VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vizualizace neuronových sítí
Visualization of Neural Networks

2014 Martin Rusek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Rusek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vizualizace neuronových sítí
Visualization of Neural Networks

Zásady pro vypracování:

Cílem bakalářské práce je získat přehled v oblasti grafové vizualizace, implementovat alespoň tři různé metody a pomocí nich vizualizovat neuronové sítě.

1. Nastudovat problematiku grafové vizualizace se zaměřením na typ neuronové sítě Self-Organizing Map a dále možnosti hierarchické vizualizace.
2. Dle pokynů vedoucího naimplementovat vybrané vizualizace v programovacím jazyce C# jako knihovnu pro další možné použití.
3. Otestovat funkčnost implementací.

Seznam doporučené odborné literatury:

[1] Elias Pampalk, Andreas Rauber, Dieter Merkl. Using Smoothed Data Histograms for Cluster Visualization in Self-Organizing Maps. In Proceedings of the Intl Conf on Artificial Neural Networks (ICANN 2002), , August 27.-30. 2002, Madrid, Spain.
[2] Alfred Ultsch and H. Peter Siemon. Kohonen's self-organizing feature maps for exploratory data analysis. In Proceedings of the International Neural Network Conference (INNC'90). Kluwer, 1990.
[3] Alfred Ultsch. Maps for the Visualization of high-dimensional Data Spaces. In Proceedings Workshop on Self-Organizing Maps (WSOM 2003), Kyushu, Japan, (2003).
[4] Alfred Ultsch. U*-Matrix: a Tool to visualize Clusters in high dimensional Data, Technical Report No. 36, Dept. of Mathematics and Computer Science, University of Marburg, Germany, (2003).

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Lukáš Vojáček**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2014 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Rád bych na tomto místě poděkoval vedoucímu mé bakalářské práce panu Ing. Lukáši Vojáčkovi za jeho rady, jelikož by bez něj tato práce nevznikla.

Abstrakt

Tato bakalářská práce se zabývá problematikou grafické vizualizace vysoce rozměrných dat do nízkorozměrného prostoru. Zaměřuje se na implementaci alespoň tří vizualizačních metod typu Samoorganizující Mapa. Jednotlivé metody budou implementovány jako DLL knihovna, která bude umožňovat hostitelské aplikaci přístup k daným metodám.

Klíčová slova: Samoorganizující mapy, SOM, neuronové sítě, Hit Histogram, U-Matrix, P-Matrix, U*-Matrix, Minimum Spanning Tree

Abstract

This bachelor thesis deals with the graphical visualization of high-dimensional data into a low-dimensional space. It focuses at least on the implementation of three visualization methods of Self-Organizing Map type. The various methods will be implemented as DLL library that will allow host application access to these methods.

Keywords: Self-Organising Maps, SOM, neural networks, U-Matrix, Hit Histogram, P-Matrix, U*-Matrix, Minimum Spanning Tree

Seznam použitých zkratek a symbolů

SOM	–	Self-Organising Map
HH	–	Hit Histogram
UM	–	U-Matrix
PM	–	P-Matrix
USM	–	U*-Matrix
ST	–	Spanning Tree
MST	–	Minimum Spanning Tree
PDE	–	Pareto Density Estimation
GUI	–	Grafické uživatelské rozhraní
DLL	–	Dynamic Link Library
FCPS	–	Fundamental Clustering Problems Suite

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Samoorganizující mapy (SOM) jsou v dnešní době jedny z nejpoužívanějších umělých neuronových sítí. Tato práce se tedy bude zabývat studiem a implementací několika vizualizačních SOM metod. Tato práce přímo navazuje na práci Petra Feichtingera o stejném názvu z roku 2013. Bude tedy vylepšovat a rozšiřovat původní práci, díky tomu zde nebude podrobně probírána problematika neuronových sítí, která již byla probrána v původní práci.

Práce se dělí na několik částí. V první kapitole je stručně vysvětlen účel a historie pojmu Samoorganizujících Map. Dále zde bude nastíněno, jaké hrubé rozdíly jsou mezi jednotlivými typy vizualizačních metod a jak je můžeme dělit.

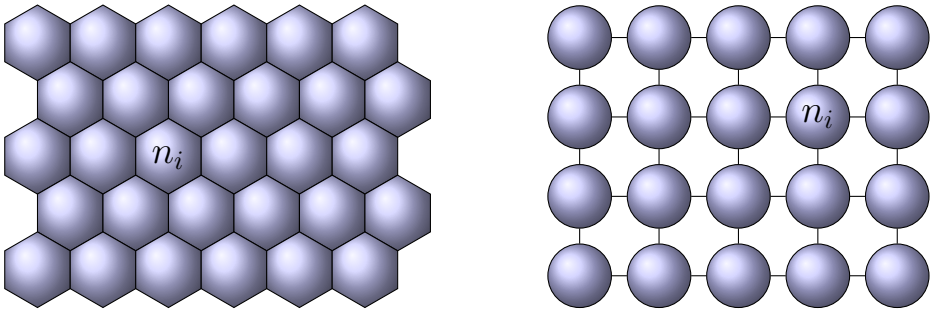
V druhé kapitole a tedy v hlavní části této práce jsou popsány a vysvětleny jednotlivé SOM metody, kterými se tato práce zabývá. Jsou to metody Hit Histogram, U-Matrix, P-Matrix, U*-Matrix a Minimum Spanning Tree (HH, UM, PM, USM, MST).

V třetí kapitole je popsán přístup k samostatné implementaci programu a jednotlivých metod. Bude v ní zmíněno, jakých změn dosáhla hostitelská aplikace, jak byly implementovány jednotlivé SOM metody a jaké implementační změny byly provedeny v metodách implementovaných Petrem Feichtingerem. Jednotlivé kapitoly jsou doplněny o grafické znázornění probírané problematiky.

2 Samoorganizující Mapy

Technika pro analýzu a vizualizaci dat Samoorganizující mapy, byla poprvé představena finským profesorem Teuvo Kohenen [5] v roce 1982, díky čemuž je také známa pod názvem Kohenenova síť. SOM je programová výpočetní metoda, nástroj pro vizualizaci vysoce rozměrných datových struktur do nízkorozměrného prostoru. Je to typ uměle vytvořené neuronové sítě, která definuje organizované mapování z vybraného datového prostoru na většinou dvourozměrnou síť zvanou mapa.

2.1 Používané formáty pro zobrazování SOM



(a) Šestiúhelníková mapa (b) Obdelníková mapa

Obrázek 1: Příklad dvou nejpoužívanějších formátů pro zobrazování SOM vizualizací

Mapa je většinou šestiúhelníková [1a] nebo obdélníková [1b]. Metody implementované v této práci využívají obdélníkové zobrazení. Mapa se skládá z neuronů (nodů, jednotek), kde každý neuron je asociován s n-rozměrným váhovým vektorem. U každého neuronu je získána výška ($Uheight, Pheight$), která je spočítaná pomocí dané SOM metody (algoritmu).

2.2 Umělá inteligence v Samoorganizujících mapách

Jak už název napovídá, SOM algoritmy jsou soběstačné, nepotřebují tedy ke svému průběhu dozor. Takové algoritmy spadají do třídy strojového učení. Strojové učení je typ umělé inteligence, která se zabývá vývojem systémů schopných učit se z dat. Například systém který se v první fázi naučí rozeznávat platný a neplatný tvar e-mailové adresy a následně je již schopen jejich samostatné kontroly.

Jelikož se vizualizační metody implementované v této práci provádí nad již naučenými SOM, tak se v této práci nebude fázi učení věnovat pozornost.

2.3 Dělení Samoorganizujících Map

Metody SOM se mohou dělit na tři skupiny. První skupina využívá k získání vah jen váhové vektory neuronů v mapě. Jako příklad můžeme uvést metodu U-Matrix.

Druhá skupina k získání výšek neuronů potřebuje jen vstupní vektory, které jsou mapovány na jednotlivé neurony v mapě dle jejich váhových vektorů. Výšky se pak získávají jen u neuronů, které byly namapovány. Ostatní výšky neuronů jsou nastaveny na nulu nebo úplně ignorovány. Tady jako příklad slouží metody HH a variace metody MST, kde se jako vrcholy používají jen namapované neurony.

V posledním případě se výšky neuronů mohou získávat z kombinace informací. Výška je získána jak z neuronů, na které byly namapovány vstupní vektory, tak i neuronů které nevyhovují žádnému vstupnímu vektoru. Příkladem může být metoda U*-Matrix, kde jsou kombinovaný informace z metod UM a PM.

3 Vizualizace

V této kapitole budou vysvětleny jednotlivé metody implementované v této bakalářské práci. Těmi jsou Hit Histogram, U-Matrix, P-Matrix, U*-Matrix a Minimum Spanning Tree (HH, UM, PM, USM, MST).

3.1 Hit Histogram

Jako první bude vysvětlena metoda Hit Histogram. Dále bude vysvětleno k čemu slouží Euklidovská metrika, a proč se využívá v SOM vizualizacích.

3.1.1 Popis metody

Metoda Hit Histogram zobrazuje mapování vstupních vektorů na nejvíce podobné neurony v mapě a vizualizuje je pomocí barevného schéma odvozeného z počtu namapovaných vstupních vektorů na jednotlivé neurony. HH je základní SOM vizualizace a v porovnání s ostatními SOM metodami je HH metoda nejjednodušší pro studium a implementaci.

