Univerzitet u Beogradu
Matematički fakultet

MINIMUM SIZE ULTRAMETRIC TREE

Računarska Inteligencija

Knežević Ana
Ponjavić Pavle

Sadržaj

1 UVOD

Konstrukcija evolutivnih stabala na osnovu udaljenosti predstavlja važan problem u biologiji i taksonomiji, pri čemu postoji mnogo različitih modela koji motivišu algoritamske probleme. Većina optimizacionih problema u konstrukciji evolutivnih stabala pokazala se kao NP-teška. Važan model pretpostavlja da je stopa evolucije konstantna, što podrazumeva da će evolutivno stablo biti ultrametrijsko stablo. Ultrametrijsko stablo je korenito, označeno listovima, i stablo sa težinskim granama u kojem svaki unutrašnji čvor ima istu dužinu puta do svih listova u svom podstablu Pokazano je da je problem konstrukcije minimalnog ultrametrijskog stabla iz ne-metričke matrice udaljenosti NP-težak. Zbog nepristupačnosti mnogih od ovih problema, biolozi obično konstruišu stabla koristeći heurističke algoritme, kao što je UPGMA (*Unweighted Pair Group Method with Arithmetic mean).*

2 OPIS PROBLEMA

Definicija i Kontekst

- **Ultrametrijsko Stablo**: Ultrametrijsko stablo je specijalizovani oblik stabla koje se koristi u hijerarhijskom klasterovanju i srodnim oblastima. U takvim stablima, udaljenost (ili neusaglašenost) između bilo koja dva lista (tačke podataka) preko njihovog najnižeg zajedničkog pretka ne premašuje direktnu udaljenost između njih kako je navedeno u matrici.
- **Instanca**: Problem je dat matricom M dimenzija n×n pozitivnih celih brojeva. Ova matrica predstavlja udaljenosti između n elemenata (ili čvorova).
- **Rešenje**: Cilj je konstruisati stablo sa težinskim granama T(V,E) sa n listova. Svaki list odgovara jednom od elemenata predstavljenih u matrici M. Stablo treba da bude strukturirano tako da za bilo koja dva lista i i j, zbir težina na putanji između njih u stablu je najmanje jednaka udaljenosti M[i,j] između njih u matrici.

Karakteristike Rešenja

- **Mera**: Cilj je minimiziranje zbira težina grana u stablu, izraženo kao $\sum e \in E w(e)$, gde je $w(e)$ težina grane e.

- **Loše vesti**: U opisu problema se navodi da je pronalaženje optimalnog rešenja za ovaj problem računski zahtevno. Specifično, napomenuto je da se ovaj problem ne može aproksimirati unutar bilo kog polinomijalnog faktora (označeno kao n^{ϵ} za bilo koje ϵ>0).

Računski izazov

Teškoća nastaje iz potrebe da se konstruiše stablo koje ne samo da se pridržava ograničenja udaljenosti data matricom M već i minimizira zbir težina grana. Takav problem se često susreće u oblastima kao što su filogenetika, bioinformatika i klasterovanje, gde je potrebno pronaći reprezentaciju podataka u obliku stabla koja hvata inherentne sličnosti ili udaljenosti među tačkama podataka dok je istovremeno ekonomična u smislu definisane metrike (u ovom slučaju, zbir težina grana).

U praktičnim primenama, mogu se koristiti heurističke metode ili aproksimacije kako bi se došlo do skoro optimalnog rešenja s obzirom na teškoću dobijanja tačnog rešenja. Tehnike kao što su pohlepni algoritmi, dinamičko programiranje ili evolucijski algoritmi mogli bi se istražiti u zavisnosti od specifičnih zahteva i ograničenja domena primene.

3 REŠAVANJE PROBLEMA

U ovom projektu ćemo koristiti algoritam grube sile, MST, simulirano kaljenje i hijerarhijsko klasterovanje kako bismo uporedili rezultate.

3.1. Algoritam grube sile

```python
def generate_all_trees(n):
    nodes = range(n)
    for edges in itertools.combinations(itertools.combinations(nodes, 2), n-1):
        G = nx.Graph()
        G.add_nodes_from(nodes)
        G.add_edges_from(edges)
        if nx.is_tree(G):
            yield G

def assign_weights(G, M):
    for (u, v, d) in G.edges(data=True):
        G[u][v]['weight'] = max(M[u][v], M[v][u])

def tree_size(G):
    return sum(d['weight'] for (u, v, d) in G.edges(data=True))

def find_minimum_ultrametric_tree(M):
    n = len(M)
    min_size = float('inf')
    min_tree = None

    for tree in generate_all_trees(n):
        assign_weights(tree, M)
        size = tree_size(tree)
        if size < min_size:
            min_size = size
            min_tree = tree

    return min_tree, min_size
```

Slika1. kod algoritma grube sile

Ovaj Python kod je dizajniran za pronalaženje i vizualizaciju minimalnog ultrametrijskog stabla zasnovanog na matrici udaljenosti pročitanoj iz datoteka. Kroz detaljnu analizu, evo šta svaki deo koda radi:

