UNIVERZITET U BANJOJ LUCI
PRIRODNO-MATEMATIČKI FAKULTET
MATEMATIKA I INFORMATIKA, INFORMATIKA
UVOD U VJEŠTAČKU INTELIGENCIJU

Seminarski rad na temu:

Problem pronalaženja najduže ponovljene podsekvence (ograničena verzija)

Student:
Danica Babić (36/20)

Predmetni profesor:
doc. dr Marko Đukanović

Banja Luka, oktobar 2025.

Sadržaj:

1. Uvod

Problem *najduže ponovljene podsekvence (ograničena verzija)* (engl. *Longest Duplicated Subsequence Problem – constrained version*) spada u klasu problema vezanih za analizu sekvenci i stringova, sa širokom primenom u računarstvu i bioinformatici. Cilj je pronaći najdužu podsekvencu datog niza koji se pojavljuje najmanje dva puta, uz dodatna ograničenja definisana ograničenom (*constrained)* verzijom problema. Konkretno, tražena podsekvenca mora sadržati sve simbole iz definisanog alfabeta, a broj pojavljivanja svakog simbola je ograničen unapred zadatim parametrom. Ova varijanta čini problem zahtevnijim, ali i realističnijim za praktične primene.

Motivacija za proučavanje problema proizilazi iz činjenice da se ponovljene podsekvence često javljaju u realnim podacima. Njihovo prepoznavanje može imati značajnu primenu: u bioinformatici omogućava identifikaciju ponavljajućih obrazaca u DNK i proteinskim sekvencama, u obradi prirodnog jezika pomaže u detekciji redundantnih struktura u tekstovima, dok u kompresiji podataka doprinosi efikasnijem iskorišćenju regularnosti ulaznih nizova.

Cilj ovog rada je implementacija i eksperimentalno poređenje dve heurističke metode za rešavanje *Longest Duplicated Subsequence Problem (constrained version)*: jednostavne pohlepne (*greedy*) strategije i naprednije metode pretrage sa promenljivim susedstvom (*Variable Neighborhood Search – VNS*). Na ovaj način ispituje se koliko različite heuristike mogu pružiti kvalitetna rešenja za problem koji je u opštem slučaju NP-težak.

Kod i podaci se nalaze na GitHub repozitorijumu. Podaci su u folderu data.

2. Pregled literature

Problem najduže ponovljene podsekvence **(LDS)** i njegove varijante predstavljaju značajnu oblast u algoritamskom istraživanju, jer kombinuju analizu stringova sa optimizacijom i heurističkim metodama. Klasični cilj problema je pronaći najdužu podsekvencu P datog niza S koji se pojavljuje najmanje dva puta. Ograničena verzija problema (constrained version) dodatno zahteva da svaki simbol iz definisanog alfabeta Σ bude zastupljen u rešenju, a broj pojavljivanja svakog simbola bude ograničen na maksimalnu vrednost d. Ova ograničenja čine problem kompleksnijim i bližim realnim primenama, kao što su bioinformatika, obrada teksta i kompresija podataka.

2.1 Klasične varijante problema

Prvi značajniji algoritamski pristupi za LDS problem koriste dinamičko programiranje (DP) i brute-force metode.

- **Brute-force pristup**: proverava sve moguće podsekvence niza S i detektuje one koji se pojavljuju više puta. Ova metoda je eksponencijalna ($O(2^n)$) i praktično neprimenjiva za srednje i velike instance.
- **Dinamičko programiranje (DP)**: koristi matricu dimenzija n×n kako bi se pratila dužina najduže podsekvence koji se ponavlja, uz izbegavanje preklapanja. Kompleksnost je $O(n^2)$ ili $O(n^3)$, zavisno od implementacije. Pogodna je za male instance, ali ne može efikasno da rešava ograničenu verziju sa dodatnim ograničenjima na simbole i maksimalna ponavljanja.

2.2 Najduža podsekvenca sa ponovljenim slovima

Uvodi se problem Longest Letter-Duplicated Subsequence (LLDS), gde tražena ponovljena podsekvenca sadrži isključivo pojedinačne simbole to jeste „slova“ (eng. „letters“).

