Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Rozšíření platformy Clueminer o grafové algoritmy

Martin Křeček

Vedoucí práce: Ing. Tomáš Bartoň

12. května 2015

Poděkování

Rád bych poděkoval především vedoucímu této práce, Ing. Tomáši Bartoňovi, za jeho vstřícný přístup, cenné rady a ochotnou pomoc při implementaci algoritmů.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. května 2015 .....

Odkaz na tuto práci

Křeček, Martin. *Rozšíření platformy Clueminer o grafové algoritmy.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Práce se zabývá návrhem a implementací modulů pro Clueminer, což je platforma, která poskytuje funkce pro porovnávání a vizualizaci různých metod analýzy dat. Vytvářené moduly podporují zpracování dat v podobě grafů a implementují dva vybrané shlukovací algoritmy. V druhé řadě je náplní také otestování jejich funkčnosti a provedení krátkého srovnání výsledků.

Klíčová slova shlukování, modul, návrh, analýza, implementace, grafy, vytěžování dat, Java

Abstract

The aim of this thesis is to design and implement modules for Clueminer, which is a platform providing functions for comparing and visualising of diverse data analysis methods. The created modules support processing of data in the form of graphs and implement two selected clustering algorithms. Secondary aim is to also test their functionality and perform a brief comparison of the results.

Keywords clustering, module, design, analysis, implementation, graphs, data mining, Java

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V současné době přibývají v mnoha odvětvích velké objemy dat, která jsou generována nebo sbírána. Existuje proto potřeba tyto data zpracovávat tak, aby z nich byly získány užitečné informace [1]. Oblast počítačové vědy, která lidem pomáhá řešit tuto problematiku, se nazývá data mining. Zabývá se odhalováním pravidel obsažených ve velkých objemech dat a jejich využitím k získání informací použitelných pro další zpracování [2].

Jednou z metod pro analýzu dat je shlukování, jehož cílem je seskupit vzorek dat do shluků. Toto seskupování probíhá na základě vzájemné podobnosti jednotlivých objektů ve vstupních datech [3]. Cílovou platformou této práce je Clueminer, opensource framework, který podporuje interaktivní data mining se zaměřením právě na shlukování.

Tato práce se zabývá implementací dvou vybraných algoritmů pro shlukování, jejich připojením jako modulů do Clueminer a porovnáním jejich úspěšnosti pro testovací data.

Cíl práce

Cílem této práce je implementace algoritmů Chinese Whispers a Fast Community a jejich integrace do platformy Clueminer. To zahrnuje analýzu platformy, návrh implementace, integraci do systému a následné testování.

Implementované moduly, které jsou spolu s výsledky testování výstupem práce, musí být schopny na vstupních datech provést shlukování a poskytnout výstup modulům pro vizualizaci. Při implementaci je nutno brát zřetel také na efektivitu algoritmů, zejména na volbu datových struktur používaných pro reprezentaci dat.

Struktura práce

Práce je rozdělena na několik celků, z nichž každý se zabývá vybraným aspektem zpracovávaného tématu.

V první kapitole je obecně nastíněna shluková analýza a její použití, přičemž jsou blíže popsány nejčastěji používané metody. Následuje popis platformy Clueminer, především z hlediska závislostí, API a požadavků na moduly. Náplní třetí kapitoly je analýza a vysvětlení fungování algoritmů, které jsou v rámci této práce implementovány. Čtvrtá kapitola se zabývá samotnou implementací algoritmů a integrací modulů. V poslední kapitole před závěrečným shrnutím jsou uvedeny výsledky testování.

Kapitola 1

Shluková analýza

Problém shlukové analýzy (též shlukování) je studován jako součást data miningu a strojového učení kvůli jeho početným aplikacím v sumarizaci, učení, segmentaci a cíleném marketingu [4]. Shluková analýza je používána v případě, že u zpracovávaných dat nemáme k dispozici informace o jejich přiřazení do tříd.

Přesná definice shlukování je nejasná a lze ji formulovat více způsoby, protože závisí na konkrétním použití. Obecně ale lze říci, že cílem shlukové analýzy je nalézt v datech skupiny objektů, které jsou si navzájem co nejvíce podobné a současně jsou co nejméně podobné objektům mimo svoji skupinu [3, 5]. V případě shlukování dat v podobě grafové struktury je pak cílem nalézt takové skupiny uzlů, aby obsahovaly co nejvíce hran vedoucích mezi vnitřními uzly a co nejméně vedoucích k uzlům v jiných skupinách [6].

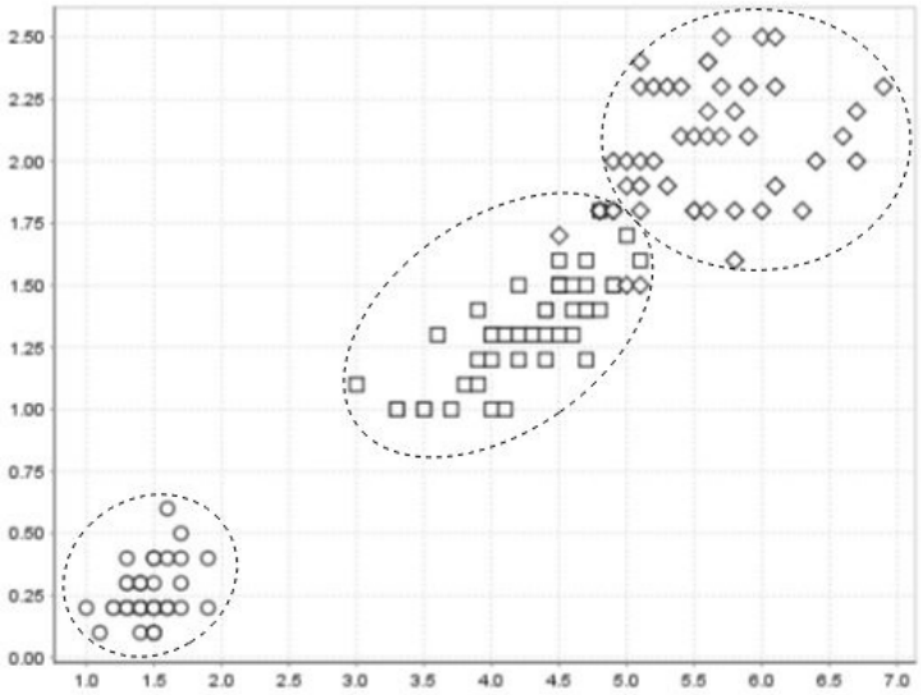
Seskupování podobných objektů do shluků může být užitečné v mnoha odvětvích [4, 5], například:

- V *ekonomii* pro hledání zemí s podobnými ekonomikami.
- Ve *finančnictví* pro hledání firem s podobnou finanční výkonností.
- V *marketingu* pro hledání skupin zákazníků s podobným nákupním chováním.
- Ve *zdravotnictví* pro hledání pacientů s podobnými příznaky.
- Při *vyhledávání dokumentů* pro hledání skupin dokumentů se souvisejícím obsahem.
- Při *analýze sociálních sítí* pro odhalení komunitních struktur a sumarizaci sítí.

1.1 Příklady shlukování

V této sekci jsou uvedeny příklady různých shlukování a jsou na nich ilustrovány základní problémy, které jsou při shlukování řešeny.

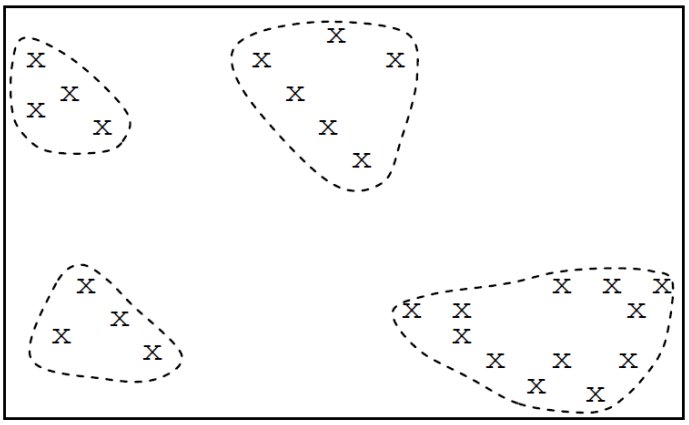
Na obrázku 1.1 jsou znázorněna data rozdělená pomocí přerušovaných čar do tří shluků. Tvar bodů vyjadřuje jejich skutečnou příslušnost k jednotlivým třídám. Je tedy patrné, že shlukování zde neproběhlo dokonale. Důvodem je fakt, že body z různých tříd jsou příliš „promíchané“ mezi sebou a není snadné je správně rozlišit.



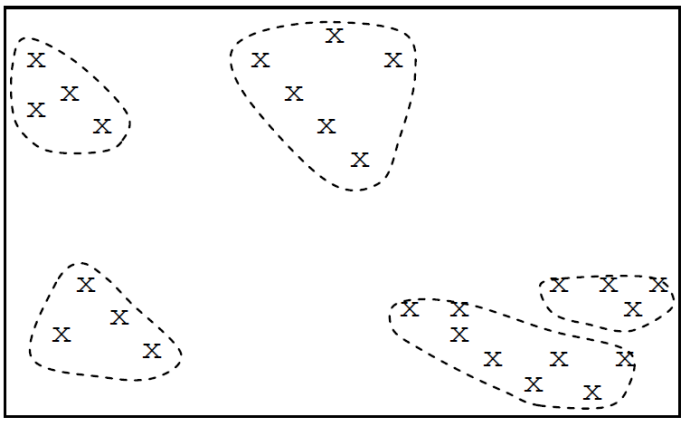
Obrázek 1.1: Příklad shlukování ukázkových dat. Tvar bodů označuje jejich příslušnost do jednotlivých tříd. Přerušované čáry znázorňují hranice vytvořených shluků.

Dalším častým problémem při shlukování je neznámý počet shluků. Data je v mnoha případech možno rozdělit více různými způsoby a většinou není jednoznačně jasné, který z nich je nevhodnější a nejlépe odpovídající realitě. Na obrázku 1.2 jsou body rozděleny do 4 shluků, které se zdají být „smysluplné“. Tytéž body jsou však rozděleny na obrázku 1.3 do 5 shluků a bez jakýchkoliv znalostí o datech reprezentovaných danými body nelze rozhodnout, která z těchto dvou možností je lepší.

