SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT - BIOINFORMATIKA

Algoritam "Neighbour joining"

Filip Beć
Zorana Ćurković
Goran Gašić
Melita Kokot
Dino Šantl
Igor Smolkovič
Mentor: *Dr. sc. Mirjana Domazet-Lošo*
Doc. dr. sc. Mile Šikić

Zagreb, siječanj 2014.

SADRŽAJ

1. Opis algoritma

Algoritam *Neighbour joining* služi za izgradnju filogenetskog stabla. Ulaz algoritma su evolucijske udaljenosti (matrica udaljenosti čvorova). Izlaz algoritma je stablo s težinskim bridovima. Cilj algoritma je izgraditi minimalno razapinjuće filogenetsko stablo. Algoritam nužno ne pronalazi takvo stablo, ali rješenja su često minimalna razapinjuća filogenetska stabla ili blizu toga [1]. Glavni razlog je smanjenje vremenske složenosti jer u praksi nije izvedivo ispitivanje svih mogućih stabala. Stablo se gradi od dna prema vrhu. Algoritam je pohlepan jer za jednom sparene čvorove ne ispituje točnost tog koraka.

Algoritam se sastoji od dva dijela: uparivanje čvorova i završni korak. Algoritam kreće od matrice udaljenosti nad kojom zaključuje koja dva čvora je potrebno upariti. Kada su poznata dva čvora koja se trebaju upariti, stvara se novi čvor koji se spaja s dva odabrana. Dva odabrana čvora brišu se iz matrice udaljenosti, a novi čvor se dodaje u matricu udaljenosti. Taj postupak se izvršava $N - 3$ puta, gdje je N početni broj čvorova. Završni korak uzima zadnja tri čvora koja su ostala u matrici udaljenosti, stvara novi čvor i spaja novi čvor s tri zadnja čvora u matrici udaljenosti. Formalno algoritam izgleda ovako:

1. **Ulaz**: matrica udaljenosti
2. Na temelju trenutne matrice udaljenosti izračunaj matricu **Q**
3. U matrici **Q** pronađi najmanju vrijednost $Q(i, j)$ i pripadajuće čvorove (i, j).
4. Stvori novi čvor w i dva nova brida: (w, i) i (w, j) te izračunaj udaljenosti $d(w, i)$ i $d(w, j)$ i zapiši ih na pripadajući brid.
5. Iz matrice udaljenosti izbriši čvorove i i j, te dodaj u matricu novi čvor w - potrebno je izračunati udaljenosti do novoga čvora w
6. Ako postoji više od tri čvora u matrici udaljenosti skoči na korak 2.

7. Stvori novi čvor w i napravi tri brida s preostalim čvorovima u matrici udaljenosti te izračunaj i pridruži vrijednost bridovima

1.1. Matrica Q

Matrica **Q** pomoćna je matrica iz koje se određuje par čvorova koji će biti susjedni. Matrica **Q** kao kriterij uzima, osim udaljenosti čvorova i i j, i utjecaj njihovih susjeda. Dokaz da najmanja vrijednost matrice **Q** pripada susjednim čvorovima dan je u [1]. Matrica **Q** izračunava se na sljedeći način:

$$Q(i,j) = (r-2)d(i,j) - \sum_{k=1}^{r} d(i,k) - \sum_{k=1}^{r} d(j,k) \tag{1.1}$$

, gdje su i, j indeksi čvorova, r je trenutni broj čvorova u matrici udaljenosti. Oznake $d(i,j)$, $d(i,k)$ i $d(j,k)$ predstavljaju vrijednosti u matrici udaljenosti. Čvorovi i i j moraju biti različiti.

1.2. Računanje vrijednosti bridova

Nakon što se stvore novi bridovi u stablu, potrebno je odrediti njihovu vrijednost. U svakom koraku uparivanja stvore se dva nova brida. U završnom koraku stvore se tri nova brida.

Nakon što se čvorovi i i j proglase susjednima, stvara se novi čvor w. Potrebno je odrediti udaljenosti $d(i,w)$ i $d(j,w)$. Udaljenosti se određuju prema formulama:

$$d(i,w) = \frac{1}{2}d(i,j) + \frac{1}{2(r-2)}\left[\sum_{k=1}^{r} d(i,k) - \sum_{k=1}^{r} d(j,k)\right] \tag{1.2}$$

, gdje je r trenutni broj čvorova u matrici udaljenosti.

