České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Libor B u k a t a

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný

Obor: Kybernetika a měření

Název tématu: Deskripce a klasifikace vícerozměrných časových řad

Pokyny pro vypracování:

1. Seznamte se s problematikou analýzy vícerozměrných časových řad.
2. Vypracujte krátkou rešerši, soustřeďte se na témata shlukování a klasifikace řad.
3. Vybrané metody aplikujte na data lodních drah (dodaná vedoucím práce).
4. Dosažené výsledky vyhodnoťte - význam a reprezentace shluků, přesnost a srozumitelnost klasifikace.

Seznam odborné literatury:

[1] Whang, L.H.: Trajectory Clustering. SIGMOD 2007.
[2] Li, L.H.: Trajectory Outlier Detection: A Partition and Detect Framework. ICDE 2008.
[3] Pieraccini, L.: Dynamic Planar Warping for Optical Character Recognition. ICASSP 1992.

Vedoucí bakalářské práce: Ing. Jiří Kléma, Ph.D.

Platnost zadání: do konce zimního semestru 2010/2011

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 22. 10. 2009

Prohlášení

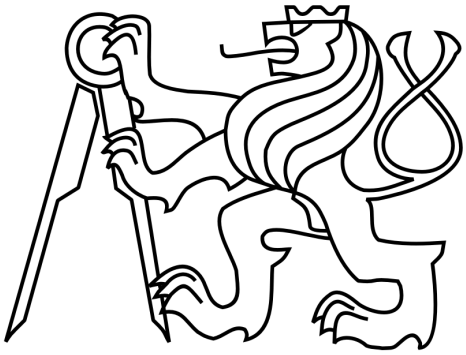
Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 20.5.2010 ..... Libor Bukata .....

Podpis

České vysoké učení technické v Praze

Fakulta elektrotechnická



Deskripce a klasifikace vícerozměrných časových řad
Description and classification multidimension time series.

(Bakalářská práce)

Libor Bukata

Vedoucí bakalářské práce: Ing. Jiří Kléma

Studijní program: Elektrotechnika a informatika

Studijní obor: Kybernetika a měření.

Datum potvrzení bakalářské práce: 22.10.2009

Abstrakt

Práce se zabývá problémem shlukování a klasifikace dvojrozměrných časových řad. Konkrétně jde o data lodních trajektorií. Výsledek klasifikace i shlukování je podmíněn zejména způsobem předzpracování a redukce řad, dále je důležitá volba vzdálenostní funkce. Předzpracování spočívá v normalizaci trajektorie do takové podoby, aby hodnota vzdálenostní funkce byla nezávislá na poloze a rotaci trajektorie. Redukce z řady vybírá pouze její charakteristické body a významně urychluje další výpočty. Pro shlukování a klasifikaci jsme zavedli 2 vzdálenostní funkce. První je převzatá funkce Edit Distance on Real sequence (EDR) s mírnou modifikaci, která zavádí korekci pro vstupní data nestejné délky. Druhá CMB (CoMBine) je nově navržená vzdálenostní funkce pro částečně chaotická data. Tato funkce je založena na hledání charakteristických příznaků trajektorie. Výsledkem je, že vzdálenostní funkce EDR dobře hledá tvarovou podobnost, zatímco vlastní vzdálenostní funkce CMB se spoléhá spíše na číselné charakteristiky trajektorií. Vzdálenostní funkce byly použity při hierarchickém shlukování s průměrováním i při klasifikaci metodou nejbližšího souseda. Podařilo se dosáhnout velmi dobrých výsledků jak z hlediska srovnání s expertními anotacemi trajektorií, tak i grafického vyhodnocení v aplikaci GoogleEarth.

Abstract

This work concerns the problems of clustering and classification of 2D time series. In particular, the thesis focuses on ship trajectory data. Before clustering and classification, the preprocessing and reduction steps are needed. Selection of a proper distance function represents another major issue. Preprocessing includes trajectory normalization into a representation independent of the trajectory location and rotation. Reduction identifies the trajectory characteristic positions which reasonably accelerates further computation. Two distance function were established and used. The first one is the well-known Edit Distance on Real sequence (EDR) function. We propose a small modification of this function, which establishes a correction factor for trajectories with a dissimilar number of observations. The second one is a newly proposed CMB (CoMBine) distance function able to work with partially chaotic data. This function uses numeric trajectory attributes. When we compare these distance functions, EDR searches for the shape similarity, while CMB distinguishes the trajectory major pattern. The distance functions were applied for hierarchical clustering with average linkage and classification by the nearest neighbor method. The reached results both match with the expert trajectory annotations and successfully pass the visual evaluation in GoogleEarth application.

Poděkování

Tato bakalářská práce by nemohla vzniknout bez významné podpory Ing. Jiřího Klémy, který práci pečlivě pročetl a poskytl cenné připomínky. Za což mu patří mé srdečné poděkování.

Také děkuji Ing. Štěpánu Urbanovi za vygenerování umělých dat lodí, na kterých bylo možné otestovat implementované algoritmy.

Obsah

Seznam použitých zkratek

TRAOD Trajectory Outlier Detection

SSE Sum of Squared Error

GCC GNU Compiler Collection

SVN Subversion

EDR Edit Distance on Real sequence

nEDR norm EDR

SAX Symbolic Aggregate approXimation

PAA Piecewise Aggregate Approximation

CMB CoMBine distance function

Seznam obrázků

Seznam algoritmů

Seznam tabulek

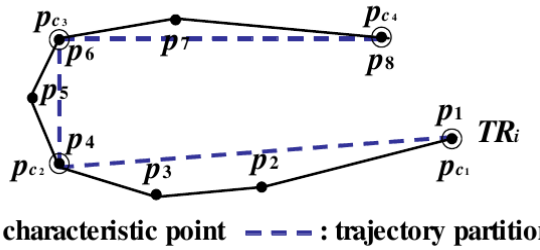
1 Rešerše

1.1 Úvod

V dnešní době se stává čím dál tím důležitějším umět správně zpracovat velký objem dat a získat z nich podstatné informace. K tomu slouží různé algoritmy, postupy, které zaručí buď ideální výsledek, a nebo výsledek skoro optimální, a to v případech, kdy je získání správných optimalizačních parametrů časově příliš náročné. Jen u zlomku těchto algoritmů je dokázáno, že jsou skutečně optimální, ale z praxe většinou víme, že dávají slušné výsledky. A to je i náš případ lodí, kde nejdříve navzorkujeme vstupní data takovým způsobem, abychom ztratili co nejméně informací, a zároveň co nejvíce snížili objem dat ke zpracování. Optimalitu některých algoritmů vlastně nelze ani dokázat, protože souvisí s vizuálním posouzením výsledku.

1.2 Úprava vstupních dat

Protože bývá často objem vstupních dat příliš vysoký, je výhodné souřadnice, které nenesou podstatnou informaci, vynechat (Obrázek 1). Celkovou trasu lodi nasekáme na úseky a vynecháme nedůležité body.



Obrázek 1: Příklad redukce dat trasy lodi. ([1] fig. 6)

Matematicky to lze popsat takto:

$$L(H) = \sum_{j=1}^{par_i - 1} \log_2 \left(len \left(p_{c_j} p_{c_{j+1}} \right) \right)$$

$$L(D|H) = \sum_{j=1}^{par_i - 1} \sum_{k=c_j}^{c_{j+1}-1} \left\{ \log_2 \left(d_\perp \left(p_{c_j} p_{c_{j+1}}, p_k p_{k+1} \right) \right) + \log_2 \left(d_\theta \left(p_{c_j} p_{c_{j+1}}, p_k p_{k+1} \right) \right) \right\}$$

První vzorec popisuje celkovou informační hodnotu v bitech (dvojkový logaritmus) dané trajektorie. Druhý kvantifikuje chybu, které jsme se dopustili hrubějším vzorkováním vůči referenčním datům. Indexy j tedy představují body hrubého vzorkování, indexy k jemného vzorkování. Našim záměrem je minimalizovat součet těchto dvou funkcí, a tedy najít optimum mezi množstvím vzorků a objemem dat:

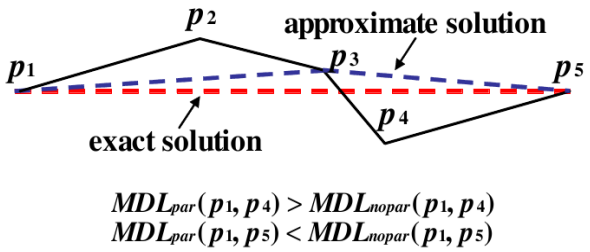
$$\min \left(L(H) + L(D|H) \right)$$

Uvedený algoritmus funguje optimálně, ale jeho časová náročnost je kvadratická, a tak se často používá přibližné řešení (Obrázek 2), které ovšem nehledá globální minimum, ale lokální minimum. Algoritmus 1 funguje ve většině případů správně ($\sim 80\,\%$) a má lineární algoritmickou složitost.

Algorithm 1 Algoritmus pro přibližné navzorkování vstupních tras. ([1] fig. 8)

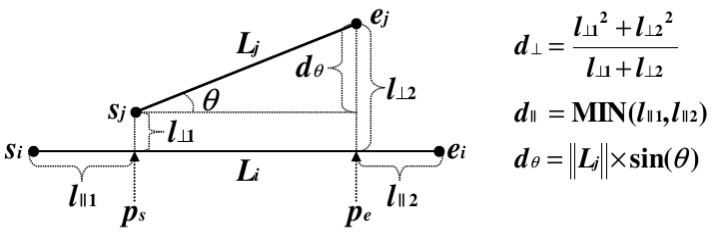
Algorithm **Approximate Trajectory Partitioning**

INPUT: A trajectory $TR_i = p_1 p_2 p_3 \cdots p_j \cdots p_{len_i}$
OUTPUT: A set CP_i of characteristic points
ALGORITHM:
01: Add p_1 into the set CP_i; /* the starting point */
02: $startIndex := 1$, $length := 1$;
03: **while** $(startIndex + length \leq len_i)$ **do**
04: $currIndex := startIndex + length$;
05: $cost_{par} := MDL_{par}(p_{startIndex}, p_{currIndex})$;
06: $cost_{nopar} := MDL_{nopar}(p_{startIndex}, p_{currIndex})$;
/* check if partitioning at the current point makes the MDL cost larger than not partitioning */
07: **if** $(cost_{par} > cost_{nopar})$ **then**
/* partition at the previous point */
08: Add $p_{currIndex-1}$ into the set CP_i;
09: $startIndex := currIndex - 1$, $length := 1$;
10: **else**
11: $length := length + 1$;
12: Add p_{len_i} into the set CP_i; /* the ending point */



Obrázek 2: Ukázka, kdy algoritmus selže - protože hledá lokální minimum. ([1] fig. 9)

1.3 Definice vzdálenostní funkce



Obrázek 3: Obrázek k odvození vzorečků. ([1] fig. 5)

