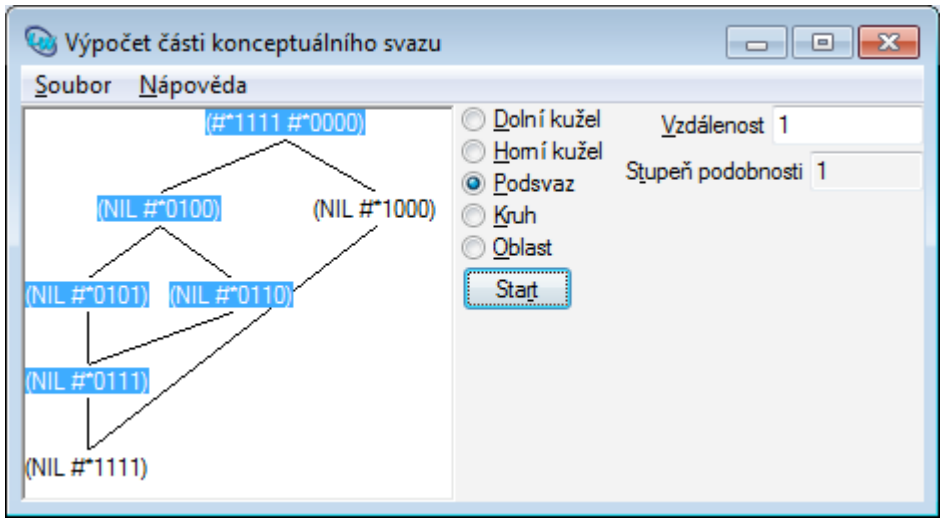
Na zobrazeném svazu můžeme zkoušet všechny naše algoritmy. Stačí vybrat algoritmus, pak pomocí myši označit ve svazu jeho vstupní argumenty a pro kužel či kruh nastavit vzdálenost. Na obrázku 13. je vše nachystáno na výpočet podsvazu.



Obrázek 13.

Po stisku tlačítka Start se označí výsledek, který vidíme na obrázku 14.

Uveďme si důležitou poznámku. I když vidíme svaz, na němž se nám prezentují výsledky zadaných algoritmů, algoritmy vycházejí pouze ze vstupních parametrů a data vypočítána dříve se neberou v potaz. Výsledek, který nám vrátí je v případě nových konceptů navázán na stávající a v případě již existujících konceptů provede jejich označení.


Výpočet části konceptuálního svazu
Soubor
Nápověda
(#*1111 #*0000)
(NIL #*0100)
(NIL #*1000)
(NIL #*0101)
(NIL #*0110)
(NIL #*0111)
(NIL #*1111)
Dolní kužel
Horní kužel
Podsvaz
Kruh
Oblast
Start
Vzdálenost
1
Stupeň podobnosti
1

Obrázek 14.

Závěr

V této diplomové práci jsem představil některé úlohy na výpočet části konceptuálního svazu. Ke každé jsem uvedl její řešení. To jsem popsal, jak teoreticky, tak i na pseudokódu. Ke kterému jsem uvedl jeho časovou složitost. Pro demonstraci jsem všechny algoritmy naimplementoval a je možné je využívat přes konzolovou aplikaci i aplikaci s grafickým uživatelským prostředím.

Při vymýšlení algoritmů jsem narazil na problém s délkou cesty a výškou konceptu ve svazu. Tyto pojmy by bylo dobré jednoznačně definovat nebo se jim vyhnout.

Výsledky mé práce mohou sloužit jako odrazový můstek pro jejich zlepšování nebo jako podnět pro vytvoření nových úloh pro situace, které nejsou dosud pokryty.

