Univerzitet u Banjoj Luci
Prirodno-matematički fakultet
Studijski program Matematika i informatika

Pretraga bima sa primjenama

Diplomski rad

Student
Nikola Balaban

Mentor
doc. dr Marko Đukanović

Banja Luka, 2025.

Sadržaj

Glava 1

Metod pretrage bima

U ovoj sekciji, prvo ćemo predstaviti osnovnu metodologiju koja se provlači kroz ovaj rad. Dakle, prvo će biti predstavljen metod pretrge bima sa bazičnim pseudokodom te objašnjene glavne komponente ovog algoritma, a potom i primjene ovog metoda u rješavanju realnih problema opisanih u naučnoj literaturi.

1.1 Graf stanja

U kombinatornoj optimizaciji mnogi problemi mogu se predstaviti kroz prizmu *pretrage grafa stanja*. Graf stanja definiše čvorove (stanja) i grane (tranzicije) koje opisuju sve moguće akcije ili odluke u prostoru pretrage [19]. U prevodu, graf stanja predstavlja matematički model problema pretraživanja, gdje se svaka moguća konfiguracija sistema opisuje kao čvor (stanje), a prelazi iz jednog stanja u drugo opisuju se kao grane (operatori ili akcije). Najčešće su stanja definisana preko parcijalnih rješenja koja indukuju potproblem koji treba dalje da se riješi u svrhu ekstenzije postojećeg parcijalnog rješenja prema ciljnim stanjima. Cilj je pronaći put od početnog stanja do ciljnog stanja (ili skupa ciljeva) koji optimizuje zadatu funkciju cilja. Taj put odgovara pronalasku skupa optimalnih odluka/akcija u prostoru pretrage.

Klasične metode pretrage obuhvataju *pretragu u širinu*, *pretragu u dubinu*, *uniform-cost pretragu* i heurističke varijante poput A* algoritma [13]. Ove metode garantuju optimalnost ili potpunost pod određenim uslovima, ali u praksi često pate od eksponencijalnog rasta prostora stanja sa porastom dimenzije instance problema. Kada su sekvence odluka ili kombinacije veoma dugačke, čak i heurističke metode mogu postati neizvodljive zbog ogromnog broja stanja.

1.2 Ideja pretrage bima

Metod pretrage bima (eng. *beam search*) uvodi kompromis između potpunosti i efikasnosti pretraživanja. Osnovna ideja je da se prostor stanja pretražuje sloj po sloj (slično pretrazi u širinu), ali se u svakom sloju zadržava samo ograničen broj najboljih kandidata. Parametar *širina bima* (β) kontroliše broj čvorova koji se čuvaju po nivou.

Ako je $\beta = 1$, algoritam se degeneriše u *pohlepnu pretragu* koja uvek bira trenutno najbolje proširenje. Ako je $\beta = \infty$, algoritam postaje puna pretraga u širinu (bez odsijecanja). Za praktične vrijednosti β, pretraga bima pruža balans između kvaliteta rješenja i potrebnih resursa (vrijeme i memorija).

Formalno, pretraga bima radi na sljedeći način:

- kreće od početnog stanja i inicijalizuje listu kandidata,
- u svakom koraku generiše sva proširenja kandidata/tranzicije sa potomcima,
- rangira ih prema heurističkoj funkciji ili evaluaciji,
- zadržava samo najboljih β proširenja i odbacuje ostale.

Pseudokod pretrage bima dat je Algoritmom 1.

Algoritam 1 Pretraga bima

Require: Početno stanje s_0, širina bima β, heuristička funkcija h

$Kandidati \leftarrow \{s_0\}$
while postoji nedovršeno stanje u $Kandidati$ **do**
    $Prosirenja \leftarrow \emptyset$
    for svako $s \in Kandidati$ **do**
        $Prosirenja \leftarrow Prosirenja \cup \{$sva dozvoljena proširenja stanja $s\}$
    end for
    Rangiraj elemente u $Prosirenja$ prema evaluacionoj funkciji $h(\cdot)$
    $Kandidati \leftarrow$ prvih β elemenata iz $Prosirenja$
end while
return najbolje pronađeno rješenje u $Kandidati$

Evaluaciona funkcija $h(\cdot)$ zavisi od problema. Može uključivati heurističku procjenu troška do cilja, aproksimaciju vrijednosti parcijalnog rješenja ili kvalitet parcijalne sekvence.

