Paralelni algoritmi za problem grupiranja podataka

Čabraja, Anto

Master's thesis / Diplomski rad

2014

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:660082

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-08-10**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Anto Čabraja

PARALELNI ALGORITMI ZA PROBLEM GRUPIRANJA PODATAKA

Diplomski rad

Voditelj rada:
doc. dr. sc. Goranka Nogo

Zagreb, srpanj 2014.

Ovaj diplomski rad obranjen je dana ____________________ pred ispitnim povjerenstvom u sastavu:

1. ____________________________, predsjednik
2. ____________________________, član
3. ____________________________, član

Povjerenstvo je rad ocijenilo ocjenom _______________.

Potpisi članova povjerenstva:

1. ____________________________
2. ____________________________
3. ____________________________

Sadržaj

Sadržaj iii

Uvod 1
Primjena i važnost grupiranja podataka 1
Pregled rada 2

1 Modeliranje problema grupiranja 5
1.1 Osnovni pojmovi 5
1.2 Matematičko modeliranje problema 6
1.3 Metode razvoja algoritama za grupiranje 11
1.4 Upravljanje podacima 14

2 Evolucijske metode 17
2.1 Meta-heuristike 17
2.2 Prirodom inspirirani algoritmi 19
2.3 Genetski algoritam 21

3 Sekvencijalni algoritmi za problem grupiranja 29
3.1 Algoritam k-sredina 29
3.2 Evolucijski algoritam 34

4 Tehnike paralelizacije 43
4.1 Paralelni procesi 43
4.2 Osnovni pojmovi tehnologije MPI 44
4.3 Topologija 46

5 Konstrukcija paralelnih algoritama 49
5.1 Paralelizacija algoritma k-sredina 49
5.2 Paralelizacija evolucijskog algoritma 56

Bibliografija 61

Uvod

Primjena i važnost grupiranja podataka

Analiza velike količine podataka jedan je od najbitnijih problema modernog vremena. Razvojem velikih servisa kao što su Google, Facebook, Twiter, Amazone započelo je prikupljanje podataka o korisnicima. Banke kao vodeći predstavnik usluga prema korisnicima čuvaju sve podatke kako bi analizirali i pružili najbolju uslugu svojim klijentima. Servisi elektroničke pošte također rade s velikim količinama podataka. Podaci se čuvaju u velikim distribuiranim bazama podataka čiji red veličine doseže 10^5 terabajta. Postavlja se pitanje kako analizirati tako velike količine podataka i iz njih dobiti bitne zaključke. Problem je još teži kada znamo da podaci koji se spremaju stižu u prirodnom poretku, dakle u bazama su spremljeni bez organizacije. Prvi korak u analizi tako velike baze neuređenih podataka je grupiranje. Grupiranje podataka omogućuje separaciju podataka na manje cjeline koje imaju slična svojstva i na njima se mogu upotrijebiti iste metode analize. Osim što grupiranje podataka otkriva sličnosti među podacima, dobivamo i važne statističke zaključke kao što su: koliko grupa postoji, koja grupa ima koliko predstavnika, kakva su svojstva koja reprezentiraju grupe. Sljedeći motivacijski primjeri najbolje opisuju važnost grupiranja.

Primjer 0.0.1. *Promotrimo slučaj banke koja ima veliki broj klijenata. Za svakog klijenta analitičari moraju procijeniti rizik kreditiranja i zaduženja. Međutim, kada bi imali model koji će, na osnovu klijenata za koje su već izračunati modeli rizika, odlučivati za novog klijenta u koju kategoriju rizika pripada uvelike bi uštedili posao. Osim vremenske uštede takav model bi smanjio rizik pogreške jer se bazira na provjerenim rezultatima prijašnjih analiza. Model koji nam je potreban je upravo grupiranje podataka. Naime, ako bi adekvatno pronašli grupe korisnika koji imaju slična svojstva lako bi novom klijentu pridružili grupu koja ga najbolje opisuje. Zatim bi pogledali kakvi su modeli rizika za klijente dotične grupe te na osnovu zaključaka kreirali predikciju za novog klijenta.*

Primjer 0.0.2. *Promotrimo servis za upravljanje elektroničkom poštom. Servis prikuplja poštu od svih korisnika i pruža skup usluga. Ignoriranje lažne pošte je jedna od takvih usluga. Problem je kako prepoznati poštu kao lažnu. Grupiranjem podataka možemo prepoznati grupu poruka koje su lažne jer gotovo uvijek imaju neka zajednička svojstva.*

Nakon toga za novu poruku provjerimo kojoj grupi pripada. Druga informacija koja je dotičnim servisima bitna je kakva je svrha korisnikove pošte. Koristi li korisnik poštu za slobodno vrijeme, posao ili obitelj. S obzirom na prikupljene informacije servis nudi dodatne usluge. Primijetimo da kod određivanja karakteristike dotičnih dokumenata možemo grupirati poruke prema nekim značajkama koje obilježavaju poslovne dokumente ili neki drugi tip dokumenata.

Primjer 0.0.3. *Promotrimo problem slaganja kutija u trodimenzionalnom Euklidskom prostoru. Kutije treba složiti u određeni prostor pravilnog oblika, tako da što više kutija stane u kapacitet prostora. Kako bi kutije mogli slagati potrebno je odrediti tip kutije. Naime, jako je bitno popunjavati prazne prostore i u svakom trenutku, kada je poznata dimenzija praznine, uzeti određenu kutiju koja najbolje odgovara praznini. Kako je broj kutija jako velik vremenski je zahtijevno svaki puta pogledati sve kutije i od njih odabrati najbolju. Rješenje danog problema za izbor kutije je grupirati kutije prema sličnim dimenzijama. Unaprijed je poznato da imamo šest tipova kutija te je ovaj problem adekvatan za rješavanje metodama grupiranja.*

Iz navedenih primjera lako je zaključiti da prilikom bilo koje analize prvi korak je grupiranje podataka i otkrivanje zajedničkih obilježja. Također vidimo kako se problem grupiranja prirodno javlja u svim područjima gdje su velike količine podataka. Cilj rada je opisati nekoliko algoritama za grupiranje podataka. Kao primjeri za testiranje poslužit će nam drugi i treći navedeni motivacijski primjer.

Pregled rada

U prvom poglavlju uvodimo osnovne pojmove kojima opisujemo problem grupiranja. Nakon osnovnih pojmova matematički formuliramo problem i formalno definiramo problem grupiranja. Formalna definicija omogućuje nam da definiramo funkcije sličnosti i razlike među podacima. Kako želimo uvesti određene metrike nad podacima, same podatke je potrebno kompaktno reprezentirati u nekom vektorskom prostoru. Posebna cjelina u modeliranju posvećena je upravo problemu konstrukcije i kompaktnog zapisivanja podataka. Definirane pojmove sada možemo upotrijebiti za konstrukciju modela rješenja. U radu predstavljamo nekoliko modela i načina konstrukcije algoritama. Prvo opisujemo osnovne modele, a to su hijerarhijsko i particijsko grupiranje.

Drugo poglavlje posvećeno je proučavanju evolucijskih metoda. Predstavljamo model za rješavanje problema grupiranja meta-heurističkim pristupom problemu.
U trećem poglavlju konstrukcijom sekvencijalnih algoritama na motivacijskom primjeru predstavljamo vremensku analizu konkretne implementacije prije definiranih modela. Nakon vremenske analize i dobivenih zaključaka, pokušat ćemo poboljšati dobivene rezultate.

Poboljšanje ćemo postići paralelizacijom implementiranih algoritama. Naime, današnja računala omogućuju vrlo moćnu paralelizaciju i istovremeno izvršavanje procesa. U četvrtom poglavlju uvodimo osnovne pojmove najvažnije tehnologije za paralelizaciju i topologiju paralelne mreže procesora. Nekim od navedenih paralelnih koncepata paralelizirat ćemo postojeće algoritme te s obzirom na vremensku analizu pokušati stvoriti dodatni model kao kombinaciju najboljih dijelova implementiranih algoritama.

Notacija

U radu koristimo određenu notaciju kako bi definirali i objasnili pojmove. U nastavku definiramo pojmove koji će u radu biti korišteni kao poznati.

Vektorska notacija:

Neka je sa S označen vektor od n elemenata. Sa $S(i)$ označavamo i-ti element danog vektora. Pretpostavljamo da vrijedi $i \in [1, n]$ i to ne provjeravamo. Neka su zadane vrijednosti a i b takve da vrijedi $a, b \in [1, n]$ i $a < b$, tada definiramo $S(: a)$, $S(a : b)$ i $S(b :)$ redom kao skup svih elemenata od početka vektora do pozicije a, skup svih elemenata između a i b te skup svih elemenata od pozicije b do zadnjeg elementa.
Neka su dana dva vektora S_1 i S_2 jednake duljine, provjera $S_1 = S_2$ označava jednakost po komponentama danih vektora.

O notacija:

Neka su $f, g : \mathbb{N} \to R^+$ dvije funkcije. Kažemo da je funkcija g asimptotska gornja međa funkcije f ako postoji $c > 0$ i $n_0 \in \mathbb{N}$ tako da za svaki $n > n_0$ vrijedi $f(n) \leq c \cdot g(n)$. Navedeno svojstvo označavamo s $f(n) = O(g(n))$.
Kada u radu spominjemo navedenu notaciju to će biti u kontekstu složenosti algoritma. Kada bude pisalo da je algoritam složenosti $O(f(n))$ to će značiti da za ulaz duljine n algoritam ne treba više od $f(n)$ jednostavnih operacija kako bi pronašao rješenje.

Poglavlje 1

Modeliranje problema grupiranja

1.1 Osnovni pojmovi

Kako bi u daljnjem razmatranju bilo jednostavnije objašnjavati strukture i same implementacije algoritama potrebno je problem grupiranja reprezentirati osnovnim pojmovima. U nastavku ćemo formalno definirati sve komponente od kojih se problem grupiranja sastoji.

Definicija 1.1.1. ***Uzorak*** *je apstraktna struktura podataka koja reprezentira stvarne podatke s kojima raspolaže algoritam za grupiranje.*

Definicija 1.1.2. ***Svojstvo*** *je vrijednost ili struktura koja predstavlja jednu značajku danog podatka unutar uzorka.*

Definicija 1.1.3. ***Udaljenost*** *između uzoraka definiramo kao funkciju $f : D \to \mathbb{R}$, gdje je D skup svojstava danih uzoraka.*

Definicija 1.1.4. *Za uzorke kažemo da su* **blizu** *jedan drugome ako je njihova udaljenost manja od unaprijed zadane veličine.*

Definicija 1.1.5. ***Klaster*** *je skup uzoraka koji su u prostoru podataka blizu. Ako su uzorci identični onda je njihova udaljenost uvijek* 0*.*

Definicija 1.1.6. ***Jedinstveno grupiranje*** *je postupak grupiranja kada svaki uzorak pripada jednom i samo jednom klasteru.*

Definicija 1.1.7. ***Nejasno ili nejedinstveno grupiranje*** *je postupak grupiranja gdje jedan uzorak može biti u više klastera.*

Napomena 1.1.8. *U radu ćemo promatrati* ***jedinstveno grupiranje*** *tako da će sve daljnje definicije i modeliranja pretpostavljati da želimo dobiti disjunktne klastere.*

1.2 Matematičko modeliranje problema

Definicija grupiranja podataka nije jedinstvena. U literaturi se na različite načine pokušava opisati ovaj postupak. Neki od pokušaja opisne definicije su:

1. *Grupiranje podataka je postupak otkrivanja homogenih*[1] *grupa uzoraka unutar skupa svih danih uzoraka.*

2. *Grupiranje podataka je postupak određivanja sličnih uzoraka te njihovo svrstavanje u isti klaster.*

Kako bi modelirali problem neće nam biti dovoljne opisne definicije. U ovom slučaju opisne definicije mogu poslužiti samo kao intuicija o čemu se zapravo radi kada govorimo o grupiranju. U nastavku ćemo pomoću definiranih pojmova u poglavlju 1.1 matematički opisati problem grupiranja podataka.

Neka je $\mathbf{U} = \{U_1, U_2, \cdots, U_n\}$ skup od n uzoraka te neka je $U_i = (s_1, s_2, \cdots, s_d)$ reprezentiran d-dimenzionalnim vektorom gdje s_i predstavlja jedno svojstvo. Ovako definiran $\mathbf{U}$ moguće je reprezentirati kao matricu

$$\mathbf{S}_{d\times n} = \begin{pmatrix} s_{1,1} & s_{1,2} & \cdots & \cdots & s_{1,n} \\ s_{2,1} & s_{2,2} & \cdots & \cdots & s_{2,n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ s_{d,1} & s_{d,2} & \cdots & \cdots & s_{d,n} \end{pmatrix}. \tag{1.1}$$

Svaki stupac matrice $\mathbf{S}$ predstavlja jedan uzorak iz danog skupa $\mathbf{U}$. Iz definicije 1.1.6 te iz navedenog formalnog zapisa dajemo formalnu definiciju problema grupiranja.

Definicija 1.2.1. ***Skup od k klastera*** $\mathbf{C} = \{C_1, C_2, ..., C_k\}$ *je skup sa sljedećim svojstvima:*

- $C_i \neq \emptyset$
- $C_i \cap C_j = \emptyset$, $\forall i, j \; i \neq j$
- $\cup_{i=1}^{k} C_i = \mathbf{U}$.

Napomena 1.2.2. *U terminima matrice* $\mathbf{S}$ *to znači da se svaki* C_i *zapravo sastoji od stupaca matrice* $\mathbf{S}$.

Definicija 1.2.3. ***Problem grupiranja*** *u skup od k klastera* $\mathbf{C}$ *je ekvivalentan problemu da* $\forall c, c' \in C_i$ *udaljenost od c do c′ je manja od udaljenosti c do bilo kojeg drugog* $c'' \in C_j$ $j \neq i$.

[1] podaci koji se ne mogu smisleno separirati

Zapravo, problem grupiranja je pronalazak najpogodnije particije za **C** u skupu svih mogućih particija. Prema napomeni 1.2.2 lako se zaključi da se problem grupiranja svodi na problem raspodjele n stupaca matrice **S** u k skupova. Uvedena matematička notacija za problem grupiranja omogućuje da postavimo model za rješavanje, koji je u kasnijem razmatranju pogodan za modeliranje i implementaciju.
Neka je $\mathcal{C} = \{\mathbf{C}^1, \mathbf{C}^2, ..., \mathbf{C}^{N(n,k)}\}$ skup svih mogućih rješenja danog problema grupiranja, gdje je

$$N(n,k) = \frac{1}{k!}\sum_{i=1}^{k}(-1)^i\binom{k}{i}(k-i)^n \tag{1.2}$$

broj mogućih rješenja za raspodjelu n uzoraka u k klastera. Rješenje problema svodi se na optimizacijski problem

$$\underset{\mathbf{C}\in\mathcal{C}}{optimiziraj}\; f(\mathbf{S}_{d\times n}, \mathbf{C}) \tag{1.3}$$

gdje je f funkcija dobrote rješenja **C**. Funkcija dobrote je funkcija koja mjeri kvalitetu rješenja za dani problem. Ovisno o tome kako je zadana, može se gledati problem maksimizacije ili minimizacije. Gotovo uvijek njome se određuje jedan od dva važna parametra:

- *koliko su blizu uzorci u danom klasteru*
- *koliko su blizu dva disjunktna klastera.*

Ako promatramo udaljenosti uzoraka unutar klastera, onda nam se problem optimizacije 1.3 svodi na problem minimizacije funkcije f, jer želimo postići što manju udaljenost uzoraka unutar jednog klastera. Međutim ako želimo postići da su nam klasteri međusobno disjunktni i da granica disjunkcije bude čvrsto definirana moramo promatrati udaljenosti među klasterima. U ovom slučaju potrebno je maksimizirati problem to jest tražiti za koju particiju će vrijednost funkcije f biti najveća.
Konačno, sada znamo uzorke reprezentirati kao n-dimenzionalne vektore, također znamo definirati klaster kao skup vektora. Time smo postavili model za određivanje kvalitete određenog klastera. Preostali posao je pronaći konkretnu funkciju f koja će na adekvatan način reprezentirati udaljenost između uzoraka. Vrlo je važno definirati od kojih se komponenti uzorak sastoji. Osnovna podjela uzoraka je na *numeričke* i *kategoričke*. Numerički uzorak je onaj uzorak čije su sve vrijednosti numeričkog tipa, dok je kategorički onaj uzorak čije vrijednosti poprimaju vrijednosti nekih kategorija. Više o mogućnostima izgleda svojstava uzorka biti će rečeno u cjelini 1.4. U ovom trenutku za definiranje funkcije dobrote potrebno je samo imati u vidu da podaci ne moraju biti jednostavni, ali i dalje se od funkcije dobrote očekuje da na adekvatan način odredi udaljenost između dva uzorka.

Uzorak s numeričkim značajkama

Ukoliko su nam uzorci takvi da ih možemo predstaviti kao vektor numeričkih podataka tada ih možemo smjestiti u vektorski prostor te na njima upotrijebiti neku od standardnih vektorskih metrika. Za problem grupiranja podataka u praksi najčešće se koristi Euklidska ili neka od p-metrika [6]. Euklidska metrika daleko je najpopularnija i glavni je predstavnik metrike koje mjere različitost među uzorcima. Drugim riječima za Euklidsku metriku vrijedi da su uzorci više različiti što je vrijednost norme veća.
Neka su S_1 i S_2 dva uzorka s n značajki, tada udaljenost d između dva uzorka u Euklidskoj metrici računamo kao:

$$d(S_1, S_2) = \sqrt{\sum_{i=1}^{n}(s_{i,1} - s_{i,2})^2} \tag{1.4}$$

gdje su $s_{i,1}$ i $s_{i,2}$ značajke u uzorcima S_1 i S_2. U kreiranju rješenja s numeričkim podacima uvijek ćemo koristiti Euklidsku metriku s malim izmjenama ovisno o vrsti problema. Međutim u praksi se vrlo često pojavljuje i metrika koja koristi svojstva kovarijacijske matrice uzoraka [4].

$$d(S_1, S_2) = (S_1 - S_2)^T \Sigma^{-1} (S_1 - S_2) \tag{1.5}$$

S_1 i S_2 su uzorci, a Σ kovarijacijska matrica uzoraka. Za ovu metriku također vrijedi da su podaci više različiti što je vrijednost d veća. Također postoji veza između Euklidske metrike i metrike s kovarijacijskom matricom što je detaljno objašnjeno u [4]. Samo ćemo reći da ako je Σ dijagonalna matrica onda se pripadna udaljenost zove normalizirana Euklidska udaljenost.
Osim metrike koje mjere razlike među podacima, postoje i metrike koje mjere sličnost. Metrike koje mjere sličnost često su bazirane na određivanju kuta između dva uzorka u vektorskom prostoru. U [4] objašnjene su neke od popularnijih metrika ovog tipa.

Uzorak s kategoričkim značajkama

Individualna usporedba dva uzorka s kategoričkim značajkama vrlo često nema važnost i jedina povratna informacija je koliko značajki u uzorcima ima jednaku vrijednost. Ako ipak želimo postići da usporedbom dvije kategorije možemo zaključiti koliko su značajke u uzorku slične, to jest blizu, moramo svakoj značajki pridružiti težine i definirati operaciju razlike. Ako bolje pogledamo, tim postupkom zapravo smo kategorije pretvorili u numeričke vrijednosti s posebno definiranom funkcijom razlike. Promotrimo sljedeći primjer.