1. Uvoz biblioteka: Kod počinje uvozom potrebnih biblioteka. `itertools` se koristi za generisanje kombinacija, `networkx` za manipulaciju i vizualizaciju grafova, `matplotlib.pyplot` za crtanje grafova, `os` za interakciju sa operativnim sistemom, i `time` za merenje vremena izvršavanja.
2. Funkcija za čitanje matrice iz datoteke (`read_matrix_from_file`): Ova funkcija otvara i čita datoteku koja sadrži matricu. Matrica se čita liniju po liniju, pri čemu se uklanjaju znakovi "[" i "]" i linija se deli na brojeve koji se konvertuju u cele brojeve pre nego što se dodaju u listu `matrix`.
3. Funkcija za generisanje svih stabala (`generate_all_trees`): Koristeći `itertools`, ova funkcija generiše sve moguće kombinacije grana za `n` čvorova, formirajući graf za svaku kombinaciju i proveravajući da li je to graf stablo koristeći `networkx.is_tree`. Ako jeste, stablo se vraća generatorom.
4. Funkcija za dodelu težina granama (`assign_weights`): Ova funkcija dodeljuje težine granama u stablu na osnovu matrice udaljenosti M. Težina grane je maksimum od `M[u][v]` i `M[v][u]`.
5. Funkcija za izračunavanje veličine stabla (`tree_size`): Računa ukupnu težinu stabla sabiranjem težina svih grana u stablu.
6. Funkcija za pronalaženje minimalnog ultrametrijskog stabla (`find_minimum_ultrametric_tree`): Generiše sva moguća stabla za dati broj čvorova i određuje stablo s minimalnom sumom težina granama. Vraća to stablo i njegovu ukupnu težinu.
7. Funkcija za obradu svih matrica u direktorijumu (`process_all_matrices`): Čita sve datoteke koje počinju sa "matrix" ili "mst" u zadatom direktorijumu, obrađuje svaku pomoću funkcije `find_minimum_ultrametric_tree`, meri vreme potrebno za obradu, i vizualizuje rezultate uz pomoć `matplotlib`.

Kod je koristan za analizu i vizualizaciju ultrametrijskih stabala, posebno u kontekstu naučnih istraživanja gde se često koriste kompleksne matrice udaljenosti. Mada je efikasan za manje instance zbog generisanja svih mogućih stabala, ovaj pristup može postati nepraktičan za veći broj čvorova zbog eksponencijalnog rasta broja mogućih stabala.

3.2. Minimum Spanning Tree

3.2.1 Opšti pregled MST (Minimum Spanning Tree - Minimalno Razapinjuće Drvo)

Minimalno razapinjuće drvo (MST) je osnovni koncept u računarskim naukama, posebno u oblastima dizajna mreža, teorije grafova i dizajna algoritama. MST ponderisanog grafa je podskup grana koje povezuju sve čvorove bez ciklusa, sa minimalnom mogućom ukupnom težinom grana.

Algoritmi za izračunavanje MST

Postoji nekoliko algoritama koji mogu izračunati MST:

1. **Kruskalov algoritam**: Ovaj algoritam funkcioniše tako što sortira sve grane grafa po težini u neopadajućem redosledu, a zatim dodaje grane MST-u, osiguravajući da se ne formiraju ciklusi. Često koristi strukturu podataka disjunktnih skupova (union-find) za praćenje koji su čvorovi u kojim komponentama.
2. **Primov algoritam**: Počinje od jednog čvora i proširuje MST dodavanjem najmanje teške grane koja povezuje MST sa nekim čvorom koji još nije u MST-u.

3.2.2 Ideja rešavanja minimalne veličine ultrametričkog stabla pomoću MST algoritma

Ultrametričko stablo ima specifična svojstva koja omogućavaju da svaka grana u stablu predstavlja ne samo direktnu povezanost između čvorova, već i najveću udaljenost (ili disimilaritet) na putu između dva čvora u stablu. Ideja korišćenja MST algoritama za konstrukciju ultrametričkog stabla se zasniva na tome što MST osigurava da ukupna težina svih grana bude minimalna, što može biti korisno pri modeliranju hijerarhijskih klastera ili filogenetskih stabala.

3.2.3. Pregled Koda

Funkcije u kodu:

- `load_matrix(file_path)`: Učitava matricu iz datoteke. Očekuje se da datoteka sadrži podatke odvojene razmacima, što je čest format za matrice u naučnim aplikacijama.

- `is_ultrametric(matrix)`: Proverava da li je data matrica ultrametrička. Ultrametrička nejednakost zahteva da za svako troje tačaka i,j,k važi da je udaljenost između i i j manja ili jednaka većoj udaljenosti između i i k i k i j.
- `create_graph_from_matrix(M)`: Kreira graf iz matrice udaljenosti, gde čvorovi predstavljaju indekse matrice, a težine grana su elementi matrice.
- `find_ultrametric_mst(M)`: Prvo proverava da li je matrica ultrametrička. Ako nije, ispisuje upozorenje. Zatim, koristi Kruskalov algoritam (preko biblioteke NetworkX) za nalazak MST-a. Ispisuje vreme potrebno za izračunavanje.
- `draw_mst(mst, filename)`: Crtanje MST koristeći biblioteku Matplotlib. Pozicije čvorova se automatski raspoređuju, a grane se označavaju sa njihovim težinama.
- `process_matrices(directory_path)`: Ova funkcija obrađuje sve datoteke u direktorijumu koje odgovaraju zadatom uzorku imena. Učitava svaku matricu, nalazi njen MST, izračunava ukupnu težinu MST-a i crta MST.