Da bi se ovaj metod primijenio na određen problem, potrebno je da se definišu sljedeće stavke:

- Definicija grafa stanja sa čvorovima i granama (tranzicijama) te njihovim težinama. Čvorovi/stanja mogu biti početna, završna ili unutrašnja.

- Funkcija proširenja čvora/stanja.
- Funkcija evaluacije h koja procjenjuje dobrotu svakog od stanja.

1.3 Primjene pretrage bima

Pretraga bima se koristi u velikom broju oblasti gdje prostor rješenja brzo eksplodira:

- *Obrada prirodnog jezika (NLP)*: najpoznatija primjena je u generativnim modelima jezika (npr. u mašinskom prevođenju, automatskom sažimanju teksta, dijaloškim sistemima). Pretraga bima ovdje pronalazi najvjerovatniju sekvencu riječi prema jezičkom modelu [26].
- *Prepoznavanje govora*: koristi se za odabir najvjerovatnijeg niza fonema ili riječi na osnovu akustičnog modela i jezičkog modela [30].
- *Bioinformatika*: primjenjuje se u poravnavanju sekvenci, predikciji strukture i problemima gdje je prostor mogućih kombinacija ogroman [12].
- *Kombinatorna optimizacija*: pretraga bima se koristi kao aproksimativna ili heuristička metoda u problemima poput planiranja, raspoređivanja (eng. *scheduling*), optimizacije putanja, pa čak i u problemskim klasama poput problema najduže zajedničke sekvence (eng. *Longest Common Subsequence*) ili problema trgovačkog putnika (eng. *Travelling salesman problem* (TSP)) [32].

U mnogim sekvencijalnim modelima, posebno u *neuronskom mašinskom prevođenju* (eng. *Neural Machine Translation*, NMT), pretraga bima i dalje predstavlja standardnu tehniku za generisanje izlaznih rečenica koje balansiraju između vjerovatnoće i raznovrsnosti. Iako su novije tehnike (poput *sampling-based* metoda ili *diverse beam search* metoda) uvedene da adresiraju nedostatke, klasična pretraga bima ostaje osnovna komponenta u najpoznatijim sistemima.

Slično, u prepoznavanju govora (ASR), pretraga bima sa prikladnim heuristikama još uvijek daje rezultate na nivou najsavremenijih sistema. U kombinatornoj optimizaciji, pretraga bima često čini jezgro *hibridnih metaheuristika*, gdje se kombinuje sa lokalnim pretragama, genetskim algoritmima ili tabu pretragom, i pokazuje konkurentne performanse u realnim aplikacijama [13, 16].

1.4 Doprinosi ovog rada i organizacija

Postoji nekoliko konkretnih doprinosa ovog rada.

- Demonstrirana je primjena pretrage bima na rješavanju jednog realnog sekvencijalnog problema iz domena bioinformatike koji do sada nije rješavan sa praktičnog aspekta. Konkretno, riječ je o problemu najdužeg zajedničkog podiza sa varijabilnim razmacima koji ima primjene u nalasku strukturalnih sličnosti između grupe bioloških molekula. Na taj način je pokazana dodatna robusnost metoda pretrage bima.

- Uvodi se model cjelobrojnog programiranja problema najdužeg zajedničkog podiza sa varijabilnim razmacima dvije niske.

- Dat je doprinos u oblasti bioinformatike, a konkretno podoblasti stringologije i sekvencijalnih problema analizirajući specijalni slučaj problema sa varijabilnim razmacima sa dvije ulazne niske, kao i njegove generalizovane varijante.

