



Univerzitet u Beogradu
Matematički fakultet

Seminarski rad iz predmeta Računarska inteligencija

Nearest String Problem

Mentor:
Stefan Kapunac

Studenti:
Vojkan Panić 138/2019
Stefan Novaković 43/2021

1 Uvod

Nearest String problem (NSP) predstavlja jedan od klasičnih optimizacionih problema u oblasti kombinatorne optimizacije i teorije algoritama. Problem se javlja u različitim praktičnim oblastima, uključujući bioinformatiku (npr. pronalaženje konsenzus sekvence u DNK analizi), kodiranje informacija, prepoznavanje obrazaca i teoriju grešaka u komunikacionim sistemima.

Formalno, dat je skup od n stringova

$$S = \{s_1, s_2, \dots, s_n\},$$

gde je svaki string dužine m i definisan nad konačnim alfabetom Σ . Cilj je pronaći string

$$s \in \Sigma^m$$

koji minimizuje maksimalno Hamingovo rastojanje do svih ulaznih stringova:

$$\min_{s \in \Sigma^m} \max_{i=1}^n d_H(s, s_i),$$

gde je d_H Hamingovo rastojanje.

Hamingovo rastojanje između dva stringa iste dužine predstavlja broj pozicija na kojima se odgovarajući karakteri razlikuju. Ova mera prirodno kvantifikuje „sličnost“ između dva stringa.

Poznato je da je Nearest String problem NP-težak, što znači da ne postoji poznat polinomijalni algoritam koji garantuje optimalno rešenje za sve instance problema. Zbog toga je u literaturi predloženo više tačnih i heurističkih metoda, uključujući brute-force pretragu, branch and bound algoritme, linearno programiranje, genetske algoritme i različite metaheuristike, poput lokalne pretrage i Variable Neighborhood Search (VNS).

U ovom radu razmatra se rešavanje Nearest String problema primenom heurističkih metoda, konkretno VNS-a sa lokalnom pretragom, kao i genetskog algoritma. Ovi pristupi se porede međusobno, kao i sa brute-force metodom na malim instancama, kako bi se procenio kvalitet i efikasnost predloženih rešenja.

2 Opis predloženog rešenja

2.1 Algoritam grube sile (brute-force)

Algoritam grube sile predstavlja tačan, ali računski izuzetno zahtevan pristup rešavanju Nearest String problema. Osnovna ideja algoritma zasniva se na potpunoj pretrazi prostora svih mogućih rešenja.

Algoritam se sastoji od sledećih koraka:

- generisanje svih mogućih stringova dužine m korišćenjem karaktera iz alfabeta Σ ,
- za svaki generisani string računanje maksimalnog Hamingovog rastojanja u odnosu na sve ulazne stringove,
- poređenje dobijenih Hamingovih distanci u potrazi za najmanjom.

Zbog eksponencijalne veličine prostora pretrage, vremenska složenost ovog pristupa je $O(|\Sigma|^m \cdot n \cdot m)$, što ga čini neprimenljivim za veće instance problema. Ipak, brute-force metoda je korisna za male instance, jer garantuje pronalaženje optimalnog rešenja i služi kao referentna tačka za poređenje i evaluaciju heurističkih i optimizacionih metoda, poput VNS-a i genetskog algoritma.

2.2 Variable Neighborhood Search (VNS) sa lokalnom pretragom

2.2.1 Opšti princip rada algoritma

Predloženi pristup zasniva se na iterativnom poboljšavanju kandidata-rešenja, pri čemu se koristi promena susedstva kako bi se izbeglo zaglavljivanje u lokalnim minimumima.

Algoritam se sastoji od sledećih faza:

- Inicijalizacija
- Shaking faza
- Lokalna pretraga
- Pravilo prihvatanja
- Promena susedstva

Ovaj pristup predstavlja implementaciju standardnog VNS okvira, prilagođenu Nearest String problemu.

2.2.2 Funkcija cilja

Funkcija cilja definisana je kao:

$$f(s) = \max_{i=1}^n d_H(s, s_i).$$

Cilj algoritma je minimizacija ove vrednosti.

2.2.3 Inicijalizacija

Početni string se generiše tako da na svakoj poziciji bira karakter iz skupa karaktera koji se pojavljuju u toj „koloni“ ulaznih stringova. Time se polazi od smislenog kandidata koji je u skladu sa datim instancama.