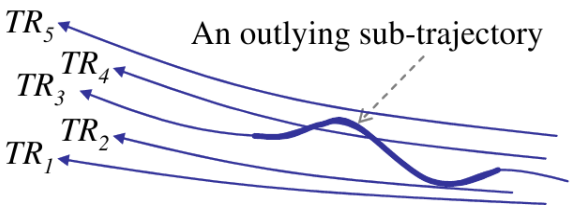
1.3.1 Pravoúhlá vzdálenost

Pomocí Lehmerova průměru je definována takto:

$d_\perp(L_i, L_j) = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}}$

1.3.2 Paralelní vzdálenost

$d_\parallel(L_i, L_j) = MIN(l_{\parallel 1}, l_{\parallel 2})$



Obrázek 4: Ukázka podezřelé trajektorie (tučná). ([2] fig. 1)

1.3.3 Rozdílnost směru

$$d_\theta(L_i, L_j) = \begin{cases} \|L_j\| \times \sin(\theta), & \text{if } 0° \leq \theta < 90° \\ \|L_j\|, & \text{if } 90° \leq \theta \leq 180° \end{cases}$$

1.3.4 Celková vzdálenostní funkce mezi dvěma úseky

$$dist\,(L_i, L_j) = \omega_\perp d_\perp\,(L_i, L_j) + \omega_\parallel d_\parallel\,(L_i, L_j) + \omega_\theta d_\theta\,(L_i, L_j)$$

$\omega_\perp$, $\omega_\parallel$, ω_θ nastavené váhy u jednotlivých vzdáleností (experimentálně zvoleny)

1.4 Detekce mimoběžných/(odchylujících se) trajektorií

Outliner je trajektorie, která se v některém místě odchýlí od většiny ostatních trajektorií, nebo když tato trajektorie je daleko od všech ostatních (Obrázek 4). V obou případech se jedná o „podezřelé" situace, které bychom chtěli detekovat.

Podle vzdálenostní funkce definované výše můžeme určit vzdálenost dvou segmentů (úseků), a tak vyhodnotit, zda jsou odlehlé. A podle celkového počtu odlehlých úseků trajektorie a počtu úseků trajektorie jsme schopni určit, zda se jedná o odlehlou/odchylující se trajektorii. Je důležité zmínit, že takto zvolená vzdálenostní funkce nesplňuje trojúhelníkovou rovnost. Tak tedy trajektorie je outliner, pokud platí:

$$Ofrac(TR_i) = \frac{suma\ vzdáleností\ odlehlých\ segmentů}{celková\ délka\ trajektorie} \geq F$$

Je jasné, že hodnotu F volíme experimentálně v závislosti na zpracovávaných datech. Pokud hodnotu F zvětšíme, budeme mírnější v hodnocení trajektorií a naopak. Neexistuje obecně optimální hodnota F, už jen z toho důvodu, že získaná data jsou ovlivněna mnoha faktory, jako jsou různá rušení, vzorkování není časově ekvidistantní, atd... A nikdy nenarazíme na přesně stejný charakter dat (nepočítáme-li binární data v počítači, kde jeden pdf dokument může být uchován v přesné podobě na několika místech). Navržený vzorec má ovšem jedno úskalí, a to je, že nepočítá s hustotou trajektorií v okolí, a tak segment v oblasti s mnoha jinými trajektoriemi velice snadno nalezne spoustu blízkých trajektorií, a tak má mnohem větší odolnost před zařazením mezi mimoběžné trajektorie. Tím je znevýhodněn segment, který se vyskytuje v mnohem řídší oblasti, kde nalezne jen minimum sousedních blízkých trajektorií, a tím je náchylnější k zařazení mezi podezřelé trajektorie. Tuto nadřazenost segmentů v hustých oblastech chceme eliminovat, a tak zavádíme upravené vzorce:

$$density\,(L_i) = \left| \bigcup_{TR_j \epsilon I} CP\,(TR_j, L_i, \sigma) \right|$$

Algorithm 2 Algoritmus TRAOD. ([2] fig. 6)

Algorithm **TRAOD** (TRAjectory Outlier Detection)

INPUT: A set of trajectories $\mathcal{I} = \{TR_1, \cdots, TR_{num_{tra}}\}$,
three parameters: D, p, and F
OUTPUT: A set of outliers $\mathcal{O} = \{O_1, \cdots, O_{num_{out}}\}$
with outlying t-partitions of each O_i
ALGORITHM:
/* I. PARTITIONING PHASE */
01: **for each** $TR_i \in \mathcal{I}$ **do**
02: Partition TR_i at a base unit;
/* II. DETECTION PHASE */
/* $\mathcal{L}$ denotes the set of t-partitions */
03: **for each** $L_i \in \mathcal{L}$ **do**
/* Definition 1 */
04: Count $|CTR(L_i, D)|$ by computing $dist(L_i, L_j)$,
/* $TR(L_i)$ means the trajectory enclosing L_i */
where $L_j \in \mathcal{L}$ and $TR(L_i) \neq TR(L_j)$;
/* Definition 2 */
05: **if** $\lceil |CTR(L_i, D)| \cdot adj(L_i) \rceil \leq \lceil (1-p)|\mathcal{I}| \rceil$ **then**
06: Mark L_i as outlying;
07: **for each** $TR_i \in \mathcal{I}$ **do**
/* Definition 3 */
08: **if** $Ofrac(TR_i) \geq F$ **then**
09: Output TR_i with its outlying t-partitions;

$$adj\,(L_i) = \frac{\sum_{L_j \epsilon L} density(L_j)/|L|}{density(L_i)}$$

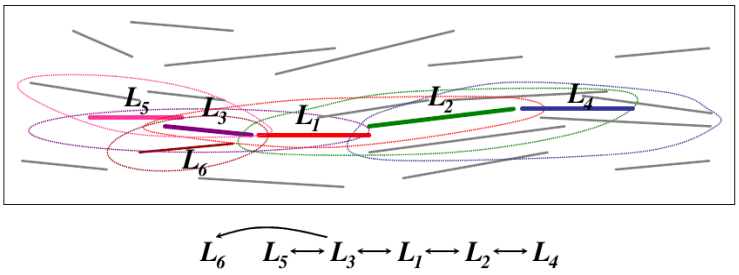
$$L = \bigcup_{TR_k \epsilon I} (TR_K)$$

První vzorec počítá celkovou hustotu segmentu L_i, parametr σ udává vzdálenost, podle které se určuje, zda trajektorie je odlehlá, nebo blízká. Tento parametr se opět nastavuje experimentálně. V druhém vzorci určíme průměrnou hustotu dané trajektorie na segment a vydělíme průměrnou hustotou na tento segment ze všech trajektorií. Tímto koeficientem přenásobíme množství trajektorií, a tak zamezíme nadřazenosti segmentů v hustých oblastech. Na získaných poznatcích je postaven algoritmus 2.

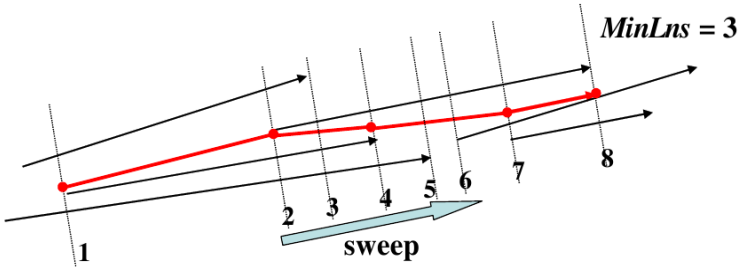
1.5 Shlukování dat

Shlukování dat slouží ke sloučení podobných dat do větších skupin. Tvary shluků jsou většinou elipsy, ale může se jednat i o jiné tvary (Obrázek 5). Každý shluk by měl obsahovat dodatečné informace (Obrázek 6), jako je vektor s průměrným směrem a velikostí všech obsažených segmentů, počet segmentů, atd... Pro účely sestavení charakteristické trajektorie shluku se s výhodou používá rotace os X a Y (Obrázek 7).

Dále je uveden algoritmus 3, který nejdříve vezme segment, zjistí, zda je v jeho okolí alespoň MinLns sousedů, pokud ano, vytvoří shluk, a vezme vnitřní segmenty v tomto shluku a pokusí se je znovu rozšířit. Pokud není dostatečný počet segmentů v okolí, tak se shluk nevytvoří/nerozšíří. Dále odstraníme ty shluky, které mají segmenty pouze od jedné trajektorie. Zásadní problém je, jak vybrat hodnotu ε, která značí vzdálenost, do jakého okolí máme hledat segmenty. Zvolíme-li tuto hodnotu příliš nízkou, je jasné, že nebudeme mít



Obrázek 5: Shluky trajektorií. ([1] fig. 10)



Obrázek 6: Příklad reprezentace trajektorie shluku. ([1] fig. 13)

skoro žádné shluky, a v opačném případě získáme jeden obrovský shluk, který nebude mít žádnou vypovídající hodnotu. Volba této hodnoty se řeší heuristicky:

$$H(X) = \sum_{i=1}^{n} p(x_i) \log_2 \left(\frac{1}{p(x_i)} \right) = -\sum_{i=1}^{n} p\left(x_i\right) \log_2 p\left(x_i\right)$$

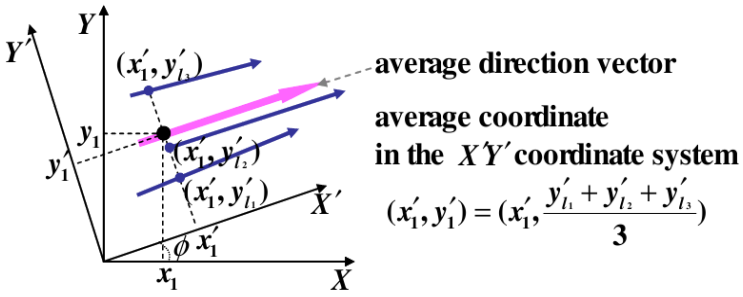
$$p\left(x_i\right) = \frac{|N_\varepsilon(x_i)|}{\sum_{j=1}^{n} |N_\varepsilon(x_j)|}$$

První vzorec počítá entropii dat shluků, je jasné, že když je entropie maximální, máme extrémně mnoho shluků, které nemají skoro žádnou vypovídající hodnotu, když je entropie nulová, tak máme příliš velké shluky. A tak bychom měli najít parametr někde mezi, a to se řeší vizuálním otestováním, a nebo výpočtem chyb SSE.

