UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Miloš P. Miković

ALGORITMI ZA REŠAVANJE PROBLEMA NAJKRAĆE ZAJEDNIČKE NADNISKE

master rad

Beograd, 2023.

Mentor:

dr Aleksandar KARTELJ, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Vladimir FILIPOVIĆ, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Stefan MIŠKOVIĆ, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Hvala profesoru Aleksandru Kartelju.

Naslov master rada: Algoritmi za rešavanje problema najkraće zajedničke nadniske

Rezime: Problem najkraće zajedničke nadniske predstavlja jedan od dobro poznatih i ispitivanih NP-teških problema optimizacije. Zbog njegove široke primene, na njegovom što efikasnijem rešenju radi se poslednje četiri decenije. Vremenom je predloženo mnoštvo algoritama koji kombinuju različite optimizacione tehnike i pristupe. U ovom radu predstavljena su dva algoritma koji rešavaju ovaj problem. Algoritam grananja sa odsecanjem uvek pronalazi optimalno rešenje problema, ali je njegovo vreme izvršavanja u najgorem slučaju eksponencijalno. Drugi predstavljeni algoritam počiva na metaheurističkoj tehnici pretraga bima (*eng.* beam search) u kombinaciji sa dve pohlepne heurističke funkcije većinsko spajanje (*eng.* majority merge) i težinsko većinsko spajanje (*eng.* weighted majority merge). U radu je detaljno opisan problem i konstrukcija pomenutih algoritama i heurističkih funkcija. Opisan je način na koji su generisane test instance problema i dat je prikaz rezultata na različitim instancama. Ukratko su opisane već korišćene tehnike za rešavanje ovog problema i date ideje za pravce daljeg usavršavanja predstavljenog algoritma ili formiranje novog.

Ključne reči: optimizacija, pretraga bima, analiza bioloških sekvenci, zajednička nadniska, težinsko spajanje, većinsko težinsko spajanje

Sadržaj

Glava 1

Uvod

Problem najkraće zajedničke nadniske (*eng.* shortest common supersequence problem) jedan je od dobro poznatih NP-teških problema optimizacije u oblasti analize reči [1]. Ukratko, PNZN[1] se može opisati kao problem pronalaženja najkraće reči ω sačinjene od simbola zadate konačne azbuke Σ, tako da su sve sekvence iz unapred zadatog konačnog skupa $\mathcal{L}$ sadržane u sekvenci ω. Kada se kaže da su sve reči iz skupa $\mathcal{L}$ sadržane, misli se na to da se svaka reč iz skupa $\mathcal{L}$ može dobiti uklanjanjem simbola iz reči ω ali u zadatom redosledu [3]. PNZN ima primene u mnogim oblastima informatike, uključujući kompresiju podataka [4], optimizaciju upita [13], analizu i poređenje teksta [2, 15]. Treba istaći njegovu široku primenu u bioinformatici gde predstavlja ključan problem u analizi bioloških sekvenci [7]. Recimo da se posmatra određena sekvenca nukleotida i da je urađeno k zapisa te sekvence nekim mernim aparatom. S obzirom da aparat greši, te sekvence će se razlikovati u određenim delovima, a opisani problem predstavlja PNZN. Kao rezultat primene u mnogim oblastima, postoji veliki broj istraživanja na temu ovog problema u pokušaju da se dođe do što boljeg rešenja. Trenutno najbolji algoritmi za rešavanje PNZN počivaju na metaheurističkoj metodi pretraga bima (*eng.* beam search), koja će biti predstavljena u ovom radu. U nastavku uvodnog poglavlja biće formalno definisan problem najkraće zajedničke nadniske i biće dat pregled dosadašnjih istraživanja na temu ovog problema.

[1] U nastavku teksta PNZN ćemo koristiti kao skraćenicu za problem najkraće zajedničke nadniske.

1.1 Problem najkraće zajedničke nadniske

U ovom poglavlju formalno ćemo definisati PNZN, ali pre toga ćemo dodatno precizirati i proširiti prethodno uvedenu notaciju. Konačna azbuka je skup koji se sastoji od konačnog broja slova i označavaćemo je sa Σ. Svaka konačna reč $\omega = \omega(1)\omega(2)...\omega(n)$ sastoji se od konačnog broja slova azbuke gde $\omega(j) \in \Sigma$ predstavlja j-to slovo reči ω. Duzinu reči ω označavaćemo sa $|\omega|$, praznu reč sa ε i važi da $|\varepsilon| = 0$. Takođe, $\omega \in \Sigma^*$ označava da je reč ω konačne dužine i da je sačinjena od slova azbuke Σ. Reč koja se dobija dodavanjem slova α na početak reči ω označavaćemo sa $\alpha\omega$, a reč koja se dobija dodavanjem slova α na kraj reči ω sa $\omega\alpha$. Sa $\omega[a : b]$ označavaćemo[2] reč koja se dobija od reči ω počevši od pozicije a do pozicije b u reči ω. Ako se izostavi vrednost pozicije a, podrazumeva se da je početna pozicija 0, slično ako se izostavi vrednost pozicije b podrazumeva se vrednost $n - 1$ ako $|\omega| = n$. Na primer, $\omega[:]$ predstavlja samu reč ω. Pristupanje slovu na poziciji i reči ω označavaćemo sa $\omega[i]$.

Neka važi da $\omega_1, \omega_2 \in \Sigma^*$, za reč ω_1 kažemo da je nadniska reči ω_2 u oznaci $\omega_1 \succ \omega_2$ ako važi sledeća rekurzivna definicija (simbol $\triangleq$ označava jednakost po definiciji) [1]:

$$\begin{aligned} \omega_1 \succ \varepsilon &\triangleq \text{Tačno} \\ \varepsilon \succ \omega_2 &\triangleq \text{Netačno}, \quad \text{ako } \omega_2 \neq \varepsilon \\ \alpha\omega_1 \succ \alpha\omega_2 &\triangleq \omega_1 \succ \omega_2 \\ \alpha\omega_1 \succ \beta\omega_2 &\triangleq \omega_1 \succ \beta\omega_2, \quad \text{ako } \alpha \neq \beta \end{aligned} \tag{1.1}$$

Zapravo, $\omega_1 \succ \omega_2$ označava da se svi simboli iz ω_2 nalaze u ω_1 u datom redosledu, ali ne nužno uzastopno. Na primer, za datu azbuku $\Sigma = \{a, c, t, g\}$, važi $agcatg \succ act$. Sada možemo formalno definisati PNZN. Instanca PNZN može se definisati kao $\mathcal{I} = (\Sigma, \mathcal{L})$, gde Σ predstavlja konačnu azbuku, a $\mathcal{L}$ predstavlja skup od m reči $\{\omega_1, \omega_2, ..., \omega_m\}$, $\omega_i \in \Sigma^*$. Potrebno je pronaći reč ω najmanje dužine tako da važi da je ω nadniska svake reči iz skupa $\mathcal{L}$ ($\omega \succ \omega_i, \forall\omega_i \in \mathcal{L}$ i $|\omega|$ je minimalna). Na primer za instancu PNZN $\mathcal{I} = (\{a, c, t, g\}, \{act, cta, aca\})$, najmanja zajednička nadniska instance $\mathcal{I}$ je $acta$.

Optimalno rešenje najčešće nije jedinstveno, a broj optimalnih rešenja raste sa porastom složenosti instance problema. Može se pokazati da je PNZN NP-kompletan

[2]Slično kao indeksiranje nizova u programskom jeziku Python.

problem, čak i ako su jaka ograničenja postavljena na $\mathcal{L}$ ili Σ. Dokazano je da je PNZN NP-kompletan problem nad svakom azbukom Σ za koju važi da $|\Sigma| \geqslant 2$ [8] ili kada su sve reči $\omega \in \mathcal{L}$ dužine dva [17]. Ovim rezultatima NP-težine se mora pristupiti sa oprezom, jer predstavljaju samo najgori scenario što često u praksi nije slučaj. U odnosu na to, realnija karakterizacija težine može se dobiti korišćenjem okvira parametrizovane složenosti (*eng.* framework of parameterized complexity). Ukratko, ovo se postiže višedimenzionim pristupom problemu, shvatanjem njegove unutrašnje strukture i izolovanjem određenih parametara. Ako se težina (koja nije polinomska) može izolovati unutar ovih parametara, problem može biti efikasno rešen za fiksne vrednosti ovih parametara. Na primer može se uzeti maksimalna dužina k nadniske kao parametar i ako je pritom veličina azbuke fiksna ili parametar takođe, problem postaje fiksno-parametarski pratljiv (*eng.* fixed-parameter tractable) jer postoji maksimalno $|\Sigma|^k$ nadniski koje se mogu proveriti kao rešenje problema [1]. Može se parametrizovati i broj reči u skupu $\mathcal{L}$.

1.2 Pregled dosadašnjih istraživanja

Problem najkraće zajedničke nadniske prvi je uveo Dejvid Mejer (*eng.* David Maier) 1978. godine u svom radu „The Complexity of Some Problems on Subsequences and Supersequences" [9]. Korišćenjem dinamičkog programiranja (*eng.* dynamic programming) PNZN nad dve reči dužine n rešen je algoritmom vremenske složenosti $\mathcal{O}(n^2)$ i prostorne složenosti $\mathcal{O}(n^2)$. Algoritam zasnovan na dinamičkom programiranju može biti unapređen, pa tako za k reči dužine maksimalno n, PNZN može biti rešen u $\mathcal{O}(n^k)$ prostornoj i vremenskoj složenosti [14]. Jasno je da ovakav algoritam nije praktičan za velike vrednosti k. S obzirom na to da ne postoji algoritam polinomske složenosti koji rešava PNZN, pribegava se optimizacionim metodama u rešavanju ovog problema. Ono što je karakteristično za optimizacioni pristup rešavanju problema jeste to da se formira algoritam koji rešava postojeći problem tako što daje rešenje koje je prihvatljivo pod određenim uslovima. Takvo rešenje ne mora nužno biti optimalno rešenje problema. Na ovaj način, korišćenjem određene optimizacione tehnike, dobija se algoritam koji se izvršava brzo u realnim uslovima i daje prihvatljivo dobra rešenja.