2.2.4 Shaking

Za dato k , algoritam nasumično izabere k pozicija i promeni njihove karaktere. To predstavlja pretragu u k -susedstvu: rešenja koja se razlikuju u k karaktera.

- malo $k \rightarrow$ male, pažljive promene
- veće $k \rightarrow$ veći „skokovi“ koji pomažu izlazak iz lokalnog minimuma

2.2.5 Lokalna pretraga

Nakon shaking-a, radi se lokalna pretraga koja pokušava da poboljša rešenje menjajući karaktere jedan po jedan i u svakoj iteraciji prihvata promenu karaktera ako ona dovodi do poboljšanja funkcije cilja. Ovim postupkom dolazimo do lokalno najboljeg rešenja.

2.2.6 Pavilo prihvatanja

Ako novo rešenje smanji vrednost funkcije cilja, ono se prihvata odmah. Ponekad se prihvata i jednako dobro rešenje (sa određenom verovatnoćom) kako bi se održala raznovrsnost i izbeglo stagniranje algoritma.

2.2.7 Promena susedstva

Ako nema napretka u manjem susedstvu, povećava se k (prelazak na šire susedstvo). Ako se pronađe bolje rešenje, algoritam se vraća na manje k i nastavlja intenzivno poboljšavanje.

2.2.8 Analiza vremenske složenosti

Neka je n broj ulaznih stringova, m dužina svakog stringa, I maksimalan broj iteracija algoritma, a K broj razmatranih susedstava u okviru jedne iteracije. U implementaciji se parametri I i K određuju heuristički, pri čemu je K najčešće mala konstanta, dok I zavisi od željenog vremena izvršavanja i kvaliteta rešenja.

Prilikom inicijalizacije algoritma generiše se početni kandidat dužine m tako što se na svakoj poziciji bira karakter iz skupa karaktera koji se pojavljuju u odgovarajućoj koloni ulaznih stringova. Ovaj postupak zahteva prolazak kroz sve ulazne stringove, kao i inicijalno izračunavanje maksimalne Hamingove distance u odnosu na svih n stringova, što ukupno daje trošak

$$O(n \cdot m).$$

U toku jedne iteracije algoritma razmatra se više susedstava različitih veličina. Za svako susedstvo izvršava se operacija *shaking*, kojom se menja k pozicija u trenutnom rešenju. Ova operacija zahteva izbor k pozicija i generisanje novog kandidata, što zahteva $O(k \cdot m)$ vremena, dok se izbor dozvoljenih karaktera po kolonama obavlja u vremenu $O(k \cdot n)$. Nakon toga sledi evaluacija novog kandidata, čiji je trošak $O(n \cdot m)$.

Dominantan deo vremena u okviru jednog susedstva otpada na lokalnu pretragu sa pravilom *first improvement*. U najgorem slučaju lokalna pretraga ispituje sve m pozicija za svaku potencijalnu korekciju, pri čemu se za svaku probu računa maksimalna Hamingova distance u odnosu na svih n stringova. Kako se vrednost ciljne funkcije može poboljšati najviše m puta, ukupan trošak lokalne pretrage iznosi

$$O(n \cdot m^3).$$

Kako se lokalna pretraga izvršava za svako od K susedstava u jednoj iteraciji, ukupni trošak jedne iteracije algoritma je

$$O(K \cdot n \cdot m^3).$$

Ukupna vremenska složenost VNS algoritma kroz svih I iteracija iznosi

$$O(I \cdot K \cdot n \cdot m^3).$$

U slučaju kada je broj susedstava K ograničen malom konstantom, složenost algoritma je kubna u odnosu na dužinu stringa i linearna u odnosu na broj ulaznih instanci. Dominantan trošak predstavlja evaluacija ciljne funkcije, dok ostale operacije imaju znatno manji uticaj na ukupno vreme izvršavanja.

2.3 Genetski algoritam

2.3.1 Opšti princip rada algoritma

Genetski algoritam predstavlja populacionu metaheuristiku inspirisanu procesima prirodne selekcije i evolucije. U ovom pristupu, svaki kandidat za rešenje Nearest String problema predstavlja jednu jedinku u populaciji, dok se iterativnim primenom selekcije, ukrštanja i mutacije populacija evoluira ka kvalitetnijim rešenjima.