Kako vrijedi $d(i,j) = d(i,w) + d(j,w)$, iz toga slijedi:

$$d(j,w) = d(i,j) - d(i,w) \tag{1.3}$$

1.3. Računanje udaljenosti do novog čvora

Pri dodavanju novog čvora u matricu udaljenosti potrebno je izračunati udaljenost od svih starih čvorova do novog čvora w. Udaljenost se računa prema formuli:

$$d(k, w) = \frac{1}{2}\left[d(i, k) + d(j, k) - d(i, j)\right] \tag{1.4}$$

, gdje je k bilo koji stari čvor u matrici udaljenosti. Čvorovi i i j su upravo spojeni čvorovi.

1.4. Složenost algoritma

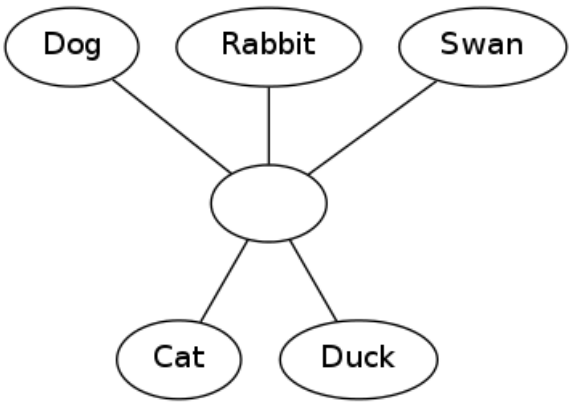
Algoritam mora izvršiti korak uparivanja $N - 3$ puta, gdje je N broj čvorova u početnoj matrici udaljenosti. U svakom koraku potrebno je izračunati matricu **Q** koja je dimenzije $r \times r$, gdje je r trenutni broj čvorova. Ako se pretprocesiraju sume u prije navedenim formulama u $O(N^2)$ vremenskoj složenosti, tada se matrica **Q** može izračunati u $O(N^2)$ vremenskoj složenosti. Zbog toga je ukupna vremenska složenost algoritma $O(N^3)$. Memorijska složenost je $O(N^2)$ jer se pamti matrica udaljenosti.

2. Primjer izvođenja

Raspolažemo skupom od 5 taksona[1] $(\text{Dog}, \text{Cat}, \text{Rabbit}, \text{Duck}, \text{Swan})$ te pridanim udaljenostima među njima definiranim matricom udaljenosti:

	Dog	Cat	Rabbit	Duck	Swan
Dog	0	5	17	15	13
Cat	5	0	9	19	14
Rabbit	17	9	0	20	16
Duck	15	19	20	0	12
Swan	13	14	16	12	0

Algoritam će biti proveden u $N - 3 = 2$ koraka.



Slika 2.1: Početni graf - moguće su sve veze

[1] Koristimo oba izraza ovisno o kontekstu - takson i čvor

2.1. Korak 1

Izračunamo matricu Q te tražimo par (i, j) koji ima najmanju vrijednost:

	Dog	Cat	Rabbit	Duck	Swan
Dog	0	-82	-61	-71	-66
Cat	-82	0	-82	-56	-60
Rabbit	-61	-82	0	-68	-69
Duck	-71	-56	-68	0	**-85**
Swan	-66	-60	-69	-85	0

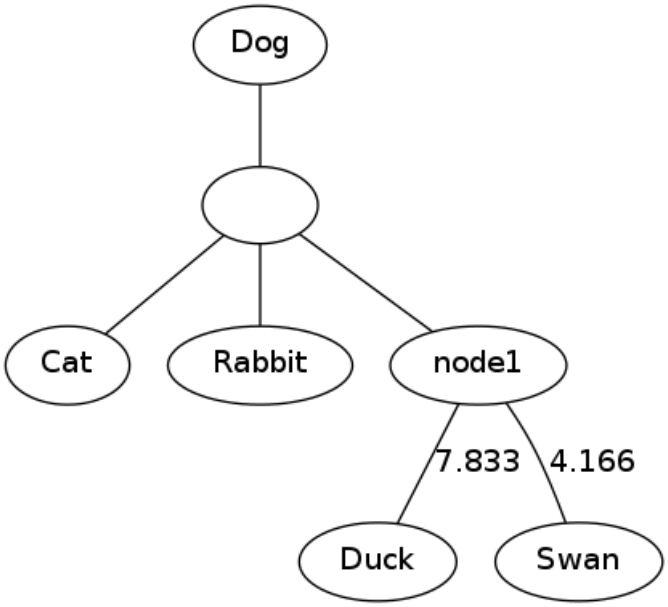
$i = \text{Duck}, j = \text{Swan}, Q_{min} = -85$. Taksone **Duck** i **Swan** spajamo u novi čvor node1 te računamo udaljenosti:

$d(\text{Duck}, \text{node1}) = 7.833$,

$d(\text{Swan}, \text{node1}) = 4.166$

Preostale udaljenosti $d(\text{k}, \text{node1})$ definirane su novom matricom udaljenosti :

	Dog	Cat	Rabbit	node1
Dog	0	5	17	8
Cat	5	0	9	10.5
Rabbit	17	9	0	12
node1	8	10.5	12	0



Slika 2.2: Trenutno stablo

2.2. Korak 2

Izračunamo matricu Q te tražimo par (i, j) koji ima najmanju vrijednost:

	Dog	Cat	Rabbit	node1
Dog	0	**-44.5**	-34	-44.5
Cat	-44.5	0	-44.5	-34
Rabbit	-34	-44.5	0	-44.5
node1	-44.5	-34	-44.5	0

$i = \text{Dog}, j = \text{Cat}, Q_{min} = -44.5$. Taksone **Dog** i **Cat** spajamo u novi čvor node2 te računamo udaljenosti:

$d(\text{Dog}, \text{node2}) = 3.875$,

$d(\text{Cat}, \text{node2}) = 1.125$

Preostale udaljenosti $d(k, \text{node2})$ definirane su novom matricom udaljenosti :

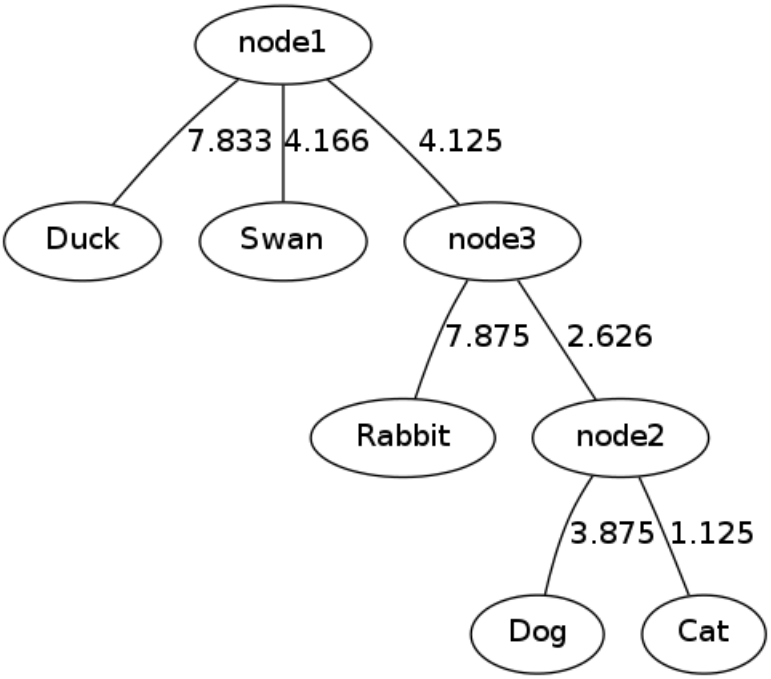
	node2	Rabbit	node1
node2	0	10.5	6.75
Rabbit	10.5	0	12
node1	6.75	12	0

Potrebno je još spojiti preostala 3 čvora. U svrhu spajanja stvaramo novi čvor node3. Poznate su nam udaljenosti $d(\text{node2}, \text{Rabbit})$, $d(\text{node2}, \text{node1})$ i $d(\text{Rabbit}, \text{node1})$. Temeljem tih udaljenosti možemo izračunati posljednja tri luka.

$d(\text{node3}, \text{node2}) = 0.5 \cdot (d(\text{node2}, \text{node1}) + d(\text{node2}, \text{Rabbit}) - d(\text{node1}, \text{Rabbit}))$
$= 2.625$

$d(\text{node3}, \text{ node1}) = 0.5 \cdot (d(\text{node1}, \text{Rabbit}) + d(\text{node2}, \text{Rabbit}) - d(\text{node1}, \text{node2}))$
$= 4.125$

$d(\text{node3}, \text{Rabbit}) = 0.5 \cdot (d(\text{node1}, \text{Rabbit}) + d(\text{node1}, \text{node2}) - d(\text{node2}, \text{Rabbit}))$
= 7.875