3.1.2 Euklidovská vzdálenost

Euklidovská vzdálenost nebo Euklidovská metrika určuje vzdálenost mezi dvěma vektory o stejném počtu prvků v n-dimenzionálním prostoru, kde jsou definovány vektory $\vec{p}$ a $\vec{q}$:

$$\vec{p} = (p_1, p_2 ... p_n) \tag{3.1a}$$

$$\vec{q} = (q_1, q_2 ... q_n) \tag{3.1b}$$

každý neuron může mít váhový vektor o 2 až n souřadnicích. K výpočtu se používá vztah Euklidovské metriky, který je odvozený z Pythagorovy věty:

$$\mathbf{d}(\vec{p}, \vec{q}) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{3.2}$$

kde d je vzdalenost mezi body p a q.

Výpis 1: Implementace funkce pro výpočet Euklidovské vzdálenosti v jazyce C#

```
1   private static double Euklid(double[] u, double[] v)
2   {
3     double tmp = 0;
4
5     for (int i = 0; i < V; i++)
6     {
7       tmp += Math.Pow((u[i] - v[i]), 2);
8     }
9     return Math.Sqrt(tmp);
10  }
```

Jak je vidět na Implementaci Euklidovské metriky [1], náročnost nespočívá v implementaci, ale v počtu kolikrát se tento výpočet Euklidovské metriky provádí. Například v metodě MST, která bude popsána později, je počet výpočtů vzdáleností n^2, kde n je počet neuronů v mapě.

3.1.3 Postup algoritmu

1. Je vybrán vstupní vektor ze vstupního datového prostoru. Daný vektor je pomocí vztahu pro Euklidovskou vzdálenost postupně porovnán s váhovými vektory neuronů v mapě. U neuronu, který je nejblíže vybranému vstupnímu vektoru, je výškové ohodnocení zvětšeno o 1. Operace je zopakována pro všechny zbylé vstupní vektory.

2. Jsou vyhledány neurony s nejmenším a největším výškovým ohodnocením. Dle nalezeného minima a maxima je nastavena barevná paleta výsledného grafu s tím, že neurony s výškovým ohodnocením rovnému 0 jsou vždy zobrazeny v bílé barvě.

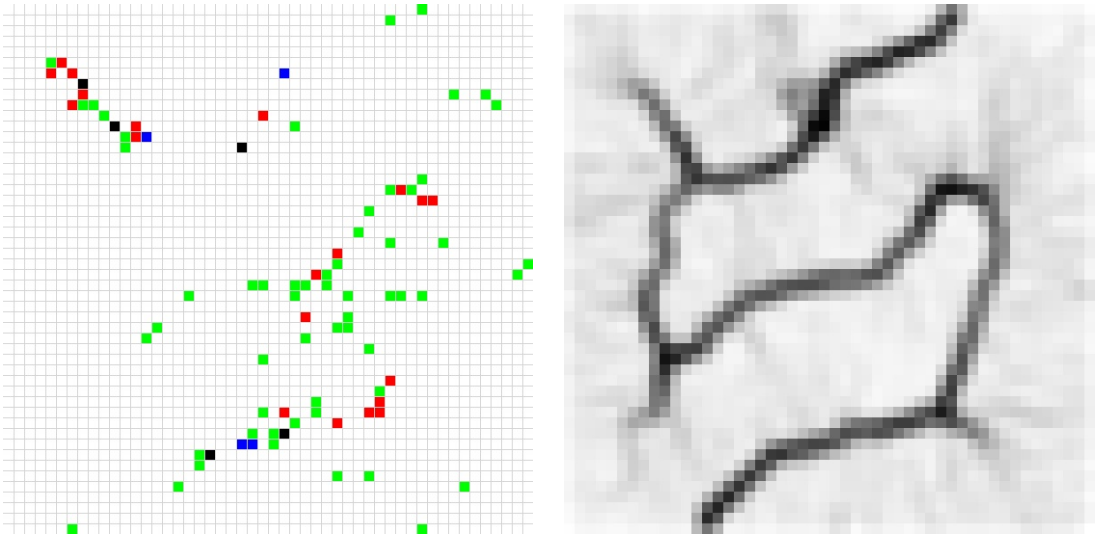
3. Vykreslení výsledného grafu na plátno.

Vybírání vstupních vektorů ze vstupního datového prostoru může být náhodné, jelikož nezáleží na pořadí, v jakém jsou vektory mapovány.

Výpis 2: Pseudokód pro získání výšek v metodě Hit Histogram

```
1  //Y, X    - rozměry mapy
2  //vv      - vstupní vektor
3  //vn      - vstupní neuron
4  //euklid  - vdálenost mezi vv a vn
5  //result  - pole pro získané výšky
6
7  foreach vstupní vektor vv begin
8    vv = načtení rozměrů vstupního vv vektoru
9    foreach řádek mapy Y begin
10     foreach sloupec mapy X begin
11       vn = načtení rozměru vstupního neuronu
12
13       euklid = vzdálenost mezi vv a vn
14       if(euklid < bestMatch) begin
15         bestY = Y
16         bestX = X
17       end
18
19     end
20   end
21   result[bestY, bestX] += 1
22 end
```

3.1.4 Výstup metody



(a) HH datového souboru Propletené kruhy (b) UM datového souboru Propletené kruhy

Obrázek 2: Metoda Hit Histogram a U-Matrix nad datovým souborem Propletené kruhy

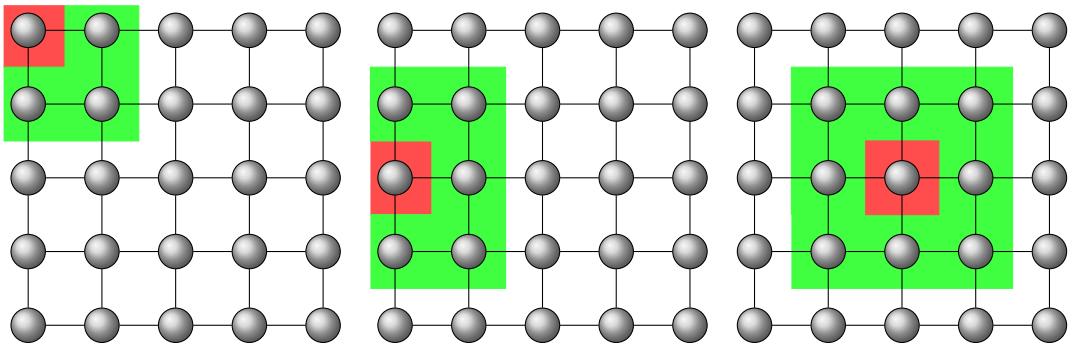
Na výstupu metody Hit Histogram [2a] je vidět výsledek mapování 150 vstupních vektorů na naučenou SOM Propletené kruhy. Na zobrazení je vidět, že výsledný graf je silně závislý na množství vstupních vektorů. Pokud je jich málo je obtížné z mapy vyčíst shluky neuronů. V porovnání s ostatními vizualizačními metodami HH neprovádí žádnou datovou analýzu. Zobrazené výšky jsou jednoduše jen počet namapovaných vstupních vektorů na jednotlivé neurony.

3.2 U-Matrix

Jako druhá bude vysvětlena metoda Unified Distance Matrix (U-Matrix), která je součástí metody U*-Matrix. Implementace samotné metody UM však není předmětem této bakalářské práce.

3.2.1 Popis metody

U-Matrix zobrazuje výšky, tedy $Uheights$ nad sítí neuronů. UM vizualizuje blízkost jednotlivých neuronů vůči ostatním dle velikosti vah daných neuronů. Výška neuronu je získána jako suma Euklidovských vzdáleností od centrálního neuronu čili neuronu, u kterého se určuje výška, k neuronům v jeho naprosté blízkosti.



(a) V rohu (b) Na okraji (c) V mapě

Obrázek 3: Příklad možných situací uspořádání neuronů při měření $Uheight$

Výška neuronu je tedy složena z tří až osmi Euklidovských vzdáleností podle toho, jestli je centrální neuron umístěný v rohu, na kraji nebo uvnitř mapy [3]. Vztah pro získání $Uheight$ je tedy definován jako:

$$\mathrm{Uheight}(n) = \sum_{m \in N(n)} \mathrm{d}(n, m) \tag{3.3}$$

kde n je neuron náležící do mapy, $Uheight(n)$ je výška daného neuronu n, $N(n)$ je množina nejbližších sousedů daného neuronu, d je Euklidovská vzdálenost mezi neurony a m je neuron náležící do množiny $N(n)$.

Výšky získané pro jednotlivé neurony jsou v podstatě aproximace vzdáleností k ostatním datům v mapě. Zobrazení je vykresleno většinou v odstínech šedé barvy. Podobné neurony jsou zobrazeny ve světlé barvě, čím tmavší odstín kterým je neuron v grafu vykreslen, tím se méně podobá sousedním neuronům, respektive je od nich ve velké vzdálenosti. Díky tomu lze v grafu dobře rozpoznat shluky neuronů a jejich oddělení od ostatních shluků v mapě.

3.2.2 Postup algoritmu

1. Je vybrán neuron v mapě. U vybraného neuronu je získána výška $Uheight$ sečtením Euklidovských vzdáleností centrálního neuronu k neuronům v jeho naprosté blízkosti. Operace je zopakována pro zbylé neurony.

2. Je vyhledána minimální a maximální $Uheight$ neuronů v mapě. Podle nalezeného minima a maxima je připravena výsledná paleta barev v odstínech šedé, pro vykreslení výsledného grafu.

3.3 P-Matrix

V této podkapitole bude vysvětlena metoda P-Matrix a její vztah s metodami U-Matrix a U*-Matrix.