Algoritam se sastoji od sledećih faza:

- Inicijalizacija populacije,
- Procena prilagođenosti (fitness),
- Selekcija roditelja,
- Ukrštanje,
- Mutacija,
- Elitizam i formiranje nove generacije.

Proces se ponavlja kroz unapred definisan broj generacija.

2.3.2 Reprezentacija rešenja

Svaka jedinka je predstavljena stringom dužine m , gde svaki karakter pripada alfabetu odgovarajuće pozicije, formiranom na osnovu ulaznih stringova. Na ovaj način se obezbeđuje da sva generisana rešenja budu validna u odnosu na datu instancu problema.

2.3.3 Funkcija prilagođenosti

Funkcija prilagođenosti definisana je kao maksimalno Hamingovo rastojanje između kandidata-rešenja i svih ulaznih stringova:

$$f(s) = \max_{i=1}^n d_H(s, s_i).$$

Pošto je cilj Nearest String problema minimizacija ove vrednosti, jedinke sa manjom vrednošću funkcije prilagođenosti smatraju se kvalitetnijim.

2.3.4 Inicijalizacija populacije

Početna populacija se generiše nasumično, pri čemu se za svaku poziciju u stringu bira karakter iz odgovarajućeg alfabeta te kolone. Veličina populacije i broj generacija zavise od dimenzija problema i određuju se heuristički.

2.3.5 Selekcija

Za selekciju roditelja koristi se turnirska selekcija. Nasumično se bira k jedinki iz populacije, a za roditelja se odabira ona jedinka koja ima najbolju (najmanju) vrednost funkcije prilagođenosti. Ovakav pristup omogućava dobar balans između eksploatacije kvalitetnih rešenja i očuvanja raznovrsnosti populacije.

2.3.6 Ukrštanje

Primenjuje se uniformno ukrštanje. Za svaku poziciju u stringu potomka, sa verovatnoćom 0.5, bira se gen od prvog ili drugog roditelja. Na ovaj način se omogućava fina kombinacija genetskog materijala roditelja i stvaranje raznovrsnih potomaka.

2.3.7 Mutacija

Nakon ukrštanja, nad potomkom se primenjuje mutacija. Za svaki gen u stringu potomka, sa unapred definisanom (jako malom) verovatnoćom, vrši se zamena karaktera nasumično izabranim karakterom iz alfabeta odgovarajuće pozicije. Mutacija doprinosi očuvanju raznovrsnosti populacije i sprečava prerano konvergiranje algoritma.

2.3.8 Elitizam i zamena generacija

Najbolje jedinke iz tekuće generacije se direktno prenose u narednu generaciju. Ostatak nove populacije formira se primenom selekcije, ukrštanja i mutacije. Na ovaj način se garantuje da se kvalitet najboljeg pronađenog rešenja ne pogoršava tokom evolucije.

2.3.9 Kriterijum zaustavljanja

Algoritam se zaustavlja nakon unapred definisanog broja generacija. Kao rezultat se vraća najbolja pronađena jedinka, odnosno string sa najmanjom vrednošću maksimalnog Hamingovog rastojanja.

2.3.10 Analiza vremenske složenosti

Neka je n broj ulaznih stringova, m dužina stringa, P veličina populacije, a G broj generacija. U implementaciji se ovi parametri određuju heuristički, pri čemu važi da je $P = \Theta(\log n + \log m)$, dok je $G = \Theta(\log m)$.

Prilikom inicijalizacije populacije generiše se P jedinki. Za svaku jedinku potrebno je formirati string dužine m i izračunati vrednost funkcije prilagođenosti, što zahteva računanje maksimalne Hamingove distance u odnosu na svih n ulaznih stringova. Trošak inicijalizacije je zato reda

$$O(P \cdot n \cdot m).$$

U toku jedne generacije dominantan deo vremena otpada na kreiranje novih jedinki. Operacije selekcije ($O(k) = O(1)$) i sortiranja populacije ($O(P \cdot \log P)$) imaju manju složenost i ne utiču presudno na ukupno vreme izvršavanja. Za svakog potomka potrebno je izvršiti ukrštanje i mutaciju, što zahteva $O(m)$ vremena, ali je ponovo najskuplja operacija izračunavanje funkcije prilagođenosti, koje iznosi $O(n \cdot m)$. Kako se ova operacija izvršava za približno P jedinki, dobijamo da je trošak jedne generacije