1.6 Závěr

Jak je vidět, detekce podezřelých trajektorií nebo vytváření shluků, není triviální záležitost. Do hry vstupuje spousta parametrů, které se většinou zjišťují experimenty. Ale jak je v praxi ukázáno, tyto algoritmy převážně fungují a dosahují slušných výsledků.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Obrázek 7: Rotace os X a Y. ([1] fig. 14)

Algorithm 3 Algoritmus pro výpočet shluků. ([1] fig. 12)

Algorithm **Line Segment Clustering**

INPUT: (1) A set of line segments $\mathcal{D} = \{L_1, \cdots, L_{num_{ln}}\}$,
(2) Two parameters ε and $MinLns$
OUTPUT: A set of clusters $\mathcal{O} = \{C_1, \cdots, C_{num_{clus}}\}$
ALGORITHM:

```
    /* Step 1 */
01: Set clusterId to be 0;  /* an initial id */
02: Mark all the line segments in D as unclassified;
03: for each (L ∈ D) do
04:     if (L is unclassified) then
05:         Compute N_ε(L);
06:         if (|N_ε(L)| ≥ MinLns) then
07:             Assign clusterId to ∀X ∈ N_ε(L);
08:             Insert N_ε(L) − {L} into the queue Q;
                /* Step 2 */
09:             ExpandCluster(Q, clusterId, ε, MinLns);
10:             Increase clusterId by 1;  /* a new id */
11:         else
12:             Mark L as noise;
    /* Step 3 */
13: Allocate ∀L ∈ D to its cluster C_clusterId;
    /* check the trajectory cardinality */
14: for each (C ∈ O) do
        /* a threshold other than MinLns can be used */
15:     if (|PTR(C)| < MinLns) then
16:         Remove C from the set O of clusters;

    /* Step 2: compute a density-connected set */
17: ExpandCluster(Q, clusterId, ε, MinLns) {
18:     while (Q ≠ ∅) do
19:         Let M be the first line segment in Q;
20:         Compute N_ε(M);
21:         if (|N_ε(M)| ≥ MinLns) then
22:             for each (X ∈ N_ε(M)) do
23:                 if (X is unclassified or noise) then
24:                     Assign clusterId to X;
25:                 if (X is unclassified) then
26:                     Insert X into the queue Q;
27:         Remove M from the queue Q;
28: }
```

2 Jednoduchý pokus o shlukování

2.1 Použité prostředí a knihovny

Byl vybrán jazyk C++ s ohledem na možnost objektového programování, efektivitu, šablony a rozsáhlou standardní knihovnu. Jako překladač byl použit systémový program GCC [4] (verze 4.3.X). Překlad je zajišťován podle souboru Makefile, tento automatizovaný překlad řídí program make [5]. Pro parsování byla použita externí knihovna libkml [6] verze 1.1. Celé testování probíhalo v systému Gentoo Linux [7]. Zdrojový kód byl zdokumentován pomocí programu Doxygen [8]. Pro lepší údržbu byl použit verzovací systém SVN [9]. Pro vizualizaci dat jsme využili program Google Earth [14].[1]

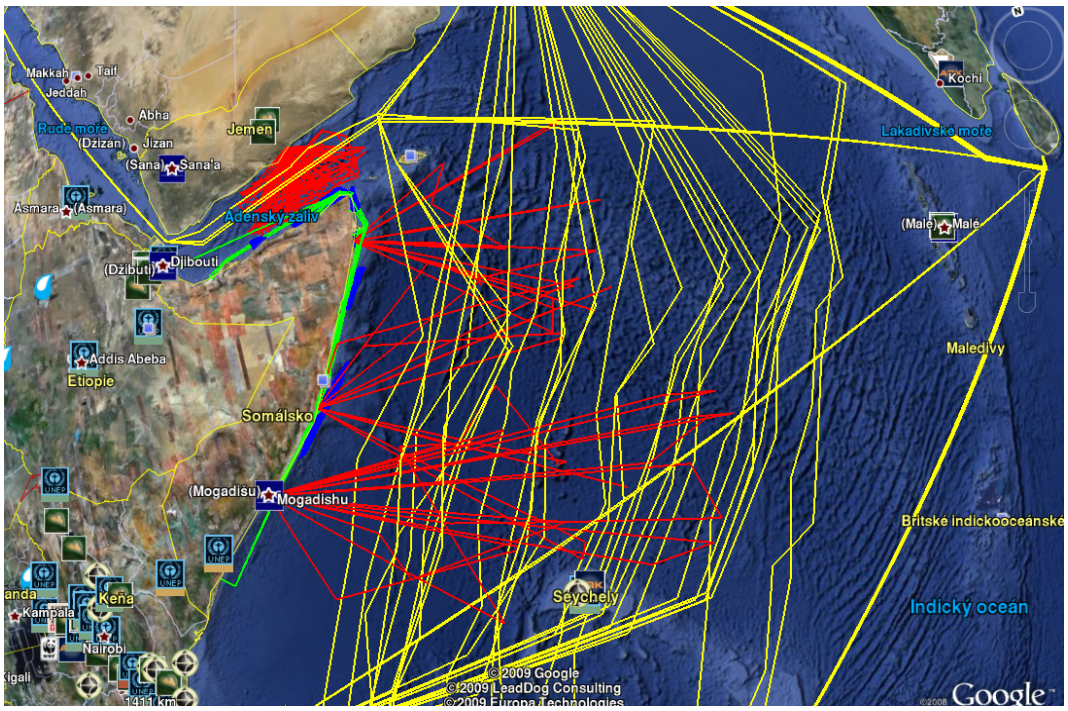
2.2 Načítání a zápis dat trajektorií

Trajektorie lodi byly poskytnuty v souboru typu kml, což je textový soubor čitelný libovolným editorem. Je to značkovací jazyk, velice podobný jazyku Xml. Trajektorie lodi mohou vypadat např. takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml>

    <Document>

        <Placemark>

            <name>3EFM6</name>
            <description>long range transport</description>
            <LineString>

              <altitudeMode>clampToGround</altitudeMode>
              <coordinates>

                18.2138529673282,-34.2825336573372,0
                18.1555176350305,-34.9611092879921,0
                ....
                80.1816737010001,5.82200894787819,0
                80.2170941768149,5.95341175978137,0

              </coordinates>

            </LineString>

        </Placemark>

            ...

    </Document>

</kml>
```

Souřadnice lodi jsou zadávány v pořadí zeměpisná délka, zeměpisná šířka a nadmořská

[1] V Linuxové verzi 5.1 byla chyba lokalizace. Pro opravení chyby stačí přidat "export LC_NUMERIC=en_US.UTF-8" do souboru googleearth.



Obrázek 8: Obarvené trajektorie. (2.2.1)

výška. Pro čtení i zápis byla použita knihovna libkml, s výjimkou metody na zápis obarvených trajektorií, kde knihovna neposkytovala dostatečné metody pro práci se styly. Detaily viz. dokumentace k implementaci. Ukázka vizualizace viz. obrázek 8.

2.2.1 Barvy použité pro obarvení trajektorií

červená	pirátské lodě
žlutá	lodě pro přepravu na dlouhou vzdálenost
zelená	přepravní lodě na krátkou vzdálenost
modrá	rybářské lodě

2.3 Jednoduchý pokus o shlukování

Nejdříve jsme museli zredukovat množství dat na trajektorii. Využili jsme algoritmus v sekci 1.2. Dále jsme provedli jednoduché shlukování, které funguje na základě vzdáleností jednotlivých trajektorií. Zde skutečně se jedná o vzdálenosti na mapě. Redukce funguje tak, že se vezme segment trajektorie, zkusí se dohledat, kolik okolních segmentů se nachází v blízkosti, je-li jich dostatek, a nejsou z totožné trajektorie, tak se zvětší shluk, takhle se rekurzivně pokračuje, až se na konci vyhodnotí, zda je ve výsledném shluku alespoň 3 a více trajektorií (lépe řečeno segmentů, které náleží těmto trajektoriím). Poté se spočítá délka segmentů pro každou trajektorii zvlášť, a vyhodnotí se, zda alespoň X % z celkové délky trajektorie leží ve shluku.

2.3.1 Konstanty pro nastavení parametrů shlukování

```
// Vzdálenost, do které se hledají okolní segmenty trajektorií..
const double NEIGHBOURHOOD_DISTANCE = 1; // Odpovídá stupním.
// Minimální počet okolních segmentů, aby se daný segment přidal do shluku.
const unsigned long MIN_NEIGHBOURS = 10;
// Minimální procentuální vzdálenost trajektorie ve shluku.
const unsigned long MIN_LENGTH_PERCENTAGE = 15;
```

2.3.2 Výsledky shlukování - výstup programu

Typ lodi	fishing	local transport	long range transport	pirate
Počet načtených lodí	10	29	100	5

Tabulka 1: Typy lodí a jejich počet.

Id shluku	fishing	local transport	long range transport	pirate
1	0	0	12	0
2	0	0	4	0
3	10	26	11	2
4	0	0	3	0

Tabulka 2: Výsledné rozložení trajektorií ve shlucích.

2.3.3 Výsledky a závěr

Ve shlucích nejsou všechny lodě, a to z toho důvodu, že některé lodě plují osamoceně a minimální počet lodí ve shluku je 3. Avšak ve výsledné tabulce záměn lze vypozorovat, že rybáři plují v blízkosti lokálního transportu, občas mezi nimi propluje pirát a nebo loď na dlouhé vzdálenosti. Podle shluků je vidět, že lodě na dlouhou vzdálenost většinou plují po vytyčených trasách, a to v blízkosti u sebe, proto ty 3 zbývající shluky. Uvedený postup sice funguje, avšak my bychom chtěli hledat trajektorie podle podobnosti, a nikoliv podle pozice na mapě. Nebo-li pro další shlukování zavedeme více symbolickou reprezentaci, a budeme zkoumat spíše tvar trajektorie, než zda loď pluje v Karibiku, nebo ve Středozemním moři...

2.3.4 Formální zápis algoritmu

Algorithm 4 Algoritmus shlukování - první pokus.