3.3.1 Popis metody

Metoda P-Matrix je definována analogicky k metodě U-Matrix. Rozdílem mezi PM od UM je ten, že PM používá jako výšku $Pheight$, což je datová hustota namapovaných vstupních vektorů do okolí vybraného neuronu v mapě, u kterého se zjišťuje hustota (density). Výška neuronu v PM metodě je tedy definována jako:

$$Pheight = p(v(n), X) \tag{3.4}$$

kde $p(w, X)$ je empirický odhad hustoty na bodu w v datovém prostoru X, n je neuron v mapě, u kterého se získává $Pheight$.

Metoda PM zobrazuje pro každý neuron v mapě hustotu namapovaných vstupních vektorů v jeho okolí, přičemž existuje mnoho různých algoritmů na odhad hustoty. Jedním z nich je například Pareto Density Estimation (PDE). PDE udává velikost poloměru měřené plochy pomocí tzv. Pareto poloměru. Bylo zjištěno, že aby PDE byl efektivní, velikost oblasti dané Pareto poloměrem, musí být natolik veliká, aby zahrnovala alespoň 20% namapovaných vstupních vektorů v mapě.

Při implementaci metody P-Matrix bylo zvoleno implementovat poloměr měřené plochy jako volitelný parametr. To umožňuje znázornit závislost PM vizualizace na velikosti měřené plochy volným škálováním poloměru. Velikost plochy pro měření $Pheight$ vybraného neuronu je daná rovnicí kružnice:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{3.5}$$

kde x a y jsou souřadnice neuronu, u kterého se zjišťuje, jestli je uvnitř kontrolované kruhové oblasti. Proměnné x_0 a y_0 jsou souřadnice neuronu, který určuje střed kruhové oblasti, r je poloměr předávaný metodě jako parametr.

3.3.2 Postup algoritmu

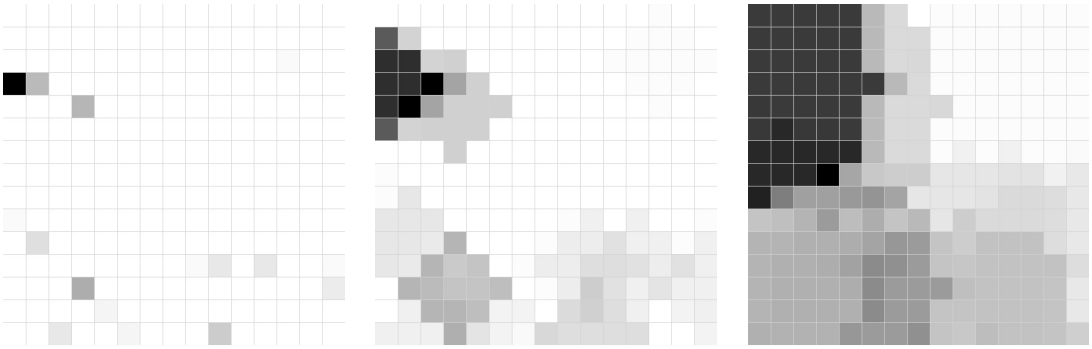
1. Je vybrán vstupní vektor ze vstupního datového prostoru. Pomocí vztahu pro Euklidovskou vzdálenost je daný vektor postupně porovnáván s váhovými vektory neuronů v mapě. U neuronu, který je nejbližší vybranému vstupnímu vektoru, je výškové ohodnocení zvětšeno o 1. Operace je provedena pro všechny zbylé vstupní vektory.

2. Následně je vybrán libovolný neuron v mapě. V kruhové oblasti kolem vybraného neuronu, kde velikost oblasti je daná poloměrem, je sečten počet namapovaných

vstupních vektorů v dané oblasti. Výškové ohodnocení neuronu $Pheight$ je nastaveno na získanou hodnotu. Operace je provedena pro všechny ostatní neurony v mapě.

3. Dle největšího a nejmenšího $Pheight$ je připravena paleta barev v odstínech šedé, ve které je vykreslen výsledný graf.

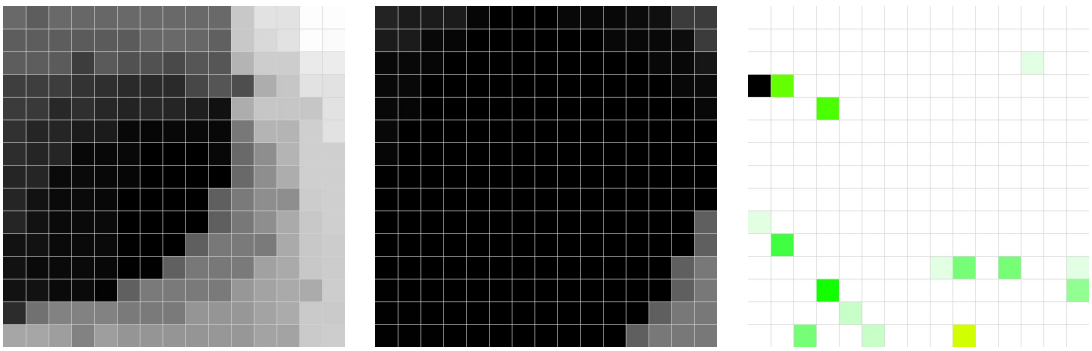
3.3.3 Výstup metody



(a) r = 0 (b) r = 5 (c) r = 10

Obrázek 4: Metoda PM nad datovým souborem Barvy

Výsledek metody P-Matrix provedené nad naučenou SOM Barvy je zobrazený na obrazcích [4] a [5]. Pro každý neuron v mapě zobrazuje odhad datové hustoty při určitém poloměru. Poloměr oblastí ve kterých je měřena $Pheight$ byl zvětšován na hodnoty $r = 0, 5, 10, 13, 15$. Je vidět rostoucí vliv neuronů, na které jsou namapované vstupní vektory.



(a) r = 13 (b) r = 15 (c) HH datového souboru barvy

Obrázek 5: Metody PM a HH nad datovým souborem Barvy

Metoda PM vychází ze základů metody Hit Histogram. Jasný příklad toho je vidět na obrazcích [4a] a [5c]. Je-li u metody PM nastaven poloměr oblasti, ve které je měřen odhad datové hustoty $Pheight$, tak výsledek metody je totožný s výstupem metody HH.

3.4 U*-Matrix

Jako předposlední bude popsána metoda U*-Matrix. Bude vysvětleno, z jakého důvodu byla tato vizualizační metoda představena a jaké výhody má oproti ostatním SOM vizualizacím. Také bude popsáno, jaké role v této vizualizaci hrají metody UM a PM.

3.4.1 Popis metody

Hlavní důvod pro definování USM vizualizace byl ten, že naměřené lokální vzdálenosti v U-Matrix většinou pocházejí zevnitř shluku neuronů. Tyto vzdálenosti jsou většinou zanedbatelné v hustě osídleném datovém prostoru. Naopak v datovém prostoru s malou hustotou jsou tyto vzdálenosti důležité. USM kombinuje vzdálenostně orientovanou metodu U-Matrix a P-Matrix, která jako výšky používá poměr velikosti měřené plochy s hustotou jejího mapování.

Vizualizační metoda U*-Matrix je odvozena od UM a PM vizualizací dle následujících pravidel:

- Pokud se datová hustota kolem neuronu rovná průměrné datové hustotě, tak výšky zobrazované v USM by měli být stejné jako v korespondující UM vizualizaci [2].
- Velká datová hustota kolem neuronu, tedy hustota větší než průměrná *Pheight* znamená, že naměřené vzdálenosti jsou převážně v oblasti datových shluků. V tomto případě má být výška $U^*height$ v USM metodě nízká [2].
- Pokud datová hustota kolem neuronu je menší než průměrná, tak lokální naměřené vzdálenosti pocházejí většinou z okrajové oblasti datového shluku. V takové situaci by výška zobrazovaná v USM metodě měla být vyšší než v korespondující UM vizualizaci [2].

Z těchto pravidel byl odvozen vztah, podle kterého se počítá velikost dané $U^*height$. Napřed se z vlastností P-Matrix spočítá tzv. $scaleFactor$. Následně je spočítána samotná $U^*height$, kde je $scaleFactor$ používán pro škálování počítané výšky.

Vztahy pro $mean(P)$, $scaleFactor$ a $U^*height$ jsou tedy definovány jako:

$$\text{mean}(P) = \frac{1}{n}\sum_{x=1}^{n} Pheight_i \tag{3.6a}$$

$$scaleFactor(n) = \frac{Pheight(n) - mean(P)}{mean(P) - max(P)} + 1 \tag{3.6b}$$

$$U^*height(n) = scaleFactor(n) * Uheight(n) \tag{3.6c}$$

kde $Pheight(n)$ je hustota oblasti daného neuronu (P-Matrix), $mean(P)$ je průměr všech hustot, $max(P)$ je maximální hustota, $Pheight_i$ je výška neuronu v PM metodě náležící do množiny všech PM vah.

Z předchozích pravidel a definic lze odvodit:

$$Pheight(n) = mean(Pheights) \Rightarrow U^*height(n) = Uheight(n) \tag{3.7a}$$

$$Pheight(n) < mean(Pheights) \Rightarrow U^*height(n) > Uheight(n) \tag{3.7b}$$

$$Pheight(n) > mean(Pheights) \Rightarrow U^*height(n) < Uheight(n) \tag{3.7c}$$

$$Pheight(n) = max(Pheights) \Rightarrow U^*height(n) = 0 \tag{3.7d}$$

3.4.2 Postup algoritmu

1. Je vybrán vstupní vektor ze vstupního datového prostoru. Daný vektor je postupně porovnán s váhovými vektory neuronů v mapě pomocí vztahu pro Euklidovskou vzdálenost. U neuronu, který je nejpodobnější vybranému vstupnímu vektoru, je výškové ohodnocení zvětšeno o 1. Operace je zopakována pro všechny zbylé vstupní vektory.