Slika 2.3: Konačno stablo

3. Testiranje i usporedbe

3.1. Korišteni jezici

Algoritam je implementiran u šest različitih jezika i to prema podjeli:

- Filip Beć - **Objective-C**
- Zorana Ćurković - **Python**
- Goran Gašić - **Java**
- Melita Kokot - **Ruby**
- Dino Šantl - **C**
- Igor Smolkovič - **C++**

3.2. Infrastruktura

Algoritam kao ulaz uzima matricu udaljenosti. U praksi se algoritam *Neighbuor joining* koristi u kombinaciji s algoritmima poravnanja. Nakon što se genetski nizovi poravnaju računa se udaljenost između njih te je taj rezultat ulaz za algoritam *Neighbour joining*. Kako ne bi svaka implementacija (jezik) posebno računala poravnanja, napravljena je infrastruktura koja sama napravi poravnanja te preda udaljenosti kao ulaz u različite implementacije. Osim pripreme ulaza, infrastruktura se pobrine za crtanje dobivenog grafa, kako bi se lakše vidjelo rješenje. Infrastruktura se nalazi u mapi *"Infra/"* u repozitoriju.

3.2.1. Priprema ulaza - poravnanja

Za pripremu ulaza koristi se skripta *alignament.py* napisana u *Pythonu*. Koristi se biblioteka *BioPython* gdje se nalazi programska podrška za *Muscle*. *Muscle* omogućava višestruko poravnanje genetskih nizova. Kao ulaz daje se datoteka u *FASTA* formatu. Izlaz je skup genetskih nizova iste duljine.

Nakon koraka poravnanja potrebno je izračunati udaljenosti među dobivenim nizovima. Udaljenost se računa kao postotak poklapanja pripadajućih znakova u nizu gdje se zanemaruju procijepi. Na kraju se iz dobivenih postotaka računa udaljenost po Jukes-Cantorovom modelu.

Na temelju indeksa genetskog niza i udaljenosti između nizova stvara se ulaz u algoritam *Neighbour joining* tako da se u prvi red zapiše broj genetskih nizova, a nakon toga u $\frac{N(N-1)}{2}$ redova se zapisuju udaljenosti između nizova kao: "i j distance", gdje je uvijek $i < j$, a i i j su pripadajući identifikatori nizova.

3.2.2. Crtanje grafa

Kako bi se rezultati algoritma mogli vizualizirati, generira se datoteka u *DOT* jeziku koji služi za opisivanje grafova. Tako napisan program (opis grafa) može se prevesti u neki od slikovnih formata (npr. png).

Implementacije algoritma *Neighbour joining* kao izlaz daju skup bridova u obliku "i j distance", gdje su i i j čvorovi u stvorenom stablu, a *distance* udaljenost između njih. Skripta *generate_graph.py* na temelju takvog izlaza (skupa bridova) generira opis grafa u *DOT* jeziku.

3.2.3. Pokretanje primjera

Skripta napisana u *bash* jeziku, *nj_process*, spaja tri komponente infrastrukture:

- poravnanje nizova
- implementaciju *Neighbour joining* algoritma
- vizualizaciju dobivenog grafa.

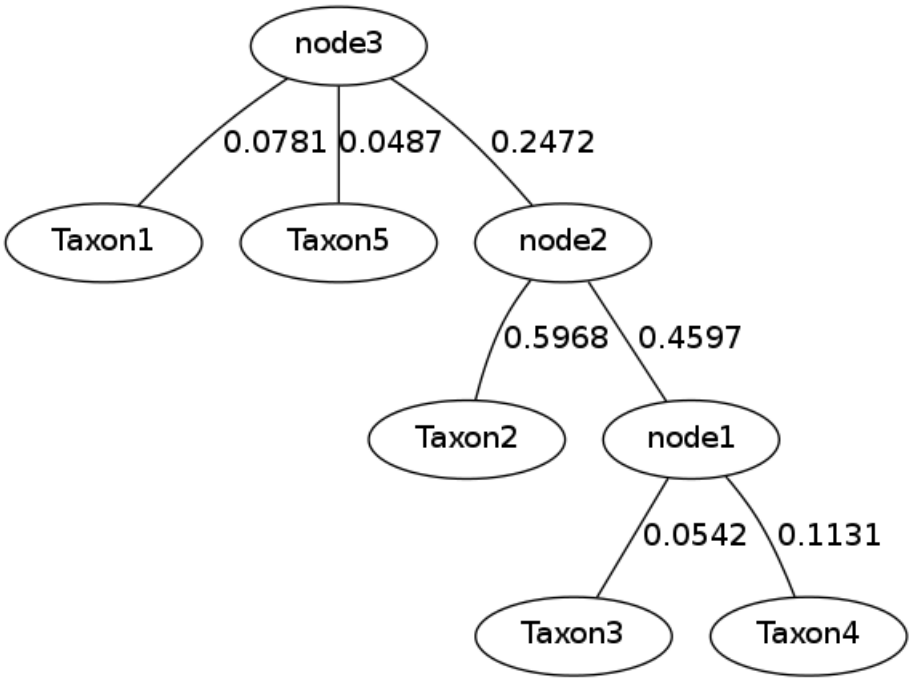
Skripta se koristi na sljedeći način:

./nj_process program fasta_datoteka.

Za određenu implementaciju u repozitoriju potrebno je pokrenuti npr:

./nj_process ../Code/Santl/Bin/neighbour_joining test1.fasta

Za druge implementacije dane su naredbe u *README.md* u repozitoriju. Kao rezultat izvođenja infrastrukture dobiva se sličan prikaz kao na slici 3.1.



Slika 3.1: Primjer slike generirane pomoću infrastrukture

3.3. Testiranje

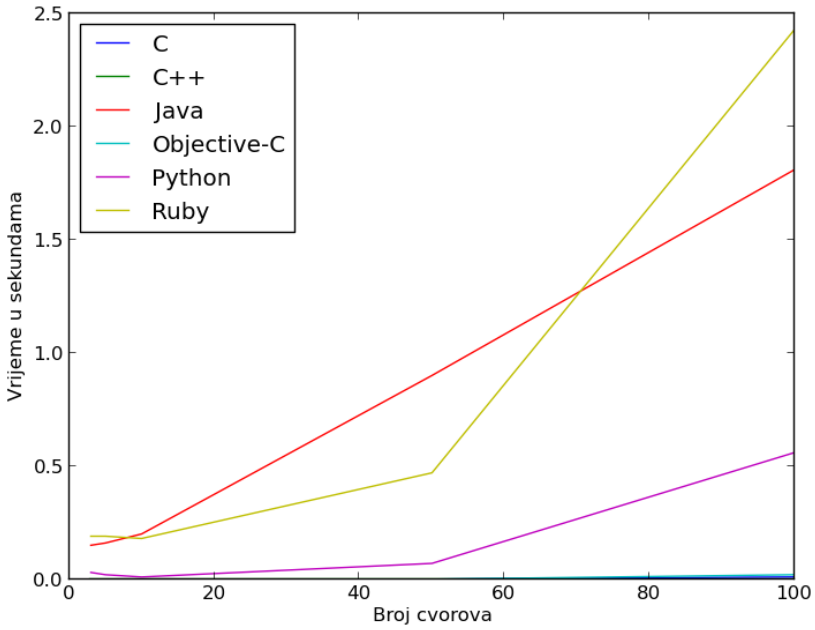
Za testiranje implementacija napisane su skripte koje omogućavaju automatizirano ispitivanje i vizualizaciju rezultata. Ispitna struktura se nalazi u repozitoriju u mapi *"Test/"*.

Testiranje obuhvaća tri cjeline: točnost, vrijeme izvođenja i korištena memorija. Komponentu točnosti nije se mogla utvrditi do kraja. Algoritam *Neighbour joining* u svojoj definiciji ima nedeterminističko svojstvo. U algoritmu nije definirano što se događa ako se u matrici **Q** pojave parovi čvorova koji imaju istu vrijednost. U tom slučaju može se uzeti bilo koji od njih i zato, u ovisnosti o implementaciji, izlazi algoritma mogu davati različita stabla. Zbog toga su se kao testni primjeri uzele matrice udaljenosti koje daju jedinstvena stabla. Za to je korištena skripta u mapi *"Test/"* pod nazivom *check_isomorphism.py* koja provjerava izomorfnost dvaju težinskih grafova.