Vremenom je predloženo mnogo heurističkih i metaheurističkih algoritama za rešavanje PNZN. Neke od poznatijih heurističkih funkcija koje su korišćene u rešavanju PNZN su alfabet (*eng.* alphabet) [10], većinsko spajanje (*eng.* majority merge)

i težinsko većinsko spajanje (*eng.* weighted majority merge) [1], turnirska (*eng.* tournament) i pohlepna (*eng.* greedy) [18], redukuj-proširi (*eng.* reduce-expand) [10] itd. Pored navedenih funkcija, korišćeni su i metaheuristički algoritmi poput genetskog algoritma (*eng.* genetic algorithm) [6] i optimizacije kolonijom mrava (*eng.* ant colony optimization) [12], koji predstavljaju složenije optimizacione tehnike i imaju tendenciju ka dužem vremenu izvršavanja ne većim instancama problema [7].

Jedna od trenutno najboljih metaheuristika za rešavanje PNZN jeste pretraga bima (*eng.* beam search). Ukratko, pretraga bima predstavlja nepotpunu pretragu drveta, koja na svakom nivou proširuje graf stanja tako što napreduje sa čvorovima koji najviše obećavaju [5]. Upravo zbog toga što se dobro pokazala u rešavanju PNZN i sličnih problema poput problema najduže zajedničke podniske (*eng.* longest common subsequence) pretraga bima će biti korišćena u ovom radu i biće detaljnije opisana u daljem tekstu. S obzirom na to da pretraga bima podrazumeva postojanje heurističke funkcije koja će oceniti kvalitet čvorova na određenom nivou, u ovom radu izabrane su dve takve funkcije, većinsko spajanje (*eng.* majority merge) i težinsko većinsko spajanje (*eng.* weighted majority merge), koje su se dobro pokazale u prethodnim istraživanjima i one će detaljnije biti opisane u nastavku teksta.

Glava 2

Algoritmi za rešavanje problema najkraće zajedničke nadniske

U ovom poglavlju biće dat pregled sledeća dva algoritama koja su korišćena za rešavanje problema najkraće zajedničke nadniske:

1. algoritam grananja sa odsecanjem (*eng.* backtracking);
2. pretraga bima (*eng.* beam search).

U nastavku teksta biće opisana konstrukcija ova dva algoritma kao i njihove karakteristike.

2.1 Algoritam grananja sa odsecanjem

Algoritam grananja sa odsecanjem poboljšava tehniku grube sile (totalne pretrage) tako što vrši provere tokom generisanja kandidata za rešenja i tako što se odbacuju parcijalno popunjeni kandidati za koje se unapred može utvrditi da se ne mogu proširiti do optimalnog rešenja problema. Dakle, grananje sa odsecanjem podrazumeva da se tokom obilaska u dubinu drveta, kojim se predstavlja prostor potencijalnih rešenja odsecaju oni delovi drveta za koje se unapred može utvrditi da ne sadrže ni jedno rešenje problema tj. da ne sadrže optimalno rešenje, pri čemu se odsecanje vrši i u čvorovima bliskim korenu koji mogu da sadrže i samo parcijalno popunjene kandidate za rešenja [16]. Dakle, umesto da se čeka da se tokom pretrage stigne do lista (ili eventualno unutrašnjeg čvora koji predstavlja nekog kandidata za rešenje) i da se provera zadovoljenosti uslova ili optimalnosti vrši tek tada, prilikom

grananja sa odsecanjem provera se vrši u svakom koraku i vrši se provera parcijalno popunjenih rešenja. Efikasnost ovakvog algoritma uveliko zavisi od kvaliteta kriterijuma na osnovu kojih se vrši odsecanje. Iako obično složenost najgoreg slučaja ostaje eksponencijalna, pažljivo odabrani kriterijumi odsecanja mogu odseći jako velike delove pretrage (koji su često takođe eksponencijalne veličine u odnosu na dimenzije ulaznog problema) i time značajno ubrzati proces pretrage.

U kontekstu PNZN korišćen je rekurzivni algoritam prikazan u Algoritmu 1. Algoritam podrazumeva postojanje dve globalne promenjive $minD$ i $maxD$ čije se

Algoritam 1 GrananjeSaOdsecanjem(ω)

```
1:  d ← |ω|                                          ▷ d - dužina trenutne reči
2:  if (d > maxD) ∨ (d ≥ nd) then
3:      return
4:  end if
5:  if (d ⩾ minD) ∧ ZajedničkaNadniska(ω) then
6:      iDaljeNadniska ← True
7:      pozicija ← 0
8:      while iDaljeNadniska do                      ▷ optimizacija
9:          if ZajedničkaNadniska(ω[pozicija + 1 :]) then
10:             pozicija ← pozicija + 1
11:         else
12:             iDaljeNadniska = False
13:         end if
14:     end while
15:     nn ← ω[pozicija :]                           ▷ nn - najbolja nadniska
16:     nd ← |nn|                                    ▷ nd - najbolja dužina
17: end if
18: for α ∈ Σ do                                     ▷ grananje po slovima azbuke
19:     GrananjeSaOdsecanjem(ωα)
20: end for
```

vrednosti postavljaju pre poziva algoritma i one označavaju redom minimalnu i maksimalnu dubinu, takođe se podrazumeva da su skupovi $\mathcal{L}$ i Σ globalno dostupni u programu. Vrednost promenjive $minD$ postavlja se na dužinu najkraće reči u $\mathcal{L}$, a vrednost promenjive $maxD$ predstavlja proizvod veličine azbuke i najduže reči u skupu $\mathcal{L}$ kao što je prikazano u formulama 2.1 i 2.2.

$$minD = \{min_{\omega \in \mathcal{L}} |\omega|\} \tag{2.1}$$

$$maxD = |\Sigma| \cdot L, \text{ gde je } L = \{max_{\omega \in \mathcal{L}} |\omega|\} \tag{2.2}$$

Sada ćemo ova uvedena ograničenja minimalne i maksimalne dubine objasniti na kratkom primeru. Za instancu PNZN $\mathcal{I} = (\{a, c, t, g\}, \{acta, cta, ac\})$ znamo da najkraća zajednička nadniska ne može biti kraća od dužine najkraće reči u skupu $\mathcal{L}$, jer takva niska ne bi sadržala ni minimalnu reč a samim time ni ostale reči u $\mathcal{L}$, pa vrednost promenjive $minD = |ac| = 2$. Takođe za instancu problema I mi sigurno znamo jedno rešenje PNZN. Kako je najduža reč $acta$ dužine 4 važi da reč $\omega = \underline{actg}_1\underline{actg}_2\underline{actg}_3\underline{actg}_4$ sigurno predstavlja jedno rešenje problema, pa važi da $maxD = |\omega| = |\Sigma| \cdot |acta| = 4 \cdot 4 = 16$.

Nakon postavljanja ograničenja dubine, rekurzivni algoritam se poziva u obliku *GrananjeSaOdsecanjem(ε)* kako bi pretraga u dubinu krenula od prazne reči. U svakom koraku algoritma vrši se provera da li je dužina trenutne reči d veća od maksimalne dubine ili je veća od dužine trenutno najbolje pronađene nadniske (na početku $nd = maxD + 1$) i ako jeste vrši se odsecanje tog pod-drveta u prostoru pretrage. Ako važi da je dužina trenutne reči d veća od minimalne dubine i pritom važi da reč ω predstavlja zajednučku nadnisku reči u skupu $\mathcal{L}$, pronađeno je jedno rešenje PNZN. Pre ažuriranja vrednosti najbolje nadniske (nn) i najbolje dužine (nd) pokušava se sa skraćivanjem reči ω u cilju dobijanja kraće nadniske, a time i većeg broja odsecanja u prostoru pretrage. Na primeru gore navedene instance problema I, kako se obilazak stabla vrši u dubinu i grananje u svakom rekurzivnom pozivu počinje prvim slovom azbuke, u ovom slučaju a, često se dobijaju rešenja oblika $\omega = aa...aR$, gde ω predstavlja rešenje PNZN, ali i R predstavlja rešenje, pa stoga možemo ukloniti prefiks reči ω, sve dok skraćivanjem i dalje dobijamo validno rešenje problema. Na kraju, vrši se grananje po svim slovima azbuke i rekurzivno poziva algoritam sa argumentom $\omega\alpha$ koji proširuje trenutnu reč ω novim slovom iz azbuke Σ. Funkcija ZajedničkaNadniska(ω) vrši proveru da li reč ω predstavlja zajednučku nadnisku reči iz skupa $\mathcal{L}$, i u ovom slučaju se podrazumeva da je skup $\mathcal{L}$ globalno dostupan. Pseudokod je dat u Algoritmu 2. U svakom koraku petlje proverava se da li je trenutna reč $s \in \mathcal{L}$ podniska reči ω, ako nije funkcija vraća $False$, a ako su sve reči podniske reči ω funkcija vraća $True$. U suštini, važi jednostavna logička veza 2.3:

$$\{Podniska(s, \omega) \mid \forall s \in \mathcal{L}\} \Longleftrightarrow \{\omega \succ s \mid \forall s \in \mathcal{L}\} \tag{2.3}$$

Ako su sve reči iz skupa $\mathcal{L}$ podniske reči ω onda važi da je ω nadniska svih reči iz $\mathcal{L}$ i obrnutno. Pseudokod funkcije Podniska(s, ω) dat je u Algoritmu 3. Na početku

se vrši inicijalizacija promenjivih, sP i ωP predstavljaju indekse početka reči s i ω, sK i ωK redom predstavljaju indekse na kraj reči s i ω. U svakom koraku petlje vrši se provera da li se stiglo do kraja reči ω, i ako je to slučaj, a prethodno nisu pronađena sva slova reči s, znači da s sigurno nije podniska reči ω. Svaki put kada se vrednosti na pozicijama sP i ωP u rečima s i ω poklope, oba indeksa se povećavaju, a u suprotnom se napreduje samo kroz reč ω pa se inkrementira njen indeks ωP. Petlja se završava kada su pronađena sva slova reči s i tada se zna da je s sigurno podniska reči ω.