Kod je strukturiran tako da automatski procesira sve matrice u određenom direktorijumu, pružajući vizualizaciju i analizu svake od njih, što je korisno za evaluaciju i poređenje različitih ultrametričkih stabala iz generisanih ili postojećih podataka.

```python
def load_matrix(file_path):
    return np.loadtxt(file_path, dtype = int, delimiter=' ')

def create_graph_from_matrix(M):
    n = M.shape[0]
    G = nx.Graph()
    for i in range(n):
        for j in range(i + 1, n):
            G.add_edge(i, j, weight=M[i][j])
    return G

def find_ultrametric_mst(M):
    G = create_graph_from_matrix(M)
    start_time = time.time()
    mst = nx.minimum_spanning_tree(G, weight='weight', algorithm='kruskal')
    end_time = time.time()
    print("Time to compute MST: {:.4f} seconds".format(end_time - start_time))
    return mst

def draw_mst(mst, filename):
    pos = nx.spring_layout(mst, seed=42)
    nx.draw(mst, pos, with_labels=True, node_size=700, node_color='skyblue')
    labels = nx.get_edge_attributes(mst, 'weight')
    nx.draw_networkx_edge_labels(mst, pos, edge_labels=labels)
    plt.title(f"MST for {filename}")
    plt.show()


def process_matrices(directory_path):
    files = [f for f in os.listdir(directory_path) if f.startswith('mst') or f.startswith('matrix') or f.startswith('hc') and f.endswith('.txt
    for filename in files:
        file_path = os.path.join(directory_path, filename)
        print(f"Processing {filename}")
        M = load_matrix(file_path)
        mst = find_ultrametric_mst(M)
```

Slika 2. kod mst algoritma

3.3. Hierarchical Clustering

3.3.1. Opšte o Hijerarhijskom klasterovanju

Hijerarhijsko klasterovanje je metod analize podataka koji grupiše slične objekte u klasterima. Postoje dva pristupa: aglomerativni (odozdo nagore) i divizivni (odozgo nadole). Aglomerativni pristup počinje tretiranjem svakog pojedinačnog objekta kao zasebnog klastera, a zatim postepeno spaja klastera na osnovu njihove sličnosti. Divizivni pristup počinje sa svim objektima u jednom klasteru, koji se zatim postepeno deli na manje klastera.

Koraci u Aglomerativnom Hijerarhijskom Klasterovanju:

1. Inicijalizacija: Svaki objekt predstavlja jedan klaster.
2. Spajanje Klastera: Izaberite dva najbliža klastera i spojite ih u jedan.
3. Ažuriranje Matrice Rastojanja: Ažurirajte matricu rastojanja da reflektuje rastojanje između novonastalog klastera i svih ostalih klastera.
4. Ponavljanje: Nastavite spajanje klastera dok svi objekti ne budu u jednom klasteru.

Metrike za Merenje Rastojanja između Klastera:

- Single Linkage: Rastojanje između dva klastera definiše se kao najmanje rastojanje između bilo koja dva objekta u klasterima.
- Complete Linkage: Rastojanje između klastera definiše se kao najveće rastojanje između bilo koja dva objekta.
- Average Linkage: Rastojanje je prosječno rastojanje između svih parova objekata u klasterima.
- Ward’s Method: Minimizacija sume kvadrata rastojanja unutar svih klastera.

3.3.2. Kako je Hijerarhijsko klasterovanje iskorišćeno u kodu - metod linkage

U našem kodu, hijerarhijsko klasterovanje se koristi za izgradnju ultrametričkog stabla, što je specifičan tip hijerarhijskog stabla gde su sva rastojanja između listova tačno određena. Kod koristi metod 'single' za hijerarhijsko klasterovanje, što znači da koristi minimalna rastojanja između klastera.

- Matrica Rastojanja: Ovo je simetrična matrica gde element na poziciji i, j predstavlja rastojanje između elemenata i i j. Veoma je važno da ova matrica tačno odražava rastojanja ili nesličnosti između elemenata koje klasterujete.
- Funkcija linkage: Ova funkcija izvodi hijerarhijsko klasterovanje koristeći dati "metod povezivanja". Ovde se koristi metod 'single' linkage (algoritam najbliže tačke), koji je prikladan za obezbeđivanje ultrametrične osobine jer spaja klastera na osnovu najkraćeg minimalnog parnog rastojanja.
- Dendrogram: Vizualizuje rezultat hijerarhijskog klasterovanja. Visina dendrograma predstavlja ultrametrična rastojanja (tj. nivoe nesličnosti na kojima se klasteri spajaju).

Ovaj skript će učitati matricu rastojanja, izvršiti hijerarhijsko klasterovanje i prikazati dendrogram koji ilustruje ultrametrično stablo.

3.3.3. Pregled koda

```python
def build_ultrametric_tree(distance_matrix):
    if distance_matrix.shape[0] == distance_matrix.shape[1] and np.all(distance_matrix == distance_matrix.T):
        condensed_matrix = squareform(distance_matrix)
    else:
        condensed_matrix = distance_matrix

    linkage_matrix = linkage(condensed_matrix, method='single')

    dendrogram(linkage_matrix)
    plt.title('Hierarchical Clustering Dendrogram')
    plt.xlabel('Index of Point')
    plt.ylabel('Distance')
    plt.show()

    root_node, node_list = to_tree(linkage_matrix, rd=True)

    tree_size = np.sum(linkage_matrix[:, 2])

    return tree_size, root_node
```

Slika 3. Hijerarhijsko klasterovanje - metoda linkage

Ova funkcija prima matricu rastojanja, proverava da li je simetrična i kvadratna, a zatim izrađuje "kondenzovanu" matricu koju `linkage` funkcija koristi za izgradnju ultrametričnog stabla metodom 'single' linkage. Vizualizacija dendrograma prikazuje rezultate klasterovanja. Funkcija takođe izračunava ukupnu veličinu stabla kao sumu težina svih grana.

Korišćenjem kondenzovane matrice, `linkage` funkcija može efikasno izračunati hijerarhijsko klasterovanje bez nepotrebnog ponavljanja informacija, što je korisno za obradu velikih skupova podataka i smanjenje vremena izvršavanja.