2. Posléze je vybrán libovolný neuron v mapě. U vybraného neuronu je získána $Uheight$ sečtením Euklidovských vzdáleností centrálního neuronu k neuronům v jeho naprosté blízkosti. Operace je provedena pro zbylé neurony.

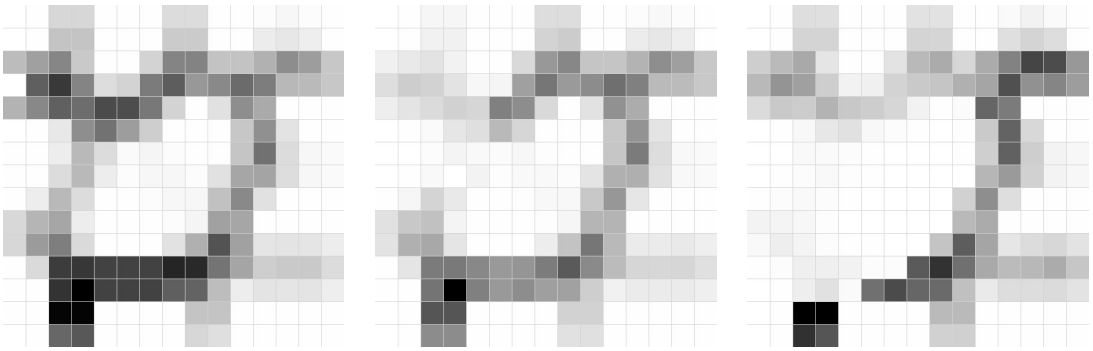
3. Je vybrán neuron v mapě. V kruhové oblasti kolem vybraného neuronu, kde velikost oblasti je daná poloměrem, je sečten počet namapovaných vstupních vektorů. Výškové ohodnocení neuronu, tedy $Pheight$ je nastaveno na získanou hodnotu. Operace je provedena pro všechny ostatní neurony v mapě.

4. Je vybrán neuron z mapy, pro který je spočítán $scaleFactor$ a následně $U^*height$. Operace je zopakována pro všechny zbylé neurony.

5. Dle největšího a nejmenšího $U^*height$ je připravena výsledná paleta barev v odstínech šedé, ve které je vykreslen výsledný graf.

Druhý a třetí krok jsou na sobě nezávislé a je tedy možné zaměnit jejich pořadí.

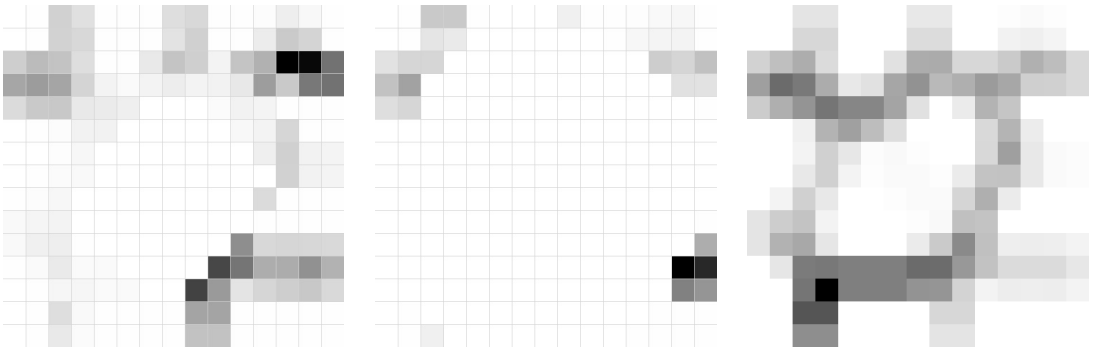
3.4.3 Výstup metody



(a) r = 0 (b) r = 5 (c) r = 10

Obrázek 6: Metoda USM nad datovým souborem Barvy

Na obrázcích [6] a [7] jsou zobrazeny výstupy metody USM. Poloměr oblasti při výpočtu $Pheight$ byl nastaven na hodnoty $r = 0, 5, 10, 13, 15$. Je vidět zmíněný efekt korekce výsledných $U^*heights$ pomocí parametru $scaleFactor$, který zajišťuje, aby jednotlivé výšky $U^*height$ byly v mezích stanovených vnitřními metodami PM a UM.



(a) r = 13 (b) r = 15 (c) UM datového souboru barvy

Obrázek 7: Metody USM a UM nad datovým souborem Barvy

Z pravidel (3.7) odvozených ze vztahu na získání hodnoty $scaleFaktor$ je vidět, že pokud je $Pheight$ menší než průměrné $Pheight$, tak je výsledná výška $U^*height$ zvětšena. Naopak pokud je $Pheight$ větší než průměrné $Pheight$, tak je $U^*height$ zmenšena. Malé procento $U^*height$ výšek neuronů je nezměněno v případě, že se $Pheight$ rovná $mean(P)$. Nakonec velmi malý počet $U^*height$ výšek je nastaven na 0, pokud je $Pheight$ maximální.

3.5 Minimum Spanning Tree

Tato kapitola se zabývá metodou Minimum Spanning Tree. Bude zde vysvětleno, z jakých důvodů byla tato metoda definovaná a jaké výhody a nevýhody má proti ostatním metodám. Bude zde vysvětlen Kruskalův algoritmus pro získání MST a jak se liší od aktuální implementace metody MST.

3.5.1 Popis metody

Spanning Tree je v grafové teorii podgraf grafu ve formě stromu, který spojuje všechny vrcholy grafu s tím, že v něm neexistují žádné cykly. Graf může mít velké množství různých ST. Úloha MST spočívá v nalezení ST s nejmenším možným ohodnocením. Je-li toto převedeno do Samoorganizujících map, tak se jako vrcholy grafu uvažují jednotlivé neurony a jako hrany mezi dvojicemi neuronů jsou uvažovány Euklidovské vzdálenosti. Existuje několik algoritmů pro získání MST. Z důvodu výkonu byla pro implementaci zvolena variace Kruskalova algoritmu.

3.5.2 Kruskalův algoritmus

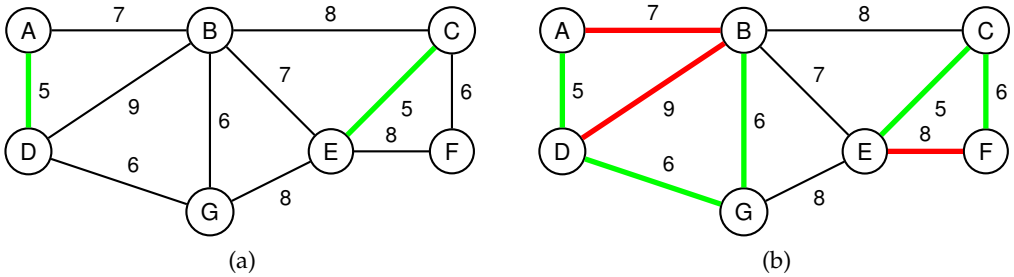
Kruskalův algoritmus patří do třídy tzv. hladových. Hladové algoritmy se uplatňují v situaci, kdy je potřeba vybrat z množiny objektů jejich podmnožinu, která splňuje určitou, předem danou podmínku a zároveň má minimální nebo maximální ohodnocení.

Mějme množinu N, kde každý objekt je ohodnocen kladným reálným číslem w. Ohodnocení je pak definováno jako:

$$w(A) = \sum_{n \in N(n)} w(n) \tag{3.8}$$

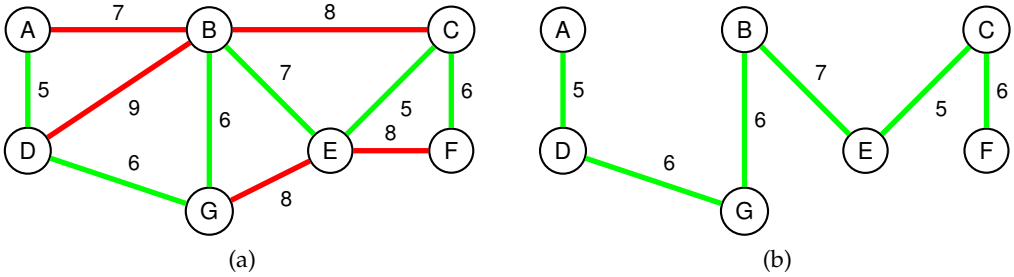
kde $w(A)$ je výsledná podmnožina, n je objekt z množiny N a $w(n)$ je reálné číslo z množiny N.

Postup Kruskalova algoritmu spočívá v tom, že jsou vybírány jednotlivé hrany s ohodnocením od nejmenšího po největší a přitom se provádí kontrola, jestli daná hrana nevytvoří s již vybranými hranami cyklus.



Obrázek 8: Postup při hledání MST pomocí Kruskalova algoritmu

Na obrázku [8a] je znázornění postupu. Začíná se od nejlevnějších hran, tedy byla vybrána hrana AD a CE s ohodnocením 5. Po vybrané hraně, je nutné zkontrolovat následující vybranou hranu, aby nevytvořila cyklus. Další krok je vybrání hran s ohodnocením 6 [8b], tedy hranu DG a BG. V tomto momentě je nutné odstranit hrany AB a BD, jelikož by vytvořili cykly ABD a BDG. Další vybraná hrana je CF, stále s ohodnocením 6 a opět je odstraněna hrana EF kvůli zamezení vzniku cyklu.