Algoritam 2 ZajedničkaNadniska(ω)

```
1: for s ∈ 𝓛 do
2:     if ¬Podniska(s, ω) then
3:         return False
4:     end if
5: end for
6:
7: return True
```

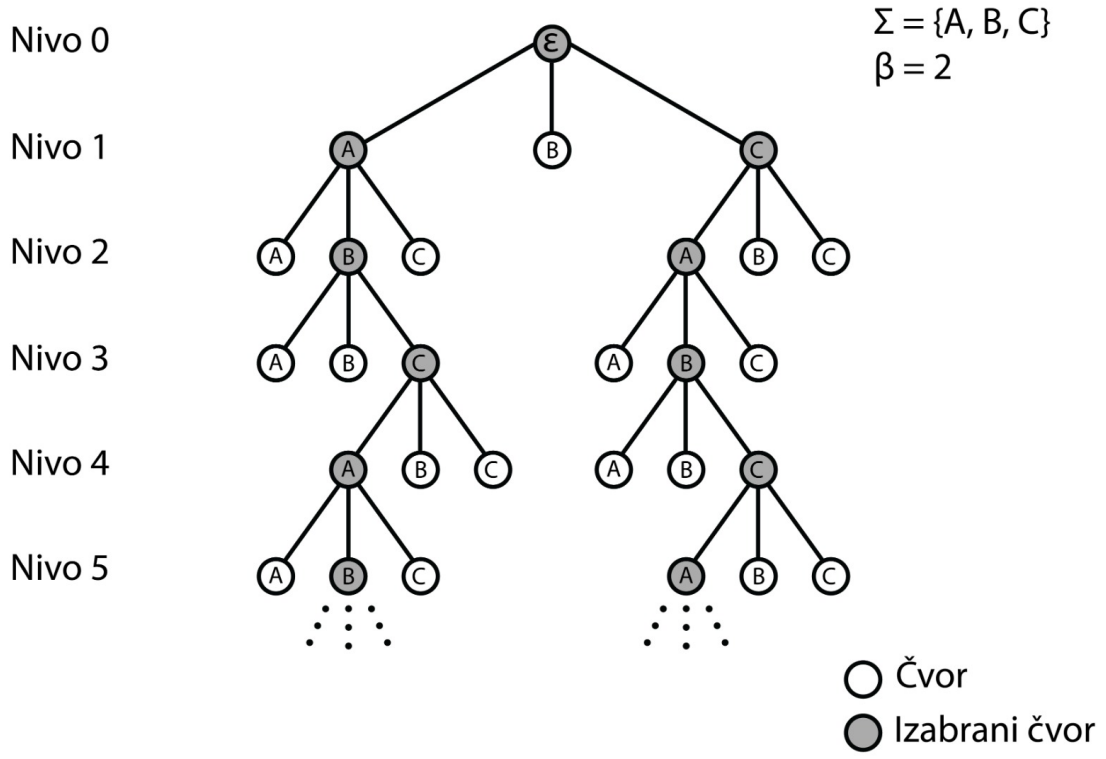
Algoritam 3 Podniska(s, ω)

```
1: sP ← 0                          ▷ indeks na početak reči s
2: sK ← |s|                        ▷ indeks na kraj reči s
3: ωP ← 0                          ▷ indeks na početak reči ω
4: ωK ← |s|                        ▷ indeks na kraj reči ω
5:
6: while sP ≠ sK do
7:     if ωP == ωK then
8:         return False
9:     else if s[sP] == ω[ωP] then
10:        sP ← sP + 1
11:        ωP ← ωP + 1
12:    else
13:        ωP ← ωP + 1
14:    end if
15: end while
16:
17: return True
```

Iako algoritam grananja sa odsecanjem u najgorem slučaju ima eksponencijalnu složenost $O(|\Sigma|^{maxD})$ kao i algoritam totalne pretrage, u praksi je prosečno vreme

izvršavanja daleko kraće. Rezultati izvršavanja algoritma grananja sa odsecanjem biće prikazani u sledećem poglavlju. Ono što je bitno istaći jeste da ovaj algoritam garantuje pronalaženje optimalnog rešenja PNZN. S obzirom na eksponencijalnu prirodu ovog algoritma, nemoguće je koristiti ga za proveru kvaliteta rešenja pretrage bima na većim instancama problema, stoga će u te svrhe biti korišćene nešto sofisticiranije metode koje će biti objašnjene u sekciji 3.1.

2.2 Pretraga bima



Slika 2.1: Šema metaheuristike Pretraga bima

Pretraga bima predstavlja metaheuristiku koja je uvedena 1976. godine u oblasti prepoznavanja govora (*eng.* speech recognition). Korišćena je u završnim slojevima mnogih modela za obradu prirodnog jezika (*eng.* natural language processing models) u donošenju odluke da se izabere najbolji izlaz date ciljne promenjive [11]. Pored navedenih primena pretraga bima se intenzivno koristi u sledećim oblastima: kombinatorna optimizacija (*eng.* combinatorial optimization), problemi plani-

ranja (*eng.* scheduling problems), problemi rutiranja vozila (*eng.* vehicle routing problems), podešavanje hiperparametara u oblasti mašinskog učenja (*eng.* machine learning hyperparameter tuning), problemi zadovoljenja ograničenja (*eng.* constraint satisfaction problems) itd.

Pretraga bima predstavlja vrstu pretrage grafa u širinu u cilju da se pronađe najbolji put od korenog čvora do ciljanog čvora u grafu. Kako bi se složenost izračunavanja držala u zadatim granicama, pretraga bima evaluira dostignute čvorove na određenom nivou ali bira samo podskup od β čvorova koji najviše obećavaju i sa tim podskupom čvorova napreduje dalje u pretrazi. Izabrani podskup od β čvorova zvaćemo bim (*eng.* beam) i označavaćemo ga sa $\mathcal{B}$, a β parametar ćemo zvati širina bima (*eng.* beam width) [5]. U kontekstu PNZN, graf koji pretražujemo $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ predstavlja usmeren aciklički graf, a pseudokod algoritma je prikazan u Algoritmu 4.

Kao i u prethodno opisanom algoritmu podrazumeva se da je promenjiva $maxD$ globalno dostupna, kao i skupovi $\mathcal{L}$ i Σ. Svaki element skupa $\mathcal{B}$ predstavlja objekat koji sadrži parcijalnu reč, heurističku ocenu i pozicije. Parcijalna reč predstavlja put od korena do nekog čvora unutar stabla pretrage, heuristička ocena predstavlja ocenu ove reči od strane heurističke funkcije, a pozicije predstavljaju niz koji ozančava pozicije do kojih se stiglo unutar skupa $\mathcal{L}$. Na primer ako je data azbuka $\Sigma = \{a, c, t, g\}$ i skup $\mathcal{L} = \{act, gta, gcc\}$ tada pozicije $p = [0, 0, 0]$ predstavljaju baš skup $\mathcal{L}$ odnosno pozicije na početak svake reči u $\mathcal{L}$, a pozicije $p^{'} = [1, 1, 2]$ predstavljaju skup $\mathcal{L}^{'} = \{ct, gt, c\}$ koji je dobijen od skupa $\mathcal{L}$ pomeranjem za vrednosti pozicija $p^{'}$. Pri proširivanju skupa $\mathcal{B}$ slovima iz azbuke Σ, dobijaju se novi elementi čija je parcijalna reč proširena novododatim slovom. Od jednog elementa dobijamo maksimalno $|\Sigma|$ novih, što znači da u svakom koraku algoritma dobijamo maksimalno $\beta \cdot |\Sigma|$ novih elemenata od kojih je potrebno izabrati β elemenata. Heurističke funkcije, koje će biti predstavljene u narednom potpoglavlju, za dati niz pozicija računaju težine za svako slovo azbuke, koje se zatim pri proširivanju bima dodaju svakom elementu koji se proširi odgovarajućim slovom. Ovim se eliminiše potreba za ocenjivanjem svakog elementa bima iznova i iznova na svakom nivou pretrage, već se heuristička vrednost svakog elementa inkrementalno gradi. Ovaj vid optimizacije omogućen je samo u slučaju pohlepnih heurističkih funkcija koje u svakom koraku algoritma u odnosu na redukovan skup $\mathcal{L}$ ocenjuju novoizabrano slovo. Nakon proširivanja parcijalne reči odovarajućim slovom azbuke, vrši se ažuriranje pozicija na taj način što se inkrementiraju pozicije onih reči koje počinju izabranim slovom.