3.4. UNION-FIND DATA STRUCTURE

3.4.1. Kako Kod Rešava Problem Minimalne Veličine Ultrametrijskog Stabla

Ideja je da se koristi Kruskalov pristup sortiranja grana po težini i postepenog dodavanja grana koje ne formiraju cikluse, što osigurava da krajnji rezultat bude stablo s minimalnom mogućom sumom težina grana. Ovo stablo bi teoretski trebalo da bude ultrametrijsko ako je matrica udaljenosti koja se koristi ultrametrička, mada sam kod ne proverava eksplicitno ultrametričnost matrice.

Ovaj Python kod koristi kombinaciju algoritama i struktura podataka kako bi konstruisao minimalno ultrametrijsko stablo na osnovu zadate matrice udaljenosti. Glavni alat koji se koristi za konstrukciju je struktura podataka Union-Find (poznata i kao Disjoint Set Union), a sama procedura uključuje sortiranje i selekciju grana koristeći pristup sličan Kruskalovom algoritmu za minimalna razapinjuća stabla.

```python
class UnionFind:
    def __init__(self, size):
        self.parent = list(range(size))
        self.rank = [0] * size

    def find(self, p):
        if self.parent[p] != p:
            self.parent[p] = self.find(self.parent[p])
        return self.parent[p]

    def union(self, p, q):
        rootP = self.find(p)
        rootQ = self.find(q)
        if rootP != rootQ:
            if self.rank[rootP] > self.rank[rootQ]:
                self.parent[rootQ] = rootP
            elif self.rank[rootP] < self.rank[rootQ]:
                self.parent[rootP] = rootQ
            else:
                self.parent[rootQ] = rootP
                self.rank[rootP] += 1
```

Slika4. union-find data structure

3.4.2. Detalji Komponenti Koda

1. **Union-Find Struktura**:
    - `__init__`: Inicijalizuje svaki element kao svoj sopstveni skup sa rangom 0.
    - `find`: Pronalazi reprezentativni element (root) skupa kome element pripada, koristeći tehniku kompresije puta za efikasnost.
    - `union`: Spaja dva skupa koristeći union by rank, minimizujući dubinu stabla.
2. **Funkcija `ultrametric_tree`**:
    - **Formiranje Liste Grana**: Kreira listu svih mogućih grana iz matrice udaljenosti, gde svaka grana sadrži težinu i indekse čvorova koje povezuje.
    - **Sortiranje Grana**: Sortira grane po težini u rastućem redosledu.
    - **Kreiranje Stabla**: Prolazi kroz sortiranu listu grana i koristi union-find strukturu da dodaje granu u stablo ako grana povezuje dva prethodno nepovezana čvora (čime se izbegava formiranje ciklusa).
3. **Funkcija `plot_tree`**:
    - Vizualizuje konstruisano stablo koristeći biblioteku NetworkX, gde čvorovi i grane su adekvatno prikazani sa težinama.
4. **Funkcija `load_matrices_from_directory`**:
    - Učitava sve matrice iz zadatog direktorijuma, pretpostavljajući da su datoteke pravilno formatirane i imenovane.
5. **Funkcija `process_matrices`**:
    - Za svaku matricu učitanu iz direktorijuma, konstruiše minimalno ultrametrijsko stablo, izračunava ukupnu težinu stabla, i meri vreme potrebno za procesiranje.

3.4.3. Zaključak

Kod je efikasan za konstrukciju minimalnih ultrametrijskih stabala kada su matrice udaljenosti već ultrametričke. Za matrice koje nisu ultrametričke, dodatne transformacije ili provere bi bile potrebne kako bi se osigurala validnost ultrametričnih svojstava dobijenog stabla.

3.5. Simulirano kaljenje

3.5.1. Opis algoritma simuliranog kaljenja (Simulated Annealing)

Simulirano kaljenje (SA) je optimizacijski algoritam inspirisan procesom kaljenja metala. Temelji se na principu sporog hlađenja kako bi se metalu omogućilo da dostigne svoje stanje minimalne unutrašnje energije, što je analogija za pronalazak globalnog minimuma u optimizacijskom problemu. Algoritam kombinuje elemente slučajne pretrage sa mehanizmom prihvatanja koji omogućava privremeno prihvatanje lošijih rešenja, čime se izbegava zaglavljivanje u lokalnim minimumima.

Parametar temperature T kontroliše verovatnoću prihvatanja lošijih rešenja. Na visokim temperaturama, algoritam je skloniji eksploraciji, dok se smanjenjem temperature povećava eksploatacija trenutne oblasti prostora rešenja. Ovaj proces se ponavlja dok temperatura ne dostigne unapred definisanu minimalnu vrednost ili se ne ispuni neki drugi kriterijum zaustavljanja.

3.5.2 Ideja i implementacija koda

Kod koji ste pružili koristi simulirano kaljenje za rešavanje problema pronalaženja minimalnog ultrametrijskog stabla na osnovu date matrice udaljenosti. Ultrametrijsko stablo je specijalni tip stabla u kojem postoji restrikcija na udaljenosti između čvorova, korisno u različitim primenama kao što su filogenetičko stablo ili klasterska analiza.

Kako kod funkcioniše:

1. Učitava matrice udaljenosti iz datoteka u zadatom direktorijumu.
2. Za svaku matricu, gradi početno stablo koristeći MST (Minimalno razapinjuće stablo) koje minimizuje zbir težina.
3. Implementira simulirano kaljenje nad tim stablom koristeći prilagođene funkcije za generisanje susednih rešenja (menjanjem ivica stabla) i evaluaciju njihovih troškova.
4. Prati najbolje rešenje i njegov trošak kroz iteracije.
5. Vizualizuje konačno stablo i ispisuje metrike performansi.

3.5.3. Detaljan pregled koda

Funkcije i njihove uloge:

1. `load_matrices_from_directory(directory_path)`: Učitava matrice udaljenosti iz tekstualnih datoteka u zadatom direktorijumu. Datoteke se filtriraju na osnovu prefiksa ('matrix', 'mst', 'hc') i sufiksa ('.txt').