- Svaka instanca niza S se analizira kroz sekvencu $x_1^{d1}x_2^{d2}\cdots x_k^{dk}$ pri čemu je svaki x_i pojedinačno slovo, $d_i \geq 2$ i $x_i \neq x_{i+1}$.
- Ograničena verzija: svi simboli iz Σ moraju biti zastupljeni, a broj ponavljanja svakog simbola ≤ d.
- Problem postaje NP-težak za veće d, pa su potrebne heuristike ili aproksimacije za praktično rešavanje.

Ograničenje LLDS vs praktična primena: pogodan za jednostavna pravila ponavljanja pojedinačnih simbola, ali nije dovoljno fleksibilan za duže blokove karaktera.

2.3 Najduža podsekvenca sa ponovljenim podsekvencom

Generalizacija LLDS u Longest Subsequence-Repeated Subsequence (LSRS).

- Tražena podsekvenca $P=x_1^{d1}x_2^{d2}\cdots x_k^{dk}$ može sadržati cele podskvenca x_i, a ne samo pojedinačna slova.
- Ograničena verzija: svi simboli iz Σ moraju biti uključeni, broj ponavljanja ≤ d.

- NP-težak za $d \geq 4$, polinomski rešiv za manje d.

Prednost LSRS nad LLDS: fleksibilniji, primenljiviji za duže obrasce u bioinformatici i obradi teksta.

2.4 Heuristički pristupi i metaheuristike

Zbog NP-težine problema u ograničenoj verziji, često se koriste heuristike:

- **Pohlepni algoritmi (Greedy algoritmi)**: iterativno grade rešenje birajući lokalno optimalni izbor. Brzi, jednostavni, ali često suboptimalni.
- **Metaheuristike**: Variable Neighborhood Search (VNS), genetski algoritmi, simulated annealing, tabu search.
    - VNS menja susedstva rešenja kako bi se istražili različiti delovi prostora mogućih rešenja, prevazilazeći lokalne optime.
    - Ove metode su pogodne za ograničene verzije, gde klasične metode ne pronalaze globalni optimum.

3. Formalna definicija problema i primeri

3.1 Definicija problema

Ulaz je niz $S=s_1s_2\ldots s_n$, alfabet Σ koji sadrži sve simbole niza i maksimalan broj ponavljanja svakog simbola $d\geq 1$. Cilj je pronaći najdužu podsekvencu P koja:

1. Se pojavljuje najmanje dva puta u S (ne nužno uzastopno).
2. Sadrži sve simbole iz Σ.
3. Nijedan simbol se ne pojavljuje više puta nego što je dozvoljeno ($\leq d$).

Zamislimo niz S kao tekst ili DNK sekvencu. Cilj je pronaći duplirane obrasce:

- Ako tražimo samo ponovljenu podsekvencu, možemo naći bilo koji segment koji se pojavljuje više puta.
- Sa ograničenjima, obrazac mora uključiti sve simbole (npr. A, C, G, T u DNK), ali nijedan simbol se ne sme pojaviti više od d puta.
- Ovo stvara složenu optimizacijsku situaciju gde jednostavna detekcija ponavljanja više nije dovoljna.

Formalno: $P=p_1p_2\ldots p_m$, $p_i\in\Sigma$, gde postoje najmanje dve pozicije $1\leq i_1<i_2\leq n-m+1$ tako da se P pojavljuje na tim pozicijama i zadovoljava ograničenja ponavljanja i zastupljenosti simbola. .

3.2 Primeri

Primer 1: mali niz

- Ulaz: S="AABCBC", Σ={A, B, C}, d=2.
- Najduža ponovljena podsekvenca bez ograničenja: P="ABC".
- Prvo pojavljivanje podsekvence je označeno roza bojom:
    - A na indeksu 0, B na indeksu 2, C na indeksu 3
- Drugo pojavljivanje podsekvence je označeno ljubičastom bojom:
    - A na indeksu 1, B na indeksu 4, C na indeksu 5