Najpre se vrši inicijalizacioni korak algoritma izvan glavne petlje. Funkcija H predstavlja heurističku funkciju kojoj se inicijalno prosleđuju pozicije na početak reči iz $\mathcal{L}$. Promenjiva težine sadrži slova i odgovarajuće težine svakog slova dobijene heurističkom ocenom. Zatim se popunjava bim elementima, gde svaki element sadrži slovo azbuke, odgovarajuću težinu i inicijalne pozicije. Nakon toga vrši se redukcija bima kao što je prikazano u Algoritmu 5. Ako je veličina bima manja od dozvoljene širine bima β, vrši se ažuriranje pozicija svakog elementa kao što je prikazano u Algoritmu 6 i vrši se provera da li postoji element koji predstavlja rešenje PNZN na isti način koji je opisan u Algoritmu 2. Ako takav element postoji, postavlja se indikator za kraj algoritma, a pronađena parcijalna reč se čuva kao rešenje. Ažuriranjem pozicija inkrementiraju se pozicije koje odgovaraj rečima iz $\mathcal{L}$ počev od trenutnih pozicija, a koje počinju poslednje dodatim slovom. U slučaju da je veličina bima veća od dozvoljene širine elementi bima se najpre promešaju, a zatim sortiraju u odnosu na heurističku ocenu. Mešanjem elemenata se eliminiše konstantni azbučni poredak kada neki elementi imaju jednaku heurističku vrednost, a time i odabir takvih elemenata u azbučnom poretku. Zatim se vrši odabir β elemenata sa najvećom heurističkom ocenom, pritom se vrši ažuriranje pozicija odabranih elemenata i već opisana provera zadovoljenja rešenja PNZN. Glavna petlja algoritma prolazi kroz elemente bima, i za svaki element se računaju heurističke ocene slova azbuke u odnosu na trenutne pozicije tog elementa u $\mathcal{L}$. Od svakog elementa dobija se $|\Sigma|$ novih elemenata proširivanjem parcijalne reči elementa slovima azbuke. Na heurističku vrednost novog elementa dodaje se heuristička ocena odgovarajućeg slova. Novi element zadržava iste pozicije, koje se ažuriraju pri redukciji bima ali samo za odabrane elemente, čime se izbegava nepotrebno ažuriranje pozicija elemenata koji neće biti izabrani. Novi elementi se dodaju u bim, nakon čega se vrši redukcija bima na prethodno opisan način. Algoritam se završava kada pronađe jedno rešenje ili ako dubina dostigne $maxD$ jer u tom slučaju kao što je već pokazano postoji jedno trivijalno rešenje problema.

Algoritam 4 PretragaBimaPNZN()

```
 1: b1 ← {}
 2: b2 ← {}
 3: pronađenaNadniska ← False
 4:
 5: težine ← H([0,...,0])
 6:
 7: for α ∈ Σ do
 8:     težinaSlova ← 0
 9:     for t ∈ težine do
10:         if α == t.slovo then
11:             težinaSlova ← t.vrednost
12:         end if
13:     end for
14:     element ← {α, težinaSlova, [0,...,0]}
15:     b1.dodaj(element)
16: end for
17:
18: b1 ← RedukujBim(b1, pronađenaNadniska)
19: dubina ← 0
20:
21: while ¬pronađenaNadniska ∧ dubina < maxD do
22:     for s ∈ b1 do
23:         težine ← H(s.pozicije)
24:         for α ∈ Σ do
25:             težinaSlova ← 0
26:             for t ∈ težine do
27:                 if α == t.slovo then
28:                     težinaSlova ← t.vrednost
29:                 end if
30:             end for
31:             element ← {(s.reč)α, s.hvrednost + težinaSlova, s.pozicije}
32:             b2.dodaj(element)
33:         end for
34:     end for
35:
36:     b1.obriši()
37:     b1 ← RedukujBim(b2, pronađenaNadniska)
38:     b2.obriši()
39:     dubina ← dubina + 1
40: end while
```

Algoritam 5 RedukujBim($\mathcal{B}$, *pronađenaNadniska*)

```
1: if B.veličina() ⩽ β then
2:     for e ∈ B do
3:         AžurirajPozicije(e)
4:         if ZajedničkaNadniska(e.reč) then
5:             nn ← s.reč                          ▷ nn - najbolja nadniska
6:             nd ← |nn|                           ▷ nd - najbolja dužina
7:             pronađenaNadniska ← True
8:         end if
9:     end for
10:    return B
11: end if
12:
13: Promešaj(B)
14: Sortiraj(B)
15: Bp ← {}
16:
17: for i = 0 → β do
18:     AžurirajPozicije(B[i])
19:     if ZajedničkaNadniska(B[i].reč) then
20:         nn ← B[i].reč
21:         nd ← |nn|
22:         pronađenaNadniska ← True
23:     end if
24:     Bp.dodaj(B[i])
25: end for
26:
27: B.obriši()
28: return Bp
```

Iako algoritam pretrage bima ne garantuje pronalaženje optimalnog rešenja, ovako konstruisan algoritam ima polinomsko vreme izvršavanja $O(\beta * maxD)$ što ga u praksi čini daleko efikasnijim od prethodno opisanog algoritma granja sa odsecanjem. Iz tog razloga će ovaj algoritam biti testiran na većim instancama PNZN, gde je algoritam grananja sa odsecanjem praktično neizvodljivo testirati.

Algoritam 6 AžurirajPozicije(*element*)

```
1: poslednjeDodatoSlovo ← element.reč[element.reč.dužina() − 1]
2:
3: for i = 0 → ℒ.veličina() do
4:     if ℒ[i][element.pozicije[i]] == poslednjeDodatoSlovo then
5:         element.pozicije[i] ← element.pozicije[i] + 1
6:     end if
7: end for
```

Heurističke funkcije većinsko spajanje i težinsko većinsko spajanje

Većinsko spajanje je pohlepni algoritam koji inkrementalno konstruiše nadnisku. Najpre se odredi slovo koje se najčešće nalazi na početku reči iz skupa $\mathcal{L}$, a zatim se izabrano slovo briše sa početka reči iz $\mathcal{L}$ koje ga sadrže. Postupak se ponavlja dok skup $\mathcal{L}$ ne postane prazan, a dobijena nadniska predstavlja rešenje PNZN [1]. U kontekstu opisanog algoritma pretrage bima, većinsko spajanje je korišćeno kao heuristička funkcija koja za dati niz pozicija u $\mathcal{L}$ vraća objekat koji sadrži slova azbuke i odgovarajuće težine u udnosu na to koliko reči od odgovarajućih pozicija počinje odgovarajućim slovom. Nakon izračunavanja težina, vrši se mešanje i sortiranje vrednosti. Mešanje nije neophodno, ali kao što je već rečeno eliminiše se konstantan azbučni poredak.

Algoritam 7 VećinskoSpajanje(pozicije)

```
1: težine ← {α ∈ Σ, [0, ..., 0]}                    ▷ inicijalizacija težina
2:
3: for i = 0 → ℒ.veličina() do
4:     if pozicije[i] ≤ (ℒ[i].dužina() − 1) then
5:         for t ∈ težine do
6:             if t.slovo == ℒ[i][pozicije[i]] then
7:                 t.vrednost ← t.vrednost + 1
8:             end if
9:         end for
10:    end if
11: end for
12:
13: Promešaj(težine)
14: Sortiraj(težine)
15:
16: return težine
```

Pseudokod algoritma prikazan je u Algoritmu 7. Mana većinskog spajanja je to što ne može da prepozna globalnu strukturu reči iz skupa $\mathcal{L}$. Većinsko spajanje izostavlja činjenicu da reči mogu biti različitih dužina. To dalje znači da će slova sa početka kraćih reči imati veću šansu da budu uklonjena iako algoritam i dalje treba da obradi preostale dugačke reči. Iz tog razloga bi skidanje slova sa početka kraćih reči trebalo da bude manje prioritetno. Drugim rečima bolje je da se prioritizira skidanje slova sa početka dužih reči. To se može postići dodavanjem težine svakom slovu azbuke u odnosu na dužinu preostalih reči iz $\mathcal{L}$ nakon uklanjanja tog slova sa početka reči koje počinju tim slovom. Dakle, korak 7 u algoritmu *VećinskoSpajanje(pozicije)* možemo zameniti sa:

$$t.vrednost \leftarrow t.vrednost + (\mathcal{L}[i].dužina() - 1) - pozicije[i] \tag{2.4}$$

Ovako modifikovan algoritam naziva se težinsko većinsko spajanje i postoje indikacije da na određenim instancama problema može da nadmaši algoritam većinskog spajana, pogotovo kada nema struktuiranosti unutar skupa $\mathcal{L}$ ili kada je ta struktuiranost haotična [1]. Predstavljene dve heurističke funkcije su korišćene u algoritmu pretrage bima.

Glava 3

Evaluacija algoritama

U ovom poglavlju biće prikazani i diskutovani rezultati predstavljenih algoritama kao i test podaci koji su korišćeni za testiranje i način na koji su test podaci generisani. Takođe će biti dat kratak pregled eksperimentalnog okruženja u kome se vršilo testiranje, hardverska i softverska specifikacija računara kao i specifičnosti implementacije.

3.1 Test podaci

Optimalno rešenje najčešće nije jedinstveno, a broj optimalnih rešenja varira od instance do instance problema i raste sa porastom broja reči u skupu $\mathcal{L}$. Ono što je bitno, jeste dužina takvog rešenja i to je jedini validan parametar optimalnosti. Algoritam pretrage bima ne garantuje pronalaženje optimalnog rešenja dok algoritam grananja sa odsecanjem garantuje. Naizgled, on može poslužiti za testiranje algoritma pretrage bima, ali vreme izvšavanja grananja sa odsecanjem eksponencijalno raste sa porastom parametra $maxD$ pa vreme izvšavanja ovog algoritma čak i na manjim instancama PNZN postaje predugo. Na primer, ako najduža reč u skupu $\mathcal{L}$ ima dvadeset slova i pritom je data proizvoljna azbuka kardinalnosti $|\Sigma| = 4$, u najgorem slučaju algoritam grananja sa odsecanjem treba da obradi $(4^{80}-1)/3$ čvorova. S obzirom na navedene činjenice i manjak test podataka u literaturi, osmišljen je algoritam koji generiše test instance. Pseudokod algoritma je prikazan u Algoritmu 8. Ideja je da se odabere reč koja će predstavljati gornju granicu optimalnosti i da se na osnovu odabrane reči generišu instance problema. Generator test instanci ima četiri parametra:

1. *putanja* — predstavlja putanju do tekstualne datoteke koja sadrži reč gg koja predstavlja gornju granicu;
2. ka — kardinalnost azbuke ($|\Sigma|$);
3. m — označava broj reči u skupu $\mathcal{L}$;
4. γ — predstavlja verovatnoću uklanjanja slova iz gg pri generisanju reči.