Primjer 1.2.4. *Neka nam je na raspolaganju skup podataka o cvijeću te neka nam je svaki cvijet zadan kao vektor s tri kategoričke komponente. Označimo sa* S *i* S' *dva cvijeta iz*

skupa zadanih uzoraka.

$$S = (crven, Amerika, iglicasto)$$
$$S' = (plav, Europa, iglicasto)$$

Lako vidimo da jedini zaključak kojeg iz ovih podataka možemo dobiti jest da se na prve dvije komponente razlikuju, dok je na trećoj vrijednost kategorije ista. Ako bi na primjer htjeli kategorijama pridružiti vrijednost morali bi definirati što znači da su Amerika i Europa različite i koliko nam je to bitno svojstvo. Važnost svojstva ima ključnu ulogu i vrlo je teško empirijski procijeniti kakve numeričke podatke dodjeliti kategorijama.

U praksi nismo uvijek u mogućnosti predvidjeti sve moguće kategoričke vrijednosti, pa samim time ne možemo svakoj kategoriji pridružiti numeričku vrijednost. Često korištena i popularna metoda za određivanje pripadnosti određenoj klasi, kada se radi o kategoričkim vrijednostima, je traženje udjela pojavljivanja određene kategoričke vrijednosti u promatranom klasteru. U tom slučaju ne promatramo udaljenost dva uzorka nego koliko u klasteru ima kategoričkih vrijednosti sličnih onima koje obilježavaju novi uzorak. Drugim riječima promatramo sličnost uzorka s klasterom. Neka je $\mathbf{C} = \{C_1, C_2, ...C_n\}$ skup od n klastera. Neka je $c = (c_1, c_2, ..., c_d)$ uzorak s d značajki istog tipa kao i uzorci iz zadanog skupa klastera. Definiramo funkciju sličnosti s kao

$$s(c, C_x) = \sum_{i=1}^{d} \frac{find(c_i, C_x)}{|C_x|} \tag{1.6}$$

gdje je $C_x \in \mathbf{C}$, funkcija $find$ vraća broj pojavljivanja značajke c_i na i-tom mjestu u svakom uzorku iz skupa C_x, a $|C_x|$ označava broj uzoraka u skupu C_x. Važno je naglasiti da se promatra samo i-to mjesto u svim značajkama. Postoji mogućnost da se vrijednost značajke, to jest kategorija, na i-tom i j-tom mjestu u uzorku podudaraju. Na primjer ako imamo uzorak koji predstavlja automobile te su dvije od značajki danog uzorka boja interijera i boja automobila. Očito dvije navedene značajke mogu poprimiti istu vrijednost, ali zapravo su potpuno disjunktne i kao takve ih treba promatrati. Konačno c pridružimo onom klasteru za koji funkcija s vrati najveću vrijednost.
Kada promatramo određeni skup podataka i kreiramo vektore uzoraka, u svakom trenutku znamo veličinu tog uzorka. Prilikom modeliranja problema sami određujemo skup značajki koji koristimo. Saznanje o broju značajki i njihovoj važnosti možemo upotrijebiti kako bi svakoj značajki odredili težinu. Naime, kako nisu sve značajke jednako bitne želimo postići da se utjecaj nekih značajki restringira, odnosno naglasi. S obzirom na navedeni zahtjev modificiramo funkciju s iz 1.6 te definiramo novu funkciju s_w

$$s_w(c, C_x) = \sum_{i=0}^{d} w_i \frac{find(c_i, C_x)}{|C_x|}. \tag{1.7}$$

Kao što vidimo svakoj značajki unutar uzorka pridružili smo težinu w_i. Važno svojstvo funkcije s_w je da omogućuje potpuno isključivanje nekih od značajki. Navedeno svojstvo u praksi često služi za empirijsko otkrivanje nebitnih značajki. Otkrivanje i uklanjanje značajki čije postojanje ne pridonosi poboljšanju rješenja uvelike smanjuje prostornu složenost što će detaljno biti obrađeno u 1.4.

Uzorak s hibridnim značajkama

Priroda problema za koje pokušavamo riješiti problem grupiranja često je složene strukture te je nemoguće podatke reprezentirati uzorcima sa značajkama samo jednog tipa. U takvom slučaju podatak se reprezentira s uzorkom čije značajke mogu biti numeričkog i kategoričkog tipa. Ovako definirana reprezentacija podataka povećava složenost i modeliranje funkcije dobrote, odnosno odluku o tome koliko su dva uzorka slična. Označimo sa $S = (s_1, s_2, ..., s_n)$ uzorak od n značajki numeričkog i kategoričkog tipa. Bez smanjenja općenitosti možemo pretpostaviti da su na prvih l komponenti numeričke značajke, a na ostalih $n - l$ kategoričke. Funkciju udaljenosti d_h između dva uzorka S_1 i S_2 s hibridnim značajkama definiramo kao

$$d_h(S_1, S_2) = d(S_1(: l), S_2(: l)) + \frac{1}{s_w(S_1(l + 1 :), C_{S_2})} \tag{1.8}$$

gdje je d funkcija definirana u 1.4, s_w definirana u 1.7, a C_{S_2} označava klaster u kojemu se nalazi S_2. Važno je napomenuti da S_1 ne mora nužno biti u klasteru dok S_2 mora. Dakle, ako dodajemo novi uzorak i želimo mu pronaći klaster to ćemo učiniti tako da ga stavimo kao prvi argument funkcije d_h. Kao što vidimo vrijednost funkcije s_w može biti jednaka nuli pa nam drugi sumand u definiciji funkcije d_h nije definiran. U praksi se eksperimentalno odredi kolika je važnost da uzorak nema niti jednu kategoričku značajku istu kao značajke unutar promatranog klastera te se funkcija d_h dodefinira s obzirom na slučaj kada je $s_w = 0$. Neka je λ eksperimentalno određena vrijednost. Tada se potpuna definicija funkcije d_h može zapisati

$$d_h(S_1, S_2) = \begin{cases} d(S_1(: l), S_2(: l)) + \frac{1}{s_w(S_1(l+1:), C_{S_2})} & s_w \neq 0 \\ \\ d(S_1(: l), S_2(: l)) + \lambda & s_w = 0. \end{cases} \tag{1.9}$$

Ovako definirana funkcija d_h je korektna. Naime, za sve parove uzoraka uvijek možemo odrediti vrijednost funkcije d_h jer su sve komponente dane funkcije dobro definirane. Funkcija d je zapravo standardna Euklidska metrika pa je kao takva definirana za sve vektore. Funkcija s_w je kompozicija funkcije $find$, funkcije zbrajanja, množenja i kardinalnog broja. Sve funkcije su dobro definirane [10] pa je i kompozicija dobro definirana. Primijetimo da se ovako definirana funkcija d_h, s pravilno postavljenim paremetrom λ može koristiti za sve slučajeve uzoraka: numeričke, kategoričke ili hibridne.

1.3 Metode razvoja algoritama za grupiranje

U postupku modeliranja rješenja za problem grupiranja koristi se više metoda. Metoda za rješavanje ima puno i znanstvenici intenzivno rade na pronalasku novih i objašnjenju kvalitete postojećih metoda. Danas je ovo područje izuzetno cijenjeno i svako novo saznanje može dovesti do revolucionarnog napretka u rješavanju izuzetno teških problema. Ipak, dvije metode su se pokazale kao općenitije i mogu poslužiti kao predložak za rješavanje većeg broja problema. Konkretno radi se o *hijerarhijskom* i *particijskom* pristupu rješavanja problema. Među ostalim metodama koje se danas ističu kao jako korisne pokazale su se: evolucijske metode, metode particioniranja grafa, metode bazirane na neuronskim mrežama te metode najbržeg silaska [8]. U ovom radu obradit ćemo navedene najpoznatije metode, a od ostalih metoda obradit ćemo evolucijske metode kojima je posvećena cjelina 2.

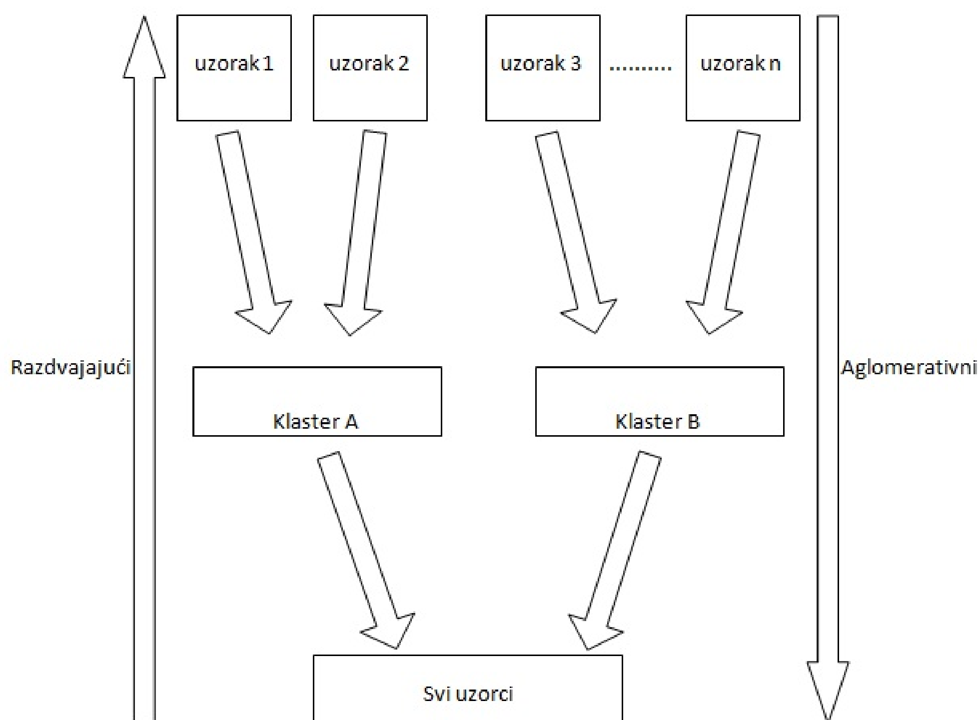
Hijerarijsko grupiranje

U hijerarhijskom grupiranju na početku procesa podaci nisu unaprijed smješteni u klastere nego postoje dva načina inicijalne raspodjele podataka:

1. svaki uzorak je klaster
2. svi uzorci su u jednom klasteru.

Postupak grupiranja može se odvijati u dva smjera ovisno o inicijalnoj raspodjeli podataka. Tako u slučaju 1. kada su uzorci svaki za sebe u posebnom klasteru postupak grupiranja nastavlja se spajanjem uzoraka s bliskim svojstvima u jedan klaster. U svakom sljedećem koraku ponavlja se postupak te se klasteri sa sličnim svojstvima spajaju u jedan novi klaster. Postupak prestaje kada odlučimo da nam je funkcija dobrote postigla željenu točnost ili smo postigli željeni broj klastera. Primijetimo ako postupak izvršimo bez uvjeta zaustavljanja dobit ćemo jedan klaster i zapravo podaci će biti u 2. inicijalnom stanju. Ako krećemo iz 2. inicijalnog stanja, to jest na početku su nam svi uzorci smješteni u jedan klaster, onda u svakom sljedećem koraku radimo separaciju uzoraka. Separacija uzoraka se provodi nad uzorcima koji se najviše razlikuju te se tako dobije više klastera. Postupak također nastavljamo dok ne dobijemo željenu točnost ili željeni broj klastera. Prva navedena metoda kada na početku imamo veliki broj klastera, točnije svaki uzorak je klaster, se naziva *spajajući postupak*[2]. Druga metoda se naziva *razdvajajući postupak*. Iz opisa hijerarhijske metode vidimo kako navedenim postupcima gradimo hijerarhiju te se navedeni postupak može prikazati kao stablo gdje su čvorovi klasteri, a veze povezuju klastere koji su nastali ili su spojeni iz čvorova djece. Tako kreirani graf kao struktura stabla naziva se *dendogram*. Na slici 1.1 dendogramom je prikazan postupak grupiranja.

[2]lat. aglomerativni postupak



Slika 1.1: Hijerarhijski proces grupiranja

Hijerarhijsko grupiranje ima dva važna svojstva:

- broj klastera ne mora biti unaprijed poznat
- klasteri su neovisni o početnim uvjetima.

Što se tiče vremenske i prostorne složenosti, ova metoda nije pogodna za velike skupove podataka. Vremenska složenost je $O(n^2 log\ n)$, dok je prostorna $O(n^2)$. Također, postupak grupiranja je statički, to jest ako neki uzorak pripada nekom klasteru ne postoji mogućnost da prijeđe u neki drugi klaster na istom nivou hijerarhije.

Particijsko grupiranje

Za postupak particijskog grupiranja uzorke inicijalno podijelimo u skup klastera. Inicijalna podjela u prvom koraku u praksi se najčešće izvodi na slučajan način. Međutim, ako su nam unaprijed poznata neka saznanja o uzorcima moguće je već u prvom koraku usmjeriti uzorke u prave klastere. Inicijalna podjela se izvodi tako da se na slučajan način

izabere k uzoraka i oni predstavljaju klastere; nakon toga se svi ostali uzorci rasporede u k klastera prema udaljenosti od izabranih predstavnika. Nakon inicijalne podjele za svaki klaster ponovo se odredi predstavnik klastera. Predstavnik je virtualni ili stvarni uzorak iz klastera koji predstavlja centar dotičnog klastera i s obzirom na njega se u klaster ubacuju novi uzorci. Uvodi se funkcija udaljenost uzoraka kako smo to definirali u cjelini 1.2 koja služi kao ocjena sličnosti. Postupak izvodimo iterativno tako da pogledamo sve uzorke i pridružimo ih predstavniku klastera kojem su najviše slični, to jest kojem su najbliži. Nakon postupka pridruživanja ponovo se bira novi predstavnik klastera i za sve podatke se pogleda sličnost s novim predstavnikom klastera. Particijsko grupiranje može se generalno prikazati pseudokodom:

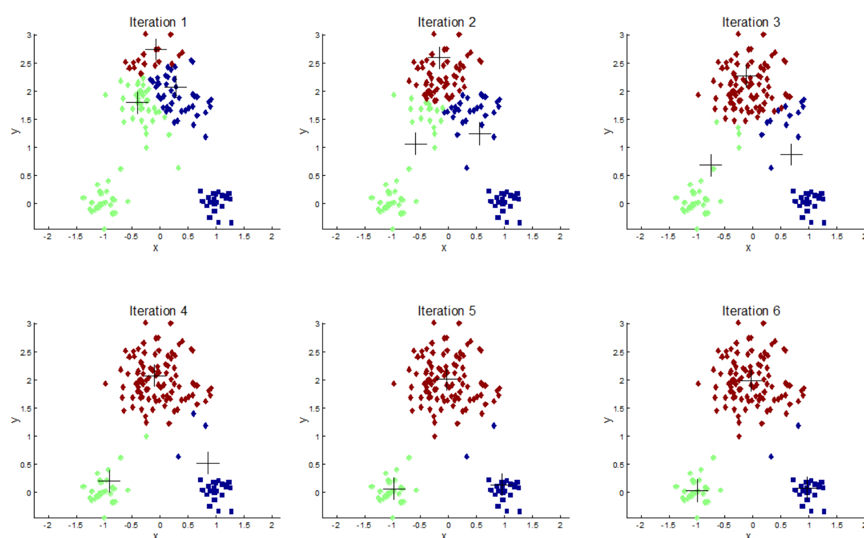
Algoritam 1: Particijsko klasteriranje

```
Na slučajan način izabrati k predstavnika m_i, i ∈ {1, 2, ..., k};
while nije zadovoljen uvjet zaustavljanja do
    for svaki uzorak Z_i iz skupa uzoraka do
        izračunaj veza(m_j, Z_i) za svaki m_i i svaki uzorak pridruži
        predstavniku klastera s najmanjom vrijednosti veza.;
    pronađi nove predstavnike klastera m_j kao srednju vrijednost
    svih elemenata klastera;
```

Funkcija veza je funkcija koja u pozadini koristi neku od definiranih funkcija u 1.2. Svaki uzorak će biti u klasteru od čijeg je predstavnika najmanje udaljen. Računanje novih predstavnika uzimanjem u obzir srednje vrijednosti ovisi o tipu problema i može se mijenjati s obzirom na zahtjeve.

Slika 1.2 prikazuje nekoliko iteracija particijske metode klasteriranja. Možemo primijetiti kako se u svakoj iteraciji promijenio predstavnik klastera te kako s obzirom na novog predstavnika uzorci mijenjaju klaster. Također, možemo primijetiti da predstavnik klastera ne mora uvijek biti element iz klastera nego to može biti virtualni uzorak koji najbolje opisuje promatrani klaster.

Kao što vidimo ovaj model grupiranja predviđa poznavanje broja klastera. Međutim složenost algoritma je znatno manja od prve navedene metode. Svaki prolazak treba n koraka, po jedan korak za svaki uzorak te je stoga složenost $O(n)$. U svakom koraku uzorci mogu promijeniti klaster i tako iterativnim postupkom do detalja profiniti particije. Ova metoda danas je najpopularnija i u praksi se najviše koristi. U kombinaciji s nekom od ostalih metoda, kao što su evolucijske metode, može varirati i broj klastera te tako uvjetno popraviti jedini nedostatak ove metode. Danas najpoznatiji algoritam za grupiranje, algoritam k-sredina, je predstavnik particijskog grupiranja. U ovom radu naglasak će biti upravo na ovoj metodi te ćemo za nju kreirati većinu algoritama i testiranja. Između ostalog opisati ćemo i navedeni algoritam k-sredina. Opis dvije najpoznatije metode daje širok pregled na



Slika 1.2: Particijski proces grupiranja

probleme koji se javljaju prilikom pokušaja grupiranja podataka. Veliki je izazov postaviti model koji će minimizirati sve nedostatke.

1.4 Upravljanje podacima

Podaci nad kojima treba provesti algoritam grupiranja gotovo uvijek su jako komplicirani. Također, nije poželjno stvoriti uzorke tako da se iskoriste sve značajke koje opisuju dani podatak. Naime, kada želimo spremiti podatke da bi ih mogli kasnije upotrijebiti u algoritmu potrebno je odabrati značajke koje su zajedničke svim podacima. Vrlo je bitno da značajki ima dovoljno da se podaci mogu separirati, ali da ih ne bude prevelik broj jer nam prostorna i vremenska složenost raste. Dakle, lako je zaključiti da moramo postići kompromis između dvije osnovne komponente. Kada bi spremili sve značajke danog podatka i koristili tako definirane uzorke, svaki uzorak bi nam bio reprezentiran s velikim brojem značajki. Gotovo nikada ne bi uspijeli postići da svi podaci imaju jednak broj značajki pa bi nam uzorci bili različitih duljina. Ako su različitih duljina nemoguće je uspoređivati takve uzorke niti donijeti smislene zaključke. S druge strane, ako odlučimo restringirati značajke tako da su svi jednake duljine kao onaj s najkraćom duljinom pojavit će se problem neadekvatnih značajki. Sve u svemu, moramo posebnu pažnju posvetiti odabiru značajki i kreiranju uzorka. Kako bi postupak bio jasan provest ćemo ga na jednom primjeru te navesti sve detalje na koje treba obratiti pažnju.

Primjer 1.4.1. *Pretpostavimo da su nam dani komentari o automobilima. Svaki komentar se sastoji od rečenica koje govore o kvaliteti lima automobila, o opremi automobila, o snazi motora, kvaliteti interijera te dostupnosti na tržištu. Zadatak je iz komentara izdvojiti rečenice i svaku rečenicu staviti u klaster ovisno o tome o čemu rečenica govori.*
Komentar:
Boja lima je idealne kvalitete, ne gubi intenzitet na suncu. Korozija je značajno zabilježena tek na modelima starijima od 10 godina. Izuzetno kvalitetan interijer s opremom kao što je električni paket, sustav protiv proklizavanja, klima i ostala oprema. Automobil je dostupan u tri varijante kao sportback, limuzina, karavan. Cijena mu se na tržištu kreće oko 10000 eura.