$$O(P \cdot n \cdot m).$$

Ukupna vremenska složenost algoritma kroz sve generacije iznosi

$$O(G \cdot P \cdot n \cdot m).$$

Zamenom izraza za P i G dobija se da je složenost genetskog algoritma polinomijalna u odnosu na n i m , uz logaritamski rast parametara populacije i broja iteracija. Dominantan trošak predstavlja evaluacija funkcije prilagođenosti, dok ostale operacije imaju znatno manji uticaj.

Ovakva struktura omogućava dobru skalabilnost algoritma i primenu na instancama za koje bi egzaktne metode, poput brute-force pristupa, bile praktično neizvodljive.

3 Eksperimentalni rezultati

3.1 Test instance

Eksperimenti su sprovedeni na malim instancama koje omogućavaju proveru korektnosti predloženih algoritama poređenjem sa optimalnim rešenjima dobijenim brute-force metodom i većim instancama problema, na kojima brute-force pristup nije praktično primenljiv, te se performanse algoritama porede isključivo međusobno.

3.2 Poređenje metoda

Upoređeni su sledeći pristupi:

- Potpuna pretraga (brute-force),
- Variable Neighborhood Search sa lokalnom pretragom,
- Genetski algoritam.

3.3 Opis test primera

U cilju evaluacije tačnosti i performansi predloženih algoritama za rešavanje *Nearest String* problema, konstruisan je skup test primera podeljenih u nekoliko smislenih kategorija. Testovi su dizajnirani tako da obuhvate trivijalne slučajeve, granične konfiguracije, kao i instance koje omogućavaju analizu uticaja pojedinih parametara problema na vreme izvršavanja algoritama.

3.3.1 Trivijalni testovi

- **Identični stringovi.** Svi ulazni stringovi su međusobno identični. Optimalno rešenje je trivijalno i jednako svakom od ulaznih stringova, uz Hamingovu distancu jednaku nuli. Ovaj test služi za proveru osnovne korektnosti implementacije.
- **Jedan string.** Ulaz sadrži samo jedan string. U ovom slučaju, optimalno rešenje mora biti upravo taj string, jer je maksimalna Hamingova distanca jednaka nuli. Test proverava ponašanje algoritama u minimalnom mogućem ulazu.

3.3.2 Granični slučajevi

- **Binarni komplement.** Ulaz čine dva stringa sastavljena od suprotnih simbola (npr. AAAA i BBBB). Nijedan kandidat ne može biti blizak oba stringa istovremeno, što dovodi do maksimalne moguće Hamingove distance. Ovaj test predstavlja ekstremni konflikt između instanci.
- **Kolonska većina.** Većina ulaznih stringova se poklapa po kolonama, dok manji broj stringova sadrži odstupanja. Očekuje se da optimalno rešenje odgovara kolonskoj većini. Test proverava sposobnost algoritama da identifikuju dominantne obrasce u podacima.
- **Dve suprotne grupe.** Ulazni skup se sastoji od dve velike i međusobno suprotne grupe stringova (npr. polovina AAAAAA, polovina BBBBBB). Ovaj slučaj je posebno izazovan za metaheuristike, jer postoji više lokalno optimalnih rešenja.

3.3.3 Mali nasumično generisani testovi

Testovi se generišu sa malim brojem instanci i kratkim stringovima, tako da je moguće izvršiti i brute-force algoritam u razumnom vremenu. Ovi testovi se koriste za validaciju rezultata metaheurističkih algoritama poređenjem sa optimalnim rešenjem dobijenim iscrpnom pretragom.

3.3.4 Srednji nasumično generisani testovi

Testovi sadrže umereni broj instanci i dužinu stringa, pri čemu je brute-force algoritam još uvek izvodljiv, ali zahteva značajno vreme izvršavanja. Cilj ovih testova je da se jasno prikaže razlika u performansama između egzaktnih i heurističkih pristupa.