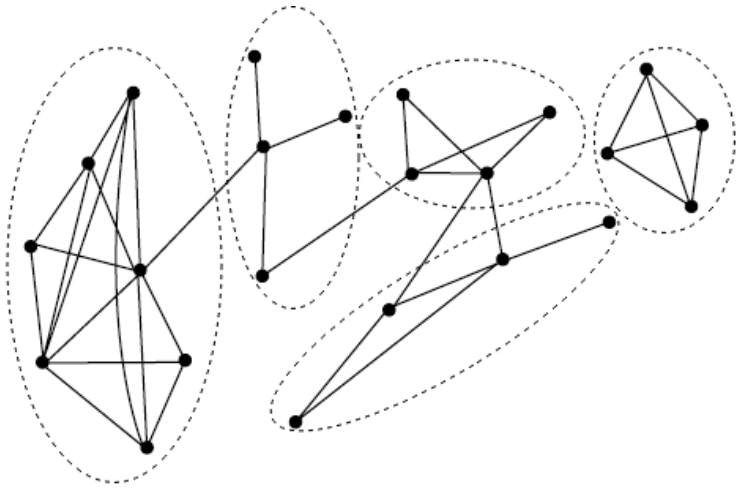
Shlukování je možné aplikovat i na data reprezentovaná ve formě grafu,



Obrázek 1.2: První možnost rozdělení bodů do shluků. V tomto případě bylo rozhodnuto, že body v pravém dolním rohu patří všechny do jedné třídy.



Obrázek 1.3: Druhá možnost rozdělení bodů do shluků. Tentokrát byly body v pravém dolním rohu rozděleny do dvou menších shluků. Tato volba se zdá být stejně validní jako předchozí možnost s jedním větším shlukem.



Obrázek 1.4: Shlukování dat v podobě grafu. Shlukovací algoritmy se v takovémto případě snaží dosáhnout toho, aby přes hranice shluků vedlo co nejméně hran (případně aby součet ohodnocení těchto hran byl minimální).

jak ilustruje obrázek 1.4. V tomto případě je snahou rozdělit uzly grafu tak, aby uvnitř shluků byla co nejhustší síť hran a mezi uzly v různých shlucích se naopak nacházelo hran co nejméně. Pokud má graf ohodnocené hrany, jsou brány v potaz i váhy jednotlivých hran a maximalizují se součty ohodnocení hran uvnitř shluků, namísto pouhého počtu hran.

1.2 Metody shlukové analýzy

V této sekci jsou popsány dvě nejčastěji používaná rozdělení metod pro shlukovou analýzu [5], spolu s jejich krátkou charakteristikou.

1.2.1 K-means

Tato metoda musí mít předem stanovený parametr k, který označuje výsledný počet shluků, do kterých mají být vstupní body přiřazeny. Na začátku algoritmu je vybráno k bodů v prostoru, které jsou označeny jako *centroidy*. Jejich pozice je možné vybrat mnoha různými způsoby, avšak lze očekávat kvalitnější výsledky v případě, že *centroidy* jsou rozmístěny dále od sebe [5].

Následuje iterativní přiřazování bodů do shluků a upravování pozic *centroidů*, které probíhá do té doby, než se přestane měnit pozice *centroidů*. Přiřazení bodu do shluku je provedeno na základě vzdálenosti k nejbližšímu *centroidu*, přičemž přiřazení bodu k *centroidu* znamená přiřazení do jeho shluku [7].

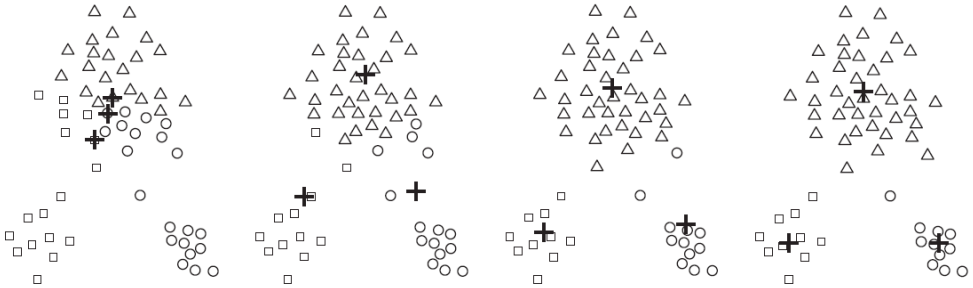
Algoritmus popisuje pseudokód 1, kde procedura `initCentroids()` provede výběr k bodů v prostoru, které jsou použity jako *centroidy*. Procedura `assignObjects()` přiřadí vstupní body vždy k nejbližšímu centroidu. Funkce

`adjustCentroids()` upraví pozice *centroidů* tak, aby vždy ležely uprostřed svého shluku, přičemž návratovou hodnotou je `true`, pokud byla pozice alespoň jednoho z *centroidů* upravena, jinak funkce vrátí `false`.

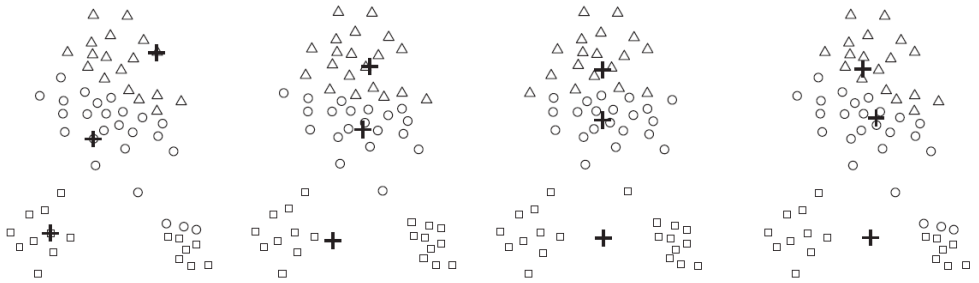
Algorithm 1 K-means

```
initCentroids();
while changes == true do
  assignObjects();
  chages = adjustCentroids();
end while
```

Průběh algoritmu vizualizuje obrázek 1.5, kde jsou znázorněny úpravy pozic *centroidů* během jednotlivých iterací. Výsledky této metody jsou citlivé na počáteční pozice *centroidů* a při jejich nevhodném zvolení jsou body přiřazeny do shluků nežádoucím způsobem, což ilustruje obrázek 1.6.



Obrázek 1.5: Vizualizace průběhu algoritmu k-means. Zleva doprava jsou postupně znázorněny pozice *centroidů* (křížky) při jednotlivých iteracích. Různé tvary jednotlivých bodů značí jejich přiřazení do shluků.



Obrázek 1.6: Průběh algoritmu k-means při nevhodném zvolení počátečních *centroidů*. Zleva doprava jsou postupně znázorněny pozice *centroidů* (křížky) při jednotlivých iteracích. Různé tvary jednotlivých bodů značí jejich přiřazení do shluků.

1.2.2 Aglomerativní hierarchické shlukování

Algoritmy z této kategorie metod pracují tak, že nejprve přiřadí každý objekt do jeho vlastního shluku a následně tyto shluky postupně slučují. Při každém kroku je nalezena dvojice shluků, které jsou k sobě navzájem nejblíže, a tyto shluky jsou sloučeny do jednoho [5].

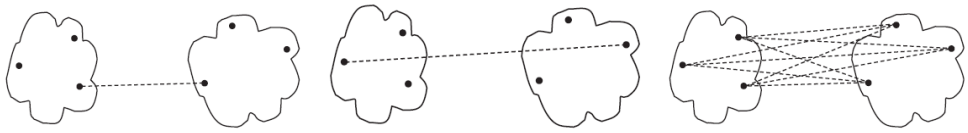
Nejbližší body jsou definovány pomocí míry vzdálenosti (například Euklidova vzdálenost), která je zpravidla vybrána na základě vhodnosti pro daná data. Pro definování vzdálenosti shluků je však třeba vzít v potaz, že se v nich nachází již více bodů. Proto se používají různé metody výpočtu vzdáleností, které určují způsob, jakým je měřena vzdálenost mezi shluky.

Používají se nejčastěji tři metody (obrázek 1.7), které definují vzdálenost dvou shluků [4]. Dvě z nich berou v potaz vzdálenost mezi dvěma body vybranými ze shluků[1], třetí počítá průměrnou vzdálenost mezi všemi body shluků[2]. Následuje výčet těchto tří metod:

- *Single link* – vzdálenost mezi nejbližšími body
- *Complete link* – vzdálenost mezi nejvzdálenějšími body
- *Average link* – průměrná vzdálenost mezi všemi body

[1] Z každého shluku je vybrán jeden bod, nelze vybrat pro měření dva body ze stejného shluku.

[2] Opět je počítána pouze vzdálenost mezi body napříč shluky, vzdálenost mezi body uvnitř shluků nehraje roli.



Obrázek 1.7: Různé metody linkování. Zleva postupně *single link*, *complete link* a *average link*.

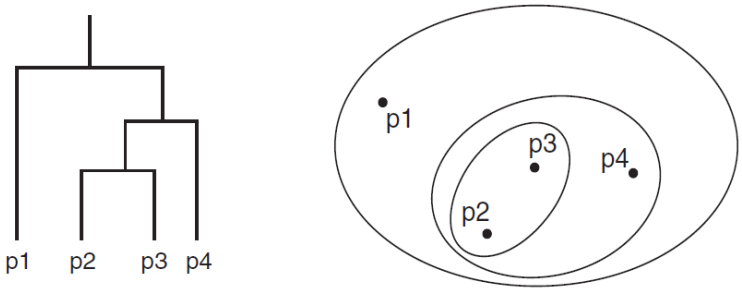
Obecný algoritmus aglomerativního hierarchického shlukování je naznačen v pseudokódu 2. Procedura `initClusters()` zde značí přiřazení každého bodu do jeho vlastního shluku a propočítání vzdáleností mezi každými dvěma shluky. Konstanta `n` označuje počet bodů, které mají být shlukovány, a tedy i počet shluků po úvodní inicializaci. Funkce `findNearest()` vrací dva sobě navzájem nejbližší shluky, které jsou následně uloženy do proměnných `x` a `y`. Tyto dva shluky jsou následně sloučeny do jednoho ve funkci `mergeClusters()` a nový shluk je uložen do proměnné `z`. Nakonec procedura `recomputeDistances()` aktualizuje vzdálenost shluku v parametru od všech ostatních shluků.