Rješenje:
Prvi korak u konstrukciji uzorka je otkriti tip podatka kojeg će biti značajke. Konkretno za primjer kojeg promatramo prirodno je da značajke budu riječi. Kako je naš zadatak klasificirati rečenice promotrimo kao primjer prvu rečenicu.
Boja lima je idealne kvalitete, ne gubi intenzitet na suncu. Prvo možemo iz rečenice izbaciti veznike, zatim riječi kao što su *ni*, *ne*. Konačno dobijemo sljedeći vektor s korijenima riječi $v_1 = (boja, lim, idealan, kvaliteta, gubi, intenzitet, sunce)$. Parsiranjem i čišćenjem druge rečenice dobijemo vektor
$v_2 = (korozija, znacajno, zabiljezena, modelima, stariji, 10, godina)$. Prva i druga rečenica govore o kvaliteti lima, imaju sedam značajki ali kao što možemo primjetiti niti jedna značajka od v_1 se ne podudara sa značajkama iz v_2. Svaki algoritam za grupiranje sa standardnim funkcijama koje mjere veze odnosno sličnost podataka neće prepoznati ove dvije rečenice kao uzorke iz istog klastera. Prvi korak u rješavanju nastalog problema je kako prepoznati značajke specifične za određenu rečenicu. Ovaj postupak nema egzaktan odgovor i u praksi se dizajneru rješenja prepušta da analizira podatke i izbaci riječi za koje procijeni da su previše specifične. Matematički ovaj problem se rješava računanjem frekvencija pojavljivanja riječi te izbacivanjem onih s najmanjom frekvencijom. Pogledajmo ponovno vektore značajki. U v_1 možemo primijetiti kako su riječi *idealna*, *kvaliteta*, *gubi* općenite i nikako ne reprezentiraju određenu kategoriju. U v_2 vidimo da su sve riječi osim *korozija* općenite te nikako ne opisuju lim automobila. Kada izbacimo sve nepoželjne riječi dobijemo sljedeća dva vektora

$$v_1 = (boja, lim, intenzitet, sunce)$$

$$v_2 = (korozija).$$

Međutim, opet imamo problem da vektori uzoraka nisu iste duljine te nemaju zajedničkih podataka. U praksi ovaj problem se rješava tako da se sve značajke napišu kao uređena n-torka te se kreiraju vektori uzoraka koji predstavljaju pojavljivanje određene značajke. U

konkretnom slučaju radi se o binarnom vektoru gdje 1 predstavlja da se značajka pojavila u uzorku, a 0 da nije.

$$(boja, lim, intenzitet, sunce, korozija)$$
$$v_1 = (1, 1, 1, 1, 0)$$
$$v_2 = (0, 0, 0, 0, 1).$$

Navedenim postupkom dobili smo jedinstveno zapisane sve rečenice kao vektor od pet komponenti. Dakle, duljina je ista i skup značajki koje promaramo je isti za sve uzorke. Isti postupak provedemo za ostale rečenice, izvučemo iz njih najbitnije značajke te ih spojimo s već uređenom petorkom i dobijemo traženu uređenu 11-torku koja predstavlja sve rečenice iz komentara:

$$(boja, lim, intenzitet, sunce, korozija, interijer, oprema, dostupan, varijante, cijena, trziste).$$

Svaku rečenicu prikažemo kao uzorak.

- Boja lima je idealne kvalitete, ne gubi intenzitet na suncu
  uzorak = $(1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0)$.
- Korozija je značajno zabilježena tek na modelima starijima od 10 godina
  uzorak = $(0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$.
- Izuzetno kvalitetan interijer s opremom kao što je električni paket, sustav protiv proklizavanja, klima i ostala oprema
  uzorak=$(0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0)$.
- Automobil je dostupan u tri varijante kao sportback, limuzina, karavan. Cijena mu se na tržištu kreće oko 10000 eura
  uzorak=$(0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1)$.

Ovako kompaktno zapisani podaci zauzimaju puno manje memorije, lakši su za pohranu ali i za analizu. Težine koje smo dodali u funkciju sličnosti u ovom trenutku imaju bitnu ulogu. Na primjer ako nam se pojavi riječ lim ili korozija sasvim je očito da se radi o kvaliteti lima te ako veću pažnu posvetimo tim značajkama veća je vjerojatnost da na kraju pripadaju ispravnom klasteru. Analogno ako promatramo riječ tržište, jedina prihvatljiva interpretacija je da se ta riječ pojavljuje samo u rečenicama koje govore o kretanju na tržištu te i ovoj značajki treba pridijeliti dodatnu važnost. S druge strane, riječ sunce može se pojaviti u raznim kontekstima, njegova je uloga bitna ali nikako ne smije biti najvažnija u odabiru klase te mu se važnost treba restringirati.

Poglavlje 2

Evolucijske metode

2.1 Meta-heuristike

Kada govorimo o teškim problemima, zapravo govorimo o problemima čiji skup rješenja eksponencijalno raste i čak ni današnja super brza računala ne mogu ispitati sva moguća rješenja kako bi pronašli ono najbolje. Problemi za koje ne možemo u realnom vremenu pronaći egzaktno rješenje nazivaju se NP-teški problemi. Ovako neformalna definicija ostavlja mnoga pitanja kao na primjer: što je to rješenje u realnom vremenu? U ovom radu nećemo promatrati složenost algoritama, kao ni konstrukciju modela za izračunavanje istih, ali zbog potpunosti i razumijevanja težine problema grupiranja i opravdanosti uvođenja heuristika definirat ćemo samo nužne pojmove.

Definicija 2.1.1. *Prihvatljivo vrijeme je vrijeme koje može proći a da rješenje još uvijek bude aktualno.*

Definicija 2.1.2. *Rješenje u realnom vremenu je rješenje čiji se rezultat može upotrijebiti u prihvatljivom vremenu od trenutka pokretanja algoritma koji traži rješenje.*

Definicija 2.1.3. *NP-teški problemi su problemi za koje ne možemo izračunati egzaktno rješenje u realnom vremenu.*

Promotrimo sada problem grupiranja za koji pokušavamo pronaći i analizirati dobar algoritam. Problem je trivijalno rješiv iscrpnom pretragom. Iscrpna pretraga je proces ispitivanja svih mogućih rješenja danog problema. U našem slučaju to bi značilo ispitivanje svih mogućih kombinacija n uzoraka u k klastera. Pretpostavimo da imamo samo dva moguća klastera i n uzoraka. Za svaki uzorak možemo odlučiti na dva načina, tako da ga pridružimo prvom ili drugom klasteru. Isti postupak možemo provesti za svaki uzorak pa

ugrubo imamo ocjenu:

$$\underbrace{2 \cdot 2 \cdot 2 \cdot \dots \cdot 2}_{\text{n puta}} = 2^n.$$

Iako je navedena ocjena za broj mogućih rješenja jako gruba ipak govori o redu veličine danog problema. Već za $k = 2$ problem grupiranja je težak problem jer imamo 2^n mogućnosti gdje smjestiti uzorke. Upravo iz navedenih razloga prilikom rješavanja problema grupiranja nikada se ne upotrebljava iscrpna pretraga osim možda u nekim slučajevima izuzetno malog skupa podataka. Osim iterativnih metoda navedenih u poglavlju 1, kada tražimo optimalno ili suboptimalno rješenje, problemu je moguće pristupiti i na heruistički način. Heuristike ne daju nikakvu garanciju na kvalitetu rješenja, za razliku od iterativnih metoda koje u svakom koraku poboljšaju već postojeće rješenje. Međutim, kod heurisika je prednost što lakše izbjegavaju lokalne minimume i na pseudo-slučajan način sa skokovima traže rješenje u prostoru svih mogućih rješenja.

Definicija 2.1.4. *Pod pojmom heuristike podrazumijevati ćemo algoritme koji pronalaze rješenje koje je zadovoljavajuće dobro te imaju polinomnu složenost izvršavanja s obzirom na ulazni skup podataka. Heuristike ne daju garanciju na kvalitetu rješenja.*

Zadovoljavajuće dobro rješenje je rješenje koje zadovoljava unaprijed zadanu točnost. Iako u definiciji navodimo kako heuristike ne daju nikaku garanciju oko kvalitete rješenja u praksi su se pokazale kao vrlo korisne. Garancija oko kvalitete rješenja nije moguća upravo zbog pseudo-slučajnih elemenata koji su sastavni dio heurističkih algoritama. Naime, prilikom analize algoritma koristi se tok algoritma kao glavni alat za dokaz korektnosti, dok kod heuristika nema glavnog toka u klasičnom smislu jer pseudo-slučajnim elementima izvodimo unaprijed nepredvidive skokove.

Definicija 2.1.5. *Meta-heuristike su skup algoritamskih koncepata koji služe za rješavanje određenog skupa sličnih problema. Za meta-heuristike su poznati neki rezultati korektnosti te su takvi algoritmi generalizirani za širu upotrebu.*

Za meta-heuristike možemo reći da su specijalizirane heuristike. Dva su osnovna pristupa rješavanju problema pomoću heurisika:

- konstrukcijski algoritmi
- algoritmi lokalne pretrage.

Primjenom konstrukcijskih algoritama rješenje se gradi dio po dio, najčešće bez mogućnosti povratka sve dok se ne izgradi konačno rješenje. Kod algoritama lokalne pretrage izabere se jedno rješenje i ono se iterativno poboljšava dok ne zadovolji uvjete. Ideja je da se baziramo oko nekog relativno dobrog rješenja i pretražujemo njemu slična kako bi našli

najbolje ili dovoljno dobro rješenje. Heuristički postupak ukratko se može opisati u nekoliko koraka:

- slučajno izaberi neko rješenje
- ako je rješenje jako loše ponovo biraj
- ako je rješenje skoro dobro poboljšaj
- ako je rješenje dobro stani.

Iz ovih nekoliko koraka vidimo kako se heuristike ponašaju na vrlo jednostavan način. Ako je neko rješenje jako loše ono ne može opstati jer će njegovo mjesto zauzeti neko bolje rješenje. Ako je rješenje preživilo prvu selekciju onda od njega pokušamo napraviti nešto bolje rješenje. Navedeni postupak se zapravo svakodnevno javlja u prirodi kao selekcijski postupak jedinki. Loše jedinke izumiru, dobre ostaju, razmnožavaju se i tako nastaju sve bolje i kvalitetnije jedinke. Upravo navedena inspiracija bit će vodilja kroz daljnji izbor algoritama za heuristike opisanih u sljedećoj cjelini.

2.2 Prirodom inspirirani algoritmi

Glavna pitanja prilikom kreiranja heurističkog rješenja su: kako provesti postupak selekcije dobrih, kako stvoriti nova rješenja ako je neko rješenje dovoljno dobro te konačno kako reprezentirati podatke. Rješenja na upravo navedene probleme možemo pronaći u prirodi. Kada govorimo o algoritmima inspiriranim prirodnim procesima mislimo na tri osnovna tipa algoritama. Konkretno radi se o evolucijskim, procesnim i algoritmima rojeva. Procesni algoritmi su algoritmi koji simuliraju procese u prirodi. Jedan od takvih procesa je kaljenje metala. Metal se zagrijava na određenu temperaturu te se postepeno hladi i tokom procesa hlađenja mijenja se struktura metala i tako se stvara poželjna struktura legure. Navedeni tip algoritama opisuje algoritme lokalne pretrage. Postoje još mnogi algoritmi ovog tipa [13], ali upravo navedeni je najpoznatiji predstavnik. Što se algoritama rojeva tiče to su algoritmi koji simuliraju neki proces uglavnom mrava i pčela. Kako su obje navedene vrste poznate kao jako radno orijentirane, njihovi procesi se vrlo često odvijaju iterativno. Tako na primjer postoji mravlji algoritam koji simulira put kojim mravi putuju tokom radne akcije. Naime, mravi kada prolaze nekim putem ostavljaju za sobom trag feromona tako da drugi mravi mogu ići istim putem. Što više mrava prođe određenim putem to je miris feromona jači te tako privlači i druge mrave na taj isti put. Ako rješenje optimizacijskog problema reprezentiramo kao jedan put kojima mravi mogu putovati možemo kreirati dobar algoritam koji rješava dani problem. Osim mravljeg algoritma ističe se algoritam roja pčela [12]. Evolucijski algoritmi su oni koji su u ovom radu puno zanimljiviji i algoritmi za grupiranje podataka nerijetko imaju elemente upravo ove metode. Evolucijski

algoritmi simuliraju životni ciklus neke populacije. Razvoj algoritma može se prikazati kao životni ciklus jedinke neke populacije. Pređimo tako na slučaj kada su rješenja skoro dobra. Pretpostavimo da svako rješenje predstavlja jednu jedinku, to znači da u prirodi imamo jedinku koja je spremna opstati. Nakon što selektiramo sve jedinke koje su sposobne opstati nužan je proces razmnožavanja. Prilikom razmnožavanja dobrih jedinki, to jest onih koje su opstale, nastaju nove jedinke s dobrim genetskim materijalom. Upravo u ovom evolucijskom koraku, ako su jedinke rješenja, vidimo analogan proces traženju poboljšanja rješenja. Također postoji mogućnost da neke jedinke prilikom nastanka mutiraju i tako znatno promjene svoje svojstvo. Ova mogućnost u rješavanju optimizacijskog problema ima dva značenja. Prvo značenje je pozitivno i pomaže kako bi izašli iz lokalnog optimuma. Naime, jednom kada imamo dva skoro dobra rješenja tada smo vrlo vjerojatno u nekom lokalnom optimumu i daljnim postupkom razmnožavanja samo onih koji su jako dobri sve više se približavamo sličnim rješenjima iz potprostora lokalnog optimuma. Mutacija se nameće kao logično rješenje bijega iz lokalnog optimuma. Mutiranjem jedinka mijenja svojstva te tako postaje ili puno gora ili barem jednako dobra kao neko već postignuto rješenje, ali ključna stavka je da mutirano rješenje vrlo često nije iz područja lokalnog optimuma gdje je nastalo. Kod evolucijskih algoritama ističe se nekoliko ključnih koraka:

- stvori inicijalnu populaciju
- izaberi najbolje iz populacije, ostale obriši
- kako bi postigli ravnotežu razmnožavaj preostale jedinke do popunjenja populacije
- mutiraj neke jedinke
- ponovi postupak selekcije
- ako neka jedinka iz trenutne populacije zadovoljava tražene uvijete stani.

Najpoznatiji predstavnici navedene metode su *genetski algoritam-GA*, *umjetni imunološki algoritam-IA*[13] i *algoritam diferencijske evolucije-DE*[12]. Ostalo je objasniti što su to jedinke, kako rješenja predstavljamo kao jedinke te kako ćemo obavljati selekciju unutar populacije. Navedena pitanja rješavaju se slično za sve algoritme, ali zbog posebnosti svakog algoritma postoje i specifičnosti tako da generaliziranog postupka za prikaz jedinki i selekcije nema. Kako ćemo u ovom radu posebnu pažnju posvetiti genetskim algoritmima za njih ćemo prikazati postupak kreiranja rješenja te provesti sve procese evolucijskog razvoja nad tako kreiranim rješenjima.

2.3 Genetski algoritam

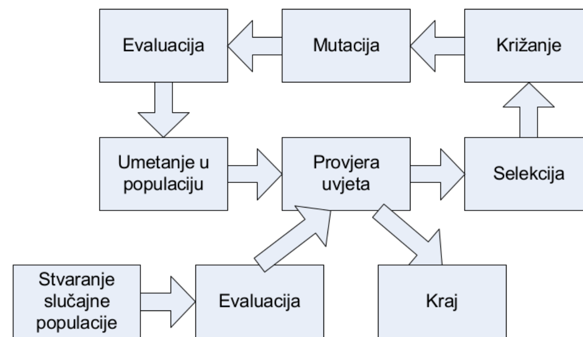
Ideja za razvoj algoritma temelji se na Darwinovoj teoriji o postanku vrste. Teorija se temelji na pet postavki:

1. plodnost vrste - potomaka uvijek ima dovoljno
2. održivost - veličina populacije je uvijek stalna
3. količina hrane je ograničena
4. prilikom procesa razmnožavanja nema potomaka jednakih roditeljima
5. najveći dio varijacije prenosi se nasljeđem.

Postoji puno izvedbi genetskog algoritma, ali generalna ideja je sljedeća. Algoritam radi s populacijom jedinki. Svaka jedinka jedno je moguće rješenje danog problema. U našem slučaju svaka jedinka je jedna particija rješenja. Jedinke u genetskom algoritmu često se nazivaju kromosomi. U kasnijim razmatranjima intuitivnije je govoriti o križanju i mutaciji kromosoma. Svakoj jedinki računa se funkcija dobrote. U našem slučaju funkcija dobrote je suma razlika svih jedinki u pojedinom klasteru. Jedinke s manjom vrijednosti dane funkcije su bolje jedinke. Nakon računanja dobrote svih jedinki izvršava se postupak selekcije. Selekcija je postupak izbora jedinki koje imaju ulogu roditelja u novoj populaciji. Jedinke djeca nastaju operatorom križanja i mutacije. Ciklus se nastavlja zadani broj puta. Iz navedenog opisa možemo izdvojiti četiri osnovne operacije koje se izvršavaju u svakom ciklusu: *selekcija*, *računanje vrijednosti funkcije dobrote*, *križanje*, *mutacija*. Postoje dvije osnovne vrste genetskih algoritama koji koriste gore opisane operacije.

Eliminacijski genetski algoritam

U svakom koraku ovog algoritma biraju se dva roditelja te se provodi postupak križanja. Nastala jedinka djeteta dalje podliježe operaciji mutacije te računanju dobrote. Nakon konačnog formiranja novonastale jedinke donosi se odluka oko njenog prihvaćanja u populaciju. Najčešće se iz populacije izbaci najlošija jedinka te se tako napravi mjesto za novu. Primijetimo da se pokušava održati stalna veličina populacije što je jedna od evolucijskih strategija u prirodi. Međutim, nije uvijek slučaj da se izbacuje najlošija jedinka. Ponekad je poželjno izbaciti najstariju jedinku ili jednog od roditelja nove jedinke. Ciklus eliminacijskog genetskog algoritma prikazan je na slici 2.1 preuzetoj iz [13].



Slika 2.1: Proces eliminacijskog razvoja

Prikazani proces na slici može se prezentirati pseudokodom koji je vrlo jednostavan za implementaciju.

Algoritam 2: Eliminacijski genetski algoritam

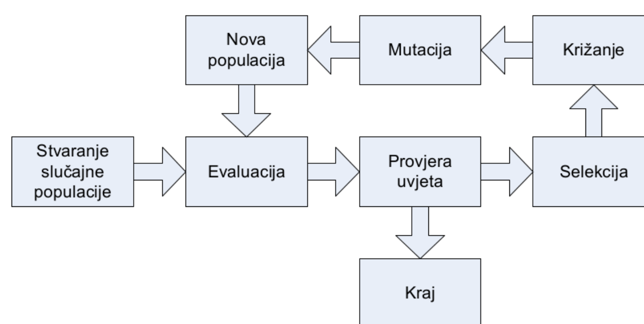
P = stvoriPocetnuPopulaciju();
Evoluiraj(P);
while *nije zadovoljen uvjet zaustavljanja* **do**
  Odaberi R_1 i R_2 iz P;
  D = Krizaj(R_1, R_2);
  Mutiraj(D);
  Evoluiraj(D);
  S = OdaberiJedinkuZaZamjenu();
  if *D bolji od S* **then**
    P.izbaci(S);
    P.ubaci(D);

Ključni operatori, to jest procesi, unutar algoritma su *mutacija*, *križanje* te relacija *bolji od*. Prilikom kreiranja algoritama navedenim operacijama i relacijama potrebno je posvetiti veću pažnju te empirijski pokušati odrediti optimalne parametre kako bi se izbjegli lokalni optimumi i loša rješenja.