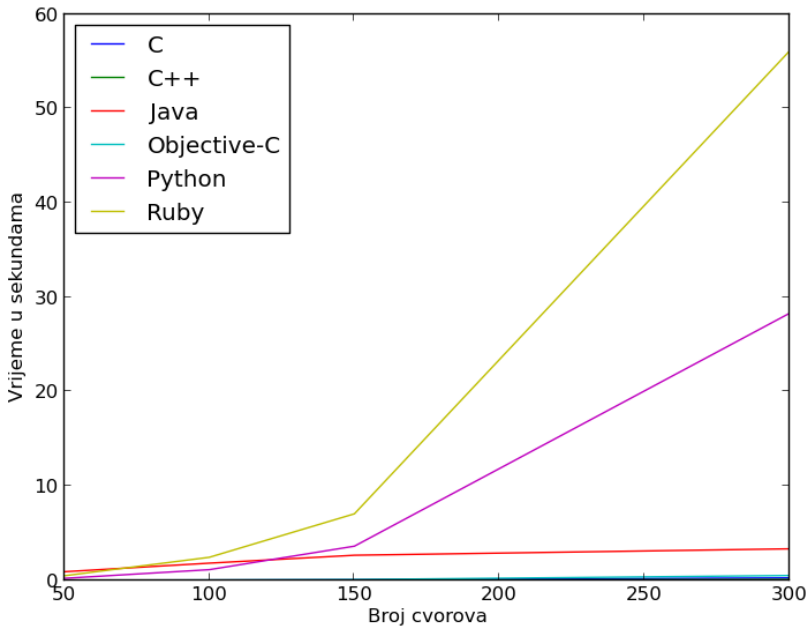
Za ispitivanje vremena izvođenja i korištene memorije generirani su test primjeri različitih veličina (broja čvorova). Ispod se nalaze grafovi koji prikazuju mjerene veličine.

3.3.1. Vrijeme izvođenja

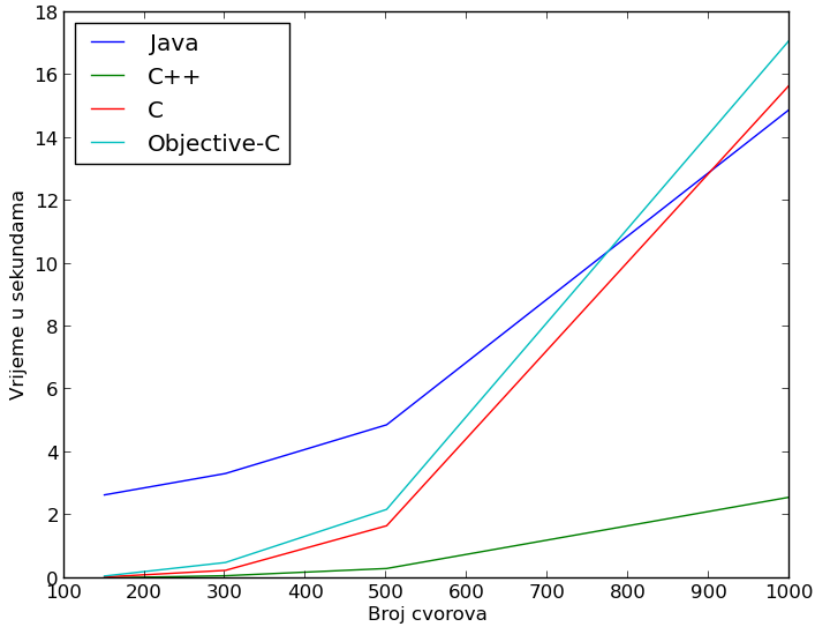
Vrijeme izvođenja algoritma testirano je nad brojem čvorova N=3, 5, 10, 50, 100, 150, 300, 500, 1000. Na slikama 3.2, 3.3 i 3.4 prikazane su različiti intervali i vremena izvođenja. Na intervalu $[150, 1000]$ nisu prikazani skriptni jezici jer je fokus stavljen na druge jezike.