```
cluster_trajectories(Q) {
 // Vstupní data: Množina všech načtený lodí Q;
 cluster_id = 0;
 // Načteme trajektorii lodi..
 foreach B in Q {
  // Postupně projdeme všechny segmenty trajektorie aktuální lodě.
  foreach S in B {
   cluster_seg = {}

     /* Pro aktuální segment najdeme počet přilehlých segmentů ze všech
     trajektorií. Zde použijeme klasickou vzdálenostní funkci. Segmenty,
     které jsme již přiřadily do určitého shluku ignorujeme. */

   [n_close_segments list_segments] = find_close_seg(S,Q,NEIGHBOURHOOD_DISTANCE);
   // Při dostatečném počtu segmentů vytvoříme shluk.
   if (n_close_segments > MIN_NEIGHBOURS) {
    // Přidáme daný segment do listu všech segmentů v aktuálním shluku.
    cluster_seg.push_back(S);
    // Voláme rekurzivní funkci pro expanzi shluku pro přilehlé segmenty.
    expand_cluster(cluster_seg, list_segments, Q);
   } else {
    // Nejedná se o shluk, příliš málo přilehlých segmentů.
    continue;
   }

     /* Funkce spočítá počet trajektorií ve shluku a jejich relativní
     délku k celkové délce každé trajektorie */

   [n_traj_in_clust perc_length_trajs] = compute_usability_cluster(cluster_seg);
   // Ze shluku vyhodíme ty trajektorie, které nemají dostatečné zastoupení..
   foreach rel_length_traj in perc_length_trajs {
    if (rel_length_traj.perc*100 < MIN_LENGTH_PERCENTAGE) {
     // Vyhodíme tuto příliš málo významnou trajektorii z aktuálního shluku.
     remove_trajectory(cluster_seg,rel_length_traj.trajectory_name);
    }
   }


     /* Pokud je počet validních trajektorií větší než 3, prohlásíme dané
     segmenty za shluk. */

   if (n_traj_in_cluster(cluster_seg) >= 3) {
    // Validní trajektorie, přiřadíme id trajektoriím ve shluku.
    set_id(id,cluster_seg,Q); id++;
   }
  }
 }
}