Generacijski genetski algoritam

Za razliku od prve navedene metode, kod koje ne postoji jasna granica između jedinki roditelja i djece, generacijski algoritmi jasno razlikuju populaciju roditelja i djece. Naime, eliminacijski pristup dopušta da u jednoj populaciji mogu biti i roditelji i djeca, dok u generacijskom pristupu nova populacija nastaje od populacije roditelja, ali se nakon toga svi roditelji brišu i imamo potpuno novu generaciju jedinki. Ovaj pristup ima jedan bitan nedostatak. Prilikom brisanja svih jedinki roditelja obrisat ćemo i neka dobra rješenja.

Navedeni problem zove se problem *elitizma*. Dakle, elitizam je svojstvo da se čuvaju najbolja rješenja. Primijetimo da eliminacijski algoritam trivijalno rješava problem elitizma jer roditelje križamo, a iz populacije izbacujemo one koji nam nisu dobri po nekom kriteriju. Dakle, najbolji nam uvijek ostaju. Kod generacijskog algoritma problem elitizma rješava se tako da se najboljih nekoliko jedinki ostavi, a ostale se generiraju nove. Evolucijski proces prikazan je grafom na slici 2.2. Navedeni proces s uključenim elitiz-



Slika 2.2: Proces generacijskog razvoja

mom se vrlo često koristi u praksi. Opisani proces prikazan je sljedećim pseudokodom.

Algoritam 3: Generacijski genetski algoritam s elitizmom

P = stvoriPocetnuPopulaciju();
Evoluiraj(P);
while *nije zadovoljen uvijet zaustavljanja* **do**
    novaPopulacija $P' = \emptyset$;
    P'.ubaci(P.najbolje());
    while *dok je broj jedniki u P' manji od veličine populacije* **do**
        Odaberi R_1 i R_2 iz P;
        $\{D_1,D_2\}$ = Krizaj(R_1, R_2);
        Mutiraj(D_1);
        Mutiraj(D_2);
        Evoluiraj(D_1);
        Evoluiraj(D_2);
        P'.dodaj(D_1,D_2);

Kao i u algoritmu 2 i u ovom algoritmu ključnu ulogu igraju operatori selekcije, mutacije i križanja. Svaki od navedenih operatora u sebi sadrži slučajne elemente i na taj način omogućuju skokove po prostoru svih rješenja. Prije objašnjenja samih operatora objasnit ćemo kako zapravo rješenja prikazujemo kao jedinke, odnosno kromosome. Naime, rješenja vrlo često nisu jednostavna i sastoje se od skupa struktura. S druge strane,

operatori u genetskom algoritmu trebaju jednostavnu strukturu kako bi operacije bile male složenosti i kako bi implementacija bila moguća. U praksi jedinke su vrlo često vektori određene duljine koji predstavljaju rješenja danog problema. Kako je vektorski prikaz najpopularniji tako u ovom radu upravo navedenu metodu koristimo za kodiranje rješenja. Prisjetimo se, problem koji promatramo je raspodjela n objekata u k klastera. Rješenje problema se na vrlo jednostavan način može prikazati kao vektor od n komponenti s vrijednostima od 1 do k. Svaka komponenta vektora predstavlja jedan uzorak, a vrijednost komponente predstavlja klaster koji je dodijeljen uzorku. Slika 2.3 prikazuje kako to zapravo izgleda.

Slika 2.3: Prikaz raspodjele n objekata u k klastera kao kromosom

Ovako definiran prikaz ima i nedostatke kao na primjer ako sve elemente drugog klastera stavimo u treći i obrnuto. Tada su nam oba rješenja jednako dobra i zapravo to su ista rješenja, ali u populaciji su to dvije različite jedinke. Optimizacijom i otkrivanjem istih rješenja može se znatno poboljšati algoritam te na to treba obratiti pažnju prilikom konstrukcije. Kada nam je poznata struktura svake jedinke moguće je konstruirati prije navedene operatore.

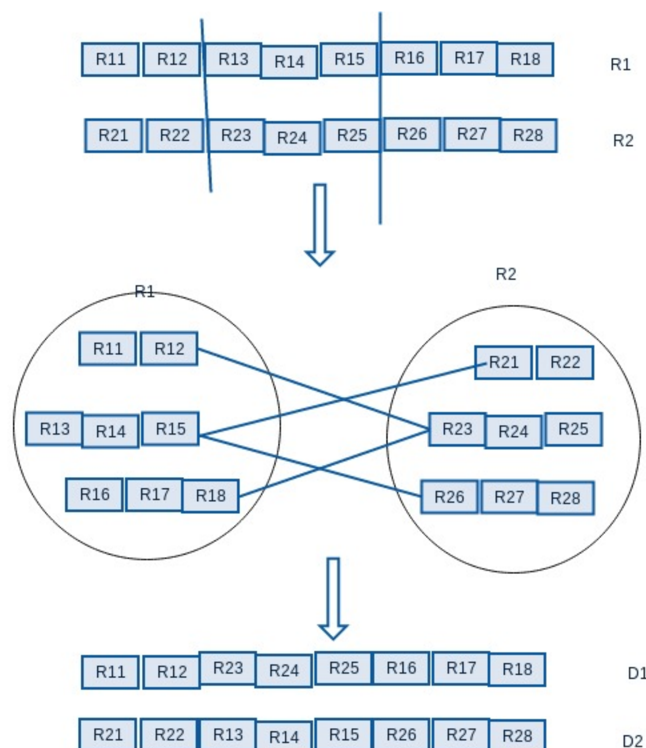
Križanje

Križanje je postupak kreiranja nove jedinke iz već postojećih jedinki. U proces križanja ulaze dvije jedinke, označimo ih s R_1 i R_2, dok kao rezultat križanja mogu biti jedna ili dvije jedinke. Križanje među jedinkama se odvija na način da se dio kromosoma jedinke R_1 i dio kromosoma jedinke R_2 spoje u novu jedinku. Primijetimo da je ovo zapravo prirodan proces gdje od genetskog materijala roditelja nastaju djeca. Postupak križanja nije jedinstven, ali dvije metode se ističu kao korisne:

1. križanje s k nasumičnih točaka prekida

2. nasumično s k uniformnih točaka prekida.

Križanje s k točaka prekida je postupak kada se unaprijed definiraju točke prekida te se kromosomi roditelji podjele na $k + 1$ podskup početnog kromosoma. Nakon inicijalne podjele uzimaju se dijelovi kromosoma od oba roditelja i formira se kromosom dijete. Slika 2.4 ilustruira proces križanja i nastanka djece D_1 i D_2 nad kromosomima roditeljima s 8 komponenti i dvije točke prekida. Nadalje kada govorimo o točkama prekida one mogu



Slika 2.4: Primjer križanja s 2 prekida

biti izabrane nasumično kao u primjeru sa slike 2.4 ili mogu biti odabrane uniformno. Uniformna podjela je često korisna kada nam je funkcija dobrote težinska i znamo koje komponente imaju koji utjecaj. Uniformna podjela nam omogućuje da unaprijed znamo koliko je moguće poboljšanje i što će se dogoditi s jedinkom dijete. Naime, ako križamo jednu lošu i jednu dobru jedinku i recimo da su nam komponente kromosoma složene po važnosti, onda znamo da će nam ako uzmemo prvi uniformni dio od lošeg kromosoma i jedinka dijete biti jako loša jer su mu najvažnije komponente loše. Kod nasumičnog odabira postoji vjerojatnost da prvi skup komponenti ima samo jednu komponentu i ako je ta komponenta preuzeta od loše jedinke to će minimalno utjecati na kvalitetu jedinke dijete. Konačno dajemo pseudokod operacije križanja s nasumičnim izborom k točaka. U slučaju

uniformnog izbora potrebno je u algoritmu samo zamijeniti funkciju za izbor točaka.

Algoritam 4: Križanje

```
R1, R2, D1 = ∅, D2 = ∅;
K = OdaberiTockePrekida(n);
for i ∈ K do
    if i mod 2 == 0 then
        D1.dodaj(R1(i : i + 1));
        D2.dodaj(R2(i : i + 1));
    else
        D2.dodaj(R1(i : i + 1));
        D1.dodaj(R2(i : i + 1));
```

Mutacija

Kada jednom dobijemo nove jedinke iz genetskog materijala roditelja svaka je na svoj način posebna i trebala bi sadržavati nešto novo. Upravo zbog navedenog razloga uvodi se operator mutacije. Mutacija se često događa s nekom vjerojatnošću. Na početku se empirijski odredi vjerojatnost mutacije te ako se zadana vjerojatnost postigne na slučajan način promijenimo jednu komponentu kromosoma djeteta. Pozadina mutacije u rješavanju optimizacijskog problema jako je važna. Naime, kada se mutacija dogodi na jedinki djeteta nastaloj od dva roditelja čija je dobrota bila gotovo dobra velika je vjerojatnost da će i dijete imati prihvatljivu funkciju dobrote. Ako smo slučajno upali u neki lokalni optimum mutacija će nam pomoći da se iz njega izvučemo.

Algoritam 5: Mutacija

```
D = jedinka spremna na mutaciju;
k = SlucajnaPozicijaUKromosomu();
S = SlucajnaVrijednost();
D[i] = S;
```

Ovisno o potrebama moguće je da se mutacija dogodi na više od jednog mjesta u kromosomu.

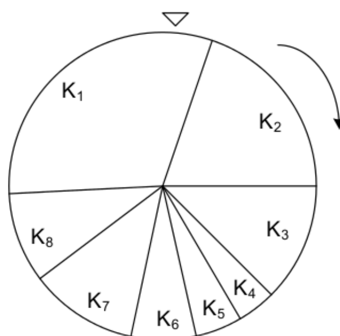
Selekcija

Kako bi mogli križati jedinke potrebno je prvo izabrati adekvatne jedinke za križanje. U nekim slučajevima kada je potrebno izabrati koja će jedinka opstati potrebno je pokrenuti proces selekcije. Dva su osnovna modela selekcije: *k-turnirska* i *proporcionalna* selekcija. Kod k-turnirske selekcije na slučajan način izabereno k jedinki iz populacije. Zatim

računamo dobrotu svake jedinke ili upotrijebimo već izračunatu dobrotu. Prema vrijednosti funkcije dobrote biramo najbolju jedinku od njih k, a ostale vraćamo nazad u populaciju. Ako nam treba više jedinki, onda postupak ponovimo toliko puta. Postoji verzija turnirske selekcije koja ne vraća $k-1$ neizabranih jedinki nego uzima novi skup od k jedinki i ponovi proces. Uz ove dvije metode postoji još metoda Turnirskih selekcija [13], ali u ovom radu kristiti će se k-turnirska s ponavljanjem i proporcionalna selekcija. Kod proporcionalne selekcije postupak je sljedeći. Gledaju se sve jedinke te se prema funkciji dobrote odredi vjerojatnost odabira. Udio se određuje prema formuli :

$$udio(K_i) = \frac{dobrota(K_i)}{\sum_{i \in |POP|} dobrota(D_i)}.$$

Nakon što vjerojatnosti prikažemo kružnim dijagramom 2.5 na slučajan način odaberemo



Slika 2.5: Primjer za populaciju od 8 jedinki

Jedinka	1	2	3	4	5	6	7	8
Dobrota jedinke	6	4	3.6	0.4	0.8	1.2	2.4	1.6
Vjerojatnost odabira	0.3	0.2	0.18	0.02	0.04	0.06	0.12	0.08

Slika 2.6: Tablica vrijednosti za sliku 2.5

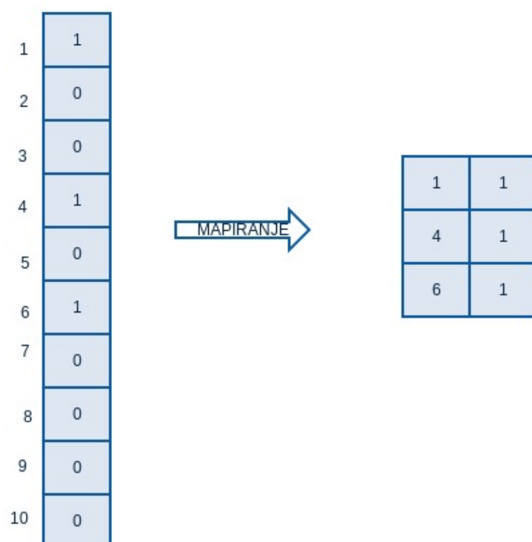
točku unutar kruga. Pronađemo područje na kružnom dijagramu gdje je pala slučajno odabrana točka, tu jedinku uzimamo kao kandidata za daljnji postupak križanja i mutacije. U praksi se vrlo često vrijednosti smjeste na jediničnu dužinu te se biraju realne vrijednosti iz dane jedinične dužine. Nakon što imamo model za selekciju jedinki, a uz prije uvedene modele za operatore mutacije i križanja, imamo sve preduvjete za stvaranje modela koji će opisati problem grupiranja te pronaći heuristički dobro rješenje pomoću navedenih algoritama.

Poglavlje 3

Sekvencijalni algoritmi za problem grupiranja

3.1 Algoritam k-sredina

Algoritam k-sredina pripada particijskoj metodi grupiranja. Kako bi testirali implementaciju navedenog algoritma poslužit ćemo se motivacijskim primjerom 0.0.2. Prije implementacije algoritma provest ćemo postupak pripreme podataka koji su nam dani na raspolaganje. Podaci su preuzeti s repozitorija [3] gdje na raspolaganju imamo preko 40000 dokumenata. Svaki dokument opisan je vokabularom od gotovo 30000 riječi. Podatke ćemo pripremiti metodama koje smo upoznali u cjelini 1.4.
Promotrimo dane dokumente. Svaki dokument je reprezentiran n-dimenzionalnim vektorom riječi. Dakle, uzorci su nam dokumenti i reprezentirani su kategoričkim značajkama. Svaka riječ iz danog vektora reprezentirana je jedinstvenim identifikacijskim brojem. Brojevi su dodijeljeni po jednostavnom postupku prema indeksu pozicije u vektoru. Na svakoj poziciji vektora zapisana je neka vrijednost iz $\mathbb{N} \cup \{0\}$. Nula predstavlja da se određena riječ ne pojavljuje u dokumentu, dok broj različit od nula predstavlja broj pojavljivanja dotične riječi u dokumentu. Kako n-dimenzionalni vektori predstavljaju vokabular riječi koji se pojavljuju u svim dokumentima to znači da će nam n-dimenzionalna reprezentacija jednog dokumenta imati vrlo često puno nula. Nule koje se pojavljuju nebitne su za analizu i grupiranje, ali zauzimaju memoriju i rješenje čine kompleksnim. Metodom mapiranja zanemarit ćemo pojavljivanje nula te podatke o n-dimenzionalnom vektoru spremiti na kompaktan način tako da pamtimo indeks gdje se nula nije pojavila u vektoru te indeksu pridružimo vrijednost koja je bila u danom vektoru. Slika 3.1 prikazuje mapiranje 10-dimenzionalnog binarnog vektora. Već na malom primjeru možemo vidjeti da je memorijska ušteda oko 40% što je izuzetno bitno za složenost algoritma. Također, kod paralelnih algoritama gdje će biti prijenosa podataka između procesora ovako kompaktni mapirani



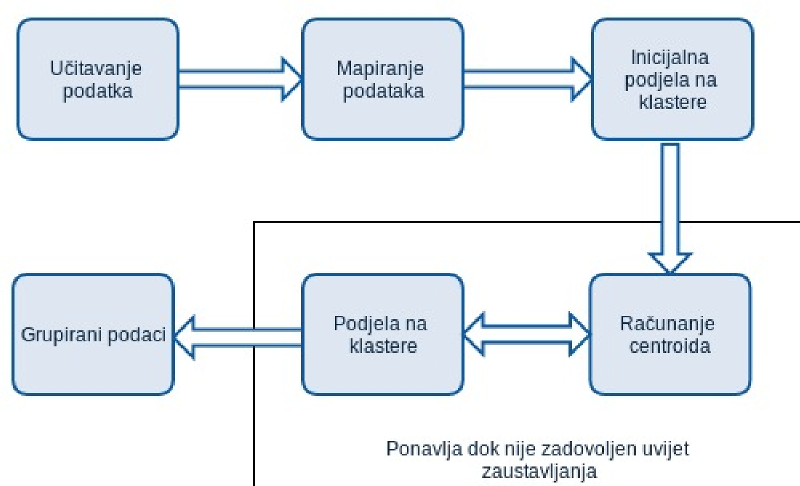
Slika 3.1: Mapiranje vektora

zapis brže će biti dostavljen na odredište.
Dakle, sve dokumente mapirat ćemo na opisani način te ćemo svakom dokumentu pridružiti identifikacijski broj. U implementaciji ćemo u klastere spremati identifikacijske brojeve dokumenata i svaki dokument ćemo interpretirati kao jedan uzorak, a mapirani podaci će predstavljati značajke danog uzorka.

Opis rješenja

Na početku algoritma na slučajan način rasporedimo dokumente u k unaprijed definiranih klastera. Nakon inicijalne podjele potrebno je odrediti predstavnika klastera koji se često naziva i centroid. Kako smo otkrili da se radi o kategoričkim podacima za ocjenu sličnosti podataka koristit ćemo funkciju sličnosti s_w definiranu u 1.7. Iz definicije funkcije s_w možemo zaključiti kako su nam potrebne frekvencije pojavljivanja svih riječi unutar danog klastera. Kada imamo izračunate frekvencije te želimo novi element pridružiti klasteru samo je potrebno pogledati koje od riječi se pojavljuju u klasteru i sumirati sve frekvencije takvih riječi. Novi element pridružujemo onom klasteru čija je suma frekvencija najveća. Primijetimo da su nam sve riječi jednako bitne te nam je $w_i = 1$ za sve i. Svaki novi element uspoređujemo s frekvencijama riječi, to jest vektor koji predstavlja skup riječi i njihove frekvencije u ovom slučaju bit će centorid danog klastera. Svaki element će biti uspoređen upravo s danim centoridom. Centroid očito nije uzorak iz danog klastera ali najbolje opisuje dani klaster. Iterativno ponavljamo postupak na način da svaki puta

kada definiramo centroide opet sve elemente pridružimo onom klasteru u kojemu je suma frekvencija centroida najveća za dani uzorak. Nakon novodefiniranih klastera, ponovo računamo centroide i ponavljamo postupak. Sljedeći dijagram opisuje tok predstavljenog algoritma. Uvjet zaustavljanja može biti unaprijed određen, tako da zadamo broj itera-



Slika 3.2: Tok sekvencijalnog algoritma k-sredina

cija. Uz ovako definiran uvjet zaustavljanja moguće je definirati granicu točnosti. Granica točnosti određuje se prema empirijskom zaključku. Naime, nakon nekog vremena uzorci su razmješteni u klastere tako da u sljedećoj iteraciji algoritma promjene budu jako male. U tom trenutku možemo reći da algoritam stane i vrati trenutno stanje. Tim postupkom ubrzali smo vrijeme izvođenja tako da smo dopustili minimalnu grešku. Granicu zaustavljanja postavit ćemo u ovisnosti prema broju klastera. Sljedećim pseudokodom algoritma opisat ćemo način realizacije.