Algorithm 2 Aglomerativní hierarchické shlukování

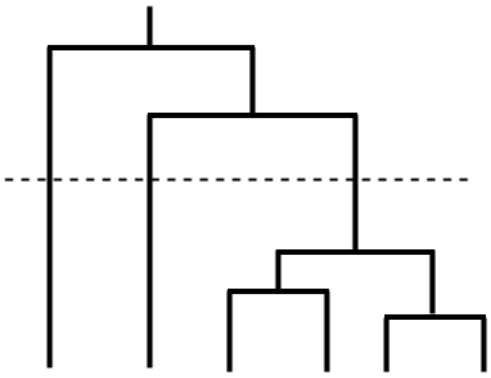
```
initClusters();
while n > 1 do
  x, y = findNearest();
  z = mergeClusters(x, y);
  recomputeDistances(z);
  n = n - 1;
end while
```

Výsledek tohoto hierarchického shlukování je běžně zobrazován jako *dendrogram* (na obrázku 1.8 vlevo), který ilustruje postupné slučování shluků až do jednoho, který pak obsahuje všechny body. Jinou metodou pro zobrazení výsledku shlukování je diagram s vnořenými shluky (na obrázku 1.8 vpravo), který je však vhodný pouze pro dvoudimenzionální data.

Pro získání konkrétního rozdělení do shluků je následně nutné výsledný *dendrogram* protnout v určité výšce v závislosti na tom, kolik shluků je vhodné vytvořit. Výšky vzniklých úseček v *dendrogramu* odpovídají vždy proporcionálně vzájemné vzdálenosti mezi shluky, které byly v daném kroku sloučeny. Bývá proto zpravidla nejvhodnější ze všech možných shlukování jako výsledné vybrat to, které má nejvyšší úsečku a předcházelo tedy největšímu „skoku“ (překonání největší vzdálenosti) při slučování shluků.



Obrázek 1.8: Vizualizace výsledků aglomerativního hierarchického shlukování. Vlevo *dendrogram*, vpravo diagram s vnořenými shluky. Původní body jsou označeny *p1* až *p4*.



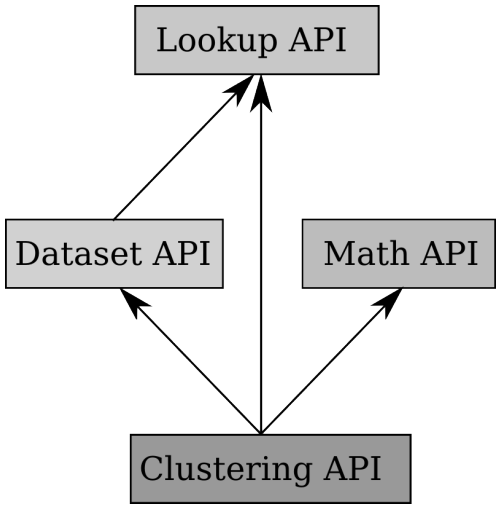
Obrázek 1.9: Řez dendrogramu. Úsečky v hladině znázorněné přerušovanou čarou jsou nejdelší, bude proto pravděpodobně nejvýhodnější provést řez dendrogramu v této úrovni a vytvořit tak 3 výsledné shluky.

Kapitola 2

Platforma Clueminer

Tato kapitola se věnuje popisu platformy Clueminer, zejména z hlediska architektury a možností rozšíření.

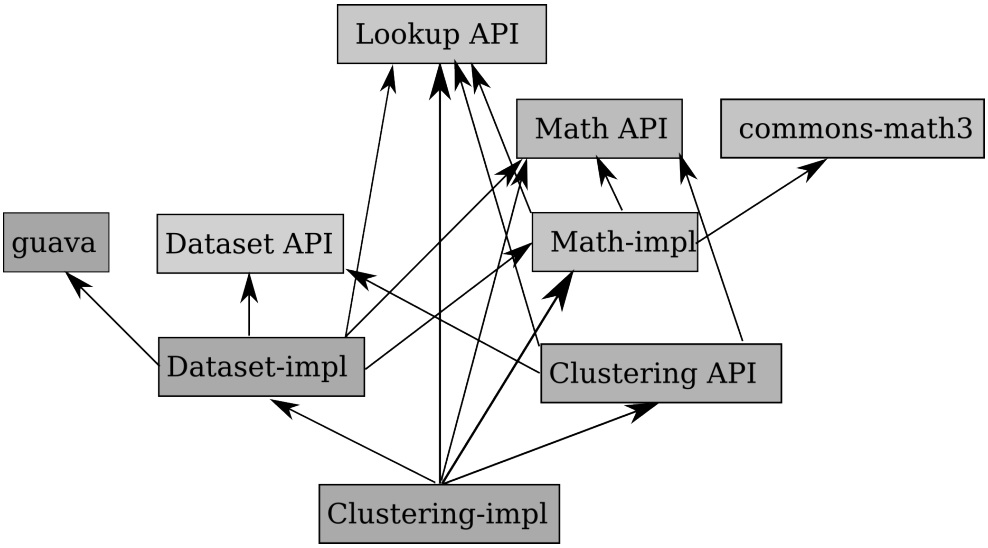
Clueminer je nástroj pro explorační datovou analýzu, který byl vytvořen v rámci magisterské práce [8]. Na rozdíl od ostatních frameworků pro data mining (RapidMiner, ELKI, Orange) Clueminer používá modulární architekturu a umožňuje jednoduché rozšíření pomocí samostatného modulu.



Obrázek 2.1: Základní moduly platformy Clueminer. Moduly definují pouze API, ale implementace je v modulu, který je závislý na tomto API.

Clueminer je postaven na NetBeans Plaform[3], což je platforma použitá mimo jiné pro NetBeans IDE. Z hlediska návrhu software se jedná o aplikování principu *loose coupling and high cohesion* [9]. Tedy jednotlivé třídy by neměly

[3] `https://netbeans.org/features/platform/`



Obrázek 2.2: Implementace základních API v rámci platformy Clueminer. V rámci architektury modulů není možné vytvářet cyklické závislosti. Modul, který používá `Clustering-impl` nemusí mít deklarovanou závislost na tento modul, stačí pouze `Clustering-api`, konkrétní implementaci je možné načíst pomocí `Lookup`.

záviset na mnoha třídách z ostatních balíků, ale využívat především třídy ze stejného balíku.

V rámci frameworku Clueminer je důležité oddělení algoritmů od grafické (prezentační) vrstvy. Grafické rozhraní závisí na API, které definuje přístup k datům (*Dataset API*) a definici rozhraní algoritmů pro shlukovou analýzu (*Clustering API*). Tento návrh umožňuje vytvoření minimální aplikace, která obsahuje pouze příkazovou řádku a definice algoritmů. Taková aplikace je vhodná např. pro spouštění analýzy na výpočetních serverech. Zároveň stejné moduly je možné využívat v grafickém rozhraní, které vizualizuje výsledky shlukové analýzy.

Díky tomuto návrhu je možné jednoduše nahradit modul, který se stará o interní reprezentaci dat a nahradit jej jiným modulem, aniž bychom museli refaktorovat další kód.

2.1 Lookup

Lookup je jednou ze základních komponent NetBeans, která umožňuje získat instance určité třídy. Následující kód najde všechny instance, které implementují požadované rozhraní.

```
Collection list = Lookup.getDefault().lookupAll(Clustering.class);
```

Modul třetí strany implementující dané rozhraní může jednoduše rozšířit funkce aplikace bez nutnosti zásahu do původních zdrojových kódů.

Tento mechanismus se používá např. pro implementaci metriky vzdálenosti. Všechny metriky vzdáleností jsou definované v jednom modulu (*high cohesion*), nicméně modul třetí strany může definovat novou metriku, která se díky stejnému API a definici `ServiceProvider` může využívat v rámci celé aplikace.

```
@ServiceProvider(service = DistanceMeasure.class)
public class EuclideanDistance implements DistanceMeasure {
    // ...
}
```

Anotace `@ServiceProvider` zajišťuje registraci třídy po přidání třídy na `CLASSPATH` Java aplikace.

2.2 Dataset API

Rozhraní Dataset API definuje přístup ke standardním „tabulkovým“ datům (tj. nejedná se např. o grafové struktury). V rámci takového datasetu se používá konvence z oblasti data-miningu, kdy jednotlivé atributy (vstupy, proměnné) jsou uvedené ve sloupcích a pozorování (měření) jsou v řádcích. Každý takový řádek může mít přiřazenou třídu (label), což se typicky používá v úlohách pro učení s učitelem, nicméně v rámci Cluemineru je informace o třídě použita pro vyhodnocení výsledků shlukové analýzy. Pro „sloupce“ (atributy) jsou automaticky počítány statistiky, jelikož je to informace, kterou potřebuje řada algoritmů.

Modul `dataset-impl` poskytuje několik implementací rozhraní Dataset API, které je postavené na rozhraní Java Collections. Nicméně standardní kolekce jako `ArrayList` se z důvodu optimalizace pro výkon příliš nevyužívají.

2.3 Clustering API

Každý shlukovací algoritmus by měl implementovat rozhraní z tohoto modulu. Rozhraní shlukovacích algoritmů předpokládá dataset definovaný v Dataset API na vstupu a výstupem je třída `Clustering`, což je kolekce shluků, které obsahují reference na řádky datasetu.

Díky obecnému rozhraní pro jednotlivé shlukovací algoritmy je pak možné psát meta-algoritmy, které pracují na principu kombinace více algoritmů, aniž by musely předem vědět o jejich existenci.

Kapitola 3

Analýza algoritmů

Oba vybrané algoritmy – Chinese Whispers a Fast Community – operují nad datasety reprezentovanými v grafové podobě, což je vhodné v případě, že data znázorňují například vazby uvnitř komunity [10]. Dalším využitím je řešení problémů z oblasti NLP[4], jako například rozpoznávání jazyků [11].

Protože objemy analyzovaných dat postupně rostou, je častým problémem běžných shlukovacích algoritmů jejich časová složitost [6]. Algoritmy zpracovávané v této práci se na tento problém zaměřují a využívají jednodušších a rychlejších postupů při shlukování. Nižší složitosti dosahují díky tomu, že během shlukování využívají pouze lokální informace v okolí právě zpracovávaného uzlu. Tím je výrazně snížen čas potřebný k výpočtům, protože není třeba při každé iteraci procházet celý graf. Dalším známým a populárním algoritmem, který využívá lokálního kontextu pro efektivní shlukování je například DBSCAN [12].