A	A	B	C	B	C
0	1	2	3	4	5

Tabela 1. Vizuelni primer najduže ponavljajuće podsekvence

Primer 2: ilustracija ograničenja

- **Ulaz:** S = "ABABCAB", Σ = {A, B, C}, d = 2
    - Nema podsekvence koja sadrži sva slova iz Σ i pojavljuje se bar dva puta.
- **Ulaz:** S = "ABABCABC", Σ = {A, B, C}, d = 2

- Moguće rešenje: P = "ABC"

S	Σ	d	Najduža ponovljena podsekvenca
ABABCAB	{A, B, C}	2	? (uključiti C)
ABABCABC	{A, B, C}	2	ABC

Tabela 2. Najduža ponovljena podsekvenca sa obaveznim prisustvom svih simbola iz alfabeta Σ

Primer 3: uticaj ogranicenja broja ponavljanja simbola u podsekvencama (d)

Za d = 1, d = 2 i d = 3:

- Ulaz: test2="ABABABABABAB", Σ={A, B}
- za d=1: rešenje je da se "AB" pojavilo 6 puta, za d=2: rešenje je da se "ABAB" pojavilo 3 puta i za d=3: rešenje je da se "ABABAB" pojavilo 3 puta, što možemo videti i u tabeli ispod

Instance	d	Rešenje
ABABABABABAB	1	AB (6)
ABABABABABAB	2	ABAB (3)
ABABABABABAB	3	ABABAB (2)

Tabela 3. Najduža ponovljena podsekvenca uz ograničenje maksimalnog broja ponavljanja po simbolu (d)

4. Opis korišćenih metoda

Zbog složenosti problema pronalaženja najduže ponovljene podsekvence sa ograničenim brojem ponavljanja po simbolu d, klasične metode često nisu dovoljno efikasne za veće nizove. Direktne permutacije i proveravanje svih mogućih kombinacija imaju eksponencijalnu složenost.

U ovom radu fokus je na dve heurističke metode: pohlepni (Greedy) algoritam i algoritam pretraživanja promenljivih susedstava (VNS). Kombinovanjem ovih pristupa moguće je postići balans između brzine izvršavanja i kvaliteta dobijenih podsekvenci.

4.1 Pohlepni algoritam

Greedy metoda funkcioniše po principu lokalne optimizacije: u svakom koraku bira opciju koja trenutno najbolje doprinosi povećanju dužine ponovljene podsekvence, poštujući ograničenje d i zastupljenost svih simbola iz alfabeta.

Greedy metoda funkcioniše po principu lokalne optimizacije: u svakom koraku bira opciju koja trenutno najbolje doprinosi povećanju dužine podsekvence, poštujući ograničenje d i zastupljenost svih simbola iz alfabeta.

Implementacija:

1. Broji se broj pojavljivanja svakog simbola u nizu.
2. Formiraju se podsekvence do maksimalnog broja ponavljanja d.
3. Svaka podsekvenca se ocenjuje funkcijom fitness, koja računa broj nepreklapajućih pojavljivanja.
4. Najčešći simboli se po potrebi dodaju kako bi se povećao broj ponavljanja.

Varijante:

- **greedy_basic**: jednostavnija i brža verzija, fokusirana na brzo pronalaženje prvog prihvatljivog rešenja. Pogodna za male instance.
- **greedy_advanced**: sistematičnija i šira pretraga, traži duže kombinacije i proverava sve moguće podsekvence. Daje kvalitetnije rešenje, ali sporije.

Prednosti: jednostavna implementacija, brzo izvršavanje, dobijanje približnog rešenja.

Ograničenja: može propustiti globalno optimalno rešenje jer se fokusira na lokalni maksimum.

4.2 Algoritam pretraživanja promenljivih susedstava (VNS)

VNS (Variable Neighborhood Search) algoritam zasniva se na principu sistematskog menjanja „okoline“ trenutnog rešenja kako bi se izbeglo zastojevanje u lokalnim optimumima. Ideja je da se ne zadovoljava samo lokalni optimum, već da se istražuje širi prostor rešenja kroz različite tipove promena u susedstvu.

VNS algoritam u ovom radu ima dve varijante: vns_basic i vns_advanced.