Algoritam 6: K-sredina bez granice točnosti

```
D = UcitajPodatke();
Mapiraj(D);
N = broj iteracija;
K = broj klastera;
klasteri = InicijalnaPodjela(D, K);
for i od 1 do K do
    centri = nadiFrekvencijeUKlasterima(klateri);
    klasteri = NoviKlasteri(klasteri, centri);
vrati(klasteri);
```

Algoritam 7: K-sredina s granicom točnosti

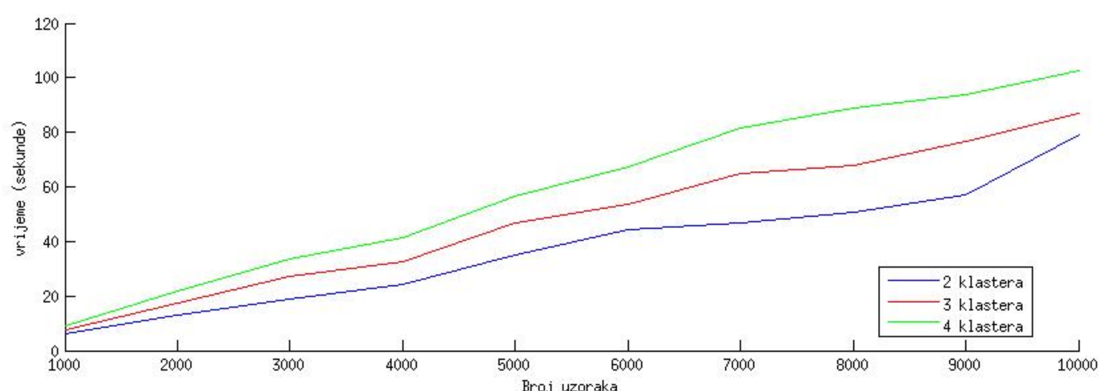
```
D = UcitajPodatke();
Mapiraj(D);
N = broj iteracija;
K = broj klastera;
klasteri = InicijalnaPodjela(D,K);
noviKlasteri = ∅;
brojRazlika = 0;
for i od 1 do N do
    centri = nadiFrekvencijeUKlasterima(Klateri);
    klasteri = NoviKlasteri(Klasteri, centri);
    brojRaazlika = Razlike(klateri, noviKlateri);
    if brojRazlika < K then
        ZaustaviAlgoritam();
    noviKlateri = klasteri;
vrati(klasteri);
```

Osim nakon određenog broja iteracija, algoritam 7 se zaustavlja ako broj razlika u klasterima padne ispod broja klastera. U analizi ovaj broj ćemo mijenjati u ik, $i \in \mathbb{N}$.

Analiza

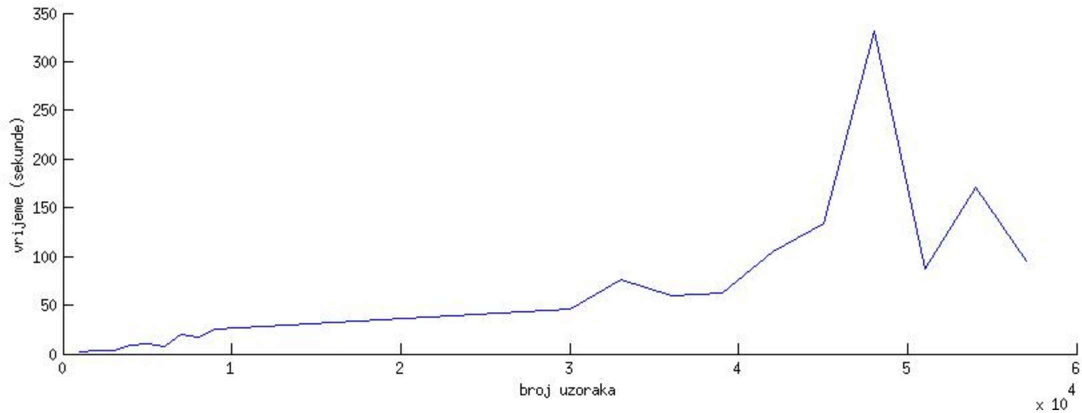
Analiza je provedena nad istim skupom uzoraka. Na raspolaganju nam je računalo s AMD Quad-Core Processorom A8-3500M, a svi algoritmi implementirani su u programskom jeziku Python. Iz prirode podataka nije jasno koliki broj klastera je optimalan za najbolje rješenje. Povećanjem broja klastera primjećuje se značajan pad točnosti kada je broj klastera veći od pet. Točnost promatramo kao razliku između funkcije dobrote za prvo inicijalno rješenje te za konačno rješenje nakon određenog broja iteracija. Zapravo, točnost

je razlika dobrote između slučajnog rješenja i rješenja dobivenog provedbom algoritma. Analizu provodimo promatrajući različiti broj klastera te uvođenjem dodatnog uvjeta zautavljanja. Na slici 3.3 prikazana su vremena sekvencijalnog algoritma za klasteriranje po-

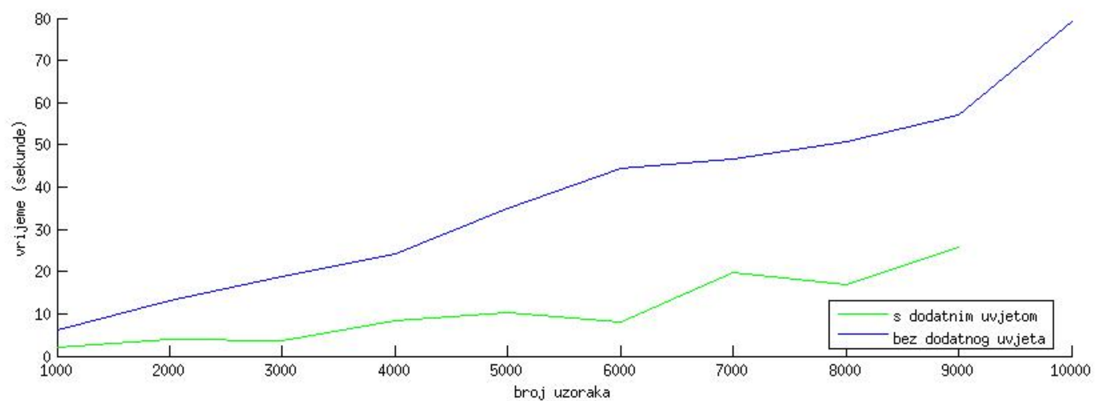


Slika 3.3: Performance skvencijalnog algoritma za različit broj klastera

dataka opisanog u 6. Broj uzoraka varira od 1000 do 10000, dok broj klastera varira od 2 do 4. Prikazana testiranja su izvođena do maksimalnog broja iteracija. Na grafu je primjetan rast vremenske složenosti obzirom na broj klastera. Također, primjećujemo i linearni rast složenosti s obzirom na broj uzoraka. Linearni rast utrošenog vremena zapravo je očekivan. Prilikom opisivanja particijskih metoda grupiranja napomenuli smo da je složenost $O(n)$. Prikazani graf potvrđuje teorijski očekivani rezultat te daje potvrdu dobre implementacije algoritma k-sredina koje je predstavnik particijske metode. Smanjenje složenosti sekvencijalnog algoritma nije moguća ispod $O(n)$ zato što moramo proći po svim danim uzorcima što nam je sigurnih n operacija. Uvođenjem dodatnog uvjeta zaustavljanja moguće je smanjiti koeficijent rasta. Međutim, nije moguće predvidjeti vrijeme potrebno za izvršavanje. Kao što je prikazano na grafu sa slike 3.4 moguće su velike vremenske oscilacije. Oscilacije su uzrok upadanja u lokalni minimum gdje algoritam nije uspio pronaći dovoljno dobro rješenje te je konačno rješenje vratio tek nakon maksimalnog broja iteracija. Bez obzira na oscilacije, uvođenjem dodatnog uvjeta zaustavljanja znatno se smanjuje vrijeme izvršavanja. Ubrzanje je vidljivo na slici 3.5 gdje je prikazana usporedba istog algoritma s uvjetom zaustavljanja i bez njega. Vrijeme izvršavanja bez dodatnog uvjeta zaustavljanja daje nam gornju među na moguće vremenske oscilacije algoritma s uvjetom zaustavljanja.



Slika 3.4: Performance sekvencijalnog algoritma s uvjetom zaustavljanja



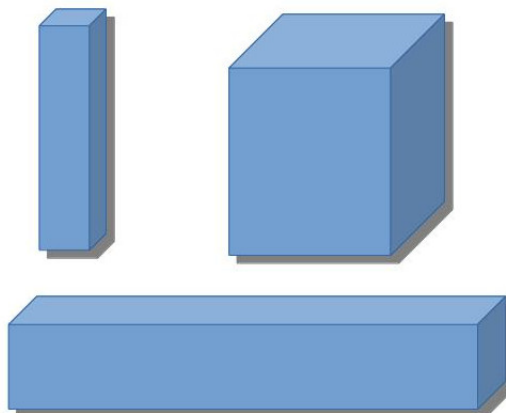
Slika 3.5: Usporedba s obzirom na uvjet zaustavljanja

3.2 Evolucijski algoritam

Za testiranje evolucijskih metoda koristit ćemo primjer 0.0.3. Na raspolaganju nam je baza podataka o kutijama. Svaka kutija spremljena je kao četiridimenzionalni vektor. Prva komponenta je identifikacijski broj kutije dok su sljedeća tri dužina, širina i visina. Cilj nam je grupirati kutije u šest kategorija prema dimenzijama. Na kraju ćemo imati velike i male kutije kockastog, stupičastog i plosnatog oblika kao što je prikazano na slici 3.6.

Opis rješenja i analiza

Rješenje ćemo implemetirati korištenjem genetskog algoritma opisanog u cjelini 2.3. Jedinke će nam predstavljati moguća rješenja. Svaka jedinka bit će vektor gdje indeks pozi-



Slika 3.6: Vrste kutija

cije u vektoru označava kutiju s identifikacijskim brojem jednakim danom indeksu. Vrijednost u vektoru predstavljat će klaster kojemu je pridružena dotična kutija. Zapravo jedinke će izgledati kao što je to prikazano na slici 2.3. Kako smo već opisali, ovako definirane jedinke pogodne su za operatore mutacije i križanja. Svaka jedinka, osim svojim zapisom, određena je i dobrotom. Dobrota je mjera kvalitete rješenja i najbitnija je funkcijonalnost u evolucijskim algoritmima. Naime, dobrota nam omogućava selekciju jedinki koje ulaze u novu generaciju. Podaci su nam širina, visina i dužina kutije, to jest možemo ih smjestiti kao točke u vektorskom prostoru i na njima upotrijebiti Eukidsku metriku definiranu u 1.4. Funkciju dobrote rješenja definirat ćemo tako da za svaki klaster pogledamo srednje vrijednosti te izračunamo Euklidsku udaljenost svih uzoraka od srednje vrijednosti. Sumiramo sve udaljenosti i nađemo srednju vrijednost udaljenosti dijeljenjem s ukupnim brojem uzoraka u klasteru. Isti postupka ponovimo za sve klastere te tako nađemo srednju udaljenost podataka od centara klastera.

$$dobrota(jedinka) = \frac{1}{6}\sum_{k=1}^{6}\frac{\sum_{el\in klaster(k)} udaljenost(centar(k), el)}{brojUzoraka(klaster(k))}.$$

Ovako opisana funkcija dobrote definira da su rješenja bolja što je srednja vrijednost funkcije manja. Svaku jedinku povećavamo za jedan element u vektorskom zapisu i na poziciji nula zapisujemo vrijednost dobrote. Operator mutacije implementiran je tako da se na slučajan način promijeni klaster određenom broju uzoraka. Mutacija se ne događa uvijek nego se unaprijed definira vjerojatnost da se mutacija dogodi. U praksi vjerojatnost mutacije kreće se između jedan i pet posto. Algoritam 8 opisuje postupak mutacije koji ćemo koristiti za konkretno rješenje problema.

Algoritam 8: Operator mutacije

```
Jedinka = jedna jedinka u populaciji;
zadanaVjerojatnostMutacije = broj iz intervala [0.01, 0.05];
vjerojatnostMutacije = slucajniBroj(0,1);
brojPromjena = broj slučajnih promjena;
if vjerojatnostMutacije < zadanaVjerojatnostMutacije then
    for i od 1 do brojPromjena do
        Jednika(SlucajniIndex()) = SlucajniBroj(1,brojKlastera)
vrati(Jedinka);
```

Algoritam 9: Operator usmjeravanja

```
Jedinka = jedna jedinka u populaciji;
c = PronadiCentroide(Jedinka);
Preraspodjeli(Jedinka, c);
vrati(Jedinka);
```

Za potrebe ovog problema uvodimo dodatnu funkciju koju pozivamo nad novim jedinkama nastalim procesom križanja. Nova funkcija zove se funkcija usmjeravanja. Funkcija prima jedinku i za nju provodi postupak traženja centroida klastera te prema nađenim centroidima izvršava proces mutacije potpunom raspodjelom uzoraka prema najbližem centroidu. Upravo navedeni postupak novu jedinku dodatno približi točnom rješenju. Princip koji koristimo za usmjeravanje sličan je jednoj iteraciji algoritma k-sredina.

Operator križanja prima dvije jedinke te kao rezultat vraća dvije nove jedinke. Križanje se odvija s k točaka prekida uniformno kako je opisano u cjelini 2.3. U operatoru križanja koristit ćemo funkciju usmjeravanja na jednoj od novonastalih jedinki.

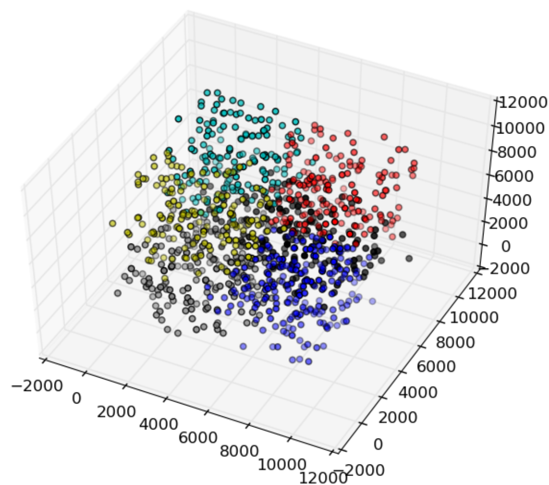
Za postupak selekcije koristit ćemo kombinaciju selekcija opisanih u cjelini o genetskom algoritmu. Konkretno, nakon sortiranja jedinki po dobroti u novu populaciju prenosimo dvije najbolje jedinke. Nakon toga popunjavamo novu populaciju križanjem prve trećine najboljih rješenja. Tim postupkom stvoren je veći dio nove populacije. Nakon toga najbolje jedinke koje nismo prebacili u novu populaciju mutiraju te prelaze u novu populaciju. Dakle, nova populacija je kombinacija eliminacijske i generacijske metode. Ovim postupkom osigurali smo da se za proces križanja koriste samo dobra rješenja. Također smo riješili problem elitizma prenošenjem dva najbolja rješenja. Konačno u novu populaciju dodajemo još rješenja koja su u prethodnoj populaciji bila dobra, ali prije toga obavimo proces mutacije. Proces mutacije može poboljšati rješenje tako da u sljedećoj generaciji upravo popravljeno rješenje bude najbolje. Realizacija rješenja opisana je algoritmom 11.

Algoritam 10: Operator križanja

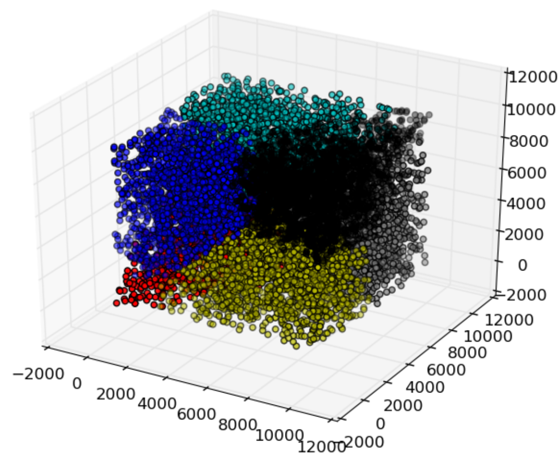
```
R_1 = uzorak iz populacije;
R_2 = uzorak iz populacije;
D_1 = ∅, D_2 = ∅;
K = OdaberiTockePrekida(n);
for i ∈ K do
    if i mod 2 == 0 then
        D_1.dodaj(R_1(i : i + 1));
        D_2.dodaj(R_2(i : i + 1));
    else
        D_2.dodaj(R_1(i : i + 1));
        D_1.dodaj(R_2(i : i + 1));
usmjeri(D_1);
vrati(D_1,D_2);
```

Algoritam 11: Genetski algoritam

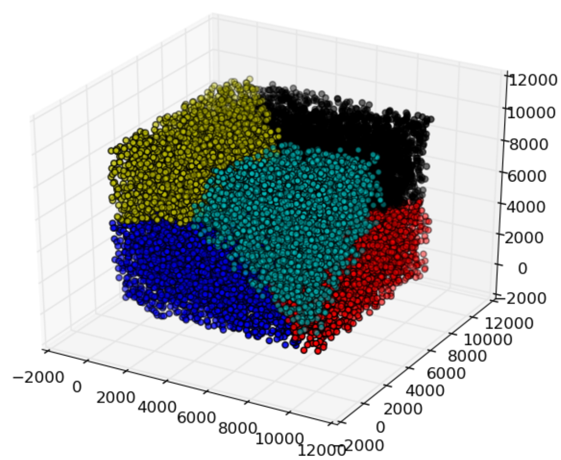
```
populacija = inicijalnaPopulacija(podaci);
dobrota(populacija);
N = broj iteracija;
brojKrizanja = S;
for i od 1 do N do
    novaPopulacija.dodaj(NajboljaDva(populacija));
    while novaPopulacija nema dovoljno jedinki do
        while brojKrizanja do
            [R_1,R_2] = DvaIzPrveTrecineNajboljih(populacija);
            [D_1, D_2] = Krizaj(R_1,R_2);
            Mutiraj(D_2);
            novaPopulacija.dodaj(D_1, D_2) brojKrizanja–;
        K = sljedeciNajboljiIzPopulacije();
        Mutiraj(K);
        novaPopulacija.dodaj(K);
    populacija=novaPopulacija;
    dobrota(populacija);
vrati(populacija.najbolji());
```



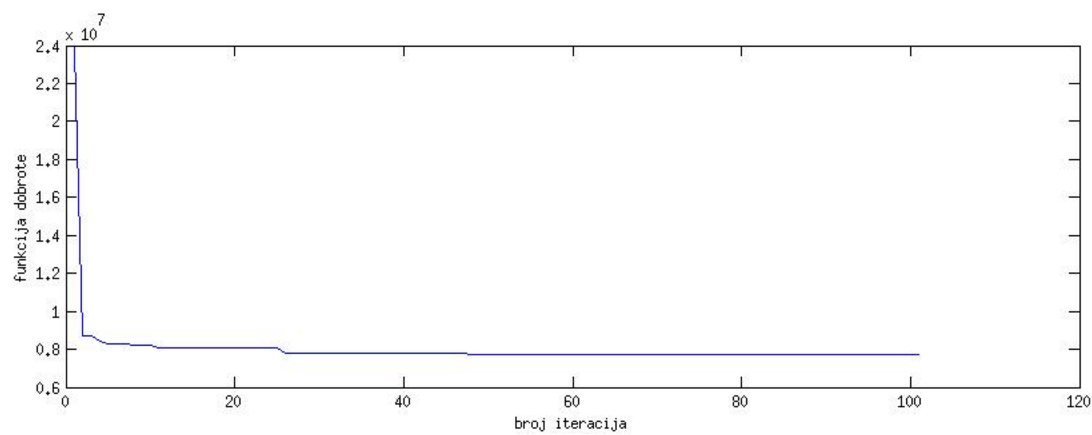
Slika 3.7: Prikaz za 1000 grupiranih uzoraka