Slika 3.2: Vremena izvođenja algoritama za veličine od 3 do 100 čvorova



Slika 3.3: Vremena izvođenja algoritama za veličine od 50 to 300 čvorova

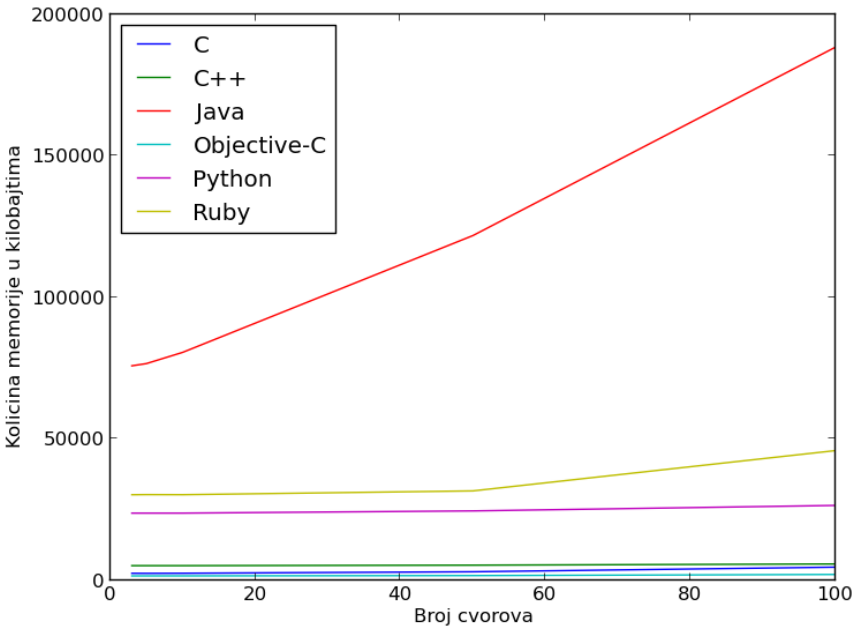


Slika 3.4: Vremena izvođenja za primjere od 150 do 1000 čvorova

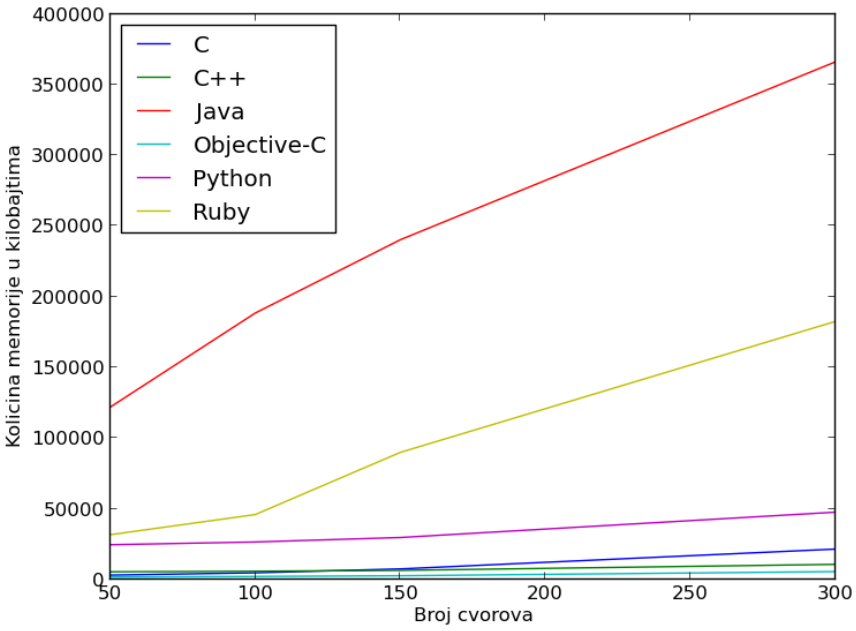
3.3.2. Korištena memorija

Mjerena je maksimalna količina memorije u nekom trentku za vrijeme izvođenja procesa. Jednako kao i za vrijeme, implementacije su ispitane nad brojem čvorova N=3, 5,

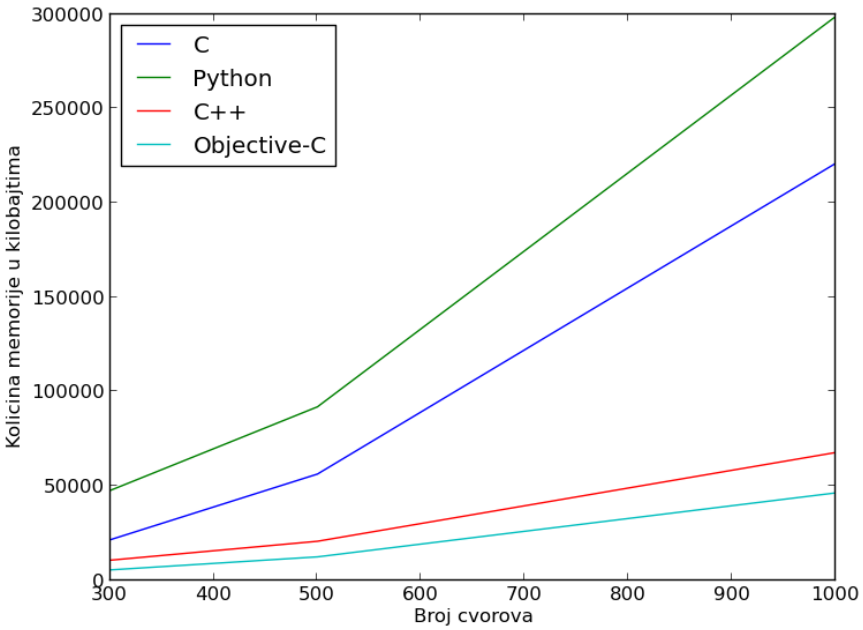
10, 50, 100, 150, 300, 500, 1000. U nastavku se nalaze različiti grafovi koji prikazuju različite intervala za broj čvorova.