- **vns_basic** radi sa jednostavnijim susedstvima i manjim brojem iteracija. Primenjuje osnovne promene na početnom rešenju (dobijenom greedy algoritmom), kao što su dodavanje, uklanjanje ili zamena jednog simbola. Ova varijanta je brža, ali može propustiti bolje rešenje ako prostor rešenja nije dovoljno istražen.
- **vns_advanced** koristi kompleksnije susedstvo i veći broj iteracija, čime omogućava širu pretragu prostora rešenja. Primenjuje više promena po iteraciji i dodatne strategije proširenja i zamene simbola, čime povećava verovatnoću pronalaženja dužih i kvalitetnijih podsekvenci.

Glavna razlika: basic je brža i jednostavnija, pogodna za manji broj iteracija, dok `advanced` detaljnije istražuje prostor rešenja i pruža bolje rezultate na srednjim i velikim instancama.

VNS algoritam ima prednost što može pronaći kvalitetnije rešenje od pohlepnog algoritma i smanjuje rizik od zastoja u lokalnom maksimumu, ali s druge strane zahteva duže vreme izvršavanja i složenija je njegova implementacija.

4.3 Hibridna metoda

Hibridni pristup hybrid_advanced kombinuje prednosti obe metode: koristi rešenje dobijeno greedy_advanced kao početno, a zatim ga unapređuje primenom vns_advanced. Na ovaj način:

- brzo dobijamo početno kvalitetno rešenje
- istražujemo širi prostor za eventualna bolja rešenja
- pronalazimo duže i bolje raspoređene podsekvence

4.4 Poređenje metoda

Metoda	Prednosti	Nedostaci	Primena
Greedy	Brzo inicijalno rešenje, jednostavna implementacija	Može propustiti globalni optimum	Male instance ili kada je potrebno brzo rešenje
VNS	Pronalaženje kvalitetnijih podsekvenci, fleksibilno istraživanje prostora rešenja	Duže izvršavanje, složenija implementacija	Srednje i veće instance
Hibrid	Kombinuje prednosti obe metode, omogućava duže i kvalitetnije podsekvence	Veća kompleksnost i duže izvršavanje	Kada je potreban balans između kvaliteta i brzine

Tabela 4. Poređenje metoda

4.5 Funkcije ocenjivanja i njihov značaj

Za ocenjivanje kandidata u problemu najduže ponovljene podsekvence sa ograničenjem d koriste se dve ključne funkcije: **fitness** i **score**.

Fitness funkcija meri koliko puta se određena podsekvenca pojavljuje u nizu S, pri čemu se poštuju dva osnovna pravila: nijedan simbol ne sme se pojaviti više puta nego što je dozvoljeno

(ograničenje d), i svi simboli iz alfabeta Σ moraju biti zastupljeni. Bitna karakteristika fitness funkcije je što se broje **nepreklapajuća pojavljivanja**, čime se izbegava dvostruko računanje istih instanci podsekvence. Ovo omogućava algoritmima da precizno kvantifikuju kvalitet kandidata, posebno kada postoji mnogo potencijalnih duplikata u nizu.

Score funkcija predstavlja dodatni kriterijum koji kombinuje dužinu podsekvence i broj ponavljanja kako bi favorizovala duže i informativnije obrasce. Formalno, score se računa po formuli:

$$\text{score} = \text{dužina podsekvence} + 0.5 \times \text{broj ponavljanja}$$

Ova metrika daje prednost podsekvencama koje su duže, čak i ako se pojavljuju ređe, čime se balansira između količine informacija i frekvencije ponavljanja.

Primer: Za niz S="ABABABAB", alfabet Σ = {A, B}, i d = 3:

- Podsekvenca "AB" ima dužinu 2 i pojavljuje se 4 puta → score = 2 + 0.5 × 4 = 4
- Podsekvenca "ABAB" ima dužinu 4 i pojavljuje se 2 puta → score = 4 + 0.5 × 2 = 5

Ovaj primer pokazuje da iako se "AB" pojavljuje češće, "ABAB" dobija veći score zbog veće dužine, što odražava preferenciju za informativnije rešenje.