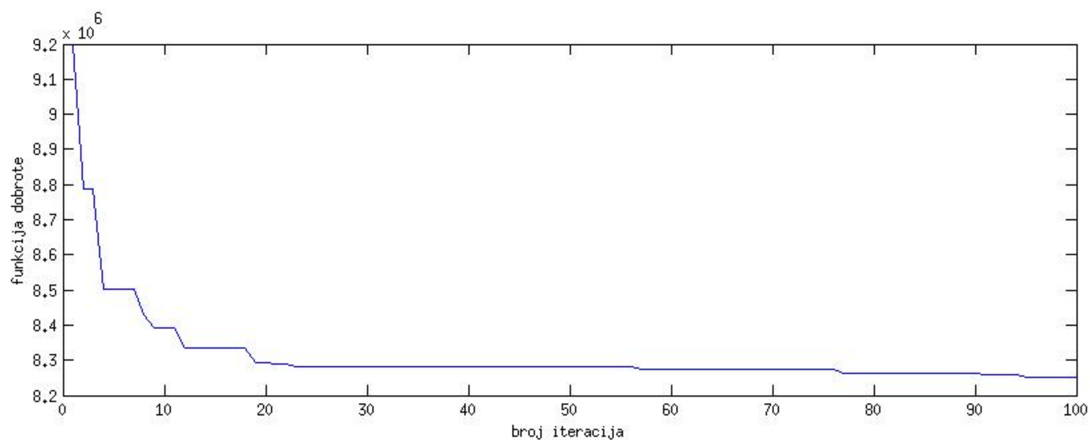
Slika 3.8: Prikaz za 10000 grupiranih uzoraka



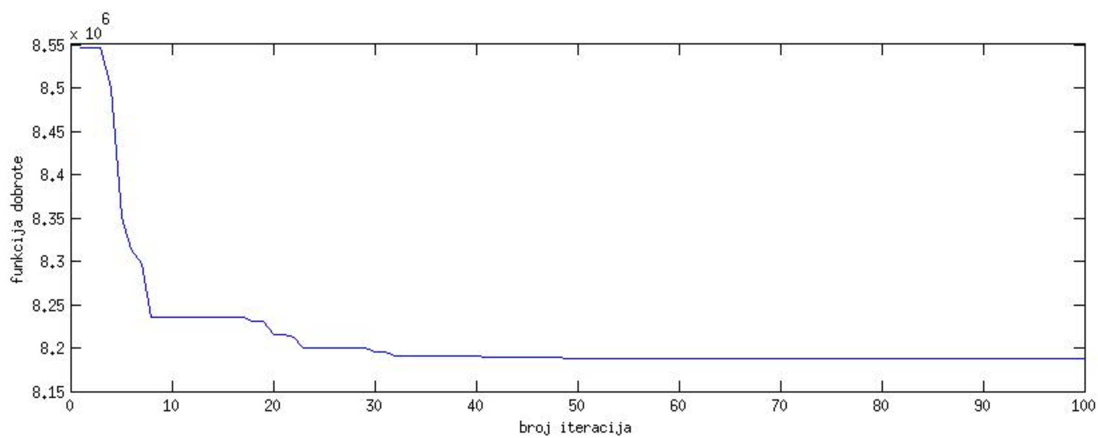
Slika 3.9: Prikaz za 20000 grupiranih uzoraka



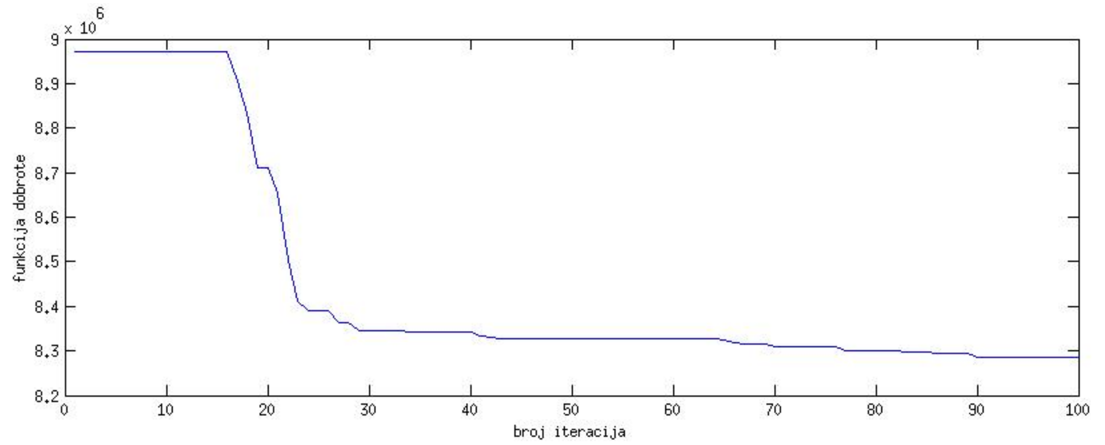
Slika 3.10: Utjecaj iteracija na kvalitetu rješenja za 1000 uzoraka



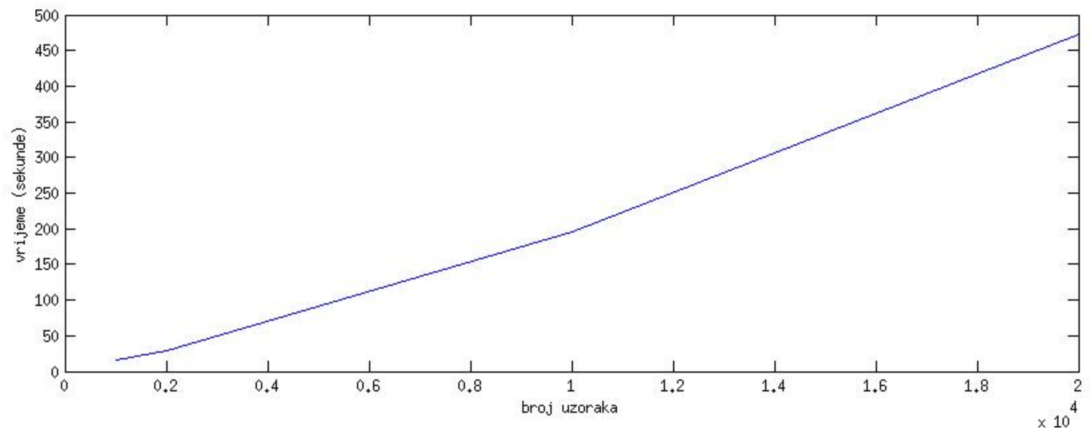
Slika 3.11: Utjecaj iteracija na kvalitetu rješenja za 2000 uzoraka



Slika 3.12: Utjecaj iteracija na kvalitetu rješenja za 10000 uzoraka



Slika 3.13: Utjecaj iteracija na kvalitetu rješenja za 20000 uzoraka



Slika 3.14: Vremenska složenost sekvencijalnog genetskog algoritma

Poglavlje 4

Tehnike za paralelizaciju algoritama

4.1 Paralelni procesi

Računala s velikom snagom danas su glavni predmet rasprave. Računalna industrija pokušava stvoriti modele koji će izvršavati sve veći broj opercija po sekundi, ali da cijena bude prihvatljiva. Zapravo, cilj je napraviti super jako računalo za komercijalne svrhe. Možemo reći da su u tome i uspijeli te danas imamo komercijalna, relativno jeftina, računala sa snagom preko 10^{12} FLOPS-a [1] . Takav industrijski razvoj računalne tehnologije omogućio nam je rješavanje nekih teških problema. Iako su računala izuzetno jaka još uvijek ne možemo egzaktno rješiti probleme koji su NP-teški. Upravo navedeni problemi potaknuli su razvoj paralelnih procesa. Naglasimo da se danas svako računalo sastoji od nekoliko procesora koji također mogu izvršavati poslove samostalno. Pod pojmom paralelni proces uvijek mislimo na proces raspodijeljen između više procesora. Procesori mogu biti smješteni na jednoj fizičkoj lokaciji ili u nekoj mreži više računala.

Osnovna ideja paralelnog procesa je podijeliti posao na skup procesora tako da svaki procesor neovisno riješi zadani problem. Također, cilj je upotrijebiti već postojeće algoritme i u njima detektirati nezavisne poslove. Na kraju se pogledaju sva nezavisna rješenja i formira se konačno rješenje. Kako je uvođenje više procesora skup proces postavlja se pitanje učinkovitost paralelnog sustava. Najčešće se učinkovitost izražava kao omjer mogućeg ubrzanja i broja procesa. Naime, neki djelovi algoritama se moraju izvršavati sekvencijalno i paralelizacija u tom trenutku nema učinka. Najpoznatija ocjena je Amdahl-ov zakon [11]. Zakon kaže da je potencijalno ubrzanje definirano onim dijelom $p \in [0, 1]$ sekvencijalnog programa koji se može paralelizirati, to jest

$$\text{ubrzanje} = \frac{1}{1-p}.$$

[1] floating point operacija po sekundi

Drugim riječima ako se 50 % programa može paralelizirati ($p = 0.5$), onda je ubrzanje 2. Ako se cijeli program može paralelizirati onda je ubrzanje beskonačno. Naravno ovo je teorijski pogled, ali u praksi ulogu ima broj procesa pa konačno ubrzanje definiramo kao:

$$\text{ubrzanje} = \frac{1}{s - \frac{p}{N_p}}$$

gdje je s sekvencijalni udio programa, p paralelni udio, a N_p broj procesora. Tablica na slici 4.1 prikazuje ubrzanja s obzirom na p kao paralelni udio i broj procesora.

ubrzanje	p = 50%	p = 90%	p = 99%
N_P = 10	1.82	5.26	9.17
N_P = 100	1.98	9.17	50.25
N_P = 1000	1.99	9.91	90.99
N_P = 10000	1.99	9.91	99.02

Slika 4.1: Ubrzanje s obzirom na udio paralelizabilosti i broj procesora

Uvođenjem paralelnog načina rješavanja počela je izgradnja potpuno novog modela za rješavanje problema. Naime, već smo spomenuli kako procesori mogu biti smješteni na jednoj fizičkoj lokaciji pa su povezani hardverski ili mogu biti razmješteni na više lokacija pa su povezani mrežno. Hardverska povezanost nije novi podatak i komunikacija među procesorima je riješena unutar samog računala. Kod povezanosti s mrežom računala, to jest procesora, pojavljuju se pitanja o brzini mrežne komunikacije te općenito o propusnosti mreže. Zbog navedenih poteškoća posebna pažnja posvećuje se topologiji mreža, načinu komunikacije i zapisivanju rješenja koja se trebaju prenositi kao informacije od procesora do procesora.

4.2 Osnovni pojmovi tehnologije MPI

U prethodnoj cjelini napomenuli smo kako komunikacija između procesora nije zanemariva. Svaki procesor mora svoj rezultat proslijediti dalje ili primiti rezultate drugih procesora. Paralelizacija algoritama za svrhu ima ubrzanje, ali ako komunikacija među procesorima oduzima previše vremena onda je besmisleno ići na paralelne algoritme. Osim same cijene komunikacije potrebno je stvoriti takav paralelni model da se neke njegove komponente mogu ponovno iskoristiti. Naime, paralelni programi su uvijek skuplji i teži za implementaciju i samim time ponovna upotreba već konstruiranih komponenti uvelike

olakšava posao. MPI[2] standard za komunikaciju među procesorima je danas najpoznatiji i najkorišteniji model komunikacije. MPI je zamišljen tako da definira skup mogućih poruka koje se mogu slati s procesora na procesor te da definira strukturu mreže među procesorima. MPI je samo standard i potrebno ga je implementirati u nekom programskom jeziku. Prve inačice tehnologije MPI implementirane su u programskim jezicima C i Fortran, a najpoznatije implementacije su openMPI [2] i MPICH2 [1]. Danas gotovo svi poznatiji jezici imaju podršku za MPI i njegove najpoznatije implementacije. Za konstrukciju paralelnih algoritama u ovom radu koristi ćemo upravo protokole iz tehnologije MPI te ovdje navodimo samo osnovne funkcionalnosti koje su nam potrebne za razvoj algoritama. Detaljna dokumentacija [7] MPI sadrži velik broj funkcija za komunikaciju i uspostavljanje topologija što ćemo opisati u cjelini 4.3. Osim funkcija za komunikaciju MPI daje osnovnu podršku za mehanizme detekcije procesora u paralelnoj mreži. Tako imamo funkcije koje:

- daju ukupan broj procesora u mreži
- identificiraju osobni broj procesora
- šalju poruku ciljanom procesoru
- primaju poruku od određenog procesora
- detektiraju grupe u kojima je promatrani procesor.

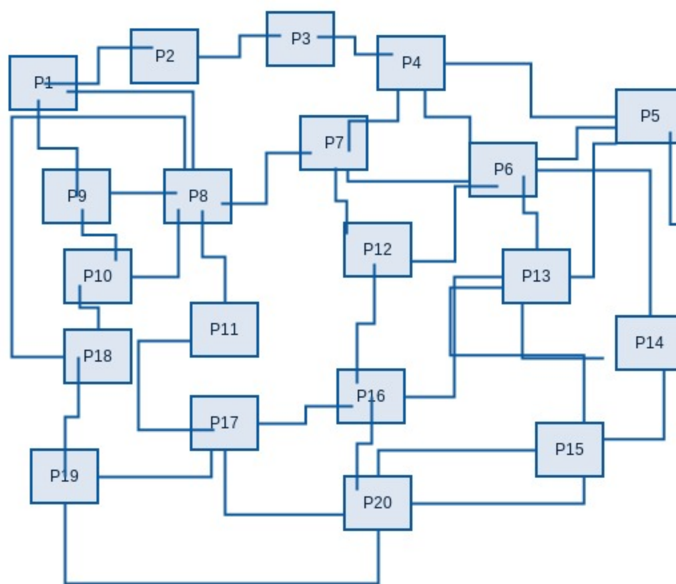
Slanje poruka može se odvijati na dva osnovna načina: *od procesora do procesora* ili *kolektivno slanje*. Poruke koje se šalju od procesora do procesora su poruke koje su vidljive samo procesoru koji šalje i procesoru koji prima poruku. Ostatak procesora za vrijeme komunikacije pošiljatelja i primatelja radi nesmetano i izvršava svoj posao. Pogodnost ovakve komunikacije je neopterećenost ostatka procesora podacima koji nisu namijenjeni njima. Kod kolektivnog slanja procesor pošiljatelj šalje poruku svim procesorima u svojoj grupi. Ostali procesori rade nesmetano. Pojam grupe definiran je u MPI-u [7] i moguće je definirati više grupa procesora ovisno o zadacima koje obavljaju. Na početku su svi procesori smješteni u jednu grupu, a posebnim topološkim operacijama moguće je podijeliti procesore. Pogodnost kolektivnog slanja je da jednim slanjem pošiljatelja svi u grupi dobiju poruku. Tako pošiljatelj izvršava samo jedno slanje što je značajna ušteda u odnosu na slanje svakom posebno. U praksi se slanje od procesora do procesora koristi ako je u pitanju slanje poruke s jednog procesora na jedan ili dva primatelja, dok za sve ostalo se kreiraju grupe procesora i koristi se kolektivno slanje. Iz opisanog postupka slanja jasno je da mora postojati određeni odnos među procesorima. Navedeni problem rješava se pravilim postavljanjem topologije mreže.

[2] Message Passing Interface

4.3 Topologije

Topologija mreže postavlja se u isto vrijeme kada se modelira i samo rješenje problema. Naime, pomoću topologije određujemo tko će s kim i pod kojim uvjetima moći komunicirati i razmjenjivati poruke. Za konstrukciju algoritama bit će nam korisna tri topološka koncepta: *povezane mreže*, *grupna povezanost* i *topologija s vođom*.

Topologija povezane mreže

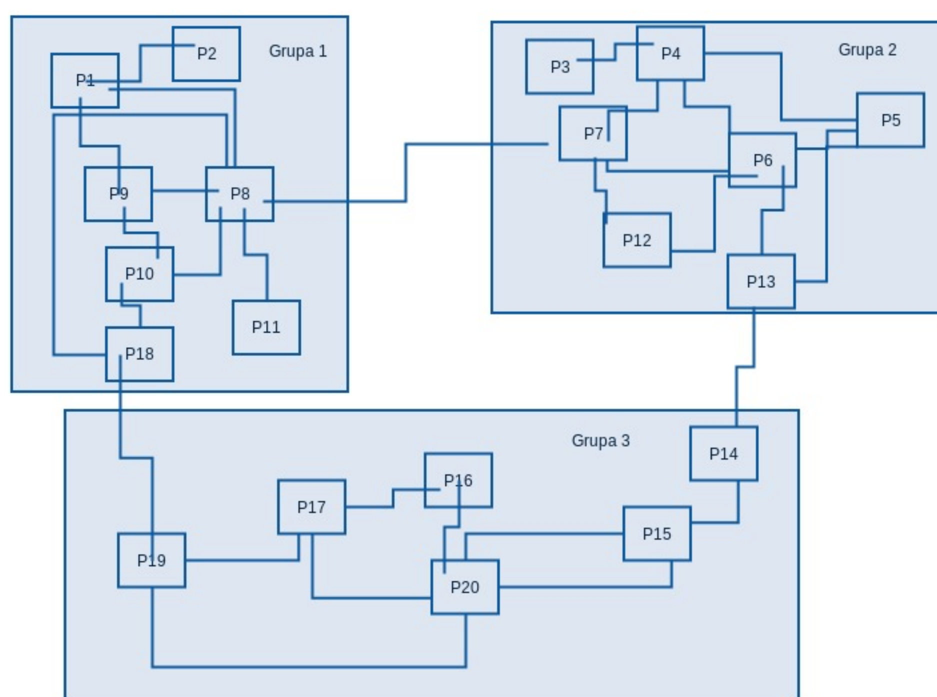


Slika 4.2: Topologija povezane mreže 20 procesora

Procesi na slici 4.2 povezani su s drugim procesorima, ali tako da poruka od svakog procesora može stići na bilo koji drugi procesor posredovanjem međuprocesora. Naime, nisu svi procesori direktno povezani nego poruka mora putovati. Kod ovako definirane topologije prijenos poruka odvija se metodom od procesora do procesora. Topologija je pogodna ako skup procesora koji obavljaju različite poslove, a ne mogu se definirati kao grupa, treba dobiti poruku procesora pošiljatelja. Da bi poruka stigla do svakog tko ju treba dobiti definira se put između pošiljatelja i prvog primatelja te onda primatelj postaje pošiljatelj do drugog primatelja i tako do zadnjeg komu je poruka namijenjena. Navedena topologija ima strukturu povezanog grafa te za nju vrijede sva svojstva iz teorije grafova [5]. U teoriji grafova je poznato da za povezane grafove možemo konstruirati minimalno razapinjuće stablo [5]. Minimalno razapinjuće stablo je podgraf danog povezanog grafa s

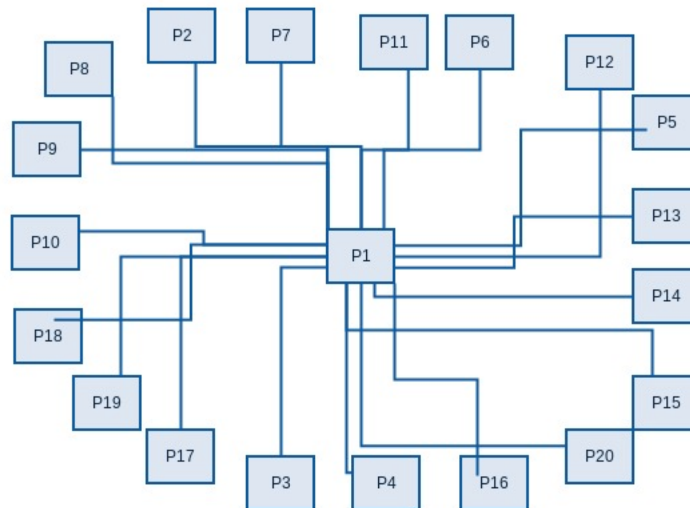
minimalnim brojem veza, tako da od svakog čvora grafa možemo doći do bilo kojeg drugog čvora. Ponekad je pogodnije konstruirati minimalno razapinjuće stablo među procesorima iako na taj način put od primatelja do pošiljatelja neće ići samo preko procesora kojima je poruka zanimljiva. Međutim, izgubljeno procesorsko vrijeme ponekad je manje skupo od prekomplicirane strukture veza.