Ovaj rad je organizovan na sljedeći način. Sekcija 2 predstavlja osnove oblasti računarske biologije sa naglaskom na jedan od bazičnih problema u oblasti, a to je problem najdužeg zajedničkog podniza. Sekcija 3 uvodi generalizaciju problema zajedničkog podniza, koji razmatra varijabilne razmake. Taj konkretan problem će i biti korišten u demonstraciji primjene pretrage bima. Dalje, Sekcija 4 uvodi generalizovanu varijantu problema uvedenog u prethodnoj sekciji. Potom se definiše graf stanja za razmatrani problem zajedno sa pretragom bima. Eksperimentalna analiza je opisana u Sekciji 5 dok su zaključci i pravci daljeg razvoja istrazivanja dati u Sekciji 6.

Glava 2

O računarskoj biologiji

Računarska biologija je interdisciplinarna oblast koja povezuje biologiju, računarstvo i matematiku, sa ciljem analize i razumijevanja bioloških podataka. Jedan od ključnih aspekata bioinformatike je analiza bioloških sekvenci – DNK, RNK i proteinskih lanaca. Poređenje i analiza sličnosti između ovih sekvenci igra važnu ulogu u identifikaciji gena, predikciji struktura i funkcija proteina, filogenetskim analizama, itd. [37, 42, 23].

Jedan od osnovnih problema u poređenju sekvenci sa stanovišta sekvencijalne sličnosti je tzv. *problem najduže zajedničke podsekvence* (LCS) [15, 4, 6, 33, 17]. Osnovna definicija ovog problema maksimizacije uključuje dvije sekvence u ulazu: $s_1 = a_1a_2...a_n$ i $s_2 = b_1b_2...b_m$. LCS predstavlja najdužu sekvencu koja je podsekvenca obje niske s_1 i s_2, ali ne nužno sa uzastopnim karakterima (tj. ne mora biti neprekidni dio ulazne sekvence). Radi jasnoće, u nastavku je dat konkretan primjer (instanca).

Primjer. Neka su u ulazu date sekvence $s_1 =$ `ACCGGTCGAGTGCGCGGAAGCCGGCCGAA` i $s_2 =$ `GTCGTTCGGAATGCCGTTGCTCTGTAAA`. Jedna zajednička podsekvenca može biti sljedeća: `GTCGTCGGAAGCCGGCCGAA`. Direktnom provjerom može se vidjeti da je to i najduža podsekvenca koja je zajednička za obje ulazne niske.

U literaturi postoje mnoge praktično zasnovane varijante bazičnog LCS problema. Neke od tih varijanti su upoštenja baznog problema, npr. povećavanjem dimenzionalnosti problema na način da se umjesto dvije sekvence u ulazu razmara proizvoljan broj sekvenci. To ima opravdanje u praksi jer je često neophodno uporediti više molekula u isto vrijeme da bi se utvrdilo pripadaju li istoj grupi molekula sa očekivano sličnim ponašanjem. Ovakva generalizacija predstavlja NP-težak problem kada se pretpostavi da je riječ o proizvoljno velikom skupu sekvenci [12, 31].

Među varijantama LCS problema koje nisu samo zasnovane na povećanju ulaza

osnovnih ulaznih niski su one koje dodaju dodatna ograničenja u definiciju LCSa. Neke od tih problema uključuju:

- **Ograničavajući LCS problem** (eng. *Constrained LCS* (CLCS)) – traži se LCS koji mora uključiti zadatu podsekvencu koja je unaprijed data [38, 5, 11].
- **Isključujući LCS problem** (eng. *Exclusion LCS* (ELCS)) – LCS koji ne smije sadržati određenu podsekvencu, opet unaprijed data [2, 40, 43].
- **Neponavljajući LCS problem** (eng. *Repetition-free LCS* (RFLCS)) – LCS sa dodatnim ograničenjem da se niti jedan od karaktera ne pojavljuje više od jednom u rezultujućoj niski [27, 14, 7].
- **Varijabilni LCS sa razmacima** (eng. *Variable Gap LCS* (VGLCS)) – ograničava rastojanje između karaktera u LCS-u, simulirajući biološke udaljenosti [35, 34].
- **Anotirani LCS problem** (eng. *Arc-annotated LCS* (AALCS)) – traži se LCS ali na anotiranim sekvencama koji bliže modeluju stvarne DNA i RNA sekvence [22, 21].