expand_cluster(cluster_seg&, need_expand, ships) {
 foreach S in need_expand {
  [n_close_segments list_segments] = find_close_seg(S,ships,NEIGHBOURHOOD_DISTANCE);
  if (n_close_segments > MIN_NEIGHBOURS) {
   cluster_seg += S;
   expand_cluster(cluster_seg,list_segments);
  }
 }
}
```

3 Rešerše symbolické reprezentace trajektorií

3.1 Úprava vstupních dat

Abychom zamezili nezávislosti vzdálenostní funkce na poloze trajektorií, je nutné data normovat.

$$(x_{norm}, y_{norm}) = \left(\frac{x - \mu_x}{\sigma_x}, \frac{y - \mu_y}{\sigma_y} \right) \tag{1}$$

Tímto zamezíme vlivu geometrických transformací, jako je posun a měřítko na vliv shlukování, či klasifikace. Směrodatná odchylka je místo rozdílu minima a maxima dané souřadnice použita proto, že částečně redukuje vliv chybného měření, kdy jeden bod „ulítne".

3.2 Vzdálenostní funkce EDR

Vzdálenostní funkce EDR je založena na počtu editujících operací (vložení, mazání, náhrad), které musíme provést, abychom z jedné trajektorie dostali druhou trajektorii. Je definovaná takto[2]:

$$EDR\left(T1, T2\right) = \begin{cases} n & if\ (m == 0) \\ m & if\ (n == 0) \\ \min & \left\{ \begin{array}{c} EDR\left(tail\left(T1\right), tail\left(T2\right)\right) + subcost \\ EDR\left(tail\left(T1\right), T2\right) + 1 \\ EDR\left(T1, tail\left(T2\right)\right) + 1 \end{array} \right\} \end{cases} \tag{2}$$

Výhoda této vzdálenostní funkce je celkem vysoká odolnost proti rušení, a to proto, že hodnota subcost je 0, pokud se segmenty přibližně rovnají, jinak 1. A tak jeden zarušený segment změní výslednou hodnotu maximálně o hodnotu 1. Čím je vzdálenost (počet editujících operací) vyšší, tím jsou si trajektorie méně podobné. Algoritmus má kvadratickou složitost, pokud se použije pokročilá programovací technika dynamické programování.

3.3 Reprezentace pomocí SAX (Symbolic Aggregate approXimation)

Vytvoření této reprezentace probíhá ve 2 fázích, první je aplikování PAA (Piecewise Aggregate Approximation) na vstupní data, druhá je přiřazení symbolů k hodnotám trajektorie.

3.3.1 Redukce vstupních dat pomocí PAA

Data jsou rozdělena do rámečků o délce w, kde je spočítána střední hodnota dat uvnitř rámečku (Obrázek 9). Výpočet probíhá takto:

$$\overline{c_i} = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} c_j$$

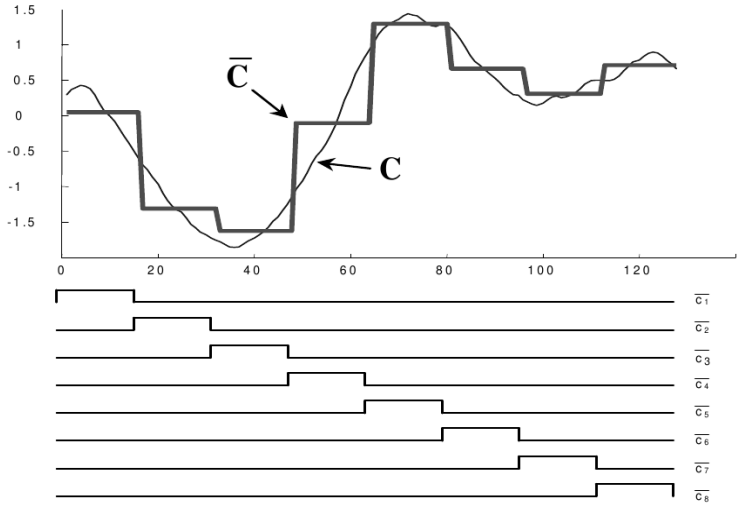
[2]Slovo tail značí, že se předají všechny body mimo prvního. Stejný význam jako v Lispu.

$\overline{c_i}$ reprezentace i-té hodnoty aproximace

w počet PAA segmentů reprezentující trajektorii

n počet vzorků časové trajektorie

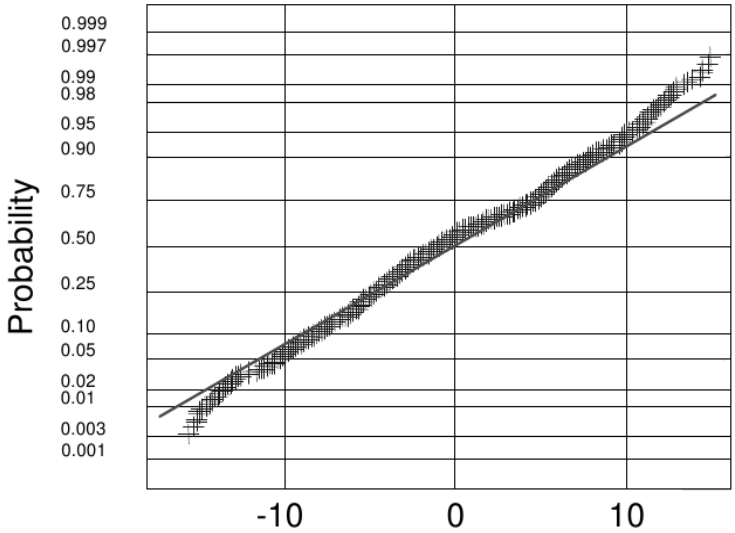
c_j j-tá hodnota časové řady



Obrázek 9: Ukázka reprezentace PAA. ([12] fig. 3)

3.3.2 Diskretizace hodnot (přiřazení symbolů)

Z analýzy signálů vyplývá, že většina signálů má Gaussovské rozdělení pravděpodobnosti (Obrázek 10). Žádoucí je generovat všechny symboly s přibližně stejnou pravděpodobností.

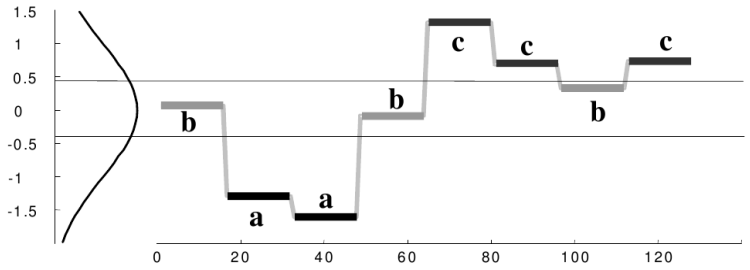


Obrázek 10: Distribuční funkce na trajektoriích délky 128 z 8 datových množin. ([12] fig. 4)

Zlomové body pro diskretizaci hodnot jsou uvedeny v následující tabulce 3. Hodnota a určuje počet symbolů diskretizace. Viz. ukázka diskretizované PAA aproximace 11.

a	3	4	5	6	7	8	9	10
β_1	-0,43	-0,67	-0,84	-0,97	-1,07	-1,15	-1,22	-1,28
β_2	0,43	0	-0,25	-0,43	-0,57	-0,67	-0,76	-0,84
β_3		0,67	0,25	0	-0,18	-0,32	-0,43	-0,52
β_4			0,84	0,43	0,18	0	-0,14	-0,25
β_5				0,97	0,57	0,32	0,14	0
β_6					1,07	0,67	0,43	0,25
β_7						1,15	0,76	0,52
β_8							1,22	0,84
β_9								1,28

Tabulka 3: Tabulka obsahující zlomové body pro rozdělení Gaussové distribuční funkce na libovolný počet stejně pravděpodobných úseků. ([12] tab. 3)



Obrázek 11: Diskretizované hodnoty PAA aproximace. Hodnoty → symboly. ([12] fig. 5)

3.3.3 Vzdálenostní funkce pro SAX

Minimální vzdálenost mezi 2 časovými řadami [12]:

$$MINDIST\left(\hat{Q},\hat{C}\right) \equiv \sqrt{\tfrac{n}{w}}\sqrt{\sum_{i=1}^{w}\left(dist\left(\hat{q}_i,\hat{c}_i\right)\right)^2}$$

Vzdálenostní funkce je pro daný počet diskretizačních hladin předpočítána ve vyhledávací tabulce.

dist	a	b	c	d	e
a	0	0	0,59	1,09	1,68
b	0	0	0	0,5	1,09
c	0,59	0	0	0	0,59
d	1,09	0,5	0	0	0
e	1,68	1,09	0,59	0	0

Tabulka 4: Ukázka vyhledávací tabulky. ([12] tab. 4)

4 Praktická realizace klasifikace a shlukování trajektorií

Data bylo nutno nejprve redukovat, k tomu jsme využili algoritmus popsaný v sekci 1.2. Poté provést normalizaci. A pak provést shlukování, nebo klasifikaci. Výstupy programů jsou buď v kml souboru, nebo v csv souborech.

4.1 Vyhodnocení redukce trajektorií

Průměrný počet bodů trajektorie jedné lodi byl v prvním data-setu (144 lodí) cca 1000 bodů, po redukci jsme se dostali na 40 bodů, což odpovídá 25 násobné redukci dat. Samozřejmě, že záleží na charakteru trajektorií, toto byl spíše ilustrativní příklad ukazující experimentálně ověřenou funkčnost algoritmu. U druhého data-setu je redukce pouze 16 násobná, a to proto, že je zde hodně pirátů, a protože piráti poměrně často zatáčí, je zde hodně význačných bodů, které nelze vypustit. Obrázek 12 porovnává vybrané redukované a neredukované trajektorie. Loď krátké vzdálenosti nebyla zahrnuta, protože vypadá velice podobně jako loď dlouhé přepravy. U redukovaných dat pro loď dlouhé přepravy není vidět celá trajektorie, část se „schovala" pod vodu. Je to způsobeno tím, že dva význačné body jsou od sebe příliš vzdáleny a program Google Earth mezi nimi natáhne přímku, která je ovšem pod hladinou moře.

4.2 Normalizace trajektorií

Nejdříve celou trajektorii orotujeme kolem začátku tak, aby začátek trajektorie byl vodorovně s koncem. Až poté provedeme normalizaci hodnot podle vzorce 1. Znázorněný postup je vidět na obrázku 13. Další věc, kterou musíme uvažovat, zda loď jede z místa A do místa B, nebo z místa B do místa A. Tvar trajektorií může být na mapě téměř identický, avšak vzdálenostní funkce berou souřadnice od počátku ke konci. Abychom se vypořádali s tímto problémem, musíme počítat hodnotu vzdálenostní funkce ještě pro případ, kdy vezmeme souřadnice jedné z dvou trajektorií v obráceném pořadí a hodnoty x,y se transformují kolem nuly (díky provedené normalizaci) na hodnoty -x,-y (čím se vlastně zrcadlí). Uvedený postup by měl být dobře patrný na obrázku 13, kde byly vybrány právě dvě takové lodě.

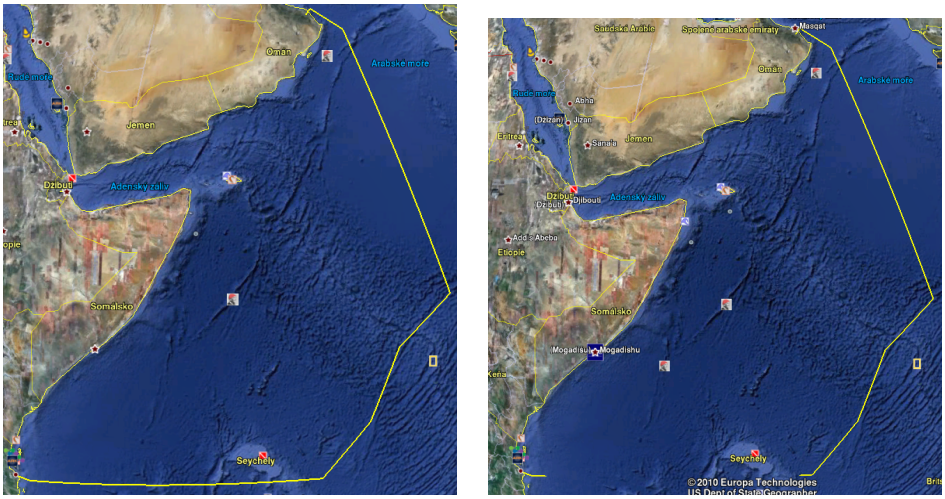
4.3 Vzdálenostní funkce

Pro shlukování a klasifikaci byly nakonec použity 2 různé vzdálenostní funkce. První je EDR, která je přesně definována podle vzorce 2, a druhá jednoduchá funkce, která byla zkonstruována speciálně na tento typ úlohy. Bližší informace v následujících podsekcích.

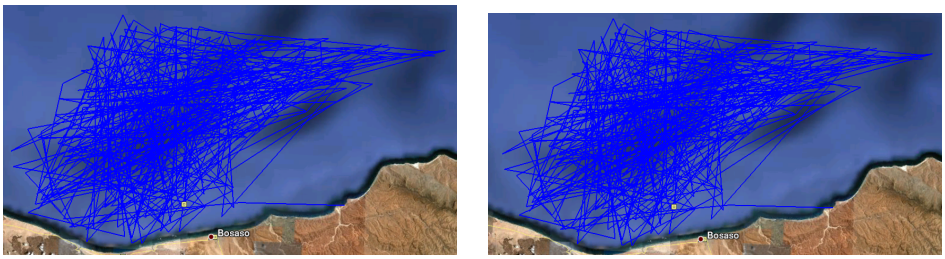
4.3.1 Vzdálenostní funkce EDR

Popis funkce EDR lze nalézt v sekci 3.2. Zde se zaměříme na způsob implementace a úskalí, které se vyskytly během shlukování.

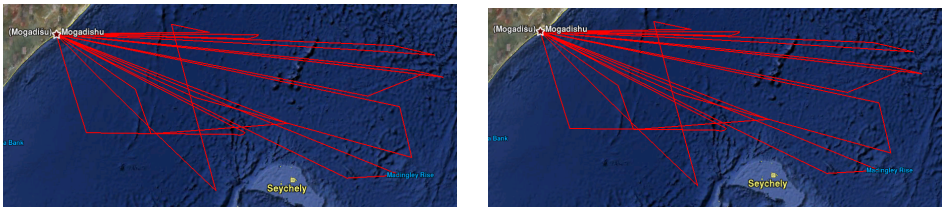
Implementace spočívá ve využití dynamického programování. Tento způsob programování spočívá v tom, že rekurzivní funkce je jakoby stavěna od spodu nahoru a přitom



(a) Loď dlouhé přepravy - neredukováno. (b) Loď dlouhé přepravy - redukováno.