Topologija povezanosti u grupi



Slika 4.3: Topologija povezanosti u grupi, 20 processora 3 grupe

Ovo je izrazito korisna topološka struktura. Često se upotrebljava upravo prilikom paralelizacije evolucijskih algoritma. Topologija omogućava da unutar grupe svi komuniciraju služeći se kolektivnim slanjem, a prema drugim grupama postoji prestavnik grupe koji onda poruku prenese metodom od procesora do procesora. U primjeni na evolucijskim algoritmima služi za prijenos najbolje jedinke između grupa koje lokalno imaju populacije nad kojima izvršavaju algoritme. Glavni problem ove mreže je optimizirati broj slanja između grupa i izabrati predstavnika grupe.



Slika 4.4: Topologija s vođom, 20 procesora jedan vođa

Topologija s vođom

Prikazani topološki model minimizira komunikaciju među procesorima. Postoji jedan procesor s kojim svi komuniciraju i samo njemu šalju poruke. U modelu se vrlo jednostavno detektiraju greške tako da je to jedna od prednosti ovog modela. Svi procesori osim vođe komuniciraju po principu od procesora do procesora, dok vođa može slati poruke svima ili ciljano. Moguća nadopuna ovog modela je da postoje grupe pa vođa može grupi poslati poruku. Model je pogodan za paralelizaciju u znanstvenom računanju te se tamo koristi za gotovo sve paralelne izračune. Od ostalih područja primjene ističu se, nama zanimljivi, evolucijski algoritmi. Naime, možemo procesorima pridružiti poslove operatora mutacije i križanja. U svakom trenutku vođa ima sve podatke o populaciji i jedinkama te ih prosljeđuje na obradu procesorima s kojima upravlja. Od rezultata koje dobije, to jest jedinki nastalih mutacijom i križanjem, kreira novu populaciju.

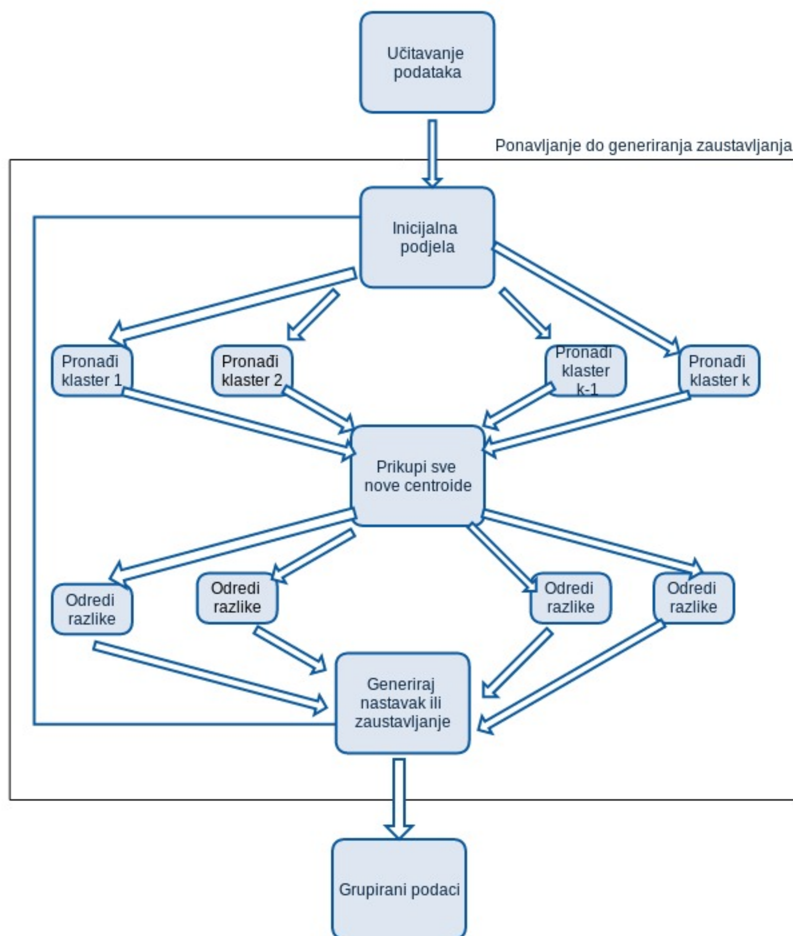
Poglavlje 5

Konstrukcija paralelnih algoritama za problem grupiranja

5.1 Paralelizacija algoritma k-sredina

Opis

U ovoj cjelini cilj je paralelizirati algoritam 7. Za paralelizaciju koristit ćemo prethodno navedene metode i tehnologije. Prvenstveno potrebno je otkriti koji su koraci u algoritmu nezavisni i kakva topologija je pogodna za paralelnu implementaciju. Predstavit ćemo paralelnu implementaciju s obzirom na broj klastera. Svakom procesoru u paraleli pridružit ćemo određeni broj klastera o kojima se brine. Idealni scenarij je da svaki procesor brine o točno jednom klasteru. Na početku jedan procesor obavlja posao učitavanja podataka te posao inicijalne podjele na klastere. Tako podijeljene podatke šalje svim procesorima koji se trebaju brinuti o nekom od klastera. Svaki procesor izračuna centroid u svom klasteru i taj podatak vrati glavnom procesoru. Centroidi se računaju na način opisan u 3.1. Glavni procesor prikupi sve centroide te na osnovu njih podijeli podatke u nove klastere. Svakom pomoćnom procesoru ponovo se proslijede klasteri na daljnju analizu. Pomoćni procesori računaju nove centroide i šalju glavnom procesoru. Glavni procesor odlučuje kada će proces grupiranja završiti te na osnovu toga zaustavlja sve ostale procesore u radu. Dodatno glavni procesor od svakog procesora koji se brine o nekom klasteru može zatražiti da izračuna razliku klastera u trenutnoj i prethodnoj iteraciji. Kada prikupi sve podatke o razlikama tada može donijeti odluku o prijevremenom prekidu. Kao što iz opisa algoritma možemo primijetiti istaknuta je uloga vođe. Procesori koji se brinu o klasterima komuniciraju jedino s vođom. Njemu šalju podatke i od njega primaju potrebne informacije za daljnji rad. Dijagram na slici 5.1 najbolje opisuje postojanje glavnog toka programa te s obzirom na mogućnost parleliza-



Slika 5.1: Paralelni algoritam k-sredina

cije pozive pomoćnih procesora. S obzirom na posao koji obavljaju, za procesor vođu i za ostale procesore implementirani su posebni algoritmi. Problem kod implementacije predstavlja sinkronizacija. Naime, procesor vođa mora čekati dok svi procesori ne završe jedan ciklus grupiranja te na osnovu toga donijeti zaključak da li nastaviti dalje ili je rješenje dovoljno dobro. Također, dok procesor vođa obrađuje podatke i donosi odluku ostali procesori moraju biti u stanju mirovanja i čekati daljnje upute. Algoritmi koji rješavaju sinkronizaciju i implementiraju topologiju s vođom prikazani su pseudokodom.

Algoritam 12: Procesor vođa

```
rank = VratiRank();
grupa = PrepoznajGrupu();
D = UcitajPodatke();
Mapiraj(D);
grupa.bcast(D);
N = broj iteracija;
K = broj klastera;
klasteri = InicijalnaPodjela(D, K);
for i od 1 do N do
    for i od 1 do K do
        grupa.send(klasteri[i], dest = i);
    centri = [];
    razlike = [];
    for j od 1 do K do
        centri[j] = grupa.recv(source = j);
        razlike[j] = grupa.recv(source = j);
    razlika = sum(razlike);
    if razlika then
        grupa.bcast(True);
        klasteri = NoviKlasteri(centri,D)
    else
        grupa.bcast(False);
        prekid;
    grupa.bcast(False);
vrati(klasteri);
```

Funkcija *VratiRank()* definirana je od strane opisane tehnologije MPI. Konkretno funkcija vraća jedinistveni broj svakom procesoru. Ako imamo k procesora oni će biti označeni brojevima od 0 do $k - 1$. Ako postoji procesor vođa onda se za vođu izabere procesor s identifikacijskim brojem 0. Funkcija *PrepoznajGrupu()* definira svakom procesoru u koju grupu procesora pripada. Vrlo je važno da svaki procesor zna svoju grupu jer u tehnologiji MPI to je osnovni princip da bi neki procesor slao i primao poruke. Nakon inicijalizacije procesora potrebno je da glavni procesor učita podatke te ih mapira prema već opisanom pravilu u 3.1. Svaki procesor mora imati uvid u podatke te zbog toga glavni procesor svima mora proslijediti sve podatke koji su na raspolaganju. Funkcija *bcast()* definirana je na određenoj grupi procesora. Procesor pozivom ove funkcije prosljeđuje određeni podatak svim procesorima u grupi. U ovom slučaju glavni procesor svima ostalima šalje mapirane

podatke. Kao i u algoritmu 7 potrebno je na slučajan način napraviti inicijalnu podjelu. Posao inicijalne podjele također obavlja procesor vođa. Nakon inicijalne podjele svakom procesoru posebno šalje onaj klaster nad kojim će obavljati operacije. Funkcija *send*() omogućuje ciljano slanje definiranjem odredišta. Funkcija je definirana od strane MPI tehnologije i radi na grupi procesora. Procesor vođa sada je spreman za primanje rezultata i obradu. Posao vođe je da prihvaća sve nove centroide i sve razlike u klasterima koje mu šalju pomoćni procesori. Nakon što prikupi sve podatke procesor vođa sumira sve razlike te provjeri uvjet zaustavljanja obzirom na razlike. Ako algoritam treba nastaviti šalje pomoćnim procesorima poruku za nastavak. Za nastavak rada vođa mora pogledati sve centroide i na osnovu njih napraviti novu podjelu na klastere. Novu podjelu radi tako da svaki uzorak pridruži klasteru čiji centroid ima najveću sumu frekvencija riječi koje se pojavljuju u uzorku. Princip pridruživanja definiran je funkcijom s_w 1.7 koja je implementirana u funkciji *NoviKlasteri()*. Nove klastere prosljeđuje na daljnju analizu. Ako je uvijet zadovoljen, to jest ako su razlike dovoljno male ili je zadovoljen broj iteracija procesor vođa šalje zahtjev za zaustavljanje. Zahtjevi za nastavak i za zaustavljanje šalju se kolektivno svima u grupi pomoću *bcast*() funkcije. Na kraju rada vođa vraća podatke podijeljene u klastere.

Algoritam 13: Pomoćni procesori

```
root = 0;
D = grupa.recv(source = root);
radi = True;
prvi = True;
stariKlaster = [];
while radi do
    klaster = grupa.recv(source = root);
    centar = nadiFrekvencijeUKlasterima(klaster);
    if prvi then
        razlika = duljina(klaster);
        prvi = False;
    else
        razlika = RazlikaKlastera(klaster, stariKlaster);
    stariKlaster = klaster;
    grupa.send(centar, source = root);
    grupa.send(razlika, source = root);
    radi = grupa.recv(source = root);
```

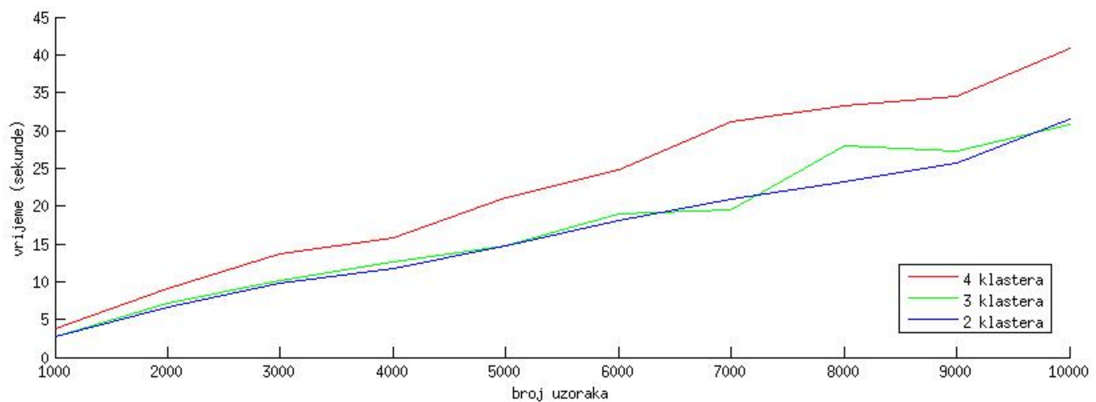
Na početku rada pomoćnog procesora definiramo tko je vođa. Vođu označavamo s *root*. Pomoćni procesor od vođe putem funkcije *recv*() prima sve podatke. Primijetimo da je funkcija *recv()* također definirana na grupi procesora. Funkcija prima kolektivne i

ciljane poruke. Na početku rada procesora uvjet na rad postavimo na istinu. Prilikom rada pomoćnog procesora razlikujemo dva stanja. Ako procesor prima inicijalne podatke onda razliku postavlja kao broj elemenata u danom klasteru. Naime, u prvom koraku kada vođa inicijalno podijeli podatke pomoćni procesori nemaju posao grupiranja i razlika je zapravo ukupan broj svih elemenata u klasteru. U svim ostalim slučajevima razlika je broj različitih elemenata u prethodnoj i trenutnoj iteraciji. Funkcija *nadjiFrekvencijeUKlasterima()* vraća centroid danog klastera. Izračunati centroid i razlika šalju se procesoru vođi te se čeka poruka o nastavku rada. Problem sinkronizacije riješen je tako da su u implementaciji korištene funkcije definirane u tehnologiji MPI s blokirajućim mehanizmom [9]. Blokirajući mehanizam definira da funkcije *recv()*, *send()* i *bcast()* čekaju odziv. Dakle, izvršavanje se zaustavlja na pozivima funkcija dok se ne dogodi odziv drugog procesora. Tako funkcija *send()* čeka poziv od funkcije *recv()*. Analogno za *bcast()*. Međutim, kod blokirajućeg mehanizma jako je bitan poredak poruka slanja i primanja. Naime, ako se dogodi da neki *recv()* čeka poruku koja se nikad neće poslati dolazimo da pojma blokade programa; program se blokira i nikada ne nastavlja s izvršavanjem. U navedenim algoritmima za vođu i pomoćni procesor možemo primijetiti kako svaki poziv slanja odgovara točno jednom pozivu prihvaćanja poruke.
Paralelizacija je provedena na svim komponentama algoritma koje su obavljale nezavisne poslove. Pomoćni procesori rade na lokalnim podacima koji su potpuno nezavisni. Nezavisnost podataka osigurana je definicijom klastera 1.2.1. Glavni procesor je sakupljao parcijalne rezultate, a definiranje novih klastera odvijalo se sekvencijalno. Naime, da smo pomoćnim procesorima omogućili da sami definiraju klastere, to bi značilo da svaki pomoćni procesor mora izračunati udaljenost svakog uzorka od svakog centroida. U tom slučaju višestruko bi računali iste rezultate. To bi dodatno usporilo algoritam. Moguće je promatrati relaksirani problem na način da unaprijed definiramo granicu prihvaćanja. U tom slučaju procesori ne bi morali znati ostale centroide, niti bi procesor vođa morao raditi raspodjelu na nove klastere. Svaki procesor bi u svoj klaster prihvaćao uzorke koji zadovoljavaju prije definiranu granicu prihvaćanja. Ovakvim postupkom smanjila bi se komunikacija te ubrzao proces grupiranja. Međutim, pitanje je koliki je omjer ubrzanja i dozvoljene greške. Moguće je da će više klastera prihvatiti isti uzorak pa nećemo imati disjunktne klastere.

Analiza

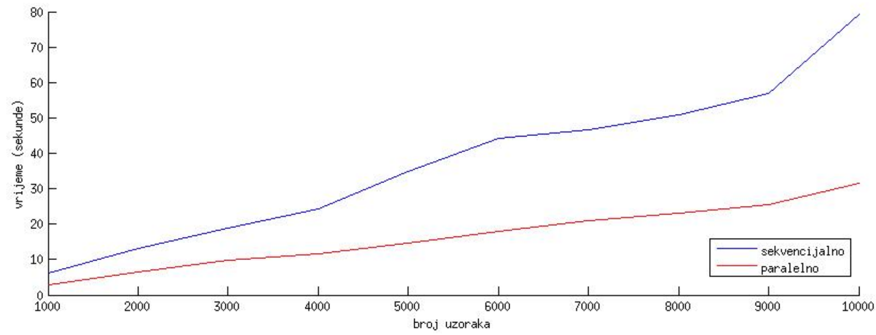
Analiza je provedena nad istim skupom podataka korištenim u 3.1 za analizu sekvencijalnog algoritma. Cilj analize je promatrati odnos složenosti paralelnog i sekvencijalnog algoritma. Implementacija algoritma predpostavlja da se svaki procesor brine o svom klasteru te svaki procesor obavlja identičan posao osim procesora vođe koji mora sakupljati podatke i raspoređivati posao. Za analizu na raspolaganju su nam četiri fizička i dva vir-

tualna procesora. Virtualni procesori su mogućnost da se jedna jezgra računala ponaša kao dvije, ali s određenim usporenjem u odnosu na realni fizički procesor. Analizu smo proveli za dva, tri i četiri klastera. Iz implementacije algoritma može se zaljučiti da je za realizaciju s k klastera potreban $k + 1$ procesor. Pomoćni procesori kojih ima k te jedan procesor za vođu. Usporedbom vremena prikazanih na slici 5.2 možemo primijetiti da je vrijeme izvršavanja za dva i tri klastera jednako. Na raspolaganju imamo četiri procesora,

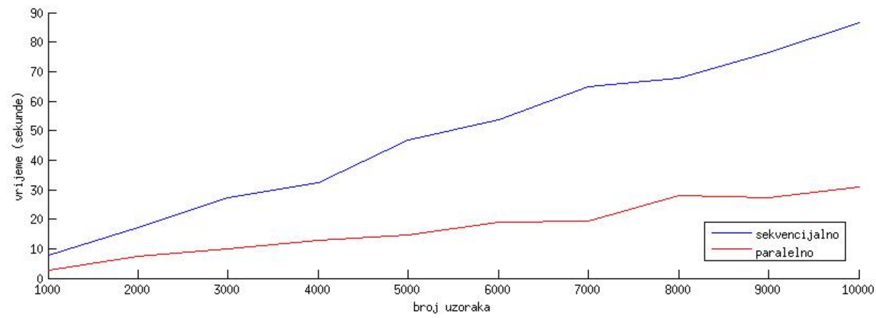


Slika 5.2: Paralelni algoritam za različit broj klastera

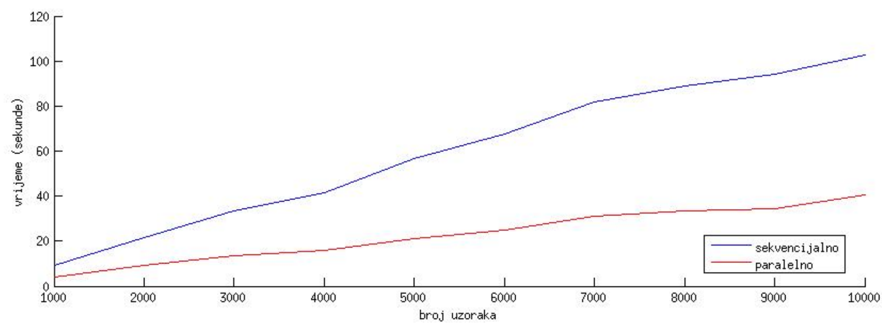
a za paralelizaciju s dva i tri klastera su nam potrebna tri odnosno četiri procesora što znači da imamo dovoljno resursa za potpunu parelizaciju što se očituje i na danom grafu. U slučaju kada trebamo četiri klastera za pomoć su angažirani i virtualni procesori koji su kako smo već napomenuli nešto sporiji. Virtualnom procesoru može se prepustiti posao vođe ili posao pomoćnog procesora. Osim što smo paralelizacijom postigli robusnost, na grafovima sa slika 5.3, 5.4, 5.5 prikazana je usporedba ubrzanja paralelnog u odnosu na sekvencijalni algoritam. Robusnost na broj klastera ponajviše ovisi o broju procesora koji su nam na raspolaganju. Važno je primijetiti da je ubrzanje značajnije što je broj klastera koje zahtijevamo veći.