3.1 Chinese Whispers (CW)

Myšlenka tohoto algoritmu je inspirována dětskou hrou v češtině známé jako „Tichá pošta“, jejíž cílem je předávat si zprávu šeptem a postupně získat směšnou zkomoleninu původní zprávy. CW se oproti tomu snaží nalézt skupiny uzlů (hráčů), které vysílají podobnou zprávu svému okolí. [11]

CW je speciální případ algoritmu MCL, který simuluje různé konečně dlouhé sledy v grafu a identifikuje shluky na základě toho, že je pravděpodobnější, že při náhodném sledu skončíme ve stejném shluku, kde jsme začali, než že překročíme hranici shluku [13].

Mějme prostý graf $G = \{U, H, \rho\}$ s množinou uzlů U a množinou ohodnocených neorientovaných hran H, který obsahuje n uzlů a m hran a ρ označuje incidenci. Na začátku algoritmu se každému uzlu přiřadí jeho vlastní třída,

[4]Natural Language Processing – Oblast počítačové vědy zabývající se zpracováváním přirozeného (lidského) jazyka počítačem a snahou, aby počítač jazyku porozuměl.

vznikne tedy n shluků, z nichž každý bude obsahovat právě 1 uzel. Poté je iterováno přes všechny uzly grafu G a každému z nich je přiřazena nová třída. Výběr nové třídy uzlu u je proveden tak, že mezi sousedy právě zpracovávaného uzlu je nalezen nejvíce zastoupený shluk. To znamená, že zpracovávaný uzel získá tu třídu, jejíž součet vah hran vedoucích k tomuto uzlu je maximální. Pokud je tento součet po průchodu všech sousedů stejný pro více tříd, je z nich výsledná třída vybrána náhodně [11].

Algoritmus popisuje pseudokód 3, kde U označuje množinu uzlů grafu, u a v jednotlivé uzly a $u.class$ třídu uzlu u. Dále $rank[X]$ značí zastoupení třídy X, $rank.max$ třídu s maximální hodnotou v poli $rank$ (při shodě jednu náhodnou) a $w[u, v]$ váhu hrany mezi uzly u a v.

Algorithm 3 Chinese Whispers

```
for all u in U do
  C[u] = u;
end for
while changes do
  for all u in U do
    for all v in neighbors(u) do
      rank[v.class] += w[u, v];
    end for
    u.class = rank.max;
  end for
end while
```

3.2 Fast Community (FC)

Tento algoritmus patří do skupiny aglomerativních hierarchických shlukovacích metod (popsáno v sekci 1.2.2), pracuje tedy na základě postupného slučování shluků.

Oproti běžnému schématu těchto metod však využívá speciální metriku vzdálenosti, resp. místo vzdálenosti používá *modularitu* Q [10]. Tato metrika udává „kvalitu“ nebo „smysluplnost“ stávajícího rozdělení bodů do shluků.

Nejprve je definováno e_{ij} jako poměr hran v grafu, které propojují uzly ve shluku i s uzly ve shluku j. Pak nechť

$$a_i = \sum_j e_{ij}$$

označuje součet těchto poměrů, tedy součet poměrů propojení shluku i se všemi shluky. *Modularitu* Q potom lze definovat jako

$$Q = \sum_i (e_{ii} - a_i^2)$$

Modularita tedy udává poměr hran uvnitř všech shluků (komunit), bez očekávané hodnoty též metriky v případě, že hrany by byly rozmístěny náhodně, bez ohledu na přítomnost komunit v grafu.

Pokud tedy dané rozdělení do shluků nedává více hran uvnitř shluků než by se dalo očekávat od náhodného rozdělení, bude $Q = 0$. Nenulové hodnoty značí odchylku od náhodnosti a hodnoty vyšší než $0,3$ značí v praxi významnou komunitní strukturu [10].

Hlavním cílem algoritmu je tedy maximalizovat *modularitu* Q, čehož je docíleno „hladovým“ postupem. Optimalizovat Q výběrem ze všech možných kombinací postupného slučování shluků by totiž bylo výpočetně příliš náročné, protože počet možností roste exponenciálně s počtem shlukovaných bodů [10]. Je proto definováno

$$\Delta Q_{ij} = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j)$$

jako rozdíl v Q po sloučení shluků (komunit) i a j. Při každé iteraci jsou pak vybrány ty shluky, jejichž sloučení bude mít za následek maximální zvýšení (nebo minimální snížení, pokud zvýšení není v daném kroku možné) hodnoty Q.

FC funguje na stejném principu jako obecné aglomerativní hierarchické metody, které byly popsány v sekci 1.2.2. Algoritmicky je zde však oproti pseudokódu 2 několik rozdílů, které souvisí s tím, že FC používá k výběru shluků, které budou sloučeny, jinou metriku než vzdálenost. Vliv těchto detailů na implementaci je detailněji diskutován v sekci 4.2.

Algoritmus je popsán pomocí pseudokódu 4, kde procedura `initClusters()` v tomto případě také přiřadí každému bodu jeho vlastní shluk, ale místo počítání vzdáleností mezi shluky vypočítá pro každé dva shluky i a j hodnotu ΔQ_{ij} označující změnu v Q, která by nastala po jejich sloučení. Maximum z možných ΔQ, resp. shluky přiřazené k tomuto maximu pro sloučení, pak v každé iteraci vrátí funkce `findHighestDeltaQ()`. Následně jsou tyto shluky sloučeny pomocí funkce `mergeClusters()`, která vrátí nově vzniklý shluk. Procedura `recomputeDeltaQ()` pak upraví hodnoty ΔQ, aby odpovídaly aktuálnímu stavu shluků.

Algorithm 4 Fast Community

```
initClusters();
while n > 1 do
  x, y = findHighestDeltaQ();
  z = mergeClusters(x, y);
  recomputeDeltaQ(z);
  n = n - 1;
end while
```

Kapitola 4

Implementace

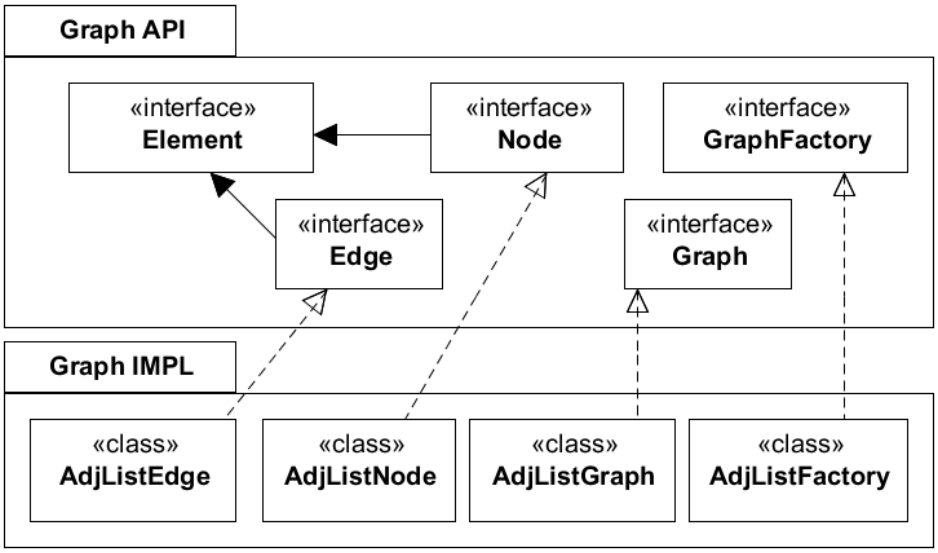
Tato kapitola se věnuje nejprve popisu používaných datových struktur, které jsou potřeba pro implementaci vybraných algoritmů. Následně jsou diskutovány specifika samotných algoritmů, otázky integrace do platformy Clueminer a s tím související závislosti modulů a tříd. V této práci je uvažováno zpracování neorientovaných grafů bez ohodnocení hran, generalizace pro obecné grafy by nebyla velkým zásahem do implementace algoritmů a mohla by být předmětem budoucích rozšíření práce.

4.1 Datové struktury

Oba implementované algoritmy využívají reprezentaci dat ve formě grafu, je tedy nutné použít k jejich implementaci vhodné datové struktury. Pro práci s grafy je využíváno rozhraní `Graph` z modulu `Graph API`, které bylo v rámci této práce implementováno ve třídě `AdjListGraph`[5] obsažené v balíčku `org.clueminer.graph.adjacencyList` v modulu `Graph IMPL`. Grafová struktura je touto třídou reprezentována jako množina uzlů, které každý obsahují množinu hran vedoucích k sousedním uzlům. To umožňuje oproti reprezentaci pomocí matice sousedností rychlejší nalezení množiny sousedů daného uzlu, čehož je využíváno zejména v algoritmu CW.

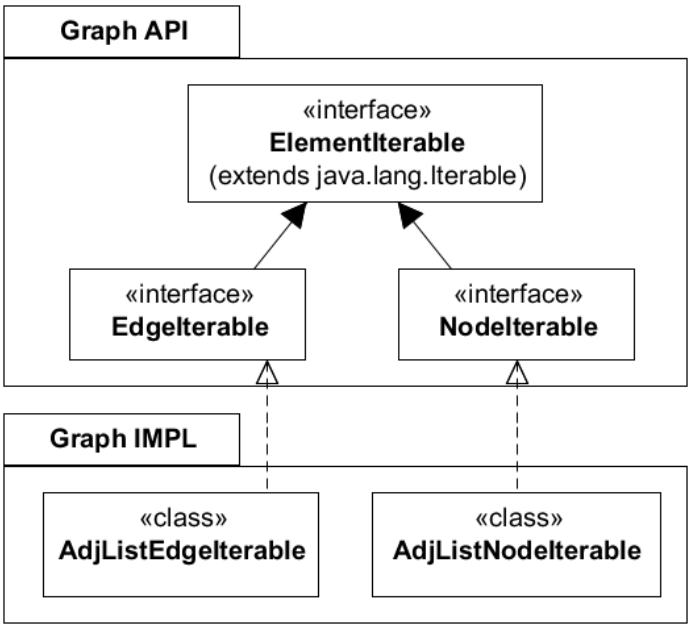
Jednotlivé třídy využívané při reprezentaci grafu jsou implementací rozhraní z balíčku `org.clueminer.graph.api` obsažených v modulu `Graph API` (např. třída `AdjListNode` implementuje rozhraní `Node`). Závislosti těchto tříd z modulu `Graph IMPL` jsou zobrazeny ve schématu 4.1.