3.3.5 Veliki nasumično generisani testovi

Testovi sadrže veliki broj instanci i duže stringove, zbog čega brute-force pristup postaje neizvodljiv. Ovi testovi se koriste isključivo za procenu skalabilnosti i performansi genetskog algoritma i VNS metode na realističnim, velikim ulazima.

3.3.6 Uticaj veličine azbuke

U ovoj grupi testova varira se veličina azbuke $|\Sigma|$, dok su broj instanci i dužina stringa fiksni. Cilj je analiza uticaja veličine prostora pretrage na kvalitet rešenja i vreme izvršavanja algoritama. Brute-force algoritam nije pokretan u ovim eksperimentima, već je fokus bio na međusobnom poređenju performansi genetskog algoritma i VNS metode.

3.3.7 Uticaj broja instanci

U ovom skupu testova varira se broj ulaznih stringova n , dok su dužina stringa i veličina azbuke konstantni. Testovi omogućavaju empirijsku analizu skaliranja algoritama u odnosu na broj instanci. Brute-force pristup u ovim uslovima nije primenjivan, jer cilj nije bio određivanje optimalnog rešenja, već poređenje efikasnosti i kvaliteta rešenja koje daju genetski algoritam i VNS.

3.3.8 Uticaj dužine stringa

Ovi testovi ispituju ponašanje algoritama pri promeni dužine stringa m , uz fiksni broj instanci i konstantnu veličinu azbuke. Kako je evaluacija Hamingove distance linearna u m , očekuje se direktan uticaj ovog parametra na vreme izvršavanja. Zbog naglog rasta prostora pretrage sa povećanjem m , brute-force algoritam nije izvršavan, te se analiza zasniva isključivo na poređenju metaheurističkih metoda.

3.3.9 Rezultati testova

Kategorija testa	Opis testa	$ \Sigma $	n	m	BF dist	BF vreme	GA dist	GA vreme	VNS dist	VNS vreme
Trivijalni	Identični stringovi	3	10	5	0	0.002s	0	0.011s	0	0.092s
Trivijalni	Jedan string	2	1	5	0	0.000s	0	0.003s	0	0.020s
Granični slučaj	Binarni komplement	2	2	4	2	0.000s	2	0.003s	2	0.027s
Granični slučaj	Kolonska većina	3	5	6	1	0.003s	1	0.009s	1	0.073s
Granični slučaj	Dve suprotne grupe	2	100	6	3	0.004s	3	0.105s	3	0.882s
Mali br. instanci	Validacija sa BF	3	5	6	4	0.003s	4	0.009s	4	0.081s
Mali br. instanci	Validacija sa BF	3	5	5	3	0.001s	3	0.007s	3	0.056s
Mali br. instanci	Validacija sa BF	3	5	7	3	0.009s	3	0.012s	3	0.106s
Srednji br. instanci	Performanse u odnosu na BF	8	25	8	7	319.254s	8	0.043s	7	0.632s
Veliki br. instanci	Test performansi	26	1000	12	N/A	N/A	12	2.626s	12	21.431s
Uticaaj $ \Sigma $	$ \Sigma = 2$	2	100	12	N/A	N/A	8	0.240s	8	5.600s
Uticaaj $ \Sigma $	$ \Sigma = 3$	3	100	12	N/A	N/A	10	0.239s	10	6.983s
Uticaaj $ \Sigma $	$ \Sigma = 5$	5	100	12	N/A	N/A	11	0.236s	11	6.164s
Uticaaj $ \Sigma $	$ \Sigma = 10$	10	100	12	N/A	N/A	12	0.235s	12	2.430s
Uticaaj $ \Sigma $	$ \Sigma = 26$	26	100	12	N/A	N/A	12	0.242s	12	2.092s
Uticaaj n	n = 10	3	10	12	N/A	N/A	9	0.037s	8	0.401s
Uticaaj n	n = 50	3	50	12	N/A	N/A	10	0.128s	10	4.033s
Uticaaj n	n = 100	3	100	12	N/A	N/A	11	0.238s	11	2.842s
Uticaaj n	n = 500	3	500	12	N/A	N/A	12	1.258s	12	9.139s
Uticaaj n	n = 1000	3	1000	12	N/A	N/A	12	2.701s	12	21.972s
Uticaaj m	m = 1	3	100	1	N/A	N/A	1	0.002s	1	0.000s
Uticaaj m	m = 5	3	100	5	N/A	N/A	5	0.088s	5	0.739s
Uticaaj m	m = 10	3	100	10	N/A	N/A	9	0.203s	9	2.495s
Uticaaj m	m = 20	3	100	20	N/A	N/A	17	0.444s	17	24.921s
Uticaaj m	m = 50	3	100	50	N/A	N/A	41	1.504s	40	137.375s