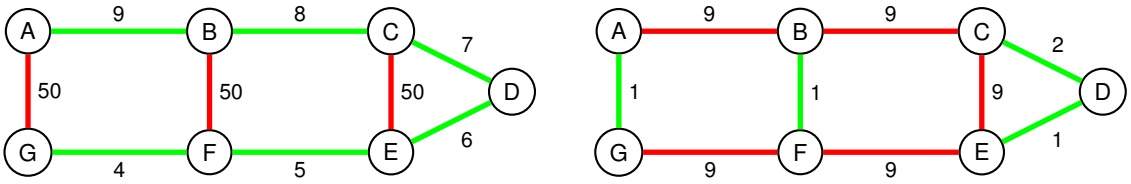
Obrázek 9: Postup při hledání MST pomocí Kruskalova algoritmu

Následující hrana s nejmenším ohodnocením je hrana BE s ohodnocením 7 [9a]. Hrana AB byla odstraněna kvůli hrozbě vzniku cyklu. Hrany BC a EG jsou také odstraněny. Na obrázku [9b] je vidět výsledný MST ADGBECF.

3.5.3 Popis algoritmu

Na rozdíl od standardní grafové situace, kde jsou k dispozici hrany a kořeny, neuronová síť obsahuje pouze neurony s váhovými vektory. Je tedy potřeba získat všechny hrany, s kterými bude metoda pracovat. Pro každý neuron se získá seznam Euklidovských vzdáleností ke všem ostatním neuronům v mapě. Pokud máme mapu o rozměrech $n * y$, tak počet hran s kterými bude MST metoda pracovat je $(n * y) * (n * y - 1)$. Z toho se využije jen $n * y - 1$ hran k sestavení aktuálního MST. Všechny hrany se setřídí vzestupně a samostatně pro každý neuron. Pak se ke každému vrcholu (neuronu) připojí hrana s nejmenším ohodnocením. Daná hrana se smaže. Pokud se dva samostatné neurony spojí hranou, vytvoří se skupina. Pokud nejmenší hrana samostatného neuronu vede

k neuronu, který je již ve skupině, daný neuron se k ní připojí. Pokud se hranou spojí dvě skupiny, obě skupiny se spojí v jednu. Tímto se rozdělí neurony do skupin o malém počtu vrcholů.



(a) nejlepší (b) nejhorší

Obrázek 10: Příklad nejlepšího a nejhoršího případu jednoho průchodu metodou MST

V nejlepším případě [10a] lze již v prvním průchodu získat samotný MST, tedy použije se $x * y - 1$ hran. Na obrázku je vidět, jak se k vrcholu a řetězovitě připojují další vrcholy díky neustále se zmenšujícímu ohodnocení hran připojených k jednotlivým vrcholům. V nejhorším případě [10b] se použije $\lceil (n * y)/2 \rceil$ hran. V následujících průchodech se jednotlivé skupiny neuronů berou opět jako samostatné vrcholy grafu s tím, že hrana s nejmenším ohodnocením se vybere ze všech neuronů tvořících skupinu. V každém průchodu se skupiny zvětšují a jejich počet se naopak zmenšuje. Nakonec zbydou jen dvě, které se spojí do výsledného MST.

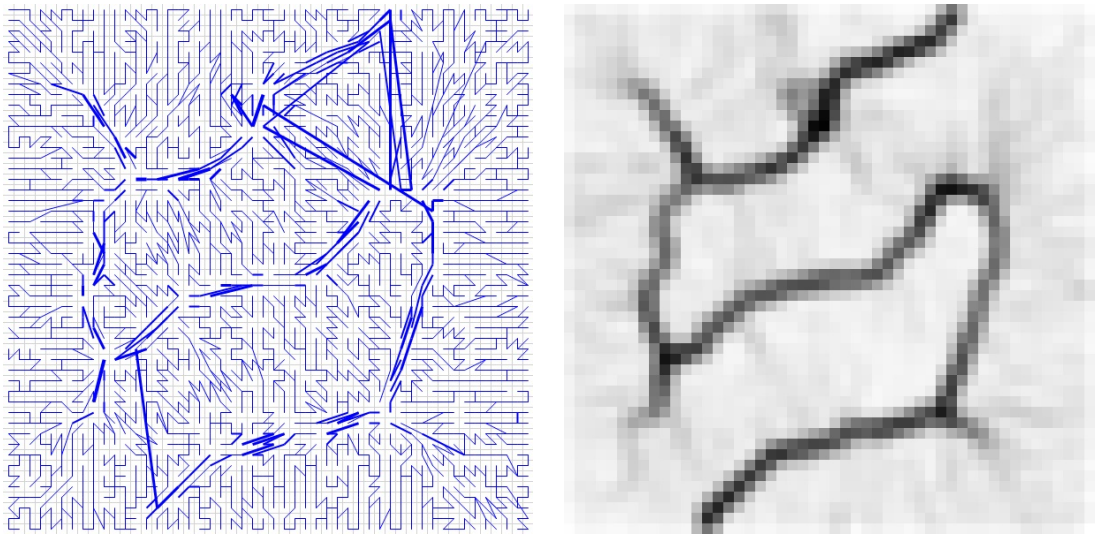
3.5.4 Postup algoritmu

1. Je vybrán neuron v mapě. U vybraného neuronu je získán seznam Euklidovských metrik ke všem ostatním neuronům v mapě. Operace je zopakována pro všechny ostatní neurony. Všechny získané seznamy jsou seřazeny vzestupně.
2. (a) Je vybrán neuron, ke kterému je připojen neuron pomocí hrany s nejmenší cenou. Operace je zopakována pro zbylé neurony.
   (b) Jsou smazány všechny hrany, které by v grafu vytvořili cyklus.
   (c) Je zjištěno kolik hran je ještě třeba na sestrojení zbytku MST. Všechny nadbytečné hrany jsou smazány.
3. (a) Je vybrána skupina neuronů, ke které je připojena další skupina neuronů pomocí hrany s nejmenší cenou. Operace je zopakována pro zbylé skupiny.
   (b) Jsou smazány všechny hrany, které by v grafu vytvořily cyklus.
   (c) Je zjištěno kolik hran je ještě třeba na spojení všech skupin neuronů. Všechny nadbytečné hrany jsou smazány.
   (d) Tento krok je opakován tak dlouho, až zbyde jen jedna skupina neuronů.
4. Je vyhledána nejmenší a největší hrana spojující dva neurony ve výsledném MST. Dle nalezeného minima a maxima jsou hrany tvořící MST rozděleny do skupin podle cen jednotlivých hran.

5. Je vykreslen výsledný graf s tím, že hrany v jednotlivých skupinách, které byly určeny v předchozím kroku, jsou vykresleny v různých tloušťkách. Hrany v první skupině, tedy hrany s nejmenší cenou jsou vykresleny tence a hrany s vyšší cenou jsou pak vykreslovány s narůstající tloušťkou.

Implementace metody MST se dá rozdělit na tři části: v první části se získají hrany, s kterými bude metoda pracovat. Druhá a třetí část jsou skoro stejné. Rozdíl mezi nimi je v tom, že druhá část se provede jen jednou, jelikož se v ní vytváří skupiny neuronů. Naproti tomu ve třetí části se již existující skupiny jen spojují do větších celků, a to tak dlouho dokud nezbyde jen jedna skupina respektive jeden MST.

3.5.5 Výstup metody

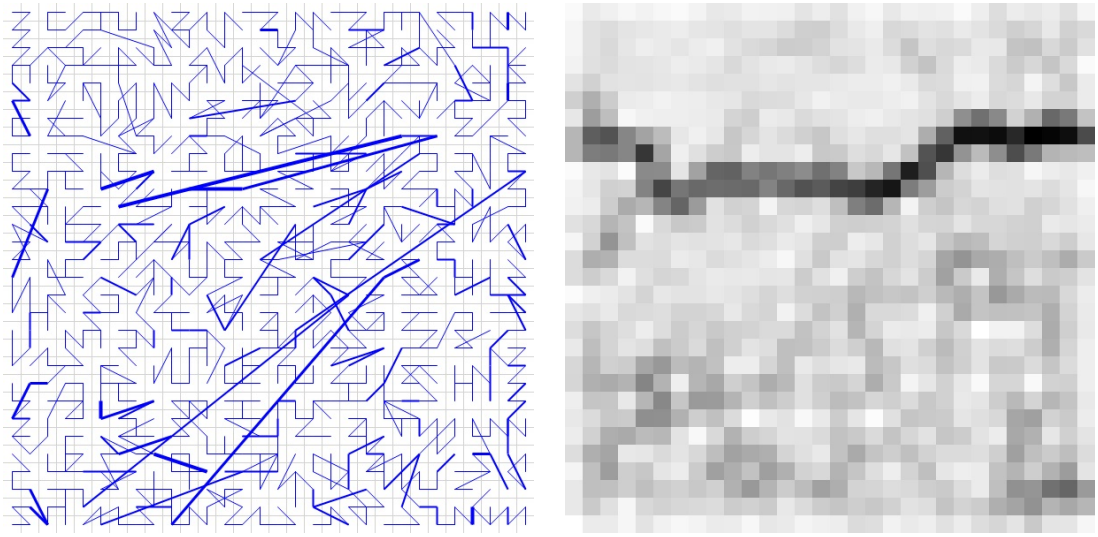


(a) MST datového souboru Propletené kruhy

(b) UM datového souboru Propletené kruhy

Obrázek 11: Metoda MST a UM nad datovým souborem Propletené kruhy

Na obrázku [11a] lze vidět výstup metody MST nad datovým souborem propletené kruhy. Na první pohled jde rozpoznat shluky podobných neuronů, které jsou spojeny tence vykreslenými hranami. Na druhou stranu hrany, které jsou vykresleny tlustě, značí velké rozdíly v datech respektive dané neurony, jsou od sebe ve velké vzdálenosti. Pokud je tato metoda porovnaná s metodou UM, kterou je možno vidět na obrázku [11b], tak výsledný vzor ve tvaru S je skoro totožný. Jak už bylo zmíněno, je to díky shlukům neuronů, které jsou navzájem velmi rozdílné.