2. `initial_tree(M)`: Kreira početno minimalno razapinjuće stablo koristeći NetworkX biblioteku, na osnovu matrice udaljenosti MM.
3. `cost_function(T)`: Računa ukupni trošak (sumu težina) za dato stablo TT.
4. `get_neighbor(T, M)`: Generiše susedno stablo menjajući jednu ivicu u trenutnom stablu. Ova funkcija pazi da novo stablo ostane validno (da ne izgubi svojstva stabla).
5. `simulated_annealing(M, T_init, T, max_iter=1000)`: Implementacija algoritma simuliranog kaljenja. Počinje sa početnim stablom i početnom temperaturom, iterativno generiše susedna stabla i ažurira trenutno najbolje rešenje u skladu sa principima algoritma.
6. `process_matrices(directory_path)`: Glavna funkcija koja koordinira učitavanje matrica, izvršavanje simuliranog kaljenja za svaku matricu, i vizualizaciju konačnih stabala.

Kroz ovaj kod, poseban naglasak je stavljen na mogućnost algoritma da istraži prostor rešenja i postepeno konvergira ka optimalnom rešenju, čak i ako to podrazumeva povremeno prihvatanje lošijih rešenja u procesu optimizacije.

```
def simulated_annealing(M, T_init, T, max_iter=1000):
    current_T = T_init
    current_cost = cost_function(current_T)
    best_T = current_T
    best_cost = current_cost

    for i in range(max_iter):
        T_new = get_neighbor(current_T.copy(), M)
        new_cost = cost_function(T_new)
        delta_E = new_cost - current_cost

        if delta_E < 0 or random.random() < math.exp(-delta_E / T):
            current_T = T_new
            current_cost = new_cost

            if new_cost < best_cost:
                best_T = current_T
                best_cost = new_cost

        T *= 0.99

    return best_T, best_cost
```

Slika 5. Simulirano kaljenje

4. ANALIZA REZULTATA

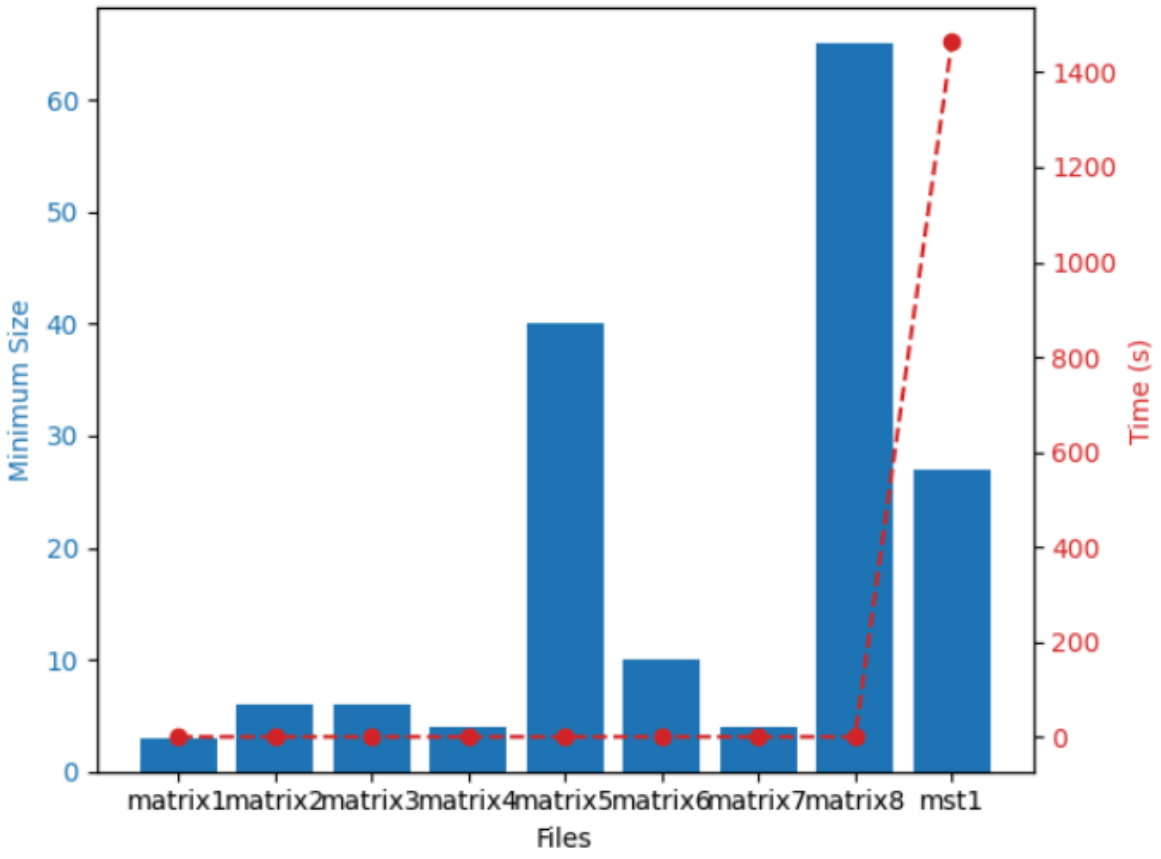
4.1. Rezultati - brute force

Datoteke matrix (matrix1.txt do matrix8.txt):

Minimalna veličinata izračunatih stabala varirala je od 3 do 65 . Dimenzija testiranih matrica je od 3x3 do 5x5. Vreme obrade ovih matrica bilo je relativno nisko, obično ispod 0,03 sekunde, što ukazuje na efikasno rukovanje ovim specifičnim veličinama i strukturama podataka.