[5] `AdjList` je zde zvoleno jako zkratka anglického „adjacency list“, tedy seznam sousedností.



Obrázek 4.1: Závislosti tříd implementujících grafové rozhraní spolu s naznačenou příslušností k modulům. Přerušované šipky značí implementaci rozhraní, vyplněné rozšíření třídy nebo rozhraní.

Dalšími strukturami obsaženými v balíčku `graph.adjacencyList` jsou pomocné třídy poskytující metody pro práci s grafem. Mezi ně patří dvě třídy, `AdjListNodeIterable` a `AdjListEdgeIterable`, které poskytují metody pro procházení seznamu uzlů, resp. hran. Závislosti těchto pomocných tříd na rozhraních z modulu `Graph API` ilustruje schema 4.2.



Obrázek 4.2: Pomocné třídy implementující iteraci přes uzly (`AdjListNodeIterable`), resp. hrany (`AdjListEdgeIterable`).

4.1.1 Chinese Whispers

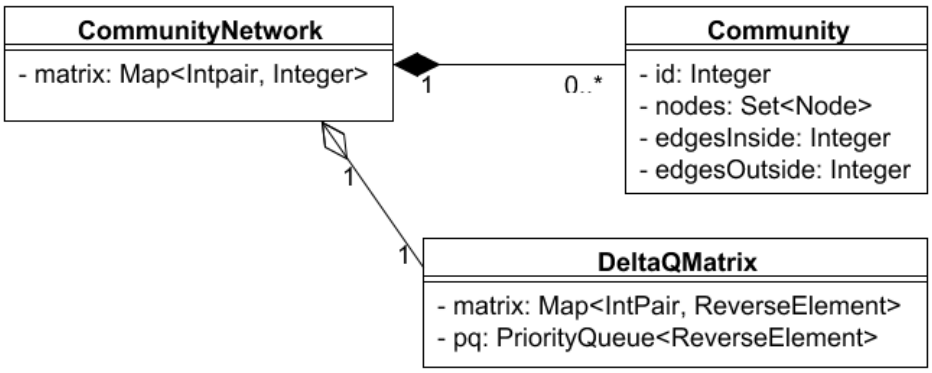
Tento algoritmus využívá pro reprezentaci grafu třídu `AdjListGraph`, jejíž reprezentace grafu je výhodná vzhledem k prováděným operacím. Implementace tohoto algoritmu je velmi přímočará, v podstatě se příliš neliší od pseudokódu 3 a neklade nároky na další pomocné struktury.

Je vhodné zmínit pouze fakt, že pro implementaci proměnné `rank` (tedy úložiště zastoupení jednotlivých tříd) z pseudokódu 3 byla použita třída `HashMap`, která poskytuje úložiště typu klíč–hodnota a je dostupná v balíčku `java.util`.

4.1.2 Fast Community

Při implementaci tohoto algoritmu bylo třeba použít několik pomocných tříd, které umožnily provádět potřebné operace dostatečně efektivně. Vzájemné relace těchto tříd znázorňuje diagram 4.3.

Základní strukturou byla množina komunit, které představují jednotlivé shluky. Ta byla implementována třídou `CommunityNetwork`, která poskytuje podobné rozhraní jako pro práci s grafy. S tím rozdílem, že za „uzly“ jsou zde požadovány komunity (tedy shluky) a je ukládán počet hran mezi nimi, tj. počet uzlů propojených napříč jednotlivými páry komunit.



Obrázek 4.3: Pomocné třídy pro algoritmus Fast Community.

Komunita je reprezentována pomocí třídy `Community`, která obsahuje kromě svého `id` ještě seznam uzlů původního grafu, které obsahuje. Pro efektivní úpravy během jednotlivých iterací je zde navíc uložena informace o tom, kolik hran původního grafu je uvnitř této komunity a kolik jich vede do jiných komunit. Využití těchto informací je blíže popsáno v sekci 4.2, konkrétně v části 4.2.2 věnující se implementaci algoritmu FC.

Další důležitou třídou je zde `DeltaQMatrix`, která implementuje matici s hodnotami ΔQ pro všechny dvojice komunit (význam hodnoty ΔQ je detailněji popsán v sekci 3.2). Zároveň tato třída zajišťuje řazení nejvyšších hodnot ΔQ v prioritní frontě (resp. jejich přidávání a odebírání), ze které jsou při každé iteraci odebírány nejvhodnější dva shluky (komunity) pro sloučení.

Třída `ReverseElement` rozšiřuje třídu `Element`, která obecně obsahuje informace o tom, jaké změny nastanou při sloučení dvou daných shluků. Třída `ReverseElement` mění původní chování třídy `Element` pouze v tom, že její metoda `compareTo()` vrací opačné hodnoty, tzn. instance této třídy jsou řazeny v obráceném pořadí. To je nutné z toho důvodu, že elementy jsou v prioritní frontě běžně řazeny od minima po maximum. Při běžném aglomerativním hierarchickém shlukování je to žádoucí chování, protože ke sloučení jsou vybírány shluky, jejichž vzájemná vzdálenost je co možná *minimální*. V případě FC je ovšem naopak snaha *maximalizovat* hodnotu ΔQ, proto je zde třeba řadit elementy od maxima po minimum.

Poslední pomocnou třídou pro tento algoritmus je `IntPair`, která pouze obaluje dvě instance třídy `Integer` a poskytuje metody pro práci s nimi.

4.2 Implementace algoritmů

Tato sekce pojednává o postupu implementace algoritmů CW a FC. Důraz je kladen na klíčové momenty v algoritmech a metody převedení logické funkcionality algoritmů do kódu.

Zvláštní datové struktury použité při implementaci jsou popsány v sekci 4.1, zde je rozebráno, jakým způsobem jsou třídy používány při programování těchto algoritmů.

4.2.1 Chinese Whispers

Při implementaci tohoto algoritmu byly pro práci s grafem použity metody třídy `AdjListGraph`. Algoritmus nemá žádné speciální nároky na pomocné datové struktury, takže rozhraní této třídy bylo dostačující.

Reprezentace grafu formou seznamu sousedností (oproti matici sousedností) byla vhodná z hlediska často prováděných operací. Protože algoritmus potřebuje při každé své iteraci nacházet sousedy každého uzlu, tento přístup umožňuje získat tento seznam sousedů od uzlu v konstantním čase.

Třída pro tento algoritmus implementuje metodu `cluster()`, která přijímá jako parametr vstupní data a vrací výsledné shlukování. Žádné další přípravné nebo optimalizační procesy nejsou v tomto případě potřeba, jak bylo již zmíněno v části 4.1.1, implementace algoritmu CW je velmi přímočará.

4.2.2 Fast Community

Tento algoritmus obsahuje několik klíčových kroků a principů, které jsou vysvětleny v této části. Jde v první řadě o způsob, jakým je aktualizována prioritní fronta, ve které jsou řazeny prvky označující jednotlivá možná sloučení shluků. Následně je ještě popsána metoda počítání hodnot potřebných k aktualizaci metriky, podle které jsou vybírány shluky ke sloučení.

Základním krokem tohoto algoritmu je sloučení dvou komunit (shluků), přičemž jedna z nich při tomto procesu zanikne a poté je potřeba aktualizovat hodnoty ΔQ, jelikož se změní počty hran vedoucích uvnitř a mezi jednotlivými komunitami. Tento krok, resp. aktualizace hodnot po něm, odpovídá přepočítání vzdáleností[6] mezi shluky při běžném aglomerativním hierarchickém shlukování popsaném v části 1.2.2.

Matice hodnot ΔQ je zde implementována ve třídě `DeltaQMatrix` jako `HashMap`, kde klíčem je dvojice celých čísel (identifikátory komunit) a hodnotou instance třídy `ReverseElement`, která mimo jiné obsahuje informaci o změně *modularity* Q po sloučení daných dvou komunit. Podrobnější informace o třídě `ReverseElement` jsou uvedeny v části 4.1.2 věnující se datovým strukturám použitým při implementaci algoritmu FC. Tento způsob implementace umožňuje provést aktualizaci hodnoty v matici, případně její přidání

[6] Konkrétně jde o volání procedury `recomputeDistances()` v pseudokódu 2.

nebo smazání, bez nutnosti realokovat v paměti zbytek hodnot, což by bylo nutné v případě, že by pro reprezentaci bylo použité jednoduše dvourozměrné pole. Po sloučení dvou komunit jsou zároveň změněné hodnoty v ΔQ upraveny třídou `DeltaQMatrix` i v prioritní frontě, ze které je následně při další iteraci opět vybrán následující pár shluků nejvíce vhodný pro sloučení.

Jak bylo uvedeno v popisu algoritmu FC v sekci 3.2, při výpočtu *modularity* Q (a potažmo i její změny ΔQ) je využíváno poměru počtu propojení se všemi komunitami (pro danou komunitu i označováno jako a_i). Při slučování dvou komunit, což je hlavní funkce třídy `CommunityNetwork`, lze pak změnu počtu hran vedoucích ven z komunity i (značeno $i.out$) po sloučení s komunitou j definovat následujícím vztahem,

$$i.out = i.out + j.out - 2e_{ij}$$

kde e_{ij} značí počet hran mezi komunitami i a j. Zjednodušeně řečeno, počet hran vedoucích mimo nově vzniklou komunitu lze spočítat jako součet počtů hran vedoucích mimo obě slučované komunity. Je však třeba odečíst dvojnásobek počtu hran mezi slučovanými komunitami[7], jelikož tyto hrany se po sloučení stanou vnitřními hranami nové komunity. Tento výpočet nahradí procházení jednotlivých hran v komunitách a počítání „statistik“, které udávají jak hustě jsou mezi sebou a uvnitř sebe komunity propojeny.

4.3 Integrace

Obě třídy používají notaci `@ServiceProvider` popsanou v sekci 2.1, která umožňuje jejich načtení na základě rozhraní, které implementují. Tato notace zaručí, že je zbytek aplikace „informován“ o tom, že tyto třídy implementují shlukovací algoritmy a mohou tak být použity ke shlukování.

```
@ServiceProvider(service = ClusteringAlgorithm.class)
```

Současně obě třídy rozšiřují abstraktní třídu `AbstractClusteringAlgorithm`, která definuje některé metody z rozhraní `ClusteringAlgorithm` a tak je třeba definovat již jen metody `getName()` a `cluster()`.

```
public class ChineseWhispers extends AbstractClusteringAlgorithm {
    // ...

    @Override
    public String getName() {
        return "Chinese Whispers";
    }
```

[7] Dvojnásobek z toho důvodu, že hrany mezi nimi jsou započítány v obou ze slučovaných komunit.