Najpre se učitava reč gg sa odabrane lokacije i azbuka popunjava odabranim slovima. Zatim se popunjava skup $\mathcal{L}$. U svakoj iteraciji petlje se kreće od prazne reči ω, prolazi se kroz slova gg i bira se slučajan broj iz uniformne raspodele (u granicama [0,1]). Ako je odabrani slučajan broj veći od γ, reč ω se proširuje trenutnim slovom. Zapravo, reč ω predstavlja reč koja se dobija uklanjanjem slova iz reči gg sa verovatnoćom γ. Na ovaj način se generiše m reči kojima se popunjava skup $\mathcal{L}$, a algoritam na kraju vraća jednu generisanu instancu problema $\mathcal{I}(\Sigma, \mathcal{L})$. Preostalo je još objasniti na koji način se generiše reč koja predstavlja gornju granicu. Reč gg dobija se slučajnim odabirom slova iz azbuke. U svakoj iteraciji izgradnje bira se slučajan broj sb iz uniformne raspodele (u granicama $[0,|\Sigma| - 1]$) i parcijalnoj reči dodaje odgovarajuće slovo azbuke $\Sigma[sb]$. Gornja granica optimalnosti garantuje jedno rešenje problema. Uz pažljivo odabran parametar γ i dovoljno veliki parametar m gornja granica će skoro uvek i predstavljati optimalno rešenje problema. U svakom slučaju gornja granica će u kontekstu testiranja predstavljenih algoritama biti sinonim za optimalno rešenje i poređenje rezultata će se vršiti u odnosu na njenu dužinu.

Na primer, neka je data azbuka $\Sigma = \{a, b\}$. Najpre se generiše reč gg željene dužine slučajnim odabirom slova iz azbuke i smešta u tekstualnu datoteku na željenu lokaciju. Neka je $gg = abba$ i neka su ostali parametri redom *putanja* do generisane reči gg, $ka = 2$, $m = 4$, $\gamma = 0.2$. Zatim se poziva opisani algoritam 8 sa uvedenim parametrima koji generiše m reči sa verovatnoćom uklanjanja γ. Neka je dobijen skup $\mathcal{L} = \{abb, bba, abba, aba\}$. Povratna vrednost generatora je instanca PNZN $\mathcal{I} = \{\{a, b\}, \{abb, bba, abba, aba\}\}$.

Sada će biti definisan skup instanci problema IP koji je korišćen za testiranje u sekciji 3.3. Postoje četiri bitna parametra koja su korišćena za generisanje instanci PNZN:

1. kardinalnost azbuke Σ ($ka = |\Sigma|$);

Algoritam 8 GeneratorTestInstanci($putanja, ka, m, \gamma$)

```
1:  gg ← Pročitaj(putanja)                          ▷ gg - gornja granica
2:  Σ ← {}
3:  L ← {}
4:
5:  for i → ka do
6:      α ← PročitajSlovo()
7:      Σ.dodaj(α)
8:  end for
9:
10: for i → m do
11:     ω ← ε
12:     for s ∈ gg do
13:         slučajanBroj ← GenerišiSlučajanBroj()
14:         if slučajanBroj > γ then
15:             ω.dodaj(s)
16:         end if
17:     end for
18:
19:     L.dodaj(ω)
20:
21: end for
22:
23: I ← (Σ, L)
24:
25: return I
```

2. broj reči u skupu $\mathcal{L}$ ($m = |\mathcal{L}|$);

3. dužina gornje granice ($|gg|$);

4. verovatnoća uklanjanja slova (γ).

Kardinalnost azbuke uzeta je iz opsega $ka \in \{2, 4, 16\}$, broj reči u skupu $\mathcal{L}$ iz opsega $m \in \{10, 20, 40, 80\}$. Dužina gornje granice definiše i potencijalnu gornju granicu za dužinu reči u skupu $\mathcal{L}$ i ona je uzeta iz opsega $|gg| \in \{50, 100, 500, 2000\}$. Na kraju verovatnoća uklanjanja slova uzeta je iz opsega $\gamma \in \{0.1, 0.2, 0.4\}$. Instance problema podeljene su u tri skupa $IP_{\gamma=0.1}$, $IP_{\gamma=0.2}$ i $IP_{\gamma=0.4}$ na osnovu verovatnoće uklanjanja slova. Sva tri skupa sadrže po 48 instanci problema koje su dobijene kao Dekartov proizvod vrednosti prethodno opisanih parametara. Vrednosti za širinu bima uzete su iz opsega $\beta \in \{100, 200, 400\}$. Na skupovima instanci IP testiran je samo algoritam pretrage bima. Algoritam grananja sa odsecanjem testiran je na

manjem skupu instanci IPG gde $ka \in \{2, 4\}$, $m \in \{8, 16, 32\}$. Gornja granica za azbuku dužine dva uzeta je iz opsega $|gg|_{|\Sigma|=2} \in \{20, 24, 28\}$, a za azbuku dužine četiri $|gg|_{|\Sigma|=4} \in \{12, 13, 14\}$ kako bi se vreme izvršavanja držalo u razumnim granicama. Kako je na skupu IPG testiran i algoritam pretrage bima, vrednosti za širinu bima uzete su iz manjeg opsega $\beta \in \{4, 8, 16\}$. Instance problema podeljene su u dva skupa $IPG_{\gamma=0.2}$ i $IPG_{\gamma=0.4}$ na osnovu verovatnoće uklanjanja slova i svaki skup sadrži po 18 instanci problema.

3.2 Eksperimentalno okruženje

Svi rezultati izvršavanja algoritama dobijeni su na računaru pod Linux operativnim sistemom sledećih sistemskih specifikacija:

- Operativni sistem — Ubuntu 20.04.6 LTS;
- Kernel — Linux 5.15.0-46-generic.

Hardverske komponente računara na kome se vršilo testiranje su sledeće:

- Procesor — AMD Ryzen 3 1300X, 3.5GHz;
- Arhitektura — x86-64;
- Broj jezgara — 4;
- Keš memorija — L1(384KiB), L2(2MiB), L3(8MiB);
- Ram memorija — Kingston, 8GB DDR4;
- Grafička kartica — Asus Radeon RX 570 (4GB GDDR5).

Treba napomenuti da iako procesor ima četiri jezgra i četiri fizičke niti, algoritmi nisu implementirani tako da podržavaju paralelno izvršavanje. Svi predstavljeni algoritmi u radu implementirani su u programskom jeziku C++ [1] kako bi se postigla što veća efikasnost u evaluaciji. Korišćen je GCC 9.4.0 prevodilac i standard jezika C++20. Ceo programski kod je organizovan unutar klasa, a svi prethodno navedeni algoritmi i pomoćne funkcije kao privatne funkcije članice (*eng.* member function)

[1] GitHub repozitorijum projekta nalazi se na `https://github.com/milosmikovic/Algorithms_For_Solving_Shortest_Common_Supersequence_Problem`

kako bi podrazumevano bile utisnute (*eng.* inline) na mestima u programu na kojima se pozivaju i kako bi se time izbegli nepotrebni troškovi poziva. U cilju što veće optimizacije određenih delova programskog koda, iako u današnje vreme nisu popularne, korišćene su i pokazivačke promenjive i odgovarajuća aritmetika. Implementacija svih struktura podataka poput skupova, vektora, uređenih parova uzeta je iz standardne biblioteke C++ jezika (*eng.* Standard C++ Library), kao i svi pomoćni algoritmi poput sortiranja ili mešanja elemenata. Svi generatori slučajnih brojeva koji su korišćeni sadrže fiksiranu vrednost semena (*eng.* seed), kako bi generisanje test instanci i evaluacija algoritama mogla da se reprodukuje u definisanim uslovima. Sve generisane test instance i odgovarajući rezultati nalaze se u numerisanim tekstualnim fajlovima. Programski kod je organizovan u dva cpp fajla i prevodi se korišćenjem pomoćnog alata Make. Prilikom prevođenja odabran je najviši nivo optimizacije od strane GCC prevodioca postavljanjem parametra O3. Prevođenje i pokretanje odgovarajućih programa vrši se iz Linux terminala na sledeći način:

- *make testInstanceGenerator* — prevodi generator test instanci
- *make SCS* — prevodi program koji pokreće opisane algoritme
- *./testInstanceGenerator (režim rada)* — pokreće generator test instanci koji nakon poziva očekuje putanju do tekstualnog fajla u kome se nalazi reč koja predstavlja gornju granicu, broj reči m, i željena slova azbuke (parametar γ se postavlja u kodu pre prevođenja), izlaz je numerisan tekstualni fajl koji predstavlja jednu instancu PNZN (postoji više režima rada generatora, na primer prethodno opisani podrazumeva ulazni argument 2, dok je za generisanje reči *gg* potrebno navesti ulazni argument 3)
- *./SCS (broj test instance)* — pokreće program koji izvršava algoritme, kao ulazni argument očekuje redni broj test instance, izlaz je numerisan tekstualni fajl koji sadrži rezultat izvršavanja ulazne test instance