(a) MST datového souboru Kosatec (b) UM datového souboru Kosatec

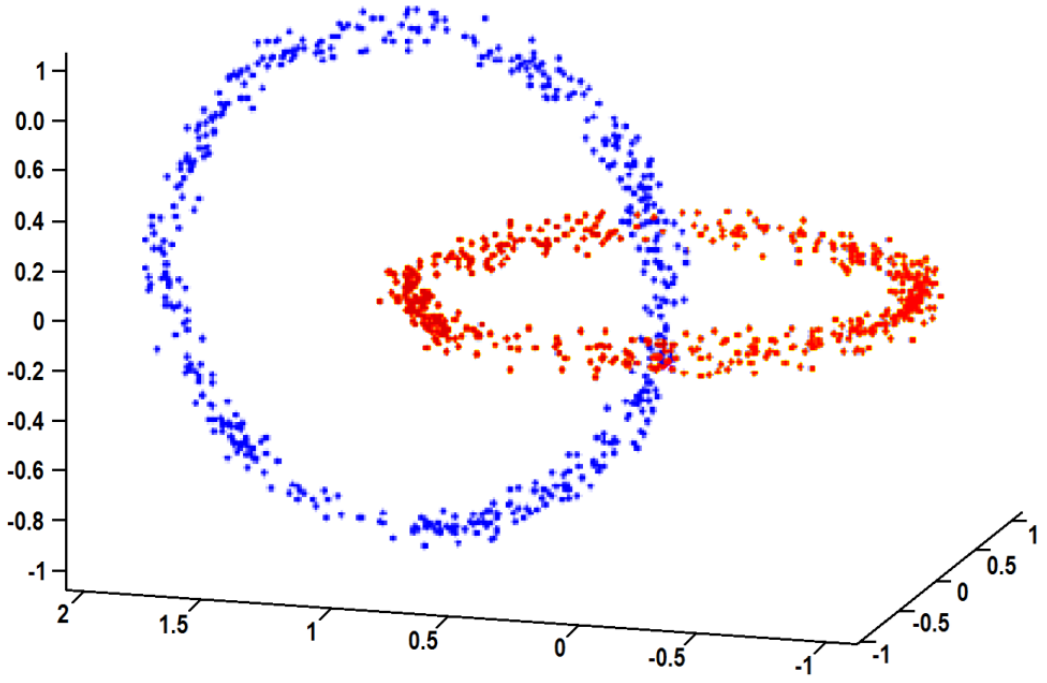
Obrázek 12: Metoda MST a UM nad datovým souborem Kosatec

Pokud ale jako vstupní datový soubor je použita jiná naučená SOM respektive datový soubor Kosatec, kde jednotlivé výšky jsou si v průměru daleko více podobné, tak je mnohem těžší v mapě metody MST rozeznat jednotlivé datové shluky. Výsledek metod je možno vidět na obrazcích [12a] a [12b]. Z těchto výsledků je patrné, že metoda MST je vhodná pro analýzu dat, kde jsou velké rozdíly mezi jednotlivými datovými shluky.

4 Data pro testování implementace

V této kapitole budou popsány jednotlivé datové soubory, které byly použity pro testování implementace vizualizačních metod.

4.1 Propletené kruhy



Obrázek 13: Datový soubor Propletené kruhy

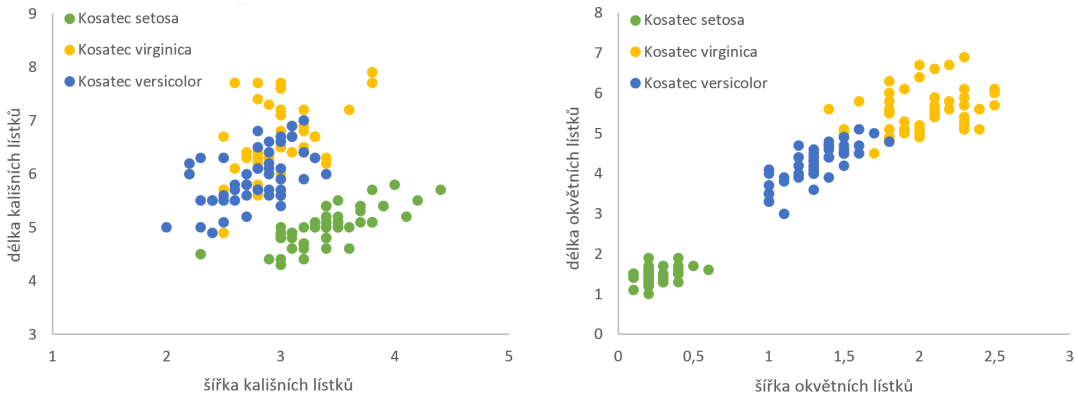
Propletené kruhy (Chain link, Interwined Rings)[13] je jeden ze známých datových souborů. Patří do skupiny FCPS (Fundamental Clustering Problems Suite) problémů. Jedná se o dva kruhy, každý z nich je uložený ve dvourozměrném prostoru. Tyto kruhy jsou navzájem propletené v trojrozměrném prostoru. Když jsou tyto data promítnuty do dvourozměrného výstupního prostoru, tak se kruhy tvořící data musí tzv. zlomit. Rozměry neuronové sítě jsou $30 * 30$, síť tedy obsahuje 900 neuronů.

4.2 Kosatec

(a) Kosatec setosa [13] (b) Kosatec virginica [14] (c) Kosatec versicolor [11]

Obrázek 14: Tři porovnávané druhy kosatců

Datový soubor Kosatec byl poprvé představen v roce 1936 Ronaldem Fisherem jako příklad diskriminační analýzy. Jedná se o jeden z nejznámějších datových souborů. Jsou v něm porovnávány tři druhy kosatců. Jmenovitě to jsou *Kosatec setosa* [14a], *Kosatec virginica* [14b] a *Kosatec versicolor* [14c]. U zmíněných kosatců byla změřena délka a šířka kališních a okvětních lístků v centimetrech. Váhové vektory neuronů v SOM mapě mají tedy čtyři rozměry. Použitý datový soubor obsahuje 900 neuronů. Rozměry mapy jsou tedy $30 * 30$.

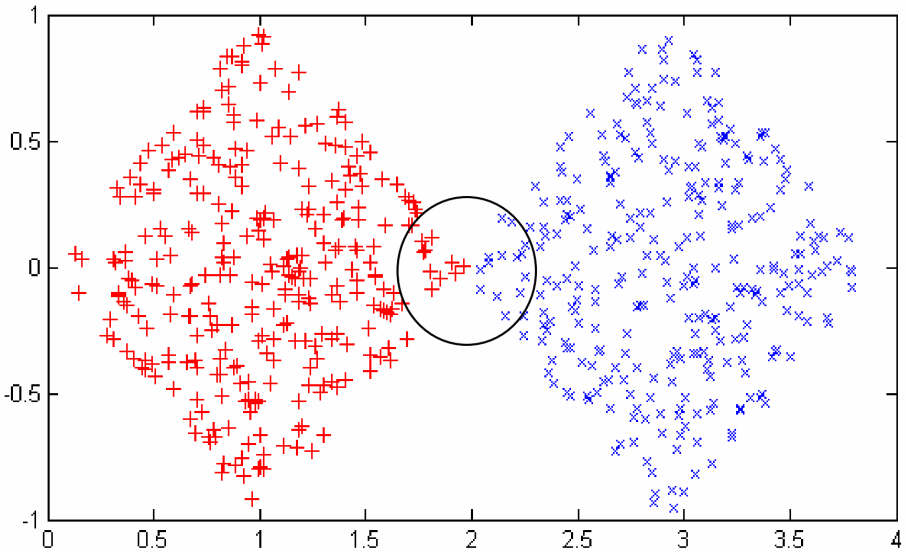


(a) Kališní lístky (b) Okvětní lístky

Obrázek 15: Znázornění datového souboru Kosatec grafem

Na obrázku [15] lze vidět grafy, které znázorňují rozdíly porovnávaných kosatců podle šířky a délky kališních a okvětních lístků. Je zřetelně vidět, že jednotlivé druhy jsou si navzájem daleko více podobné v rozměrech kališních lístků, než okvětních lístků, kde jsou patrné daleko větší rozdíly.

4.3 Dva diamanty



Obrázek 16: Ukázka datového souboru Dva diamanty

Dva diamanty je datových soubor, který pochází z FCPS. Jedná se uměle generovaná data vytvarovaná do tvaru dvou diamantů. Oba diamanty se navzájem stýkají pomocí svých rohů. Ukázku tohoto souboru je možné vidět na obrázku [16]. Rozměry mapy jsou $30 * 30$ obsahuje tedy 900 dvourozměrných neuronů.