Svaka od ovih varijanti problema ima svoju praktičnu primjenu, predstavljajući profinjenije procjene mjera sličnosti nad molekularnim strukturama.

2.1 Pregled algoritama za rješavanje bazičnog LCS problema

U ovoj sekciji navodimo neke osnovne algoritme raznih tipova u rješavanju baznog LCS problema sa dvije ulazne niske iz literature. Kako je ovaj problem polonomijalno rješiv, razmatraćemo samo egzaktne pristupe [6]. Što se tiče egzaktnih algoritama, u prethodnih 50-ak godina razvijeno je više njih, a među najpoznatiji su:

- **Dinamičko programiranje** (DP) – vremenska složenost $O(|s_1| \cdot |s_2|)$ za dvije sekvence s_1 i s_2.
- **Hiršbergov algoritam** [15] – *divide-and-conquer* princip.
- **Hunt-Szymanski** pristup [17, 3] – umjesto izračunavanja svih DP stanja, prati samo podnizove kroz liste podudaranja i ažuriranja, drastično smanjujući posao kada su podudaranja rijetka.

- **Apostolico** pristup [4] – koristi preprocesiranje sa dominantnim tačkama podudaranja između ulaznih sekvenci.
- **Wagner & Fischer** pristup [39] — baziran na ideji rješavanja "edit-distance" problema.
- **A*** pristup [13] – baziran na paradigmi najbolji prvi (eng. best-first).

Kao najpoznatiji pristup, u detalje ćemo prikazati metod dinamičkog programiranja nad dvije sekvence. Definišemo DP tabelu $c \in \mathbb{N}_0^{(n+1)\times(m+1)}$, gdje je $|s_1| = n, |s_2| = m$:

$$c[i][j] = \text{dužina } LCS\big(s_1[1..i], s_2[1..j]\big), \qquad 0 \le i \le n,\ 0 \le j \le m,$$

gdje $s[i, j]$ predstavlja neprekidan podstring niza s koji počinje od karaktera na poziciji i a završava sa karakterom na poziciji j.

Osnovni slučajevi su

$$c[i][0] = 0 \ \ (\forall i), \qquad c[0][j] = 0 \ \ (\forall j),$$

jer je LCS sa praznim prefiksom dužine 0.

Optimalna podstruktura je data narednom lemom.

Lema 1 (Optimalna podstruktura)**.** *Važi:*

$$c[i][j] = \begin{cases} c[i-1][j-1] + 1, & ako\ s_1[i] = s_2[j], \\ \max\{c[i-1][j],\, c[i][j-1]\}, & ako\ s_1[i] \neq s_2[j]. \end{cases}$$

Dokaz. Ako je $s_1[i] = s_2[j]$, svaki LCS za prefikse dužine i ($s_1[1..i]$) i prefikse dužine j ($s_2[1..j]$) može da završi tim znakom, pa se problem svodi na $s_1[1..i-1]$ i $s_2[1..j-1]$ uz uvećanje dužine za 1. Ako $s_1[i] \neq s_2[j]$, bilo koji zajednički podniz optimalne dužine mora izostaviti bar jedan od elemenata $s_1[i]$ ili $s_2[j]$, pa je optimalna dužina maksimum između izostavljanja $s_1[i]$ i izostavljanja $s_2[j]$.