Datoteka MST (mst1.txt):

Veličina ove matrice 9x9. Vreme obrade bilo je izuzetno visoko, 1463,02 sekundi.



Slika6. Rezultati koji su dobijeni algoritmom grube sile

4.2. Rezultati - MST

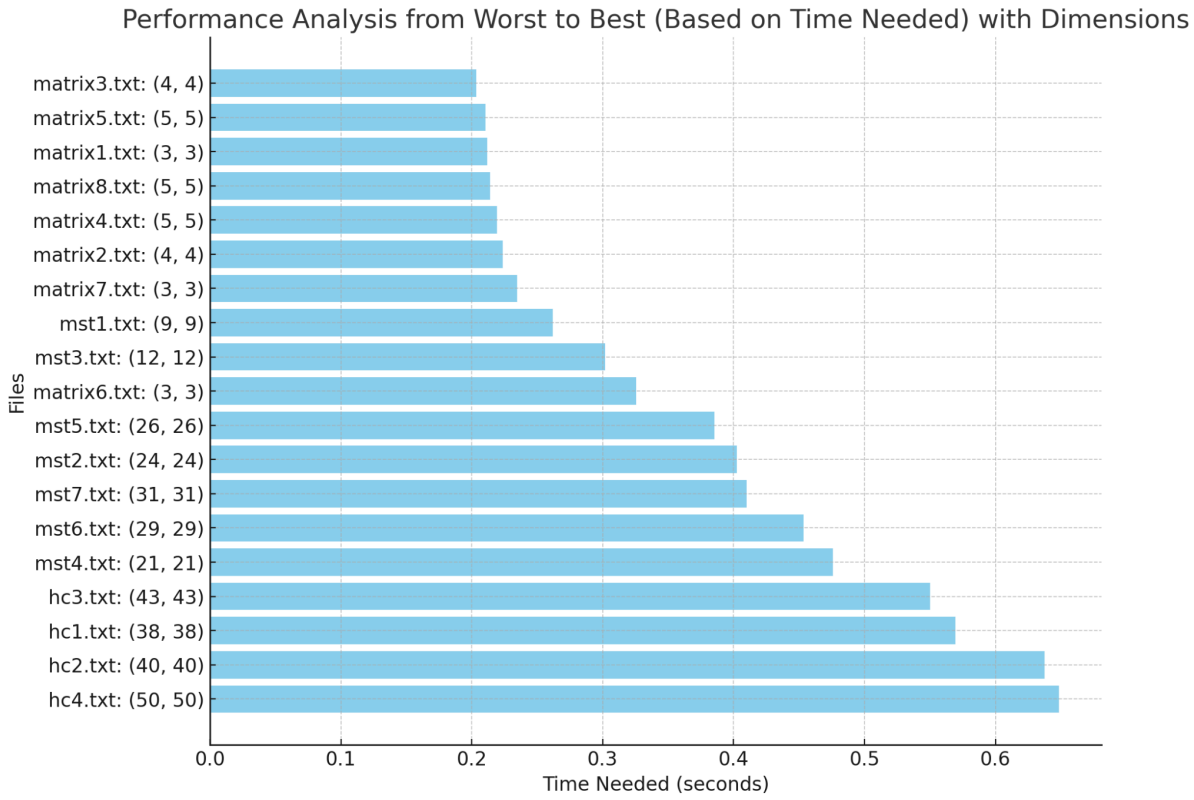
U sledećoj tabeli se može videti da su najbrže završile matrice iz datoteka matrix, za koje smo već rekli da su malih dimenzija. Najviše vremena je trebalo za datoteku hc3.txt u kojoj se nalazi matrica veličine 43x43, međutim to vreme je neuporedivo brže u odnosu na vreme koje je bilo potrebno za matricu dimenzije 9x9 algoritmom grube sile.

File	Elapsed Time (seconds)
matrix3.txt	0.000100
matrix7.txt	0.000100
matrix6.txt	0.000100
matrix1.txt	0.000100
matrix8.txt	0.000200
matrix5.txt	0.000200
matrix4.txt	0.000200
matrix2.txt	0.000200
mst1.txt	0.000300
mst3.txt	0.000500
mst4.txt	0.000600
mst2.txt	0.001100
mst7.txt	0.001300
mst5.txt	0.001300
hc2.txt	0.002400
hc1.txt	0.002900
hc4.txt	0.003600
mst6.txt	0.004600
hc3.txt	0.005700