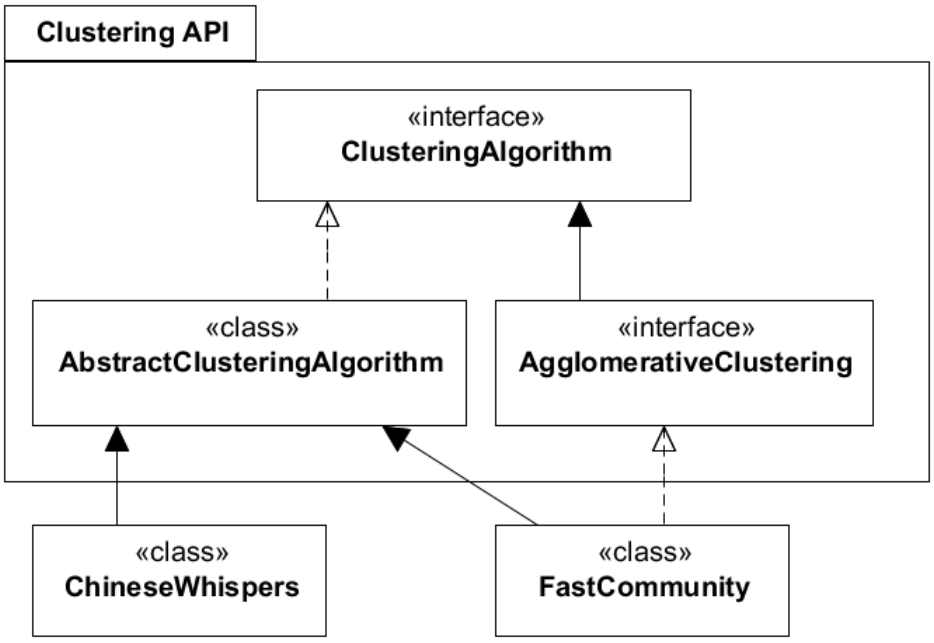
```
    @Override
    public Clustering<Cluster> cluster
            (Dataset<? extends Instance> dataset) {
        // ...
    }
}
```

V případě třídy `FastCommunity` je ještě přidána notace indikující implementaci rozhraní `AgglomerativeClustering`, protože výstupem algoritmu FC je *dendrogram* značící hierarchický výsledek. Důležitou metodou, kterou je zde třeba implementovat, je `hierarchy()`, která v parametru přijímá vstupní data a nastavení parametrů a vrací hierarchický výsledek.

```
public class FastCommunity extends AbstractClusteringAlgorithm
                           implements AgglomerativeClustering {
    // ...

    @Override
    public HierarchicalResult hierarchy
            (Dataset<? extends Instance> dataset, Props pref) {
        // ...
    }
}
```

Závislosti implementovaných tříd na modulu `Clustering API` znázorňuje schema 4.4.



Obrázek 4.4: Závislosti implementovaných tříd na Clustering API. `ChineseWhispers` i `FastCommunity` rozšiřují třídu `AbstractClusteringAlgorithm`, která implementuje rozhraní `ClusteringAlgorithm`. `FastCommunity` navíc implementuje rozhraní `AgglomerativeClustering`, které rozšiřuje rozhraní `ClusteringAlgorithm`.

Kapitola 5

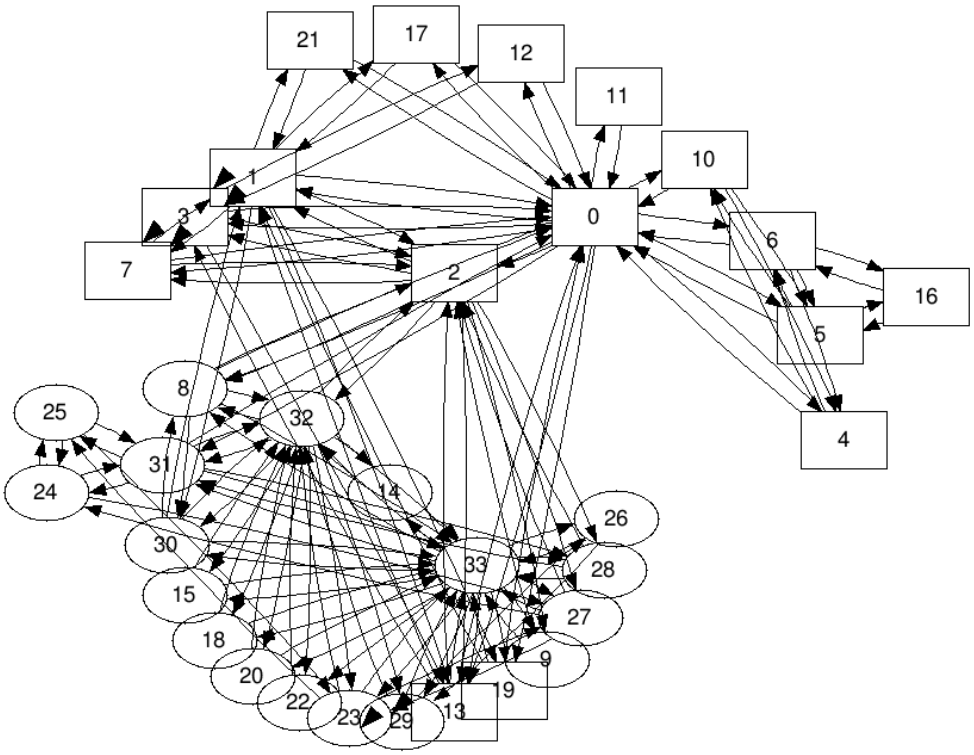
Testování

Tato kapitola se věnuje testování funkčnosti implementovaných algoritmů a jejich výsledcích na testovacích datech.

K ověření základní funkčnosti byla použita testovací data vytvořená v několika verzích v testovacích třídách obou algoritmů. Výstupy těchto testů zde nebudou diskutovány, protože v nich používaná data byla vytvořena pouze k ověření základní funkčnosti a kompletnosti očekávaných výstupů.

Pro srovnání výsledků obou algoritmů byl použit dataset ilustrovaný na obrázku 5.1, který vyjadřuje pomocí grafové struktury přátelské vazby mezi 34 členy karate klubu na americké univerzitě [14].

Tento klub byl následkem vnitřních sporů rozdělen na dvě skupiny, z nichž jedna klub opustila a založila si svůj vlastní [10]. Očekávaným výsledkem shlukování těchto dat jsou tedy dva shluky obsahující 16 a 18 bodů, které reprezentují dvě skupiny členů v klubu.



Obrázek 5.1: Karate dataset. Tvary uzlů vyjadřují jejich příslušnost do jedné ze dvou skupin. S tímto rozdělením byly při testování provnávány výstupy algoritmů.

5.1 Chinese Whispers

Jeden z výsledků běhu tohoto algoritmu je patrný na obrázku 5.2. V tomto případě se podařilo přesně rozdělit data do dvou komunit, které odpovídají rozdělení klubu diskutovanému na začátku kapitoly 5.

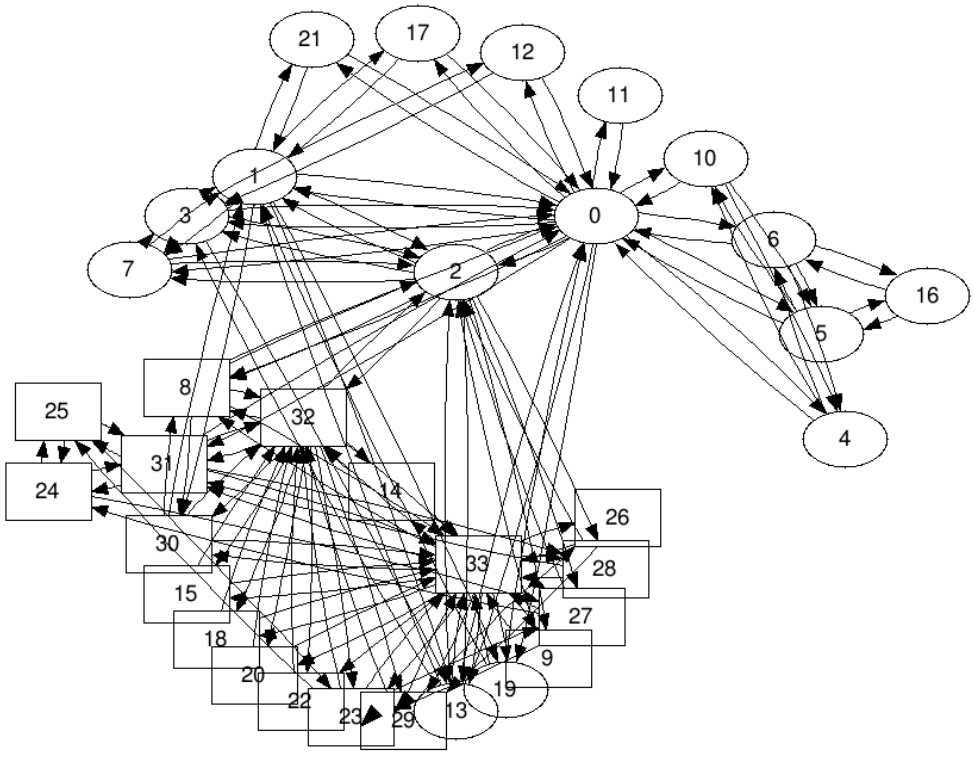
Z dalších výsledků je ale patrné, jak v tomto algoritmu hraje roli také náhoda. V případě běhu ilustrovaného na obrázku 5.3 se nepodařilo správně zařadit uzel číslo 2, který by dle referenčních dat měl patřit do druhého shluku [10].

Ve třetím běhu (obrázek 5.4) se dokonce změnil počet nalezených shluků, které tentokrát byly 3. Je patrné, že ačkoliv reálně byla data rozdělena pouze na dvě skupiny, třetí skupina, nalezená algoritmem CW, je v datech skutečně přítomna a rozdělení do 3 shluků je tedy „smysluplné“.