Slika 3.5: Korištena memorija u kilobajtima do 100 čvorova



Slika 3.6: Korištena memorija u kilobajtima od 50 do 300 čvorova



Slika 3.7: Korištena memorija od 300 do 1000 čvorova

3.4. Rezultati

Algoritam se može implementirati u apriornoj vremenskoj složenosti $O(N^3)$, gdje je N broj čvorova. Mjerenja su pokazala da implementacija algoritma ovisi o jeziku u kojem je algoritam ostvaren. Tako generalno možemo reći da niži programski jezici daju bolje rezultate, dok skriptni jezici očekivano daju nešto lošija vremena izvođenja.

Graf vremena izvođenja do sto čvorova (slika 3.2) pokazuju da viši programski jezik *Java* i skriptni jezici *Ruby* i *Python* imaju puno veća vremena izvođenja od ostalih jezika, što je i očekivano. Zanimljiv je graf na slici 3.3 gdje se vidi da za malo veći broj čvorova programski jezik *Java* ima puno manje vrijeme izvođenja od skriptnih jezika. Za nešto veći broj čvorova, do njih 1000 (slika 3.4), pokazuje se da se *Java* može mjeriti s ostalim jezicima. Najbolje vrijeme izvođenja imala je implementacija u *C++* programskom jeziku, što je i očekivano jer je *C++* dovoljno nizak jezik, a opet koristi biblioteke struktura koje su optimizirane. Jezici *Objective-C* i *C* pokazuju gotovo identične rezultate.

Rezultati ispitivanja korištene memorije jednako tako su očekivani. *Java* programski jezik troši najviše memorije zbog popratnih troškova virtualnog stroja. Skriptni jezici troše nešto više memorije ali ne znatno od ostalih jezika. Implementacija u jeziku *C* troši nešto više memorije zbog toga što je napravljen *trade-off* između čitkosti koda i efikasnosti. Jezici *C++* i *Objective-C* daju najbolje rezultate jer su korištene

efikasne strukture, odnosno, *Objective-C* prilagođen je za mobilne uređaje koji ne bi trebali trošiti puno memorije.

4. Zaključak

Ishod projekta pokazao je kako određeni programski jezici utječu na implementaciju algoritama, konkretno na algoritam koji se koristi u bioinformatici. Pokazalo se da niski programski jezici poput *C* jezika mogu dati dobre rezultate, ali implementacija algoritma je kompleksnija. *C++* jezik dobar je izbor. Iako na nižoj razini, omogućuje pisanje manjeg broja linija i korištenje gotovih struktura podataka koje provjereno dobro rade. *Java* je također dobar izbor jer se pokazalo da vrijeme izvođenja na većim test primjerima nije lošije od ostalih jezika, a dovoljno je na visokom nivou da se, primjerice, izbjegava ručno upravljanje memorijom. Skriptni jezici nisu dobar izbor za krajnji produkt, ali dobar su izbor za implementaciju prototipa algoritma. Razvoj algoritma u skriptnim jezicima je brz što omogućava efikasno eksperimentiranje. Pokazalo se da se može koristiti u implementaciji algoritama, ali uz uvjet da se većim dijelom koristi samo podskup jezika koji je identičan jeziku C. Kada smo koristili objekte kao prikaz podataka rezultati su bili lošiji. *Objective-C* zbog svoje sintakse teško je čitljiv korisnicima koji se primarno ne bave njime.

5. Literatura

[1] S. N. and N. M., "The neighbor-joining method: a new method for reconstructing phylogenetic trees.," *Molecular Biology and Evolution 4*, pp. 406–425, 1987.

[2] "Phylogeny programs." `http://evolution.genetics.washington.edu/phylip/software.html`.

[3] "Neighbor joining." `http://en.wikipedia.org/wiki/Neighbor_joining`.

[4] J. Felsenstein, "Phylogeny methods, part 3 (distance methods)." `http://evolution.gs.washington.edu/gs541/2002/lecture3.pdf`, 2002.