Slika7. Rezultati koji su dobijeni algoritmom mst

4.3. Rezultati - Hierarchical Clustering

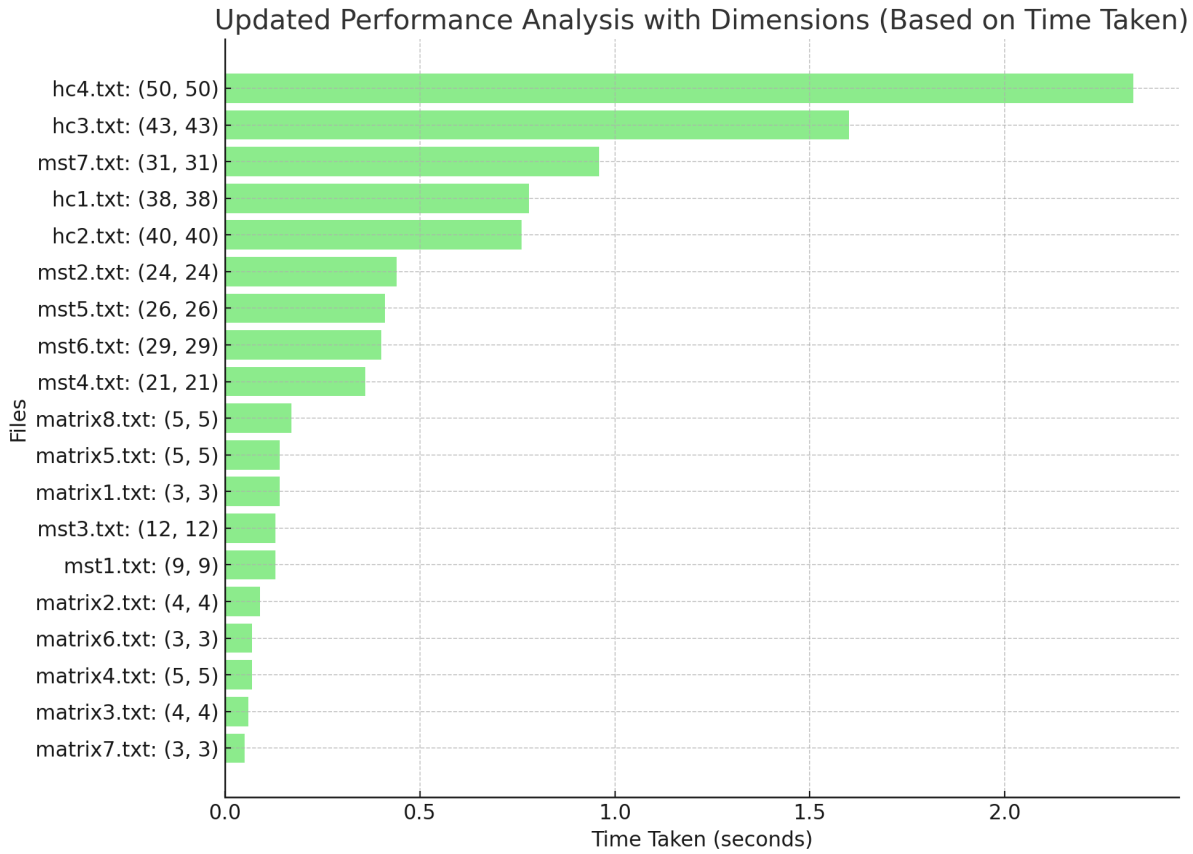
Na sledećem grafikonu možete videti koliko je bilo potrebno vremena za izvršavanje različitih dimenzija matrica upotrebom algoritma hijerarhijsko klasterovanje.