3.3 Eksperimentalni rezultati

U ovoj sekciji biće prikazani eksperimentalni rezultati izvršavanja algoritama. U tabelama 3.1 i 3.2 prikazani su rezultati algoritma grananja sa odsecanjem i algoritma pretrage bima na skupu test instanci $IPG_{\gamma=0.2}$ i $IPG_{\gamma=0.4}$. Kolone redom označavaju: kardinalnost ulazne azbuke Σ ($ka = |\Sigma|$), broj reči u skupu $\mathcal{L}$ ($m = |\mathcal{L}|$),

dužinu gornje granice ($|gg|$), dužinu nadniske koju je algoritam granja sa odsecanjem pronašao (GSO), vreme izvršavanja algoritma u sekundama (t), minimalnu dubinu pretrage ($minD$), maksimalnu dubinu pretrage ($maxD$). Kolona VS predstavlja dužinu nadniske koju je pronašao algoritam pretrage bima koristeći heurističku funkciju većinsko spajanje, a TVS koristeći heurističku funkciju težinsko većinsko spajanje. Rezultati pretrage bima su podeljeni u sekciji u zavisnosti od vredsnoti širine bima β. U svim predstavljenim tabelama svaki red predstavlja jednu instancu problema. U tabelama koje sadrže rezultate pretrage bima u svakom redu su podebljane dve vrednosti, jedna se odnosi na najboli rezultat većinskog spajanja, a druga na najbolji rezultat težinskog većinskog spajanja. U tabelama 3.3, 3.4, 3.5 predstavljeni su rezultati izvršavanja algoritma pretrage bima na test instancama $IP_{\gamma=0.1}$, $IP_{\gamma=0.2}$, $IP_{\gamma=0.4}$. Na svim test podacima korišćene su sledeće azbuke:

- $\Sigma_2 = \{a, b\}$
- $\Sigma_4 = \{a, c, t, g\}$
- $\Sigma_{16} = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p\}$

$IPG_{\gamma=0.2}$						
$\|\Sigma\|$	m	$\|gg\|$	GSO	$t(s)$	$minD$	$maxD$
2	8	20	20	1	19	38
2	8	24	24	6	21	42
2	8	28	28	101	24	48
2	16	20	20	0	19	38
2	16	24	24	6	21	42
2	16	28	28	105	26	52
2	32	20	20	0	19	38
2	32	24	24	6	22	44
2	32	28	28	108	26	52
4	8	12	12	4	11	44
4	8	13	13	15	13	52
4	8	14	14	64	14	56
4	16	12	12	4	11	44
4	16	13	13	15	13	52
4	16	14	14	63	14	56
4	32	12	12	4	12	48
4	32	13	13	15	13	52
4	32	14	14	64	14	56
$IPG_{\gamma=0.4}$						
2	8	20	18	0	14	28
2	8	24	21	1	16	32
2	8	28	25	8	18	36
2	16	20	19	0	14	28
2	16	24	21	1	16	32
2	16	28	26	27	21	42
2	32	20	20	0	16	32
2	32	24	23	4	20	40
2	32	28	27	53	23	46
4	8	12	12	3	9	36
4	8	13	13	17	11	44
4	8	14	14	60	10	40
4	16	12	12	4	9	36
4	16	13	13	16	11	44
4	16	14	14	60	10	40
4	32	12	12	4	9	36
4	32	13	13	17	11	44
4	32	14	14	65	11	44

Tabela 3.1: Rezultati grananja sa odsecanjem na test instancama $IPG_{\gamma=0.2}$ i $IPG_{\gamma=0.4}$

$IPG_{\gamma=0.2}$														
			$\beta = 4$				$\beta = 8$				$\beta = 16$			
$\|\Sigma\|$	m	$\|gg\|$	VS	$t(s)$	TVS	$t(s)$	VS	$t(s)$	TVS	$t(s)$	VS	$t(s)$	TVS	$t(s)$
2	8	20	**20**	0	**20**	0	20	0	20	0	20	0	20	0
2	8	24	27	0	26	0	**24**	0	**24**	0	24	0	24	0
2	8	28	**28**	0	**28**	0	28	0	28	0	28	0	28	0
2	16	20	**20**	0	**20**	0	20	0	20	0	20	0	20	0
2	16	24	**24**	0	**24**	0	24	0	24	0	24	0	24	0
2	16	28	**28**	0	**28**	0	28	0	28	0	28	0	28	0
2	32	20	**20**	0	**20**	0	20	0	20	0	20	0	20	0
2	32	24	**24**	0	**24**	0	24	0	24	0	24	0	24	0
2	32	28	**28**	0	**28**	0	28	0	28	0	28	0	28	0
4	8	12	**12**	0	**12**	0	12	0	12	0	12	0	12	0
4	8	13	**13**	0	**13**	0	13	0	13	0	13	0	13	0
4	8	14	**14**	0	**14**	0	14	0	14	0	14	0	14	0
4	16	12	**12**	0	**12**	0	12	0	12	0	12	0	12	0
4	16	13	**13**	0	**13**	0	13	0	13	0	13	0	13	0
4	16	14	**14**	0	**14**	0	14	0	14	0	14	0	14	0
4	32	12	**12**	0	**12**	0	12	0	12	0	12	0	12	0
4	32	13	**13**	0	**13**	0	13	0	13	0	13	0	13	0
4	32	14	**14**	0	**14**	0	14	0	14	0	14	0	14	0
$IPG_{\gamma=0.4}$														
2	8	20	19	0	**18**	0	**18**	0	18	0	18	0	18	0
2	8	24	**21**	0	**21**	0	21	0	21	0	21	0	21	0
2	8	28	**25**	0	**25**	0	25	0	25	0	25	0	25	0
2	16	20	21	0	20	0	20	0	**19**	0	**19**	0	19	0
2	16	24	**22**	0	22	0	22	0	**21**	0	22	0	21	0
2	16	28	**26**	0	**26**	0	26	0	26	0	26	0	26	0
2	32	20	24	0	22	0	22	0	**21**	0	**21**	0	21	0
2	32	24	27	0	26	0	**26**	0	**25**	0	26	0	25	0
2	32	28	32	0	30	0	31	0	**27**	0	**30**	0	27	0
4	8	12	**12**	0	**12**	0	12	0	12	0	12	0	12	0
4	8	13	**13**	0	**13**	0	13	0	13	0	13	0	13	0
4	8	14	**14**	0	**14**	0	14	0	14	0	14	0	14	0
4	16	12	**12**	0	**12**	0	12	0	12	0	12	0	12	0
4	16	13	**13**	0	**13**	0	13	0	13	0	13	0	13	0
4	16	14	**14**	0	**14**	0	14	0	14	0	14	0	14	0
4	32	12	**12**	0	**12**	0	12	0	12	0	12	0	12	0
4	32	13	**13**	0	**13**	0	13	0	13	0	13	0	13	0
4	32	14	**14**	0	**14**	0	14	0	14	0	14	0	14	0

Tabela 3.2: Rezultati pretrage bima na test instancama $IPG_{\gamma=0.2}$ i $IPG_{\gamma=0.4}$

$IP_{\gamma=0.1}$														
			$\beta = 100$				$\beta = 200$				$\beta = 400$			
$\lvert\Sigma\rvert$	m	$\lvert gg\rvert$	VS	$t(s)$	TVS	$t(s)$	VS	$t(s)$	TVS	$t(s)$	VS	$t(s)$	TVS	$t(s)$
2	10	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
2	10	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
2	10	500	**500**	0	**500**	0	500	0	500	0	500	0	500	0
2	10	2000	**2000**	1	**2000**	0	2000	1	2000	1	2000	3	2000	2
2	20	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
2	20	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
2	20	500	**500**	0	**500**	0	500	0	500	0	500	1	500	0
2	20	2000	**2000**	1	**2000**	0	2000	2	2000	1	2000	3	2000	3
2	40	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
2	40	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
2	40	500	**500**	0	**500**	0	500	1	500	0	500	0	500	1
2	40	2000	**2000**	1	**2000**	1	2000	2	2000	1	2000	4	2000	3
2	80	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
2	80	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
2	80	500	**500**	1	**500**	0	500	1	500	0	500	1	500	0
2	80	2000	**2000**	1	**2000**	1	2000	2	2000	3	2000	5	2000	5
4	10	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	10	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
4	10	500	**500**	0	**500**	0	500	0	500	1	500	0	500	1
4	10	2000	**2000**	1	**2000**	0	2000	2	2000	1	2000	4	2000	3
4	20	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	20	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
4	20	500	**500**	0	**500**	0	500	0	500	0	500	1	500	0
4	20	2000	**2000**	1	**2000**	1	2000	2	2000	2	2000	4	2000	4
4	40	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	40	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
4	40	500	**500**	0	**500**	0	500	1	500	0	500	1	500	0
4	40	2000	**2000**	2	**2000**	1	2000	3	2000	2	2000	5	2000	5
4	80	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	80	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
4	80	500	**500**	0	**500**	0	500	1	500	0	500	1	500	1
4	80	2000	**2000**	2	**2000**	1	2000	3	2000	3	2000	6	2000	7
16	10	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	10	100	**100**	0	**100**	0	100	0	100	0	100	1	100	0
16	10	500	**500**	0	**500**	0	500	1	500	0	500	1	500	1
16	10	2000	**2000**	2	**2000**	2	2000	5	2000	5	2000	11	2000	11
16	20	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	20	100	**100**	0	**100**	0	100	0	100	0	100	1	100	0
16	20	500	**500**	0	**500**	0	500	1	500	0	500	1	500	1
16	20	2000	**2000**	2	**2000**	2	2000	5	2000	5	2000	11	2000	11
16	40	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	40	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
16	40	500	**500**	0	**500**	0	500	1	500	0	500	2	500	1
16	40	2000	**2000**	3	**2000**	2	2000	6	2000	5	2000	13	2000	12
16	80	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	80	100	**100**	0	**100**	0	100	0	100	0	100	1	100	0
16	80	500	**500**	0	**500**	0	500	1	500	1	500	2	500	2
16	80	2000	**2000**	4	**2000**	3	2000	7	2000	7	2000	15	2000	15