Povezanost sa algoritmima:

- **Greedy algoritmi** koriste fitness i score za rangiranje kandidata i heurističko proširenje najperspektivnijih podsekvenci.
- **VNS algoritmi** oslanjaju se pretežno na score funkciju da vode pretragu ka kvalitetnijim rešenjima, prevazilazeći lokalne optime.
- **Hibridni pristup** kombinuje oba – Greedy brzo generiše početno rešenje, dok VNS score funkciju koristi za dodatno unapređenje kroz sistematsko pretraživanje susedstva.

Ovaj dvojni pristup omogućava algoritmima da balansiraju između brzine i kvaliteta rešenja, posebno kada se radi o srednjim i velikim instancama problema.

5. Eksperimentalni rezultati

5.1 Eksperimentalno okruženje

Eksperimenti su izvršeni na računaru sa operativnim sistemom Windows 10, procesorom Intel i5, 16GB RAM-a. Implementacija algoritama je urađena u programskom jeziku Python 3.11.7, koristeći standardne biblioteke (random, os) za generisanje test instanci i merenje vremena izvršavanja.

5.2 Test instance

Za eksperimentalnu evaluaciju algoritama generisane su instance različitih veličina i tipova, kako bi se obuhvatile različite karakteristike nizova i omogućilo objektivno poređenje performansi.

Alfabet i veličine instanci:

- Koristi se DNK alfabet: A, C, G, T.
- Dužine stringova su definisane po kategorijama: male (100 karaktera), srednje (600), velike (3000) i vrlo velike (10000).
- Za svaku veličinu generisano je po 5 instanci za svaki tip niza.

Tipovi instanci:

1. **random:** nasumični nizovi bez garantovanih duplikata, koji služe za testiranje osnovne sposobnosti algoritama da prepoznaju bilo kakve ponovljene podsekvence.
2. **one_duplicate:** niz sa tačno jednom ponovljenom podsekvencom, što omogućava testiranje kako algoritmi pronalaze jednostavne duplikate.
3. **multiple_duplicates:** niz sa više nezavisnih duplikata, što proverava sposobnost algoritama da detektuju više ponovljenih podsekvenci i da pravilno upravljaju ograničenjem d.
4. **mixed:** niz sa kombinacijom duplikata različitih dužina i pozicija, čime se simulira složenija struktura realnih podataka i testira fleksibilnost algoritama.

Princip generisanja nizova:

- Svaki niz se inicijalno sastoji od svih simbola iz alfabeta, kako bi se osigurala potpuna zastupljenost.
- Duplikati se dodaju sistematski, ili bez preklapanja postojećih duplikata, ili sa kontrolisanim preklapanjem, zavisno od tipa eksperimenta.
- Dužina duplikata se bira tako da bude značajna za algoritme: minimalno 10% ukupne dužine niza ili barem 5 karaktera, a maksimalno ograničeno na 50 karaktera.
- Pozicije duplikata se biraju nasumično, ali tako da se poštuju pravila preklapanja, čime se garantuje da svaki duplikat bude jasno definisan i prepoznatljiv.

Proces generisanja fajlova:

- Nizovi se generišu programom u Python-u i čuvaju kao tekstualni fajlovi u folderu data.
- Svaki fajl je imenovan prema veličini, tipu i rednom broju instance, npr. instance_medium_3_multiple_duplicates.txt.
- Ovim pristupom dobijaju se različite i kontrolisane instance koje omogućavaju pouzdano poređenje performansi algoritama greedy i VNS na nizovima različite dužine i složenosti.

Napomene:

- Male instance služe za ilustraciju i verifikaciju ispravnosti algoritama, dok srednje i velike instance omogućavaju testiranje skalabilnosti i efikasnosti heurističkih metoda.
- Randomizacija pozicija i dužina duplikata osigurava da se algoritmi testiraju na različitim konfiguracijama i scenarijima, što čini eksperiment realističnijim.

5.3 Rezultati i analiza

Eksperimenti su izvršeni nad instancama različitih veličina, a rezultati su izraženi prosečnim score vrednostima pronađenih podsekvenci. Rezultati jasno pokazuju razlike u kvalitetu rešenja koje generišu pojedine varijante algoritama.