Slika8. Rezultati koji su dobijeni algoritmom hijerarhijsko klasterovanje

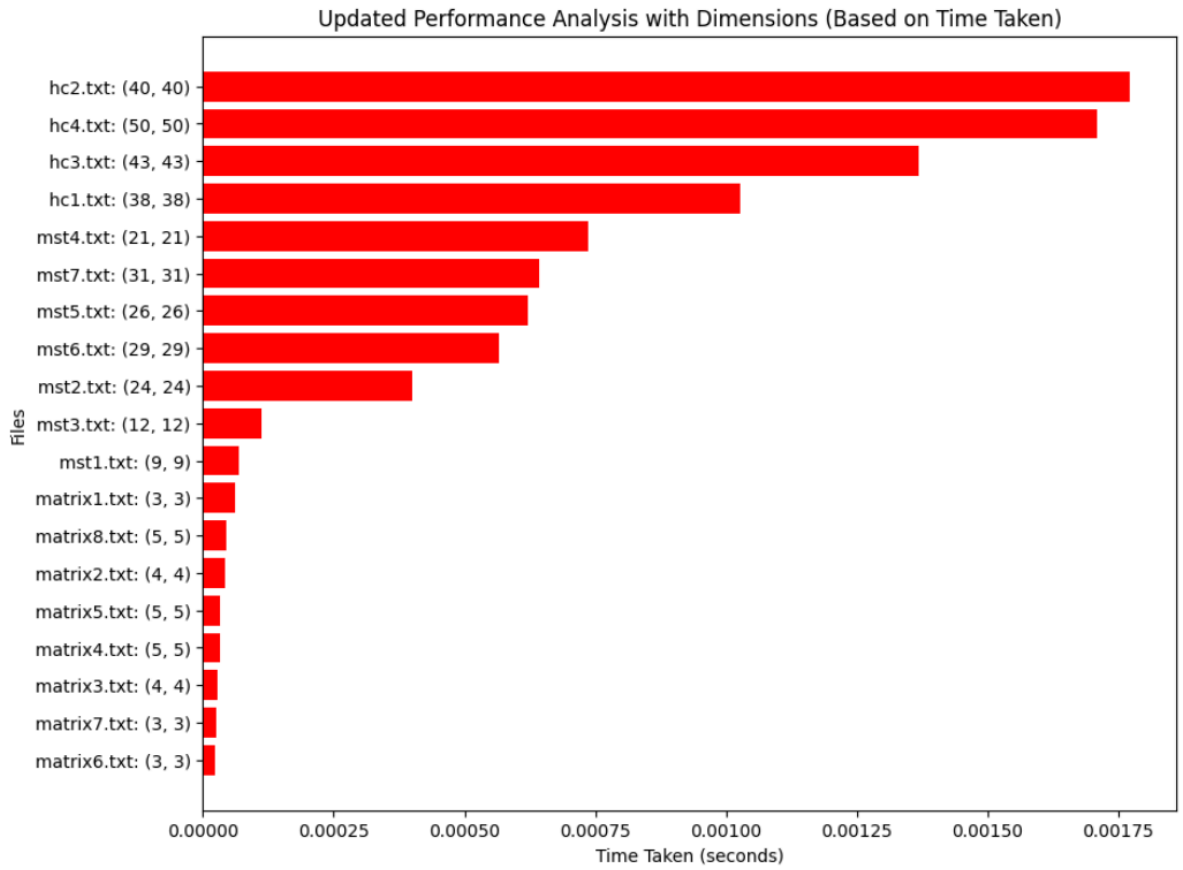
4.4. Rezultati - Simulated Annealing

Grafikon vizualizuje performanse svake datoteke na osnovu potrebnog vremena za obradu, sa dimenzijama matrice uključenim u oznake datoteka radi jasnoće. Svaka traka predstavlja vreme potrebno za obradu datoteke, što omogućava brzo identifikovanje datoteka koje zahtevaju duže vreme obrade. Datoteka hc4.txt, sa najvećim dimenzijama, pokazuje najduže vreme obrade, dok su manje datoteke matrica kao što su matrix7.txt i matrix3.txt obrađene najbrže. Ovaj raspored pomaže u proceni uticaja veličine matrice na vreme obrade.



Slika8. Rezultati koji su dobijeni algoritmom simulirano kaljenje

4.5. Rezultati - Union-find data structure



Slika9. Rezultati koji su dobijeni korišćenjem union-find strukture

5. Složenost algoritama

5.1. Simulirano kaljenje (Simulated Annealing)

Ovaj pristup uključuje iterativno pokušavanje poboljšanja rešenja pravljenjem malih slučajnih promena (pretraga u okolini) i uslovno prihvatanje ovih promena na osnovu verovatnoće koja se smanjuje tokom vremena (temperatura). Evo kako izgleda računanje kompleksnosti:

- Izgradnja početnog stabla (initial_tree): Izgradnja početnog minimalnog razapinjućeg stabla (MST) koristeći Kruskalov algoritam koji je tipično O(ElogE), gde je E broj ivica.

- Funkcija suseda (get_neighbor): Uključuje modifikaciju strukture stabla, što ima kompleksnost O(1) za dodavanje i uklanjanje ivica, ali proveravanje strukture stabla (povezanost i acikličnost) može biti do O(N), gde je N broj čvorova.
- Petlja simuliranog kaljenja: Iterira maksimalno max_itermax_iter puta. Svaka iteracija potencijalno uključuje ponovno izračunavanje troškova stabla, što je O(N), i promenu strukture stabla kako je opisano gore.

Ukupna kompleksnost: O(max_iter×(N+NlogN)), gde je faktor NlogN dominantan zbog kreiranja MST-a u najgorem slučaju.

5.2. Union-Find (Minimalno ultrametričko stablo)

Ovaj metod koristi strukturu union-find za izgradnju stabla, osiguravajući da ostane aciklično i povezano dodavanjem najmanje skupih ivica prvo.

- Sortiranje ivica: O(ElogE), gde je E ukupan broj ivica.
- Operacije union-find: Svaka operacija unije i pronalaženja može se izvršiti skoro konstantno, O(α(N)), zbog kompresije puta i unije po rangu, gde je α inverzna Ackermann funkcija, vrlo sporo rastuća funkcija.

Ukupna kompleksnost: O(ElogE+Eα(N)), dominirana sortiranjem ivica.

5.3. Brute Force

Ovaj pristup generiše sva moguće stabla i proverava svako od njih za ukupnu težinu da bi se našlo minimalno ultrametričko stablo.

- Generisanje svih stabala: Postoji N^{N-2} razapinjućih stabala za kompletni graf sa N čvorovima (Kejlijeva formula), i svako je generisano i provereno.
- Izračunavanje težine stabla: O(N) za svako stablo.

Ukupna kompleksnost: O(N^{N-1}), eksponencijalna kompleksnost, čineći ovaj pristup nepraktičnim za velike N.

5.4. Hijerarhijsko klasterovanje

Ova tehnika gradi dendrogram koristeći algoritam hijerarhijskog klasterovanja, koji se može izvršiti na nekoliko načina; ovde se koristi metoda jednostrukog povezivanja.

- Konstrukcija matrice povezivanja: Tipično O(N^2logN) pošto uključuje sortiranje parova udaljenosti.
- Izgradnja dendrograma: O(N^2) u slučaju izgradnje kompletnog stabla.

Ukupna kompleksnost: O(N^2logN), dominirana konstrukcijom matrice povezivanja.

5.5. MST (Minimalno razapinjuće stablo)

Direktno računa MST, što je slično prvom koraku pristupa simuliranog kaljenja ali se fokusira samo na proizvodnju MST-a.

- Kreiranje grafa iz matrice i računanje MST-a: O(ElogE), gde je E broj ivica, koristeći Kruskalov algoritam.

Ukupna kompleksnost: O(ElogE), efikasno za retke grafike.

6. Zaključak

- Efikasnost: Brute force metoda je najmanje efikasna zbog svoje eksponencijalne kompleksnosti. MST i Union-Find metode su najefikasnije po pitanju kompleksnosti, posebno pogodne za grafike gde je E mnogo manje od N^2.
- Primena: MST i Union-Find su odlični za grafike gde postoji jasna hijerarhija težina ivica. Simulirano kaljenje pruža fleksibilnost u izbegavanju lokalnih minimuma, što ga čini pogodnim za složenije pejzaže. Hijerarhijsko klasterovanje je posebno korisno za analizu klasterovanja više od same izgradnje stabla.
- Skalabilnost: MST, Union-Find i Hijerarhijsko klasterovanje dobro skaliraju sa povećanjem N, za razliku od brute force pristupa. Skalabilnost simuliranog kaljenja značajno zavisi od parametra max_iter i specifične implementacije funkcije suseda.

Svaki metod ima svoje prednosti i treba ga izabrati na osnovu specifičnih potreba problema, kao što su veličina ulaza i preciznost zahteva izlaza.