Tabela 3.3: Rezultati pretrage bima na test instancama $IP_{\gamma=0.1}$

$IP_{\gamma=0.2}$														
			$\beta = 100$				$\beta = 200$				$\beta = 400$			
$\|\Sigma\|$	m	$\|gg\|$	VS	$t(s)$	TVS	$t(s)$	VS	$t(s)$	TVS	$t(s)$	VS	$t(s)$	TVS	$t(s)$
2	10	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
2	10	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
2	10	500	589	0	563	0	**558**	0	574	0	561	0	**546**	0
2	10	2000	2346	1	2320	1	2364	3	2326	2	**2311**	4	**2273**	4
2	20	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
2	20	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
2	20	500	503	0	**499**	0	**499**	0	499	0	499	1	499	0
2	20	2000	2474	3	**2269**	1	**2380**	2	2288	2	2432	4	2310	4
2	40	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
2	40	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
2	40	500	**500**	0	**500**	0	620	1	610	1	500	1	500	0
2	40	2000	**1999**	3	**1999**	1	1999	2	1999	2	1999	4	1999	4
2	80	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
2	80	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
2	80	500	**500**	0	**500**	0	500	1	500	0	500	1	500	0
2	80	2000	2208	3	2200	3	2207	6	2193	5	**2000**	5	**2000**	4
4	10	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	10	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
4	10	500	**500**	1	**500**	0	500	1	500	0	500	1	500	0
4	10	2000	**2000**	1	**2000**	0	2000	2	2000	1	2000	4	2000	4
4	20	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	20	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
4	20	500	**500**	0	**500**	0	500	0	500	0	500	1	500	0
4	20	2000	**2000**	1	**2000**	1	2000	2	2000	2	2000	4	2000	4
4	40	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	40	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
4	40	500	**500**	0	**500**	0	500	0	500	0	500	1	500	0
4	40	2000	**2000**	1	**2000**	1	2000	2	2000	2	2000	5	2000	5
4	80	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	80	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
4	80	500	**500**	0	**500**	0	500	1	500	0	500	1	500	1
4	80	2000	**2000**	2	**2000**	1	2000	3	2000	3	2000	7	2000	7
16	10	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	10	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
16	10	500	**500**	1	**500**	0	500	1	500	1	500	1	500	1
16	10	2000	**2000**	2	**2000**	2	2000	4	2000	5	2000	11	2000	10
16	20	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	20	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
16	20	500	**500**	0	**500**	0	500	1	500	0	500	1	500	1
16	20	2000	**2000**	2	**2000**	2	2000	5	2000	5	2000	11	2000	11
16	40	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	40	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
16	40	500	**500**	1	**500**	0	500	1	500	1	500	2	500	2
16	40	2000	**2000**	3	**2000**	2	2000	6	2000	6	2000	13	2000	13
16	80	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	80	100	**100**	0	**100**	0	100	0	100	0	100	1	100	0
16	80	500	**500**	1	**500**	0	500	1	500	1	500	2	500	2
16	80	2000	**2000**	3	**2000**	4	2000	7	2000	7	2000	16	2000	15

Tabela 3.4: Rezultati pretrage bima na test instancama $IP_{\gamma=0.2}$

$IP_{\gamma=0.4}$														
			$\beta = 100$				$\beta = 200$				$\beta = 400$			
$\|\Sigma\|$	m	$\|gg\|$	VS	$t(s)$	TVS	$t(s)$	VS	$t(s)$	TVS	$t(s)$	VS	$t(s)$	TVS	$t(s)$
2	10	50	**44**	0	**44**	0	44	0	44	0	44	0	44	0
2	10	100	**92**	0	89	0	92	0	89	0	92	0	**88**	0
2	10	500	436	0	434	0	**435**	0	428	0	436	0	**427**	0
2	10	2000	1747	1	1710	1	1743	1	1700	2	**1714**	3	**1696**	2
2	20	50	51	0	49	0	51	0	49	0	**50**	0	**48**	0
2	20	100	99	0	**93**	0	97	0	93	0	**94**	0	93	0
2	20	500	467	0	448	0	**452**	0	446	0	455	0	**443**	0
2	20	2000	**1785**	1	1790	1	1804	2	**1782**	1	1807	4	1782	3
2	40	50	**53**	0	52	0	53	0	**50**	0	53	0	50	0
2	40	100	99	0	97	0	**98**	0	96	0	98	0	**95**	0
2	40	500	477	0	466	0	**474**	1	**463**	0	475	0	468	1
2	40	2000	1834	1	1823	1	**1826**	2	**1815**	2	1848	5	1830	5
2	80	50	54	0	**52**	0	**53**	0	52	0	53	0	52	0
2	80	100	100	0	99	0	**99**	0	97	1	99	0	**96**	0
2	80	500	478	0	472	0	**476**	1	**470**	0	477	1	470	1
2	80	2000	1902	4	1884	2	1893	6	**1875**	4	**1887**	10	1883	10
4	10	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	10	100	116	0	102	0	103	0	**98**	0	**98**	0	98	0
4	10	500	650	0	605	0	607	0	604	0	**606**	0	**588**	1
4	10	2000	2502	1	2499	1	2533	3	**2475**	3	**2500**	6	2549	5
4	20	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	20	100	110	0	109	0	**100**	0	**100**	0	100	0	100	0
4	20	500	692	0	673	0	688	1	671	0	**684**	1	**663**	1
4	20	2000	2782	2	2704	2	**2760**	4	2709	3	2768	10	**2693**	7
4	40	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	40	100	**100**	0	**100**	0	100	0	100	0	100	1	100	0
4	40	500	737	1	706	0	**718**	1	**692**	0	719	2	692	1
4	40	2000	2910	3	2900	3	2960	7	2832	5	**2883**	13	**2820**	10
4	80	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
4	80	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
4	80	500	791	1	643	0	596	0	**500**	1	**500**	1	500	1
4	80	2000	3052	6	2840	4	2863	8	**2683**	6	**2558**	12	2702	13
16	10	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	10	100	**100**	0	**100**	0	100	0	100	0	100	0	100	1
16	10	500	**499**	0	**499**	0	499	1	499	1	499	2	499	1
16	10	2000	3827	8	5269	11	**3729**	15	**4896**	20	3835	32	4919	44
16	20	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	20	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
16	20	500	**500**	0	**500**	1	500	1	500	0	500	1	500	2
16	20	2000	**2000**	2	2206	3	2000	6	**2000**	5	2000	11	2000	12
16	40	50	**50**	0	**50**	0	50	0	50	0	50	1	50	0
16	40	100	**100**	0	**100**	0	100	0	100	0	100	1	100	0
16	40	500	**500**	0	**500**	0	500	1	500	1	500	1	500	2
16	40	2000	**2000**	3	**2000**	3	2000	6	2000	6	2000	13	2000	13
16	80	50	**50**	0	**50**	0	50	0	50	0	50	0	50	0
16	80	100	**100**	0	**100**	0	100	0	100	0	100	0	100	0
16	80	500	**500**	1	**500**	0	500	1	500	2	500	3	500	2
16	80	2000	**2000**	4	**2000**	3	2000	8	2000	8	2000	17	2000	16

Tabela 3.5: Rezultati pretrage bima na test instancama $IP_{\gamma=0.4}$

3.4 Diskusija eksperimentalnih rezultata

Najpre je potrebno analizirati generisane test instance problema kako bi se stvorila šira slika pre evaluacije algoritama. Ključni parametar je verovatnoća uklanjanja slova γ. Za fiksiranu dužinu nadniske gg, vrednosti ovog parametra kontrolišu dužinu generisanih reči u $\mathcal{L}$. Ako je $\gamma = k$ to znači da će generisane reči u proseku biti k puta kraće od gornje granice gg. Praćenjem vrednosti iz tabele 3.1, jasno se vidi razlika parametara $mindD$ i $maxD$ za različite vrednost γ, što pokazuje da su za veće vrednosti ovog parametra reči u $\mathcal{L}$ kraće, ali su pritom međusobno različitije. Odnosno, za manje vrednosti ovog parametra dobija se skup reči koje su međusobno sličnije i duže. Takođe ako se posmatra optimalna dužina nadniske koja je dobijena algoritmom granja sa odsecanjem može se primetiti da gg najčešće jeste optimalno rešenje problema za manje vrednosti γ, dok je za veće vrednosti ključan parametar m. Za male vrednosti parametra m primećuje se da gg nije optimalno rešenje, ali porastom vrednosti m raste i verovatnoća da gornja granica predstavlja rešenje koje je optimalno ili jako blizu optimalnog. Drugim rečima, iako su za veće vrednosti γ reči kraće i heterogenije, generisanjem većeg broja reči dobijamo jasniju sliku u evaluaciji. Instance PNZN generisane za veće vrednosti γ predstavljaju teže isntance problema zbog veće heterogenosti reči u $\mathcal{L}$, i to se može videti iz tabele 3.1. Na primer za instancu problema $\{n = 4, m = 32, |gg| = 14\}$ vreme izvrsavanja $t_{\gamma=0.2} = 64s$, a $t_{\gamma=0.4} = 65s$ je skoro identično iako su granice dubine dosta manje za $\gamma = 0.4$. Naravno, težina problema raste do vrednosti $\gamma = 0.5$, a zatim proporcionalno opada.