Slika 5.3: Usporedba sekvencijalnog i paralelnog algoritma s 2 klastera



Slika 5.4: Usporedba sekvencijalnog i paralelnog algoritma s 3 klastera



Slika 5.5: Usporedba sekvencijalnog i paralelnog algoritma s 4 klastera

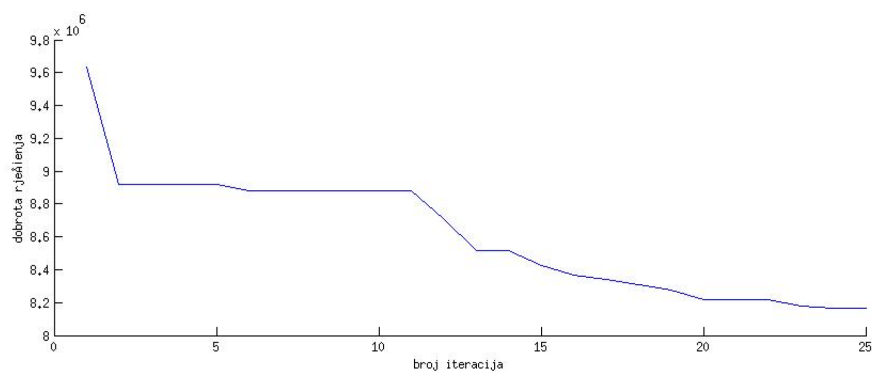
5.2 Paralelizacija evolucijskog algoritma

Opis rješenja i analiza

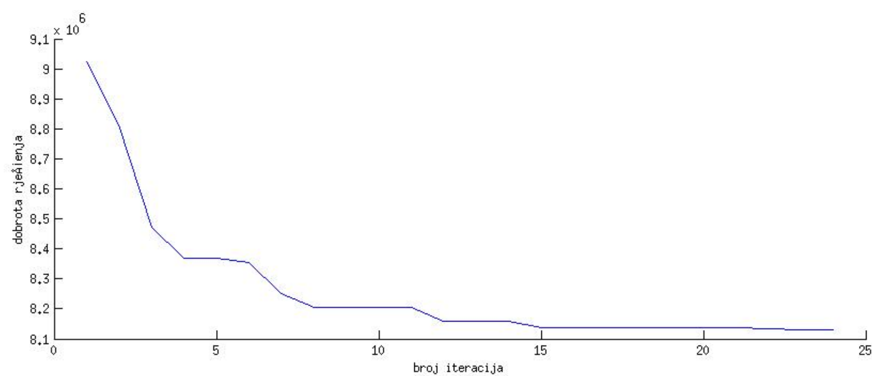
U ovoj cjelini cilj je paralelizirati prije opisani algoritam 11. Za realizaciju paralelizacije koristit ćemo topologiju povezanosti u grupi. Na raspolaganju su nam četiri procesora te će nam svaki procesor predstavljati jednu grupu. Svaki procesor će lokalno izvoditi algoritam 11 te s ostalim procesorima tehnologijom MPI razmjenjivati najbolje rješenje. Zamislimo da imamo četiri nezavisne populacije koje se razvijaju svaka na svom otoku. U svakoj generaciji iz populacije najbolja jedinka migrira na druge otoke. U prirodi su ovakve migracije česte i upravo ova ideja poslužit će za realizaciju paralelizacije. Kako bi implementirali traženi algoritam potrebno je modificirati osnovni algoritam. Potrebno je broj iteracija ravnopravno rasporediti po procesorima te kreirati paralelni ciklus. Dakle, na početku svake nove iteracije algoritam treba poslati svoje rješenje i primiti rješenja s tri preostala procesora. Prilikom slanja i primanja potrebno je pažljivo definirati poredak razmjene poruka. Procesor nula prvi šalje poruku svim ostalim procesorima. Ostali procesori moraju imati definirano primanje kako se ne bi dogodila blokada programa. Nakon procesora nula pravo na slanje ima procesor jedan, dok za ostale definiramo da prihvate poruku. Realizacija je prikazana algoritmom 14. Primijetimo da je problem elitizma i dalje ostao očuvan. U svakoj novoj iteraciji procesor čuva svoje najbolje rješenje, ali i rješenja svih drugih procesora. Kako u svojoj populaciji nakon određenog vremena ima velik broj kvalitetnih rješenja, procesom križanja nastaju kvalitetna rješenja što doprinosi bržem i kvalitetnijem poboljšanju. U svakoj iteraciji svaki procesor ima točno jedno slanje podataka i tri primanja podataka. Podaci su jedinke dane populacije. Analizu provodimo obzirom na broj uzoraka, to jest jedinki u populaciji. Na kraju prikazujemo ubrzanja u odnosu na sekvencijalni algoritam, kao i vizualni prikaz rješenja. Analizu dobrote rješenja provodit ćemo za jedan od procesora. Na ostalim procesorima zabilježeno je identično ponašanje.

Algoritam 14: Genetski algoritam

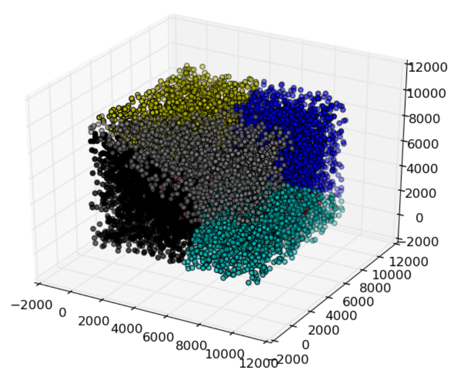
```
comm = GenrirajGrupe();
proc = comm.Rank();
populacija = inicijalnaPopulacija(podaci);
dobrota(populacija);
N = broj iteracija;
brojKrizanja = S;
for i od 1 do N/4 do
    if proc = 0 then
        comm.bcast(populacija.najbolji());
        migrant = comm.recv(source=1); novaPopulacija.dodaj(migrant);
        migrant = comm.recv(source=2); novaPopulacija.dodaj(migrant);
        migrant = comm.recv(source=3); novaPopulacija.dodaj(migrant);
    if proc = 1 then
        migrant = comm.recv(source=0); novaPopulacija.dodaj(migrant);
        comm.bcast(populacija.najbolji());
        migrant = comm.recv(source=2); novaPopulacija.dodaj(migrant);
        migrant = comm.recv(source=3); novaPopulacija.dodaj(migrant);
    if proc = 2 then
        migrant = comm.recv(source=0); novaPopulacija.dodaj(migrant);
        migrant = comm.recv(source=1); novaPopulacija.dodaj(migrant);
        comm.bcast(populacija.najbolji());
        migrant = comm.recv(source=3); novaPopulacija.dodaj(migrant);
    if proc = 3 then
        migrant = comm.recv(source=0); novaPopulacija.dodaj(migrant);
        migrant = comm.recv(source=1); novaPopulacija.dodaj(migrant);
        migrant = comm.recv(source=2); novaPopulacija.dodaj(migrant);
        comm.bcast(populacija.najbolji());
    while novaPopulacija nema dovoljno jedinki do
        while brojKrizanja do
            [R1,R2] = DvaIzPrveTrecineNajboljih(populacija);
            [D1, D2] = Krizaj(R1,R2);
            Mutiraj(D2);
            novaPopulacija.dodaj(D1, D2);
            brojKrizanja−−;
        K = sljedeciNajboljiIzPopulacije();
        Mutiraj(K);
        novaPopulacija.dodaj(K);
    populacija=novaPopulacija;
    dobrota(populacija);
vrati(populacija.najbolji());
```



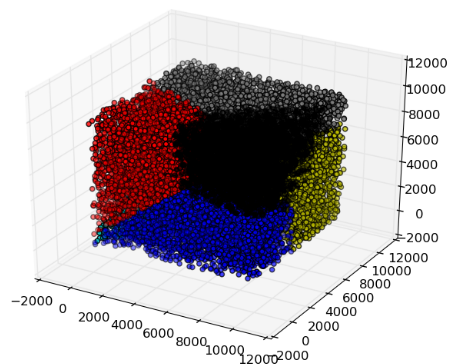
Slika 5.6: Utjecaj iteracija na kvalitetu rješenja za 10000 uzoraka



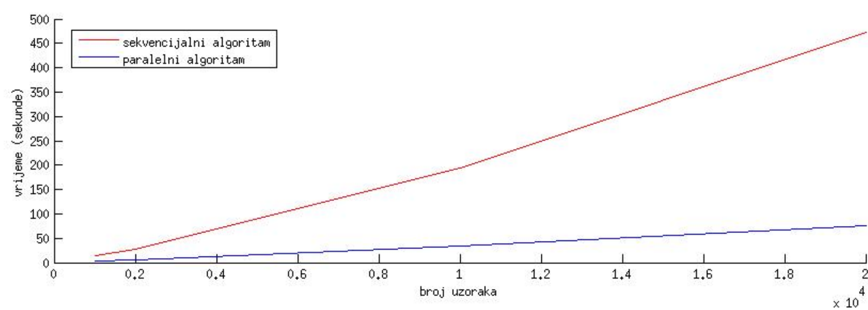
Slika 5.7: Utjecaj iteracija na kvalitetu rješenja za 20000 uzoraka



Slika 5.8: Prikaz za 10000 grupiranih uzoraka



Slika 5.9: Prikaz za 20000 grupiranih uzoraka



Slika 5.10: Vremenska usporedba sekvencijalnog i paralelnog algoritma

Iterativno poboljšanje algoritma vidljivo je na grafovima sa slika 5.6 i 5.7. Točnost algoritama vizualno je prikazana na slikama 5.8 i 5.9. Iz prikazane analize možemo zaključiti kako paralelizacijom nismo izgubili na točnosti u odnosu na sekvencijalni pristup rješenju. Međutim, vrijeme izvršavanja paralelnog algoritma daleko je manje od vremena potrebnog sekvencijalnom algoritmu za istu točnost. Graf vremenske složenosti prikazan je na slici 5.10.

Bibliografija

[1] *MPICH2 user documentation*, 2012, http://www.mcs.anl.gov/research/projects/mpich2staging/goodell/documentation/.

[2] *OpenMPI user documentation*, 2013, www.open-mpi.org/doc/.

[3] K. Bache i M. Lichman, *UCI Machine Learning Repository*, 2013, http://archive.ics.uci.edu/ml.

[4] S. Das, A. Abraham i A. Konar, *Metaheuristic Clustering*, Springer, 2009.

[5] R. Diestel, *Graph Teory*, Springer, 2012.

[6] Z. Drmač, M. Marušić, S. Singer, V. Hari, M. Rogina i S. Singer, *Numerička analiza*, (2003), web.math.pmf.unizg.hr/~rogina/2001096/num_anal.pdf.

[7] W Gropp i R. Graham, *MPI: A Message-Passing Interface Standard*, (2012).

[8] K. Jajuga i A. Sokołovski, *Classification, Clustering, and Data Analysis*, Springer, 2002.

[9] Z. Juhász, P. Kacsuk i D. Kranzlmüller, *Distributed and Parallel Systems*, Springer, 2005.

[10] J. Kogan, C. Nicholas i N. Teboulle, *Grouping Multidimensional Data*, Springer, 2006.

[11] F. Willmore, *Introduction to Parallel Computing*, (2012), tacc.utexas.edu/c/document_library.

[12] M. Čupić, *Prirodom inspirirani optimizacijski algoritmi*, (2009), fer.unizg.hr.

[13] M. Čupić, *Prirodom inspirirani optimizacijski algoritmi-Heuristike*, (2012), java.zemris.fer.hr/nastava/pioa/knjiga-0.1.2012-08-28.pdf.

Sažetak

Grupiranje podataka jedna je od najbitnijih metoda analize podataka te je u gotovo svim slučajevima prvi korak daljnje analize. U današnje vrijeme veliki servisi kao što su Google i Facebook u svojim bazama spremaju terabajte podataka. Podaci pristižu u prirodnom poretku, to jest bez definiranog redoslijeda te ih je potrebno grupirati u smislene cjeline. Problem grupiranja trivijalno je rješiv iscrpnom pretragom. Međutim, rezultati analize podataka potrebni su u realnom vremenu što nikako nije izvedivo iscrpnom pretragom budući da je problem grupiranja NP-težak problem.

Rad započinje s definiranjem osnovnih pojmova i matematičkim modeliranjem problema grupiranja. Nakon uvodnih definicija opisane su osnovne metode grupiranja: *hijerarhijsko* i *particijsko* grupiranje. Osim osnovnih metoda uveden je pojam evolucijske metode. Navedena metoda danas se gotovo uvijek koristi kada se na efikasan način pokušavaju pronaći rješenja teških problema. Posebna cjelina posvećena je upravo evolucijskoj metodi te su definirani osnovni operatori potrebni za razvoj algoritma. Evolucijske metode zajednički je naziv za skup algoritama inspiriranih prirodnim procesima. U radu je posebno opisan genetski algoritam kao predstavnik metode. Uveden je pojam *heuristike* te su konstruirani iterativni i evolucijski algoritam za problem grupiranja. Implementirane metode dobro rješavanju zadane probleme što je prikazano analizom. Vremenska složenost sekvencijalnih algoritama je prilično velika te su zbog toga paralelizirani. Nakon uvodnih napomena, opisana je tehnologija MPI te sve prednosti koje odlikuju paralelizaciju. Na kraju su uvedene razne topologije mreža procesora: *topologija povezane mreže*, *topologija povezanosti u grupi* te *topologija s vođom*. Objašnjeni su osnovni pojmovi sinkronizacije paralelnih procesora te opisane konkretne implementacije paralelnih algoritama. Iterativni sekvencijalni algoritam paraleliziran je korištenjem topologije s vođom, dok je za paralelizaciju evolucijske metode korištena topologija povezanosti u grupi. Implemenacije su testirane te su analizirani dobiveni rezultati.

Summary

Data clustering is one of the most important methods for data analysis and in most cases the first step in further analysis. Nowadays big services like Google and Facebook store terabytes of data. Data arrives in natural order, usually without a defined sequence and they need to be grouped into meaningful clusters. The problem of data clustering is trivial if we use thorough examination. However, we need a result in real time, which thorough examination can't offer since data clustering is an NP-hard problem.
This thesis offers basic definitions for data clustering and introduces mathematical modeling of data clustering problems. After definitions we describe basic methods for clustering: hierarchical and partitional clustering. In addition, we introduce the concept of evolutionary programming which represents the most popular method in solving NP-hard problems efficiently.
In section two particular attention is dedicated to genetic algorithms as a representative of the evolutionary algorithms class. Evolutionary method is a common name for methods inspired by natural processes. A description of heuristic algorithms concept and a construction of sequential algorithms for data clustering problem is given in section three. It is shown that the implemented methods are good in solving problems, but the big time complexity of sequential algorithms leads to parallel processes. Therefore, after the sequential solution, we introduce the concept of parallel programming via MPI technologies and the benefits it offers.
These technologies allow many types of network topologies for processors. This thesis describes three topological concepts: connected network, network with a master and group network. Also, we explain the synchronization among processors and demonstrate some parallel algorithm solutions for real-life problems. Synchronization is a hard problem because of a restricted set of functions (send, receive and broadcast). Using those functions we implemented all parallel algorithms in section five. After the implementation and testing we conducted a thorough analysis of results.

Životopis

Anto Čabraja rođen je 23. kolovoza 1990. godine u Derventi. Sa šest godina s roditeljima doseljava u Zagreb. U Zagrebu upisuje osnovnu školu Rudeš. Nakon uspješnog završetka osnovne škole upisuje matematičku gimnaziju X Gimnazija "Ivan Supek". Gimnaziju uspješno završava u proljeće 2009. godine te maturira s odličnim uspjehom. Tokom pohađanja gimnazije pokazuje izuzetno zanimanje za matematiku te maturira na temu *Pravokutni trokut u primjeni*. Svoje matematičko obrazovanje odlučuje nastaviti upisivanjem Prirodoslovno-matematičkog fakulteta u Zagrebu. Na fakultet se uspješno upisuje na jesen 2009. Tokom preddiplomskog studija aktivan je u udruzi eStudent gdje obavlja dužnosti voditelja i člana tima za organizaciju predavanja. Kroz djelovanje u udruzi sudjeluje na nekoliko natjecanja kao finalist: *Best Code Chalange*, *Case Study Competition*. Na preddiplomskom studiju razvija svoje programerske vještine. Nakon uspješno završenog preddiplomskog studija upisuje diplomski studij računarstva i matematike na istom fakultetu. Posebno zanimanje pokazuje za paralelno izračunavanje te za obradu i analizu velikih baza podataka. S obzirom na interese sudjeluje na natjecanju Mozgalo te zajedno s kolegicama Anamarijom Fofonjkom i Herminom Petric Maretić razvija aplikaciju za analizu dokumenata. S aplikacijom osvaja drugo mjesto. Osim navedenih aktivnosti pohađa predavanja o strojnom učenju i umjetnoj inteligenciji gdje izrađuje aplikaciju za detekciju tablica automobila. Za vrijeme trajanja studija vodi radionice i predavanja iz programerskih kolegija u sklopu Zagrebačkog centra za poduku (ZCP). Također sudjeluje na startup projektima. Jedan od istaknutijih projekata je izrada društvene mreže Egg. Za temu diplomskog rada odabrao je obradu *paralelnih algoritama za problem grupiranja podataka*.