4.4 Barvy

Barva	Červená	Zelená	Modrá	Barva	Červená	Zelená	Modrá
1	0.25	0	0	7	0.75	0	0
2	0	0.25	0	8	0	0.75	0
3	0	0	0.25	9	0	0	0.75
4	0.5	0	0	10	1	0	0
5	0	0.5	0	11	0	1	0
6	0	0	0.5	12	0	0	1

Tabulka 1: Příklad barev obsažených v datovém souboru Barvy

Datový soubor Barvy obsahuje sadu dvanácti barev, které jsou od sebe striktně oddělené. Příklad barev v datovém souboru je vidět v tabulce [1]. Barvy jsou uložené ve formátu RGB (červená, zelená, modrá). Tento datový soubor má velikost $15 * 15$ a obsahuje tedy 225 neuronů.

5 Implementace

V této kapitole bude vysvětleno, jakým způsobem jsem přistupoval k implementaci programu a jaké návrhové a implementační změny jsem provedl v programu implementovaným kolegou Petrem Feichtingerem.

5.1 Programovací jazyk

Jako programovací jazyk jsem použil C# z platformy .NET. Jeho použití je uvedeno v zadání bakalářské práce. Navíc původní verze programu je již implementovaná v jazyce C#. Aplikaci jsem implementoval v .NET Frameworku 4.5 Client oproti .NET Frameworku 4.0 Client, v kterém byl implementován původní program. Nutnost přechodu na novější verzi .NET Frameworku je z důvodu toho, že jsem při implementaci použil $Tasks$ [8].

Změnou verze použitého .NET Frameworku došlo ke změně softwarových a hardwarových požadavků na systém.

- Podporované operační systémy pro .NET Framework 4.5:
  - Windows Vista SP2 (32-bit a 64-bit)
  - Windows 7 SP1 (32-bit a 64-bit)
  - Windows 8 (32-bit a 64-bit)
  - Windows 8.1 (32-bit a 64-bit)
  - Windows Server 2008 R2 SP1 (64-bit)
  - Windows Server 2008 SP2 (32-bit a 64-bit)
  - Windows Server 2012 (64-bit)
  - Windows Server 2012 R2 (64-bit)
- Požadavky na hardware pro .NET Framework 4.5:
  - Procesor s frekvencí 1 GHz nebo vyšší
  - 512 MB paměti RAM
  - 850 MB volného místa na pevném disku (32-bit)
  - 2 GB volného místa na pevném disku (64-bit)
- Operační systémy které nadále nejsou podporované:
  - Windows XP

5.2 Struktura aplikace

Obrázek 17: Struktura vazeb aplikace

Struktura aplikace [17] zůstává v podstatě stejná. Tedy na hostitelskou aplikaci jsou připojené DLL knihovny s implementovanými vizualizačními metodami. Aplikace má přístup k jednotlivým metodám a může je libovolně volat.

5.3 Implementační změny programu

V původní verzi aplikace byly jednotlivé metody obsluhovány stejným vláknem jako grafické uživatelské rozhraní. Díky tomu bylo GUI neresponzivní po celou dobu výpočtu jednotlivých vizualizačních metod. Z tohoto důvodu jsem jednotlivé metody implementované Petrem Feichtingerem a mé vlastní upravil, respektive implementoval jako statické třídy. Po této úpravě jednotlivé metody volám pomocí tzv. *Tasks* [8]. Na rozdíl od spouštění metod v konkrétním vláknu, je *Task* vyšší úroveň abstrakce. Při použití *Tasku* uživatel jen oznamuje, že daný kód se má provést separátně. O konkrétní vlákno se již nestará.

Výpis 3: Implementace volání vizualizačních metod pomocí *Task*

```
1   private async Task minSpanningTreeTaskAsync()
2   {
3     // nastevení programu před zavoláním vizualizační metody
4     beforeFceSet();
5     statusProgres.Maximum = pX * pY * 2;
6
7     // příprava proměnné pro získávání progresu z vlákna
8     var progress = new Progress<int>(i=> statusProgres.Value = i);
9
10    // spuštění metody v tasku a získaní výsledku metody
11    im = await Task.Run(() => MinimumSpanningTreeStatic.
       MinimumSpannigTreeMain(progress, myMap, pX, pY, pV));
12
13    // vykreslení výsledku na plátno
14    PicImage = im;
15
16    // nastavení programu po dokončení metody
17    allParamHid();
18    lastSet();
19  }
```

Ve výpisu zdrojového kódu [3] je vidět příklad implementace volání metody pomocí *Tasku*. Volání vizualizační metody se skládá z několika kroků:

1. Aktivací tlačítka metody v uživatelském grafickém rozhraní v záložce vizualizace je poslán požadavek na vykonání vybrané vizualizační metody.

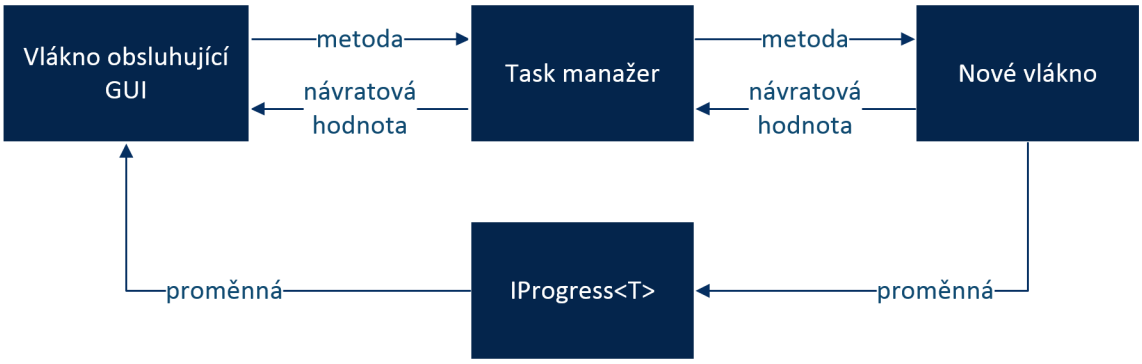
2. Je provedena kontrola, jestli výpočet jakékoliv vizualizační metody již neprobíhá. Pokud probíhá je požadavek ignorován. Tento krok je nutný, aby se zabránilo spuštění více vizualizačních metod najednou.

3. Nastavení proměnných aplikace které jsou nutné k úspěšnému vykonání volané metody.

4. Spuštění metody pomocí *Tasku* a vykreslení výsledku metody na plátno.

5. Nastavení proměnných aplikace po úspěšném vykonání metody a příprava prostředků k obsluze metody.

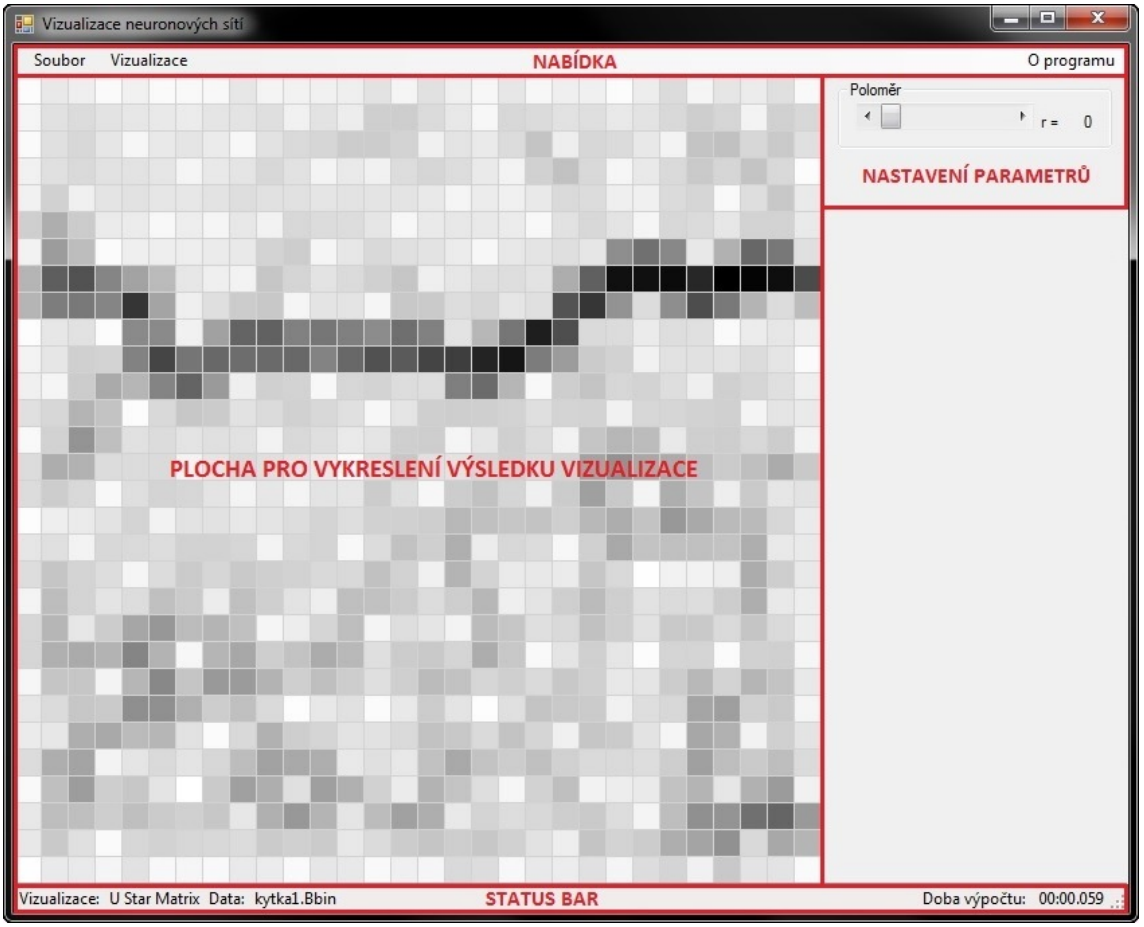
Vlákno na kterém běží vizualizační metoda předává informace o svém průběhu hlavnímu vláknu, na kterém běží GUI pomocí rozhraní *IProgress<T>*.