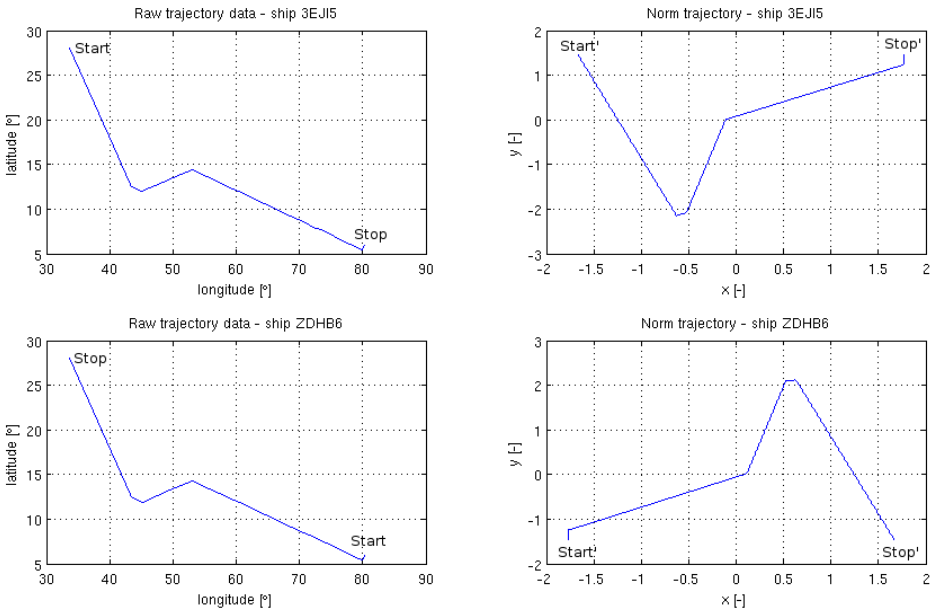
(c) Rybářská loď - neredukováno. (d) Rybářská loď - redukováno.



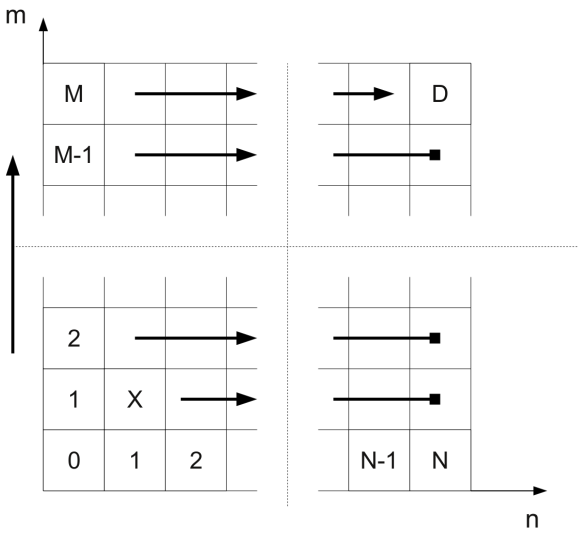
(e) Pirátská loď - neredukováno. (f) Pirátská loď - redukováno.

Obrázek 12: Porovnání redukovaných a neredukovaných trajektorií.

využívá paměť (v drtivé většině pole) pro ukládání dílčích mezivýsledků. A tak z exponenciální algoritmické náročnosti se dostaneme na polynomiální, která je již mnohem snesitelnější. Zní to velice optimisticky, avšak ne všechny rekurzivní algoritmy lze takto předělat. EDR naštěstí nepatří mezi ně. Obrázek 14 ukazuje způsob programování rekurzivní funkce 2. N a M představuje počet souřadnic dvou trajektorií, a n a m jsou jejich indexy. Vyplněné vzdálenosti na krajích jsou již známy hned od počátku, a to podle prvních dvou vztahů rovnice 2. Další výpočet spočívá postupně od místa označeného jako X, a to podle třetího vztahu rovnice 2. Nutno říci, že velikost celého dvojrozměrného pole je $(\text{počet souřadnic trajektorie1} + 1)\,(\text{počet souřadnic trajektorie2} + 1)$, protože buňka na indexu [0][0] znázorňuje prázdné trajektorie, nikoliv první (nulový index) trajektorie. Pořadí výpočtů je znázorněno šipkami, výsledek vzdálenostní funkce je v obrázku označen písmenem D. Náročnost výpočtu pro EDR je kvadratická s počtem souřadnic jednotlivých trajektorií. Ve skutečné implementaci bylo (z důvodu efektivity) dvourozměrné pole realizováno jako jednorozměrné (přístup přes jeden index, optimalizace bufferovaní paměti).



Obrázek 13: Rotace trajektorie + normalizace.



Obrázek 14: Dynamické programování - EDR.

Problém vzdálenostní funkce EDR spočívá v tom, že hodnoty, které vrací, nemají definovaný rozsah. A tak dvě naprosto rozdílné trajektorie s malým počtem souřadnic jsou od sebe méně vzdálené, než dvě naprosto rozdílné trajektorie s velkým počtem souřadnic. A tak jsem mírně modifikoval vzdálenostní funkci EDR, a označil jsem ji jako **nEDR**, tato funkce již vrací hodnoty v rozsahu $< 0; 1 >$. Způsob modifikace je následující:

$$Norm\,EDR = \frac{EDR\,distance}{max\left(length\left(trajectory1\right), length\left(trajectory2\right)\right)} \tag{3}$$

Další problém je, že vzdálenostní funkce EDR hledá tvarovou podobnost, nikoliv charakterovou podobnost. A tak lépe funguje na lodích přepravy, a o mnoho hůře na pirátech a rybářích. Například rybáři plují podle toho, kde jsou ryby, a tak tvar trajektorií je více-

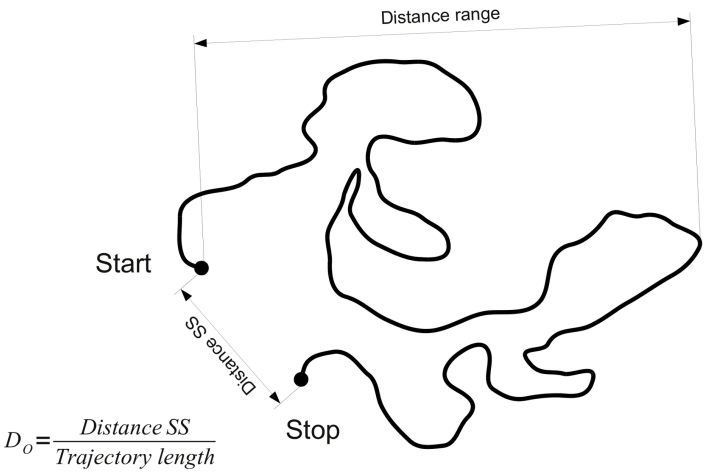
méně náhodný. Vzdálenost takových trajektorií je pak velká, i přesto, že jsou si trajektorie na první pohled podobné. To, že EDR alespoň částečně funguje na rybářích a pirátech, je dle mého názoru způsobeno tím, že pravděpodobnostní rozložení souřadnic na jednotku plochy je u pirátů a rybářů různé. A tak se s mírně vyšší pravděpodobností náhodně strefíme do blízkosti dvou souřadnic u dvou rybářů, respektive dvou pirátů, než rybář-pirát.

4.3.2 Vzdálenostní funkce CMB

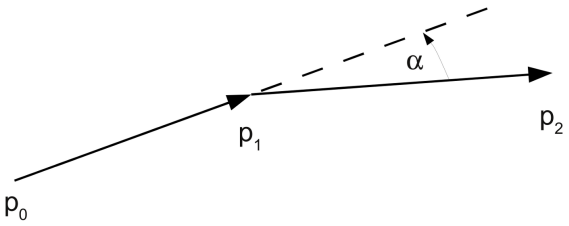
Tato vzdálenostní funkce na rozdíl od EDR nehledá tvarovou, ale charakterovou podobnost. Byla speciálně vytvořena tak, aby byla jednoduchá, měla lineární časovou náročnost a dobře fungovala na vygenerovaných datech lodí. Funkce má celkem 4 příznaky:

- Plutí lodě kolem počátku.
- Normovaný histogram úhlů zahýbání lodí.
- Dosah trajektorie od počátku.
- Délka trajektorie.

Každý příznak má stejnou váhu, funkce vrací hodnotu od 0 do 1, kde 1 je největší vzdálenost/nepodobnost dvou trajektorií. Graficky znázorněné příznaky jsou na obrázku 15. Zobrazená data lodi s normovaným histogramem úhlů jsou na obrázku 16. Nutno zmínit, že normovaný histogram úhlů je dobré počítat pro redukovaná data, abychom dostali pouze významné body. Jinak by byla naprostá většina úhlů v nule (rovný směr) a histogram by již neměl žádnou váhu - vypovídající hodnotu. Mimochodem právě u histogramu úhlů je nejvíce poznat (za předpokladu jemného histogramu), že data jsou syntetická, a to proto, že lodi zatáčejí pouze o diskrétní hodnoty úhlů.

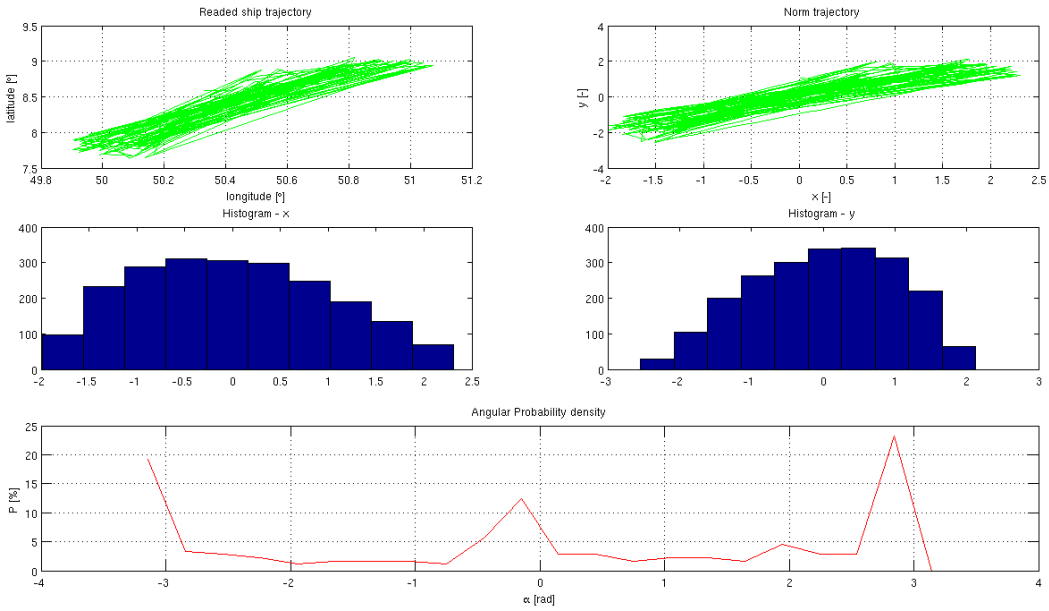


(a) Plutí lodi kolem počátku, dosah.



(b) Výpočet úhlu mezi 3 body trajektorie.

Obrázek 15: Demonstrace výpočtu příznaků vzdálenostní funkce CMB.



Obrázek 16: Zobrazení dat rybářské lodi 3EAP7 (redukovaná data).

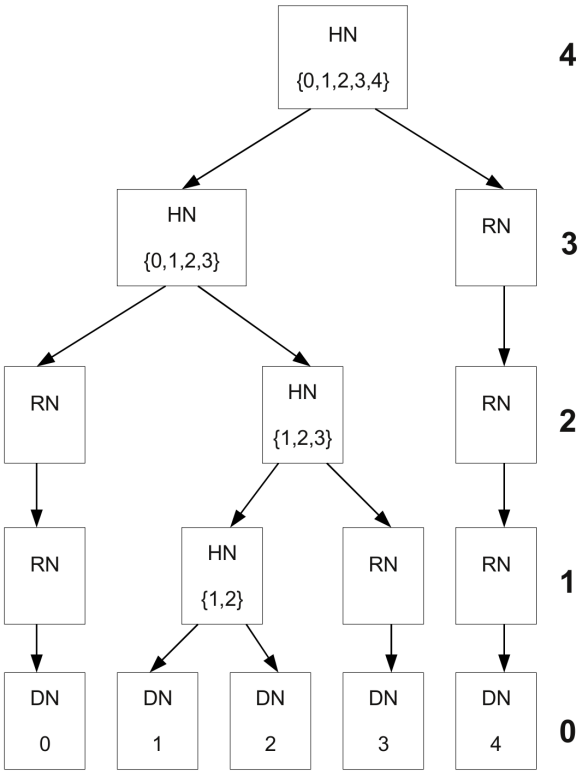
4.4 Hierarchické shlukování

Při tomto způsobu shlukování se vytváří strom, jehož struktura je závislá na vzdálenostech jednotlivých trajektorií. Na obrázku 17 je ukázka takového stromu, kde jsou jednotlivé uzly označeny takto:

HN rozbočovací uzel (hub node)

RN opakovací uzel (repeat node)

DN datový uzel (data node)

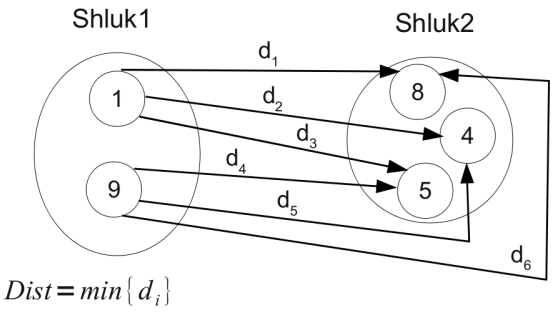


Obrázek 17: Shlukovací strom.

V každé hladině (tučné číslo napravo) může být maximálně jeden rozbočovací uzel, výška zbytku stromu se dorovná opakovacími uzly. Datový uzel obsahuje index na data jedné trajektorie, se kterou je spojen. Potřebný počet shluků se pak získá řezem skrz příslušnou výškovou hladinu stromu. Pro rozbočovací uzel se vždy berou vzdálenostně nejbližší dva shluky. Toto je realizováno binární haldou.

4.4.1 Metoda nejbližšího souseda (single-linkage)

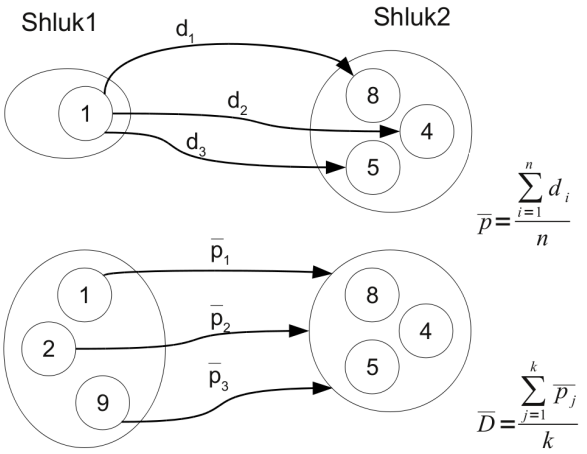
Tato metoda spočívá v tom, že jako vzdálenost dvou shluků se bere nejmenší vzdálenost mezi libovolnými 2 trajektoriemi z různých shluků. Tuto metodu znázorňuje obrázek 18. Metoda nejbližšího souseda má tendenci dělat shluky o velkém počtu lodí, protože stačí podobnost pouze mezi 2 trajektoriemi.



Obrázek 18: Metoda nejbližšího souseda.

4.4.2 Metoda průměrování (average-linkage)

Na rozdíl od nejbližšího souseda nemá tato metoda sklony k upřednostňování velkých shluků. To je docíleno tím, že namísto minimální vzdálenosti se počítá s průměrnou vzdáleností. Způsob výpočtu a grafické znázornění je patrno na obrázku 19.



Obrázek 19: Metoda průměrování.

4.4.3 Vyhodnocení správnosti funkce

Správnost funkce implementovaného shlukování byla ověřena pomocí porovnání výsledků s Matlabem. Program na shlukování umí zapsat vzdálenostní matici ve formátu pro Matlab ve formě csv souboru.

4.5 Klasifikace

Klasifikace probíhá tak, že jsou načteny dvě množiny dat, jedna učící, druhá testovací. Typ testovací lodi se přiřadí podle nejbližší lodi v datech na učení (nejbližší soused) nebo podle majority ze 3 nejbližších lodí. Využívá se přitom již navržených vzdálenostních funkcí.

5 Výsledky shlukování a klasifikace

5.1 Shlukování (průměrování) - vzdálenostní funkce CMB

Nejmenší počet shluků, kdy dostáváme rozumné výsledky, je 7. Občas se totiž stane, že nějaká trajektorie je natolik rozdílná od ostatních, že se dříve sloučí dva jiné shluky rozdílných typů lodí. Výsledky jsou shrnuty tabulkou 5 a obrázkem 20. Hned v prvním shluku je vidět, že některé lodě místního transportu se podobají některým lodím dlouhé přepravy. Horší je, že ve 4 shluku jsme chybně přiřadili dva piráty k deseti rybářům. Pokud bychom použili větší počet shluků, tak by došlo k rozdělení i tohoto shluku.

číslo shluku	rybáři	místní transport	dlouhá přeprava	pirát
1	0	21	28	0
2	0	5	0	0
3	0	1	72	0
4	10	0	0	2
5	0	0	0	3
6	0	1	0	0
7	0	1	0	0

Tabulka 5: Tabulka shluků - CMB.