Male instance (100 karaktera)

Algoritam	Prosečan score
VNS Advanced	20.275
Hybrid Advanced	19.875
Greedy Advanced	18.850
VNS Basic	16.725
Greedy Basic	0.000

Tabela 5. Rezultati različitih algoritama na malim instancama

- VNS Advanced daje najbolje rezultate čak i na malim instancama.
- Hybrid Advanced je skoro jednako dobar, pokazujući efikasnost kombinacije pohlepnog i VNS pristupa.
- Greedy Advanced daje solidne rezultate brže, dok Greedy Basic ne pronalazi korisne podsekvence na malim instancama.

Srednje instance (600 karaktera)

Algoritam	Prosečan score
VNS Advanced	116.200
Hybrid Advanced	116.000
Greedy Advanced	112.000
VNS Basic	56.975
Greedy Basic	0.000

Tabela 6. Rezultati različitih algoritama na srednjim instancama

- Razlika između VNS Advanced i Hybrid Advanced je minimalna, što pokazuje da hibridni pristup uspešno koristi početno pohlepno rešenje.
- Greedy Advanced je nešto slabiji, ali i dalje značajno bolji od basic varijanti.
- Basic varijante pokazuju da jednostavne heuristike nisu dovoljne za srednje instance.

Velike instance (3000 karaktera)

Algoritam	Prosečan score
Hybrid Advanced	552.200
VNS Advanced	552.000
Greedy Advanced	548.000
VNS Basic	74.875
Greedy Basic	0.000

Tabela 7. Rezultati različitih algoritama na velikim instancama

- Za velike instance, hibridni pristup daje najbolji rezultat, mada su VNS Advanced i Hybrid Advanced skoro identični.
- Greedy Advanced još uvek daje solidnu aproksimaciju, dok basic varijante gotovo da ne pronalaze korisne podsekvence.
- Ovi rezultati potvrđuju da je kombinacija Greedy + VNS optimalna za skalabilnost i kvalitet rešenja.

Vrlo velike instance (10.000 karaktera)

Algoritam	Prosečan score
Hybrid Advanced	1819.80
VNS Advanced	1818.80
Greedy Advanced	1815.20
VNS Basic	170.85
Greedy Basic	0.000

Tabela 8. Rezultati različitih algoritama na vrlo velikim instancama

- Trend iz prethodnih instanci se nastavlja: hibridni pristup postiže najduže i kvalitetnije podsekvence, dok Greedy Advanced pruža brzu aproksimaciju.
- Basic varijante se ne skaliraju dobro i značajno zaostaju u kvalitetu rešenja.