Rezultati algoritma grananja sa odsecanjem predstavljaju optimalne rezultate na test instancama, a vreme izvršavanja očekivano eksponencijalno raste u odnosu na veličinu azbuke. Može se reći da je za ovaj algoritam najbitnije da što pre pronađe barem jednu nadnisku, a zatim odsecanjem smanji prostor pretrage. Na test instancama $IP_{\gamma=0.1}$ i $IP_{\gamma=0.2}$ algoritam pretrage bima skoro uvek pronalazi nadnisku dužine $|gg|$. S obzirom na veću homogenost reči u ovim test instancama, može se pretpostaviti da su pronađene nadniske optimalne ili jako blizu optimalnih, a pogotovo za veće vrednosti parametra m. S obzirom na izbor pohlepnih heurističkih funkcija koje su korišćene, dobri rezultati na ovim test instancama su očekivani. Heuristička funkcija većinsko spajanje daje najbolje rezultate kada većina reči u $\mathcal{L}$ ima slične segmente uzastopnih slova. Slično je i kod heuristike težinsko većinsko spajanje, stim što se u ovom slučaju teži balansiranju dužina reči. Na test instanca-

ma $IP_{\gamma=0.2}$ primećuju se nešto lošiji rezultati na binarnoj azbuci za manje vrednosti parametra m. To je iz razloga što azbuku čine samo dva slova, pa iako je verovatnoća uklanjanja relativno mala, s obzirom da je azbuka mala i da je broj reči mali, uklanjanjem slova vise se narušava poredak u rečima što nije slučaj na većim azbukama. Test instance $IP_{\gamma=0.4}$ predstavljaju teže isntance PNZN. Rezultati su očekivano lošiji zbog velike heterogenosti reči. Pronađene nadniske nad binarnom azbukom su uvek kraće od dužine gg, što je očekivano uzevši u obzir to da su i generisane reči na ovim instancama problema u proseku 40% kraće od gg pa i optimalno rešenje mora biti kraće. Ovo važi i za veće azbuke ali u manjem intenzitetu, odnosno optimalno rešenje na većim azbukama je bliže gg. Nad binarnom azbukom se takođe primećuje da porastom m raste i dužina pronađenih nadniski i približava se dužini gornje granice što potvrđuje prethodno donesene zaključke o test instancama problema. Analizom predstavljenih tabela primećuje se da težinsko većinsko spajanje najčešće daje kraće nadniske u odnosu na većinsko spajanje što je i očekivano. Posmatranjem tabele 3.2 takođe se primećuje da ako pronađe optimalno rešenje težinsko većinsko spajanje to čini za manje vrednosti širine bima u odnosu na većinsko spajanje. Iz tabele 3.5 primećuje se da je za veće instance problema ($|gg| \in \{500, 2000\}$) potrebna i veća širina bima. S obzirom na to da su najbolje nadniske dobijene skoro podjednako često za $\beta = 200$ i $\beta = 400$, na ovim instancama problema bi se mogla izabrati vrednost za širinu bima unutar ovih intervala koja bi bila kompromis između kvaliteta dobijenih rešenja i vremena izvršavanja.

Glava 4

Zaključak i pravci daljeg rada

Problem najkraće zajedničke nadniske predstavlja izazovan i težak optimizacioni problem. Na njegovom što efikasnijem rešavanju radi se već četiri decenije. U ovom radu predstavljena je kombinacija trenutno najbolje predložene metaheuristike pretraga bima sa pohlepnim heurstičkim funkcijama većinsko spajanje i težinsko većinsko spajanje. Predstavljeni algoritam je efikasan jer inkrementalno gradi rezultijuću nadnisku, u cilju minimalnog vremena izvršavanja. Predložena heuristička funkcija težinsko većinsko spajanje se može proširiti gledanjem unapred (*eng.* look-ahead). To znači da se u svakoj iteraciji algoritma pretrage bima za svako parcijalno rešenje bira najbolje ocenjeno slovo u k prolaza težinskog većinskog spajanja. Jedini uslov je to što bi se u kombinaciji sa pretragom bima, težine slova morale rangirati ako želimo da inkrementalno gradimo nadniske, odnosno ne bi mogle da se koriste težine koje su dobijene direktno od heurističke funkcije, jer postoje delovi pretrage u kojima lošija rešenja počinju da dominiraju. Predloženi algoritam nije predstavljen zbog lošijih rezultata, a sličan algoritam rangiranja težina opisan je u radu iz 2007. godine "A Probabilistic Beam Search Approach to the Shortest Common Supersequence Problem" [1]. Ono što može biti predmet daljeg rada jeste pokušaj da se koristi gledanje unapred, ali da se težine slova ne rangiraju (rangiranjem težina gubi se na kvalitetu dobijenih težina). U tom slučaju nadniska ne bi mogla da se inkrementalno gradi zbog pomenutog problema, već bi se heuristička ocena svake parcijalne reči računala u svakom koraku pretrage bima. Postoje indikacije da ovako konstruisan algoritam može da nadmaši predstavljeni algoritam, ali bi vreme izvršavanja bilo daleko veće. Predstavljeni algoritam se takođe može modifikovati tako da umesto da u svakom koraku bira β elemenata sa najvećom heurističkom ocenom, taj izbor izvrši ruletskom selekcijom (*eng.* roulette selection) ili nekom drugom vrstom selekcije.

Pritom se u svakoj iteraciji može k ($k < \beta$) najbolje ocenjenih elemenata direktno prebaciti u novi bim. Predstavljeni su i algoritmi koji ne grade inkrementalno nadsniku počevši od prazne reči ε već najpre formiraju skup nadniski koje pokušavaju da redukuju što je više moguće. Takav algoritam je predložen u radu iz 2006. godine "The deposition and reduction algorithm" [7]. Opisani pohlepni algoritam pretrage bima bi mogao da se proširi tehnikama redukcije dobijene nadniske u cilju dobijanja kvalitetnijih rešenja i to može biti predmet daljeg istraživanja. Predstavljene pohlepne heurističke funkcije mogu se zameniti sa funkcijama koje ocenjuju verovatnoću da slučajno izabrana reč iz uniformne raspodele određene dužine predstavlja nadnisku. Takve heurističke funkcije počivaju na osnovnim zakonima verovatnoće i pretpostavljaju da su sve reči u $\mathcal{L}$ međusobno nezavisne [5]. Trenutno najkvalitetnija takva funkcija je AEL koja aproksimira očekivanu dužinu slučajno izabrane nadniske. Kombinovanje ovakvih heurističkih funkcija sa pohlepnim bi možda dovelo do boljih i kvalitetnijih rešenja. U svakom slučaju sve navedene tehnike treba ispitati i napraviti širu sliku o tome koje od njih su međusobno kompatibilne i u kojoj meri. Što se tiće generatora test instanci koji je predložen u ovom radu, dalja ispitivanja bi mogla biti na temu ulaznih parametara β, $n = |\Sigma|$ i m. Zapravo, mogao bi se ispitivati njihov međusoban odnos kako bi stepen sigurnosti da je gg optimalno rešenje bio maksimizovan, ili se predstavljena slučajna tehnika izbora može zameniti nekom kompleksnijom i sigurnijom.

Bibliografija

[1] Antonio J. Fernandez Christian Blum, Carlos Cotta and Francisco Gallardo. A Probabilistic Beam Search Approach to the Shortest Common Supersequence Problem. In *Lecture Notes in Computer Science*, 2007.

[2] Joseph Kruskal David Sankoff. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Center for the Study of Language and Inf, 1983.

[3] Garey and Johnson. Shortest common supersequence. on-line at: `https://www.csc.kth.se/~viggo/wwwcompendium/node165.html`.

[4] Storer JA. *Data compression: methods and theory.* Computer Science Press, 1988.

[5] Marc Huber Gunther Raidl Jonas Mayerhofer, Markus Kirchweger. A Beam Search for the Shortest Common Supersequence Problem Guided by an Approximate Expected Length Calculation. 2022.

[6] Frerk Schneider Jurgen Branke, Martin Middendorf. Improved heuristics and a genetic algorithm for finding short supersequences, 1998.

[7] Hon Wai Leong Kang Ning. Towards a better solution to the shortest common supersequence problem: the deposition and reduction algorithm, 2006.

[8] Esko Ukkonen Kari-Jouko Raiha. The shortest common supersequence problem over binary alphabet is NP-complete. In *Theoretical Computer Science*, 1981. Volume 16, Issue 2, Pages 187-198.

[9] David Maier. The Complexity of Some Problems on Subsequences and Supersequences, 1978.

[10] Gianluca Della Vedova Giancarlo Mauri Paolo Barone, Paola Bonizzoni. An Approximation Algorithm for the Shortest Common Supersequence Problem: An Experimental Analysis, 2001.

[11] Matt Payne. What is Beam Search? Explaining The Beam Search Algorithm. on-line at: `https://www.width.ai/post/what-is-beam-search`.

[12] Martin Middendorf Rene Michel. An island model based ant system with lookahead for the shortest supersequence problem. 2006.

[13] Timos K. Sellis. Multiple-query optimization, 1988. ACM Transactions on Database Systems (TODS), 13(1):23-52.

[14] Ming Li Tao Jiang. On the Approximation of Shortest Common Supersequences and Longest Common Subsequences, 1995.

[15] Ronald L. Rivest Clifford Stein Thomas H. Cormen, Charles E. Leiserson. *Introduction to Algorithms, Second edition.* MIT Press and McGraw-Hill, 2001.

[16] Petlja tim. Bektreking i gruba sila. on-line at: `https://petlja.org/biblioteka/r/Zbirka2/04%20bektreking_i_gruba_sila`.

[17] V. G. Timkovskii. Complexity of common subsequence and supersequence problems and related problems, 1989.

[18] Vadim G. Timkovsky. Some Approximations for Shortest Common Nonsubsequences and Supersequences. In *String Processing and Information Retrieval*, 2006.

Biografija autora

Miloš Miković je rođen u Beogradu 15. aprila 1997. godine. Išao je u osnovnu školu „Branko Radičević" u Batajnici, završio je odličnim uspehom i bio nagrađen Vukovom diplomom. Nakon završene osnovne škole upisao je prirodno-matematički smer Devete beogradske gimnazije „Mihailo Petrović Alas". Paralelno sa gimnazijom pohađao je mnogobrojne sportove. Najinteresantnija oblast u srednjoj školi mu je bila programiranje, tako da odlučuje da upiše smer Informatike na Matematičkom fakultetu u Beogradu. Na drugoj godini studija konkuriše i dobija studentsku stipendiju. Osnovne studije završava u roku od četiri godine i upisuje master studije na istom smeru. U međuvremenu se zapošljava kao game developer u firmi „International Game Technology" u Beogradu.