5.2 Shlukování (průměrování) - vzdálenostní funkce nEDR

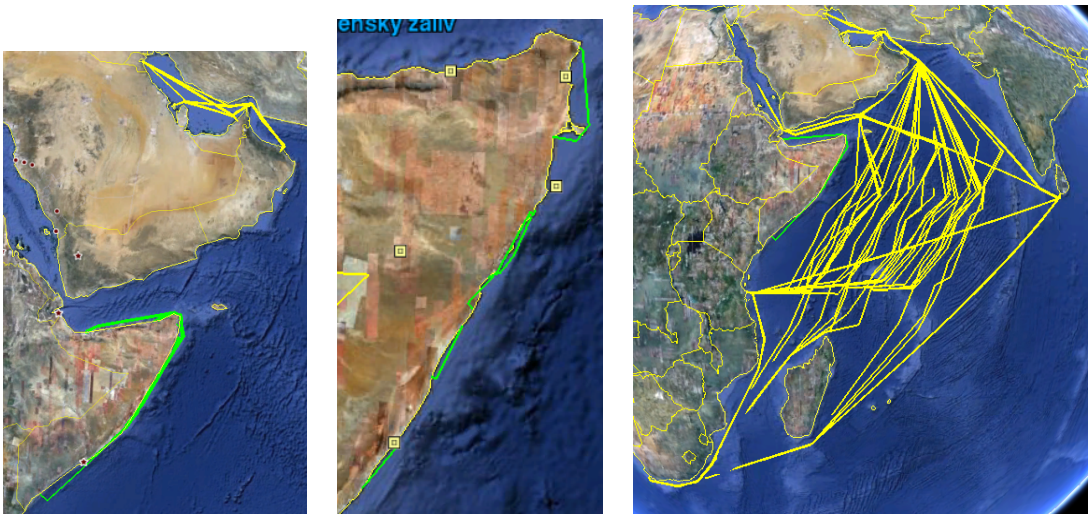
Vzdálenostní funkce nEDR provádí shlukování na základě tvaru. Z výsledků lze usoudit, že lodě pro místní přepravu jsou téměř k nerozeznání od lodí pro dlouhý transport. Jediné, co je může odlišit je délka trajektorie. Proto vzdálenostní funkce CMB byla v rozpoznání těchto dvou typů lodí úspěšnější. I přes naznačený problém v sekci 4.3.1 byli piráti a rybáři celkem úspěšně odděleni. Výsledky shlukování jsou v tabulce 6 a na obrázku 21.

číslo shluku	rybáři	místní transport	dlouhá přeprava	pirát
1	0	0	0	3
2	4	0	0	0
3	0	0	0	1
4	0	0	12	0
5	6	0	0	1
6	0	26	31	0
7	0	3	57	0

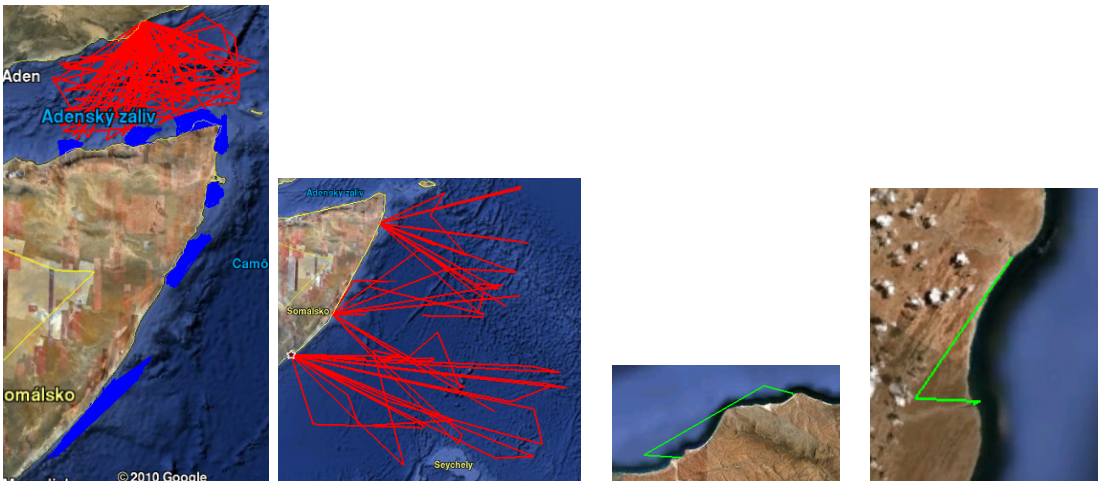
Tabulka 6: Tabulka shluků - nEDR.

5.3 Klasifikace - vzdálenostní funkce CMB

Jako soubor na učení byl použit *red_ data1.kml* (144 lodí), a jako testovací soubor *red_ data2.kml* (843 lodí). Funkce CMB dosáhla v klasifikaci vynikajících výsledků, zvlášť pro nejbližšího



(a) Shluk 1. (b) Shluk 2. (c) Shluk 3.



(d) Shluk 4. (e) Shluk 5. (f) Shluk 6. (g) Shluk 7.

Obrázek 20: Výsledné shluky - CMB.

souseda (Tabulka 7), kde přesnost klasifikace odpovídá 96,9 %, u třech nejbližších sousedů je to mírně horší (Tabulka 8), kde přesnost klasifikace odpovídá 94,2 %. U výsledků klasifikace se potvrdilo, že největší problém je rozpoznat loď místního transportu od lodě dlouhé přepravy. Občas se chybně přiřadil rybář k pirátské lodi, což může být způsobeno tím, že bylo příliš málo dat na učení pro tento typ lodě (konkrétně 10).

	rybáři	**místní transport**	**dlouhá přeprava**	**pirát**
rybáři	59	0	0	5
místní transport	0	158	21	0
dlouhá přeprava	0	0	100	0
pirát	0	0	0	500

Tabulka 7: Výsledky klasifikace (k=1) - CMB.

	rybáři	místní transport	dlouhá přeprava	pirát
rybáři	58	0	0	6
místní transport	0	136	43	0
dlouhá přeprava	0	0	100	0
pirát	0	0	0	500

Tabulka 8: Výsledky klasifikace (k=3) - CMB.

5.4 Klasifikace - vzdálenostní funkce nEDR

Vstupní data identická s klasifikací pomocí CMB. Zde se ovšem ukázala slabina nEDR popsaná v sekci 4.3.1, a proto zde tato vzdálenostní funkce špatně klasifikuje piráta ve více než 50 %, sice to může být malým počtem pirátů na učení (je jich 5), ale vzdálenostní funkce CMB jich měla stejný počet a dokázala je rozpoznat se 100 % jistotou. Zajímavé ovšem je, že rybáři se většinou klasifikovali správně. Přesnost klasifikace pro k=1 (Tabulka 9) odpovídá 65,2 %, pro k=3 (Tabulka 10) je to opět o něco horší, a to 53,7 %. Záměny mezi loděmi místního transportu a loděmi dlouhé přepravy nejsou tak kritické, a to proto, že ve skutečnosti jsou si tyto trajektorie velmi podobné.

	rybáři	místní transport	dlouhá přeprava	pirát
rybáři	63	0	0	1
místní transport	0	141	38	0
dlouhá přeprava	0	0	100	0
pirát	222	0	32	246

Tabulka 9: Výsledky klasifikace (k=1) - nEDR.

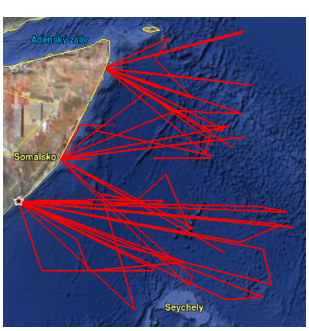
	rybáři	místní transport	dlouhá přeprava	pirát
rybáři	62	0	2	0
místní transport	0	114	65	0
dlouhá přeprava	0	0	100	0
pirát	284	0	39	177

Tabulka 10: Výsledky klasifikace (k=3) - nEDR.

5.5 Zhodnocení výsledků

Pro shlukování byl použit soubor *red_ data1.kml* pro rozumné množství trajektorií a dobré možnosti vizualizace (tento soubor je přiložen na CD). Pro výpočet vzdálenosti mezi shluky byla použita metoda průměrování, protože neupřednostňovala velké shluky a dávala rozumnější výsledky. Z výsledných shluků lze vypozorovat rozdílný přístup vzdálenostních funkcí, jedna hledá podobné charakteristiky (CMB), druhá hledá podobné tvary (nEDR), a právě proto nelze říci, která z těchto funkcí je lepší, každá se hodí na něco jiného. Zajímavé je, že problém popsaný v 4.3.1 se mnohem více projevil při klasifikaci, než při shlukování, což by šlo vysvětlit tím, že při shlukování se bere právě ta průměrná vzdálenost mezi shluky,

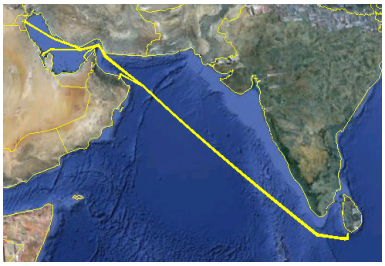
a když se například jedna rybářská loď více podobá pirátovi, tak to tolik neovlivní výsledek vzdálenosti mezi shluky (za předpokladu, že každý shluk obsahuje více než jednu loď), než když se porovnávají pouze tyto dvě lodě. A tak v průměru jsou si ve shlukování o kousek blíže lodě typu pirát-pirát resp. rybář-rybář, než pirát-rybář.



(a) Shluk 1.

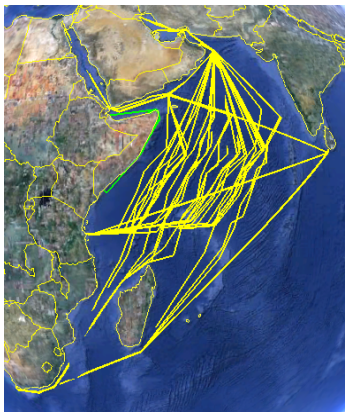
(b) Shluk 2.

(c) Shluk 3.



(d) Shluk 4.

(e) Shluk 5.



(f) Shluk 7.

(g) Shluk 6.

Obrázek 21: Výsledné shluky - nEDR.

6 Závěrečné poznámky

6.1 Náročnost a typy použitých algoritmů

Než začneme s daty cokoliv dělat, je nutné je předpřipravit[3], což je úprava trajektorií do takového tvaru, aby byly nezávislé na svých polohách a případném úhlu natočení. Pokud hledáme pouze tvarovou podobnost, tak provedeme i rozměrovou normalizaci. Všechny tyto operace mají lineární časovou náročnost v závislosti na počtu vstupních souřadnic trajektorie. Takže pro množinu lodí dostáváme kvadratickou složitost.

Experimentálně jsme ověřili, že redukce dat definovaná pomocí algoritmu 1 funguje téměř optimálně na různých typech trajektorií a navíc má lineární časovou složitost v závislosti na počtu souřadnic trajektorie (kvadratická pro množinu lodí). Redukce dat byla nutná obzvlášť pro vzdálenostní funkci nEDR, která má kvadratickou složitost, když si k tomu uvědomíme, že výpočet vzdálenostní matice má také kvadratickou složitost, dostáváme kvartickou složitost, a ta už není pro 843 lodí s průměrným počtem 1000 neredukovaných souřadnic snesitelná. Obecně se dá říci, že shlukování je výpočetně složitá operace. U vzdálenostní funkce CMB (lineární složitost) je složitost shlukování „jen" kubická.

U klasifikace je výpočetní složitost podobná shlukování a závisí na počtu lodí na učení a testování. Je rozumné, aby počet lodí na učení byl výrazně menší, než počet testovacích lodí. Klasifikace používá metodu k-nejbližších sousedů. Konkrétně implementováno pro k=1 a pro k=3 (majorita ze 3).

6.2 Zhodnocení výsledků[4]

Výsledky shlukování nebo klasifikace nejvíce ovlivňuje vzdálenostní funkce. Špatně fungující vzdálenostní funkci nezachrání sebechytřejší shlukovací/klasifikační algoritmus. V naší práci jsme použili dvě vzdálenostní funkce, první nEDR, druhá CMB. Pokud jsme použili vzdálenostní funkci nEDR, tak jsme zjistili, že shluky tvoří tvarově podobné trajektorie. Ovšem piráti a rybáři si sami sobě tvarově podobní moc nejsou, a tak tady tato funkce u klasifikace selhávala. U shlukování fungovala částečně, což se pokouším vysvětlit v sekci 5.5. Výhoda této funkce spočívá v naprosté univerzálnosti a odolnosti proti rušení, protože pokud ulítne jedna souřadnice do obrovské hodnoty, tak počet editujících operací se změní maximálně o jedničku a minimálně ovlivní výslednou hodnotu vzdálenostní funkce. Naproti tomu vzdálenostní funkce CMB počítá číselné charakteristiky trajektorií. Výhoda je, že tato funkce funguje i na částečně chaotických trajektoriích, jako jsou např. rybáři, kteří plují náhodně tam, kde je hodně ryb. Tato funkce je velmi rychlá. Ovšem největší nevýhoda této funkce je, že pokud se změní charakter vstupních dat, tak již přestanou dané příznaky fungovat a je tedy nutné najít nové příznaky a funkci modifikovat. A tak můžeme bezpečně říci, že ne-