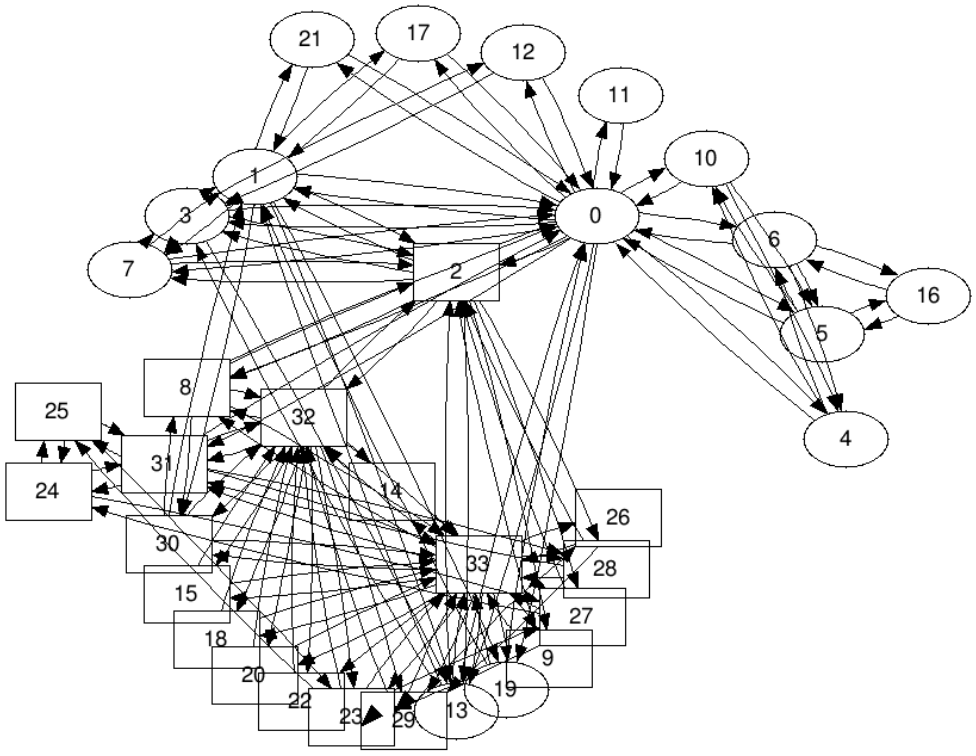
Při některých bězích nastala i situace, kdy byly oba hlavní shluky sloučeny dohromady a výsledkem byl jediný shluk obsahující všechny body. Tabulka 5.1 obsahuje procentuální zastoupení počtu výsledných shluků při testovacích bězích algoritmu.

Počet shluků	**Poměr běhů s tímto výsledkem**
1 shluk	5%
2 shluky	40%
3 shluky	55%

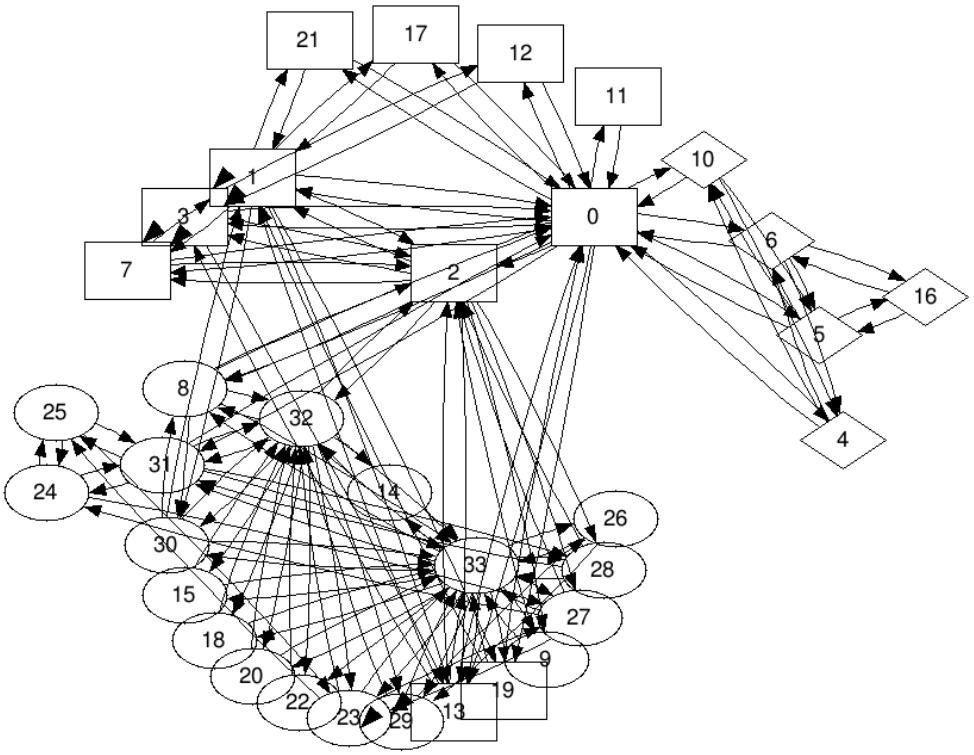
Tabulka 5.1: Četnost výskytů různých počtů shluků při testovacích bězích algoritmu CW.



Obrázek 5.2: První výsledek běhu Chinese Whispers. Tvary uzlů vyjadřují jejich příslušnost do jedné ze dvou skupin. Při porovnání s referenčním rozdělením na obrázku 5.1 je patrné, že v tomto případě je výsledek zcela přesný.



Obrázek 5.3: Druhý výsledek běhu Chinese Whispers. Tvary uzlů vyjadřují jejich příslušnost do jedné ze dvou skupin. V tomto případě náhodné vlivy v algoritmu CW zapříčinily, že výsledek je jiný než v případě běhu ilustrovaného na obrázku 5.2 a lze zde identifikovat chybně zařazený uzel číslo 2.

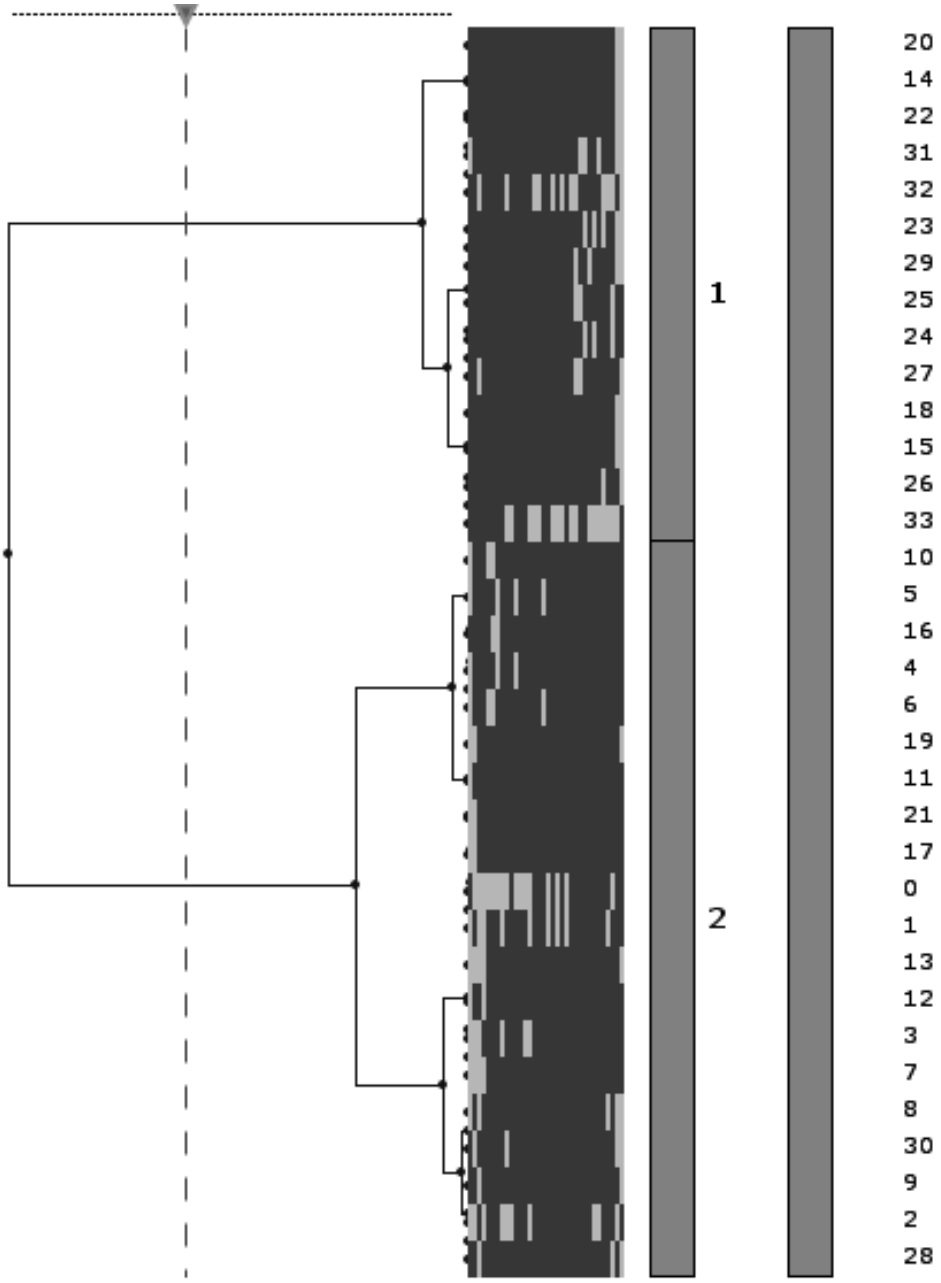


Obrázek 5.4: Třetí výsledek běhu Chinese Whispers. Tvary uzlů vyjadřují jejich příslušnost do jedné ze tří skupin. Při tomto běhu byly nalezeny tři shluky, což sice neodpovídá reálnému známému stavu dat, ale z vizualizace je patrné, že tyto shluky jsou „smysluplně“ vytvořené.