3.4 Eksperimentalno okruženje

Eksperimenti su izvršeni u sledećem okruženju:

- operativni sistem: Linux / Windows,
- programski jezik: Python,
- razvojno okruženje: Jupyter Notebook,
- hardver: standardni laptop sa višejezgrenim procesorom i 16 GB RAM-a.

4 Zaključak

U ovom radu razmatran je Nearest String problem, kao reprezentativan NP-težak problem kombinatorne optimizacije, i analizirano je njegovo rešavanje primenom tri različita pristupa: potpune pretrage (brute-force), Variable Neighborhood Search metode sa lokalnom pretragom, kao i genetskog algoritma. Cilj istraživanja bio je ispitivanje tačnosti, performansi i skalabilnosti ovih pristupa kroz pažljivo dizajniran skup test primera.

Rezultati trivijalnih i graničnih testova potvrđuju korektnost svih implementiranih algoritama. U slučajevima identičnih stringova i minimalnog ulaza, svi algoritmi pronalaze optimalno rešenje sa Hamingovom distancom jednakom nuli, dok se u graničnim konfiguracijama (binarni komplement, dve suprotne grupe) ispravno identifikuje neminovno veća optimalna distanca. Time je potvrđeno da algoritmi pravilno rešavaju i ekstremne konfiguracije ulaza.

Na malim i srednje velikim instancama, gde je brute-force pristup još uvek izvodljiv, pokazano je da i VNS sa lokalnom pretragom i genetski algoritam skoro uvek pronalaze optimalna rešenja, uz višestruko kraće vreme izvršavanja. Posebno se ističe drastična razlika u performansama kod srednjih instanci, gde

brute-force metoda zahteva vreme reda veličine od nekoliko minuta, dok heuristički pristupi daju rezultate u deliću sekunde. Ovi testovi jasno ilustruju praktična ograničenja egzaktnih metoda i opravdanost primene metaheuristika.

Kod velikih instanci, gde brute-force pristup postaje potpuno neizvodljiv, genetski algoritam i VNS metoda pokazuju različite karakteristike skaliranja. Analiza uticaja veličine azbuke, broja instanci i dužine stringa pokazuje da oba algoritma pravilno reaguju na povećanje složenosti problema, ali da genetski algoritam u proseku postiže bolje performanse, naročito pri većim vrednostima parametara n i m . VNS metoda, iako stabilna u pogledu kvaliteta rešenja, pokazuje znatno veće vreme izvršavanja za duže stringove, što je u skladu sa njenom intenzivnom lokalnom pretragom.

Eksperimenti vezani za uticaj pojedinih parametara dodatno potvrđuju očekivana teorijska svojstva problema: povećanje veličine azbuke i dužine stringa direktno proširuje prostor pretrage i povećava vreme evaluacije, dok rast broja instanci utiče na funkciju cilja kroz složeniju procenu maksimalne Hamingove distance. Uprkos tome, heuristički algoritmi zadržavaju prihvatljive performanse i stabilan kvalitet rešenja.

Na osnovu sprovedene analize može se zaključiti da su metaheuristički pristupi, posebno genetski algoritam, veoma pogodni za rešavanje Nearest String problema u praktičnim scenarijima. Iako ne garantuju optimalnost za sve instance, oni nude dobar kompromis između kvaliteta rešenja i vremena izvršavanja, što ih čini neuporedivo praktičnijim od egzaktnih metoda za veće ulaze.

Mogući pravci daljeg unapređenja uključuju hibridizaciju genetskog algoritma sa lokalnom pretragom, adaptivno podešavanje parametara tokom izvršavanja, kao i paralelizaciju evaluacije populacije ili susedstva. Ovakva unapređenja mogla bi dodatno poboljšati performanse i skalabilnost predloženih pristupa.