[3] Vyjímka je vzdálenostní funkce CMB, kde není úprava vstupních dat nutná, protože tato funkce hledá číselné charakteristiky trajektorií, které jsou ovšem nezávislé na poloze na mapě.

[4] Zde jsem záměrně zhodnotil výsledky maximálně obecně, pro konkrétní výsledky stačí nahlédnout do sekce 5.

existuje ideální vzdálenostní funkce na obecná data, existuje pouze optimální vzdálenostní funkce na určitý druh dat. A tedy tvrdit, že nEDR je lepší než CMB je stejně nesmyslné, jako to tvrdit naopak. Jediné, co můžeme říci, že CMB měla lepší klasifikační výsledky a lépe odlišila jednotlivé typy lodí ve shlucích, a tak pro naše vstupní data byla úspěšnější.

Pokud již máme dobře fungující vzdálenostní funkci, je nutné vypočítat vzdálenostní matici a na té provést shlukování. Použil jsem hierarchické shlukování. Největší problém během shlukování bylo spočítat vzdálenost mezi shluky, nejdříve jsem používal metodu nejbližšího souseda, ta ovšem nadržovala velkým shlukům, a tak kazila výsledky. Není nic horšího, než mít jeden shluk, a v něm více než 90 % všech lodí. Proto bylo nutno naimplementovat jinou metodu, a to metodu průměrování, která počítá průměrnou vzdálenost mezi dvěmi trajektoriemi ve dvou shlucích. Tato metoda funguje téměř optimálně, nenadržuje velkým shlukům, ani malým shlukům, a dává rozumné výsledky. Výhoda tohoto shlukování je univerzálnost - stačí předat vzdálenostní funkci a data.

Klasifikace využívá toho, že již máme definované vzdálenostní funkce. Metoda k-nejbližších sousedů byla vybrána proto, že je jednoduchá a funguje celkem spolehlivě. Výhoda klasifikace je univerzálnost, stačí data a vzdálenostní funkce.

6.3 Další kroky

1. V dalších krocích je dobré otestovat implementované algoritmy na reálných datech a určit robustnost vzdálenostních funkcí.

2. Pokusit se zvýšit obecnost vzdálenostní funkce CMB.

3. Vyhodnotit klasifikační přesnost v závislosti na procentuálním ořezání testovacích dat (ne vždy dostaneme celé trajektorie).

Reference

[1] J.-G. Lee, J. Han and K.-Y. Whang. Trajectory Clustering: A partition-and-Group Framework. In *Proc. 2007 ACM SIGMOD Int'l Conf. on Management of Data*, Beijing, China, June 2007, pp. 593-604. The extendet version is available from http://www.cs.uiuc.edu/research/techreports.php?report=UIUCDCS-R-2007-2828.

[2] J.-G. Lee, J. Han, X. Li. Trajectory Outlier Detection: A Partition And Detect Framework. In *ICDE 2008*, pages 140-149.

[3] E. Levin, R. Pieraccini: Dynamic Planar Warping for Optical Character Recognition. In *Signal Processing Research Department, AT&T Bell Laboratories, Murray Hill, NJ 07974, U.S.A.* November 1992.

[4] *GCC, the GNU Compiler Collection* [online]. c1997 [cit. 2010-05-10]. Dostupné z WWW: <http://gcc.gnu.org/>.

[5] *GNU Make* [online]. c1997 [cit. 2010-05-10]. Dostupné z WWW: <http://www.gnu.org/software/make/make.html>.

[6] *Knihovna libkml* [online]. c2010 [cit. 2010-05-10]. Dostupné z WWW: <http://code.google.com/p/libkml/>.

[7] *Gentoo Linux* [online]. c2001 [cit. 2010-05-10]. Dostupné z WWW: <http://www.gentoo.org>.

[8] *Doxygen* [online]. 27-9-1997 [cit. 2010-05-10]. Dostupné z WWW: <http://www.doxygen.org/>.

[9] *Apache Subversion* [online]. c2010 [cit. 2010-05-10]. Dostupné z WWW: <http://subversion.apache.org/>.

[10] *KML Documentation Introduction* [online]. c2010 [cit. 2010-05-10]. Dostupné z WWW: <http://code.google.com/intl/cs/apis/kml/documentation/>.

[11] L. Chen, M. Tamer Özsu, and V. Oria. Robust and Fast Similarity Search for Moving Object Trajectories. In *CS. Tech Report. CS-2004-33, School of Computer Science, University of Waterloo.*

[12] J. Lin, E. Keogh, S. Lonardi, B. Chiu: A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In *DMKD'03, June 13, 2003, San Diego, University of California.*

[13] L. Chen, M. Tamer Özsu, and V. Oria. Symbolic representation and retrieval of moving object trajectories. In *Proc. of the ACM SIGMM international workshop on multimedia information retrieval*, pages 227-234, 2004.

[14] *Google Earth* [online]. c2010 [cit. 2010-05-17].
Dostupné z WWW: <http://earth.google.com/>.

A Popis implementovaných programů a dokumentace

Pro instalaci je potřebná knihovna *libkml* a Linuxové prostředí s překladačem *GCC*. Přeložené programy umí vypsat nápovědu přepínačem ***--help***.

Možnosti instalace a kompilace:

make Provede kompilaci programu pomocí překladače GCC. Nepovinné volby jsou např. *-j3*, která zapne využití 3 procesorů. Volba *DEBUG=1* zkompiluje program s ladícími informacemi.

make clean Smaže nepotřebné objektové soubory.

make clean_prog Smaže binární soubory programů.

make install Nainstaluje programy do adresáře */usr/local/bin/*. Potřebuje oprávnění roota.

make uninstall Smaže binární soubory přeložených programů z cesty */usr/local/bin/*.

make distrib Vytvoří soubor *AnalyseTrajectory.tar.bz2* se zdrojovými soubory projektu.

Implementované programy v C++ a skripty pro Matlab

CreateReduceTrajectory Program slouží k redukci dat lodních trajektorií a navíc je umí obarvit pro aplikaci Google Earth.

ComputeDistance Počítá z načtených dat vzdálenost mezi 2 trajektoriemi pomocí vybrané vzdálenostní funkce. Jednoduchý interaktivní shell.

GroupingTrajectory Provede shlukování dat pomocí vybrané vzdálenostní funkce a zapíše výsledky ve formě kml souborů.

ClassificationTrajectory Klasifikace testovacích lodí pomocí vybrané vzdálenostní funkce. Potřebuje data na učení.

Skripty pro Matlab Vytvořil jsem dva skripty pro Matlab. První *print_ship_data.m* pro grafické zobrazení dat lodi a druhý *compute_clusters.m* pro shlukování pomocí Matlabu. Oba skripty již potřebují vygenerovaná data ve formě csv souborů od programů na redukci a shlukování trajektorií.

Dokumentace

Ze zdrojového kódu byla vygenerována dokumentace pomocí programu *Doxygen* a to ve formě html a pdf souboru. Vzhledem k rozsáhlosti dokumentace (přes 80 stránek) nebyla vytištěna jako příloha k bakalářské práci.

B Obsah přiloženého CD

1. Adresář *bak_prace* obsahuje:
   (a) Elektronickou verzi tohoto dokumentu.
   (b) Krátkou prezentaci bakalářské práce.
2. Adresar *src* obsahuje:
   (a) Zdrojové kódy C++ programů.
   (b) Skripty pro Matlab v adresáři *matlab*.
   (c) Adresář *input_data* s daty lodí ve formě kml.
   (d) Dokumentace kódu ve formátu html nebo pdf v adresáři *doc*.
   (e) Statistiky SVN v adresáři *svnstat*.

C Použitý software

Google Earth Program na vizualizaci dat v kml souborech.

Matlab Pomocný program pro vizualizaci dat a výpočty.

GCC Použitý překladač pro jazyk C++.

Valgrind Program na hledání chyb paměti.

Doxygen Program schopný generovat dokumentaci pro různé programovací jazyky, a to jak ve formě html, tak také ve formě pdf. Využívá sazební systém LaTeX 2_ε.

Subversion Správce verzí softwaru, využíván pro údržbu softwaru.

Statsvn Generuje statistiku z SVN repozitáře. Použit spíše pro zajímavost.

Make Program, který řídí překlad.

Lyx Latex editor použitý při psaní této bakalářské práce.

Gimp Použit pro jednoduchou úpravu obrázků.

Libkml Knihovna pro parsování a manipulaci s kml soubory.

OpenOffice Draw Kreslení obrázků nebo diagramů.