Praktickým využitím této práce by mohlo být její spojení s grafickým zobrazovačem svazů jLatVis [7] pro vytvoření elektronické mapy pracující s velkými koncepty. Její ovládání by mohlo být stejné jako současné mapy dostupné na internetu. Nejdříve by se nám zobrazil faktorový svaz (na mapě světa taky nevidíme všechny vesnice). Kolečkem myši bychom se posouvali mezi jednotlivými stupni faktorizace (zobrazování menších měst). To by bylo možné realizovat pomocí algoritmu na výpočet kruhové oblasti, kde střed bude nejbližší koncept kurzoru myši s vhodným poloměrem. Posun mapy by v případě chybějících dat vyvolal dopočítání chybějících konceptů. To by proběhlo na základě konceptů v této oblasti, které ale patří do jiného stupně faktorizace. K tomu by mohl být použit algoritmus na výpočet oblasti.

Mapa by si ale musela poradit s optimalizacemi vykreslování. Při vykreslování může koncepty všelijak přehazovat, aby docílila přehlednějšího (menší počet křížení hran) zobrazení. Tyto optimalizace je nutné zohlednit při zadávání vstupních parametrů algoritmům na dopočet části svazu.

Conclusions

In this diploma thesis I presented several tasks for computing part of a concept lattice. There is a solution attached to each of them, described theoretically and in the pseudo-code to which I added its time requirements as well. For better demonstration, I implemented all the algorithms and it is thus possible to use them through a console application or through an application with graffical user interface.

When developping the algorithms, I encountered a problem with the route lenght and with the height of the concept in the lattice. These terms would require a really specific definition. Otherwise it would be better to avoid them.

The results of my thesis can serve as a good base for its further improvement or as an impulse for creation of new tasks solving situations that has not been covered so far.

A practical usage of this thesis could be in its connection with grafical tool for displaying lattices called jLatVis which can be use for creation of an electronical map working with big lattices. Its control method could be the same as current maps available on the internet. First the factor lattice would be displayed (on the world map, we do not see all the villages either). Then we could get further in particular level of factorisation by mouse scroll (displaying smaller towns). This would be possible thanks to algorithm for calculcation of circle area where the center will be the closest concept of the mouse coursor with the appropriate radius. In case there are some missing data, the map shift would launch a calculation of missing concepts. This would work on the basis of the concepts in this area that belong to different level of factorisation. The algorithm for area calculation could be used for that.

However the map would have to cope with drawing optimalisations. During the drawing, it can variously switch the concepts to achieve a clear view (with smaller number of edge crossing). These optimalizations have to be taken into account when inputting entry parameteres of algorithms for final computing of the lattice part.

Reference

[1] Bělohlávek R.: *Introduction to formal concept analysis.* Elektronické skriptum, 2008.

[2] Lindig C.: *Fast Concept Analysis. Working with Conceptual Structures – Contributions to ICCS 2000.* Shaker Verlag, Aachen, 2000, 152–161.

[3] Carpetito C., Romano G.: *Concept data analysis. Theory and Applications.* J. Wiely, 2004.

[4] Bělohlávek R., De Baets B., Outrata J., Vychodil V.: *Lindig's algorithm for concept lattices over graded attributes.* Torra V., Narukawa Y., Yoshida Y. (Eds.): Modeling Decisions for Artificial Intelligence: 4th International Conference, Lecture Notes in Artificial Intelligence 4617, 2007, pp. 156-167. [Springer-Verlag Berlin Heidelberg 2007, DOI 10.1007/978-3-540-73729-2_15, ISBN 978-3-540-73728-5].

[5] Bělohlávek R., Dvořák J., Outrata J.: Fast factorization by similarity in formal concept analysis of data with fuzzy attributes. *Journal of Computer and System Sciences* **73**(6)(2007), pp. 1012-1022. [Elsevier Science, DOI 10.1016/j.jcss.2007.03.016, ISSN 0022-0000]

[6] FCALGS: Data Formats. Popis formatu vstupnich a vystupnich dat pro FCA. http://fcalgs.sourceforge.net/format.html

[7] Popis programu jLatvis. http://www.jlatvis.com/cs/