Obrázek 18: Znázornění vazeb vláken a jejich komunikace

Na obrázku [18] jsou znázorněny vztahy mezi objekty, které se starají o vlákna nebo s nimi spolupracují. Hlavní vlákno předá metodu s parametry Task manažeru. Mezi nimi je i parametr typu *Progress<T>*, který poskytuje rozhraní *IProgress<T>*. Task manažer určí, v jakém vlákně bude metoda spuštěna. V průběhu metody poskytuje rozhraní *IProgress<T>* spojení s hlavním vláknem, přes které jsou posílány zprávy o průběhu metody. Po dokončení metody je výsledek vrácen hlavnímu vláknu.

5.4 Změny v grafickém uživatelském rozhraní



Obrázek 19: Náhled grafického uživatelského rozhraní

Do grafického uživatelského rozhraní [19] jsem přidal několik prvků podávajících informace o stavu aplikace.

- Prvky přidané do aplikace:
  - *label* se jménem právě aktivní vizualizační metody.
  - *label* pro měření doby výpočtu jednotlivých vizualizačních metod.
  - *label* s názvem datového souboru se kterým se právě pracuje.
  - *progress bar* poskytující grafickou indikaci o právě probíhajícím výpočtu vizualizační metody. Zobrazený je jen při samotném výpočtu.

  Všechny tyto objekty jsou umístěny v tzv. *status baru* na spodním okraji aplikace.

5.5 Ukázky implementace

Výpis 4: Implementace interní třídy pro práci s daty v metodě Hit Histogram

```
internal class HitData : IComparable
{
    public double l { get; set; }
    public int x { get; set; }
    public int y { get; set; }

    public HitData(double l, int x, int y)
    {
        this.l = l;
        this.x = x;
        this.y = y;
    }

    public int CompareTo(object obj)
    {
        HitData d = (HitData)obj;
        if (this.l < d.l) return -1;
        else if (this.l > d.l) return 1;
        else return 0;
    }
}
```

Ve všech mnou implementovaných metodách využívám k uchování a manipulaci dat vlastní *interní* třídy. Příklad takové třídy je možné vidět ve výpisu kódu [4]. Konkrétně se jedná o třídu metody Hit Histogram. Daná třída se skládá z několika proměnných, které uchovávají její vlastnosti. Dále obsahuje konstruktor pro inicializaci dat. Samotná třída dědí z rozhraní *IComparable* a tedy obsahuje metodu *CompareTo()*, aby bylo možné využívat metodu *Sort()* k setřídění.

Výpis 5: Implementace výpočtu datové hustoty v metodách P-Matrix a U*-Matrix

```
for (int i = 0; i < Y; i++){
  for (int j = 0; j < X; j++){
    sum = 0;
    for (int ii = -iy; ii <= iy; ii++){
      for (int jj = -jx; jj <= jx; jj++){
        if (i + ii >=0 && i + ii < Y && j + jj >=0 && j + jj < X){
          if (Math.Pow(((j-jj)-j),2) + Math.Pow(((i-ii)-i),2) >
              Math.Pow(radius,2))
            continue;
          sum += matTmp[i + ii, j + jj];
        }
      }
    }matResult[i, j] = sum;
  }
}
```

Na výpisu [5] lze vidět moji implementaci výpočtu hustoty v okolí vybraného neuronu v metodách PM a USM. První dvě vnější řídící struktury *for* se starají o postupné vybírání neuronů v mapě. Vnitřní dvojice struktur *for* je zodpovědná za vybírání neuronů ve čtvercové oblasti kolem vybraného neuronu. První podmínka kontroluje, jestli vybraný neuron nepřekračuje rozměry mapy. Pokud ano je ignorován. Druhá podmínka ošetřuje čtvercový tvar oblasti na kruhový pomocí rovnice kružnice. Velikost oblasti je daná proměnnou *radius*.

Výpis 6: Implementace vytváření skupin vrcholů (neuronů) v metodě MST

```
1   while (it < edges.Count)
2   {
3   if (!dict.ContainsKey(new Tuple<byte, byte>(edges[it][1].mx, edges
        [it][1].my))){
4     if (dict.ContainsKey(new Tuple<byte, byte>(edges[it][1].wx,
       edges[it][1].wy)))
5     {
6       dict.Add(new Tuple<byte, byte>(edges[it][1].mx, edges[it][1].
       my), dict[new Tuple<byte, byte>(edges[it][1].wx, edges[it][1].
       wy)]);
7       lines.Add(new TreeData(edges[it][1].l, edges[it][1].mx, edges[
       it][1].my, edges[it][1].wx, edges[it][1].wy));
8     }
9     else
10    {
11      dict.Add(new Tuple<byte, byte>(edges[it][1].mx, edges[it][1].
       my), group);
12      dict.Add(new Tuple<byte, byte>(edges[it][1].wx, edges[it][1].
       wy), group);
13      lines.Add(new TreeData(edges[it][1].l, edges[it][1].mx, edges[
       it][1].my, edges[it][1].wx, edges[it][1].wy));
14      group++;
15    }
16  }
17   it++;
18  }
```

V popisu metody MST byl popsán postup vytváření skupin neuronů. Implementaci tohoto jevu je možné vidět na výpisu [6]. Algoritmus začíná cyklem *while*, který probíhá tak dlouho, dokud nejsou všechny neurony přiřazeny do skupin. První podmínka kontroluje, jestli daný neuron již není přiřazen do skupiny. Pokud je, tak se vybraný neuron přeskočí. Vnitřní podmínka kontroluje, zda neuron na druhém konci hrany již také není ve skupině. Pokud ano, tak se volný neuron připojí do dané skupiny. V druhém případě, tedy v bloku *else* jsou oba neurony spojené hranou přiřazeny do nové skupiny.

6 Závěr

Cílem této bakalářské práce bylo seznámení se s problematikou umělých neuronových sítí typu Samoorganizující mapa a její praktickou aplikací při implementaci alespoň tří vizualizačních metod. Implementoval jsem čtyři metody. Jsou to Hit Histogram, P-Matrix, U*-Matrix a Minimum Spanning Tree. V teorii jsem se zabýval i metodou U-Matrix, která je spolu s metodou P-Matrix obsažena v metodě U*-Matrix. Implementované metody jsem přidal do hostitelské aplikace jako dynamicky linkovanou knihovnu.

Dále jsem provedl v hostitelské aplikaci několik implementačních změn. Ta nejzásadnější je, že jsem implementoval všechny metody jako statické třídy a tedy umožnil jejich běh v samostatných vláknech. Při implementaci se výběr metod PM a USM ukázal lehce nevhodný. Zmíněné metody jsou si v částech podobné, díky tomu se při vytváření dokumentace některé informace opakují.

Jak už kolega naznačil v předchozí práci, nejzajímavějším rozšířením by bylo přidání učení. Dále bych doporučil přidat implementaci metod, které by používaly šestiúhelníkovou mapu k jejich zobrazení. Díky tomu by případný uživatel měl širší výběr možností při jejich simulaci.

7 Reference

[1] Ultsch, A. (2003). *Maps for the Visualization of high-dimensional Data Spaces*. Germany: Marburg

[2] Ultsch, A. (2003). *U*-Matrix: a Tool to visualize Clusters in high dimensional Data*. Germany: Marburg

[3] Ultsch, A. (2003). *Pareto Density Estimation: Probability Density Estimation for Knowledge Discovery*. Germany: Marburg

[4] Ultsch, A., & Simeon, H.P. (1990). *Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis*. Germany: Dortmund

[5] Kohenen, T. (1982). Self-Organized formation of topologically correct feature maps. *Biological Cybernetics, 43*, 59 - 69

[6] Mayer, R., & Rauber, A. (2010). Visualising Clusters in Self-Organising Maps with Minimum Spanning Trees. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN10), LNCS, 6353*, 426 - 431, Greece: Thessaloniki

[7] *Data Mining with the Java SOMToolbox*. (n.d.). Dostupné z `http://www.ifs.tuwien.ac.at/dm/somtoolbox/visualisations.html`

[8] *Task Class*. (n.d.). Dostupné z `http://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.110).aspx`

[9] *.NET Framework System Requirements*. (n.d.). Dostupné z `http://msdn.microsoft.com/en-us/library/8z6watww(v=vs.110).aspx`

[10] *Progress<T> Class*. (n.d.). Dostupné z `http://msdn.microsoft.com/en-us/library/hh193692(v=vs.110).aspx`

[11] *Iris flower data set*. (n.d.). Dostupné z `http://en.wikipedia.org/wiki/Iris_flower_data_set`

[12] *Iris Data Set*. (n.d.). Dostupné z `http://archive.ics.uci.edu/ml/datasets/Iris`

[13] *Ray's Auriculas*. (n.d.). Dostupné z `http://raysauriculas.blogspot.cz/2012/06/fred-booley-night-owl-more.html`

[14] *Our Native Irises: Blue Flag Irises*. (n.d.). Dostupné z `http://www.fs.fed.us/wildflowers/beauty/iris/Blue_Flag/iris_virginica.shtml`

8 Obsah CD

- Dokumentace - pdf obsahující textovou část bakalářské práce.
- Program - obsahuje složky zdroj, exe, testy.
- Program/zdroj - obsahuje zdrojové kódy programu.
- Program/exe - obsahuje spustitelný program.
- Program/testy - obsahuje soubory s testovacími daty pro program.