5.4 Zaključci iz rezultata

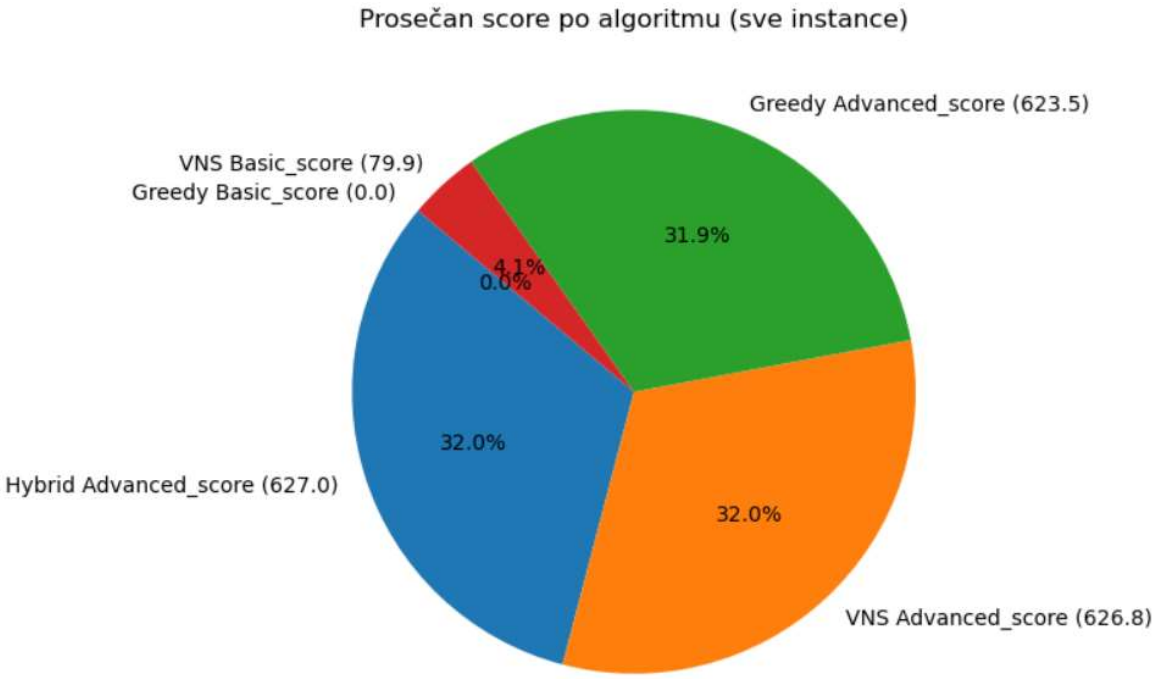
1. **Hibridni pristup (Hybrid Advanced)** dosledno daje najbolje rezultate na srednjim, velikim i vrlo velikim instancama, kombinujući brzinu pohlepnog algoritma sa širinom pretrage VNS-a.
2. **VNS Advanced** blizu je hibridnog rešenja, ali bez početnog Greedy rešenja može zahtevati više vremena za konvergenciju.
3. **Greedy Advanced** nudi kompromis između brzine i kvaliteta, naročito korisno za brza inicijalna rešenja ili kada je potrebno testirati veliki broj instanci.
4. **Basic varijante** nisu pogodne za srednje i velike instance jer ne uspevaju da pronađu značajne ponovljene podsekvence.

Ovi rezultati potvrđuju hipotezu da kombinacija heurističkih metoda (pohlepni + VNS) omogućava balans između kvaliteta rešenja i vremena izvršavanja, dok jednostavne ili osnovne heuristike nisu dovoljne za kompleksnije nizove.

Tabela ukupnih rešenja (za sve algoritme)

Algoritam	Prosečan score
Hybrid Advanced	626.96875
VNS Advanced	626.81875
Greedy Advanced	623.51250
VNS Basic	79.85625
Greedy Basic	0.000

Tabela 9. Poređenje rezultata različitih algoritama na svim testiranim instancama



Slika 1. Prosečan rezultat po algoritmu za sve instance

6. Zaključak

U ovom radu analiziran je problem pronalaženja najduže ponovljene podsekvence sa ograničenim brojem ponavljanja po simbolu. Implementirane su i upoređene heurističke metode: pohlepni (Greedy) algoritam i algoritam pretraživanja promenljivih susedstava (VNS), kao i hibridni pristup koji kombinuje prednosti obe metode. Greedy algoritmi brzo pronalaze približna rešenja, ali se fokusiraju na lokalne maksimume, dok VNS omogućava kvalitetnije i duže podsekvence uz duže vreme izvršavanja. Hibridni pristup pokazao se naročito efikasnim za srednje i velike instance, balansirajući brzinu i kvalitet rešenja. Takođe, složenije funkcije score i fitness verovatno bi dovele do još boljih rešenja.

Kao pravci daljeg istraživanja, moguće je optimizovati hibridne metode parametarskim podešavanjem i adaptivnim heuristikama, primeniti genetske i druge metaheuristike, testirati metode na realnim bioinformatičkim podacima, kao i istražiti skalabilnost i paralelizaciju algoritama za velike nizove.

7. Literatura

1. Mladenović, N., & Hansen, P. (1997). Variable Neighborhood Search. *Computers & Operations Research*
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.)
3. ArXiv: 2112.05725 (2022). Beyond the Longest Letter-duplicated Subsequence Problem
4. ArXiv: 2304.06862 (2023). The Longest Subsequence-Repeated Subsequence Problem
5. LearnersBucket. (n.d.). *Longest Repeated Subsequence Algorithm Example*. Retrieved from https://learnersbucket.com/examples/algorithms/longest-repeated-subsequence/
6. Longest Common Subsequence with Gap Constraints (LCS-FIG) algorithm for DNA sequence analysis