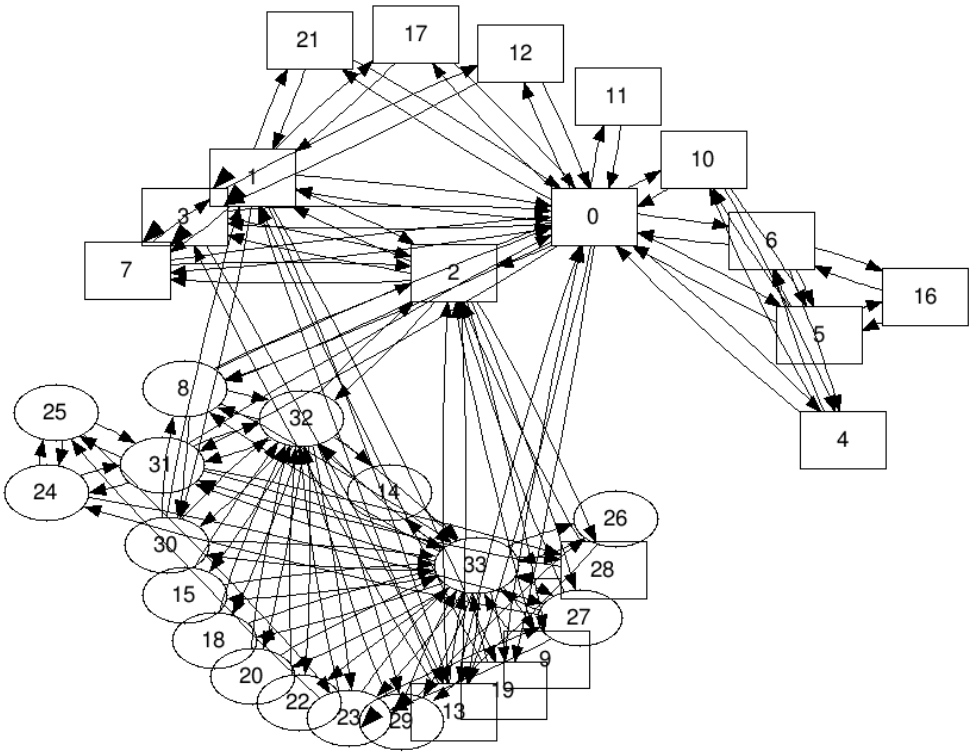
5.2 Fast Community

Výstupem tohoto algoritmu je dendrogram, který je posléze nutné pro získání shluků protnout v určité hladině. Postup řezu dendrogramu je popsán v části 1.2.2 o aglomerativním hierarchickém shlukování. Výsledek běhu FC je znázorněn na obrázku 5.5 v podobě dendrogramu exportovaného z Clueminer. Z tohoto výstupu je patrné, že se podařilo správně identifikovat a zvolit nejvýhodnější počet shluků, kterým jsou dva. V jiných testovacích běžích FC se tento fakt neměnil, vždy hodnoty poukazovaly na to, že nejkvalitnější počet shluků v těchto datech jsou dva.

Finální výsledek shlukování po řezu dendrogramu je na obrázku 5.6. Nesprávně zařazené zde byly uzly 9 a 28, pravděpodobně kvůli sousednosti s uzly číslo 2, 13 a 19. Tento výsledek se ukázal být stabilní, ani při opakovaném testování se výstup z algoritmu neměnil.



Obrázek 5.5: Dendrogram výsledku běhu Fast Community. Čísla na pravé straně označují původní vstupní data, přerušovaná čára nalevo pak hladinu, ve které je nejvýhodnější provést řez.



Obrázek 5.6: Výsledek běhu Fast Community. Tvary uzlů vyjadřují jejich příslušnost do jedné ze dvou skupin. Nesprávně zařazeny v tomto případě byly uzly číslo 9 a 28.

5.3 Srovnání algoritmů

V této sekci je uvedeno porovnání výsledků běhů obou algoritmů. Hodnoty uvedené v tabulce 5.2 jsou průměrem pro 20 běhů každého algoritmu. Srovnání bylo prováděno za použití datasetu reprezentujícího členy karate klubu, který byl popsán na začátku kapitoly 5.

Algoritmus	**CW**	**FC**
Průměrná doba běhu [ms]	121	559
Průměrná přesnost	90%[8]	94%
Nejvyšší naměřená přesnost	100%	94%
Nejnižší naměřená přesnost	68%	94%

Tabulka 5.2: Srovnání různých metrik meřených při testování algoritmů. Přesnost zde značí procentuální poměr počtu správně zařazených bodů vůči celkovému počtu bodů v datasetu.

[8] V případech, kdy byl nalezen třetí shluk zobrazený na obrázku 5.4, byl při výpočtu přesnosti počítán jako správně rozdělený. Vzhledem k tomu, že v tomto případě šlo o specifický graf a shluk byl „smysluplný“, by jinak byl algoritmus CW v porovnání znevýhodněn.

Závěr

Cílem práce bylo implementovat algoritmy Chinese Whispers a Fast Community, integrovat je do platformy Clueminer jako moduly, otestovat jejich funkcionalitu a navzájem je krátce srovnat.

Samotné implementaci předcházela analýza platformy, pomocí které byly zjištěny možnosti integrace a požadavky na nové moduly. Následně byla provedena analýza algoritmů, při níž byly identifikovány klíčové nároky na datové struktury, které bylo nutné použít při implementaci. Po vytvoření modulů následovalo testování jejich funkcionality a přesnosti jimi prováděného shlukování na srovnávacích datech.

Výsledky ukazují, že oba algoritmy pracují dle očekávání a jsou schopny správně identifikovat shluky v grafech. Celkově úspěšnějším se při testování ukázal být algoritmus Fast Community, který dosáhl na srovnávacích datech stabilní úspěšnosti 94%. Algoritmus Chinese Whispers však v některých případech dokázal provést shlukování tak, že shluky přesně odpovídaly známému reálnému rozdělení vstupních dat (tedy úspěšnosti 100%). Zároveň je ale nutno dodat, že tento algoritmus obsahuje element náhody, takže výsledky se mohou při jednotlivých bězích vzájemně výrazně lišit, což se potvrdilo i při testování.

Budoucí práce

Implementace v této práci byla omezena na zpracování neorientovaných grafů s neohodnocenými hranami, rozšíření na obecné grafy je proto předmětem budoucích vylepšení vytvořených modulů, které by umožňovaly zpracování širšího výběru grafových dat.

Pro shlukování dat v podobě grafů existuje celá řada různých algoritmů, které se mohou hodit každý pro jiný typ dat. Přidání nových modulů s dalšími algoritmy by proto mohlo být náplní budoucího pokračování v této práci.

Na testovací část této práce by pak bylo možné navázat porovnáním implementovaných algoritmů na více různorodých typech dat, aby byly důkladněji prozkoumány jejich přednosti a nevýhody.

Literatura

[1] Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.: From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, ročník 17, č. 3, 1996: s. 37–54. Dostupné z: http://dblp.uni-trier.de/db/journals/aim/aim17.html

[2] Agrawal, R.; Imielinski, T.; Swami, A.: Database mining : A performance perspective. *IEEE Transansaction on Knowledge and Data Engineering : Special issue on learning and discovery in knowledge-based databases*, ročník 5, č. 6, December 1993: s. 914–925. Dostupné z: http://www.bibsonomy.org/bibtex/2f56462d3f6713d94846a29c12f26b390/stumme

[3] Jain, A. K.: Data Clustering : 50 Years Beyond K-Means. *Pattern Recognition Letters*, 2010. Dostupné z: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.18.2720&rep=rep1&type=pdf

[4] Aggarwal, C. C.; Reddy, C. K.: *Data clustering: algorithms and applications*. CRC Press, 2013.

[5] Bramer, M.: *Principles of Data Mining*. Springer, 2007, ISBN 1-84628-765-0.

[6] Mishra, R.; Shukla, S.; Arora, D. D.; aj.: An Effective Comparison of Graph Clustering Algorithms via Random Graphs. *International Journal of Computer Applications*, ročník 22, č. 1, May 2011: s. 22–27.

[7] Kumar, V.: An Introduction to Cluster Analysis for Data Mining. February 2000. Dostupné z: http://www.cs.umn.edu/~han/dmclass/cluster_survey_10_02_00.pdf

[8] Barton, T.: *Cluster analysis of cell profile responses*. Diplomová práce, Czech Technical University in Prague, 2011. Dostupné

z: https://dip.felk.cvut.cz/browse/details.php?f=F3&d=K13136&y=2011&a=bartoto2&t=dipl

[9] Fowler, M.: Reducing coupling. *IEEE Software*, ročník 18, č. 4, 2001: s. 102–104.

[10] Newman, M.: Fast algorithm for detecting community structure in networks. *Physical Review E*, ročník 69, September 2003. Dostupné z: http://arxiv.org/abs/cond-mat/0309508;http://www.bibsonomy.org/bibtex/256de7e6d214faebdbf2f2ef0fce09d7d/marianne

[11] Biemann, C.: Chinese Whispers - an Efficient Graph Clustering Algorithm and its Application to Natural Language Processing Problems. In *Proceedings of TextGraphs: the Second Workshop on Graph Based Methods for Natural Language Processing*, New York City, USA, 2006, s. 73–80. Dostupné z: http://www.bibsonomy.org/bibtex/210699e6fa5efdf7b8b10d1b052ae54be/fluctuator

[12] Ester, M.; Kriegel, H.-P.; Sander, J.; aj.: A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, ročník 96, 1996, s. 226–231.

[13] van Dongen, S.: *A Cluster algorithm for graphs*. Dizertační práce, Centrum voor Wiskunde en Informatica, 2000. Dostupné z: http://www.bibsonomy.org/bibtex/2d7d305114e2d45acdea509c28769881c/jullybobble

[14] Zachary, W.: An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, ročník 33, 1977: s. 452–473. Dostupné z: http://www.bibsonomy.org/bibtex/20614639786a007bef0f79204b3e77bbf/tfalk

Příloha A

Seznam použitých termínů

API Application Programming Interface – Programátorské rozhraní pro poskytování nebo využívání funkcionality

Clustering Seskupování podobných objektů ve vzorku dat

CW Algoritmus Chinese Whispers

Data mining Získávání informací z velkých objemů dat

Dataset Sada zpracovávaných dat

DBSCAN Shlukovací algoritmus

FC Algoritmus Fast Community

IDE Integrated Development Environment – Programátorské vývojové prostředí

MCL Markov Cluster Algorithm – Shlukovací algoritmus pro grafy

NLP Natural Language Processing – Oblast počítačové vědy zabývající se zpracováním lidského jazyka

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `clueminer` ..... adresář s NetBeans projektem Clueminer
- `data` ..... adresář s testovacími daty
  - `*.pairs.txt` ..... dataset ve formě dvojic uzlů propojených hranou
  - `*.matrix.txt` ..... dataset převedená do formy matice
- `src` ..... adresář se zdrojovým kódem práce
  - `thesis.tex` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... adresář s textem práce
  - `thesis.pdf` ..... text práce ve formátu PDF