2.2 Pregled algoritama za generalizovani LCS problem

Generalizovani LCS problem je pseudo NP težak. Kada je broj ulaznih nisku fiksiran, problem je polinomijalno rješiv pomoću dinamičkog programiranja i vremenu $O(n^k)$ gdje je n dužina najduže sekvence u ulazu, a k broj sekvenci. Pristup se konstruiše uopštavajući DP pristup za dvije niske. Pokazano je da je ta granica

uska (eng. *tight*), i da ako se problem LCS-a sa k niski može riješiti u $O(n^{k-\epsilon})$ onda vrijedi $P = NP$, a pretpostavlja se da to nije tačno. Zbog toga, u literaturi je razvijen veliki broj heurističkih, metaheurističkih i aproksimativnih tehnika.

U vezi heurističkih pristupa, bazni pohlepni algoritam za Generalizovani LCS problem dizajniran je u radu [20] i ne garantuje optimalno rješenje. Dalje, za rješavanje ovog problema u generalnom obliku sa proizvoljnim brojem ulaznih niski, napredne metaheuristike su se pokazale kao neprikosnovene. Među njima izdvajamo pristupe kao što su pristup mravljim kolonijama (ACO) [10], zatim pretraga bima koja je dizajnirana u [9], te hibridni metod pretrage bima i ACO metoda predstavljen u radu [8]. Napredna pretraga bima sa probabilističkom funkcijom za navođenje pretrage je predstavljena u [28]. Do sada najefikasnija heuristika na slučajnim i blisko-slučajnim sekvencama je bazirana na aproksimaciji očekivane dužine najduže zajedničke podsekvence bilo kojeg potproblema [12]. Kasnije je ideja dinamičke pretrage bima dodatno poboljšala do sada najbolje rezultate za LCS problem [31], koji se pokazao naročito efikasnim na sekvencama koje ne potpadaju pod uniformnu distribuciju, tzv. POLY skup instanci. I egzaktni pristupi su dobro izučavani te njihova efikasnost dodatno poboljšana. Npr. u radu [41] dizajniran je metoda na dominantnim tačkama i naprednim k-indeksnim stablima. U [36] predstavljen je pristup koji koristi tehniku linearnog sortiranja za smanjenje skupa relevantnih dominantnih tačaka. U skorije vrijeme, razvijeni su egzaktni hibridi A* tehnike sa pretragom bima i idejama "anytime" pretrage kolona [13]. Potonji pristup je zvanično najbolji metod što se tiče egzaktnih pristupa.

Postoji još jedna klasa algoritma razmatramna u rješavanju Generalizovanog LCS problema, tzv. aproksimativni algoritmi. Ovi algoritmi posjeduju garanciju da je njihovo rješenje unutar određenog faktora kvaliteta od optimalnog. Više o ovim pristupima može se naći u radu [13].

U osnovi, glavni metodi za rješavanje LCS problema i velikog broja njegovih varijanti su pretaga bima kada je u pitanju aproksimativni, robusni rješavač, dok je to A* pristup kada je u pitanju kompletan rješavač.

2.3 Primjene LCS i njegovih varijanti

U nastavku nabrajamo nekoliko slučajeva realnog korišćenja LCS problema i njegovih varijanti.

- Upoređivanje DNK/RNK sekvenci radi identifikacije gena i mutacija.
- Filogenetske analize i konstrukcija evolutivnih stabala.

- Poređenje proteinskih lanaca i detekcija funkcionalnih domena.
- Verifikacija softverskog koda i detekcija plagijarizma.
- Kompresija bioloških podataka.

Neki od radova gdje se može pogledati više o ovim primjenama su dati sa [31, 29, 24].

2.4 Notacija

Prije nego što krenemo sa definisanjem problema kojeg namjeravamo rješavati, uvedimo notaciju koju ćemo koristiti u nastavku ovog rada.

Sa $|s|$ označimo dužinu sekvence s. Sa $s[i]$, označimo karakter na i-toj poziciji sekvence s gdje je pozicija prvog karaktera označena indeksom $i = 1$. Podniz sekvence s koji počinje sa indeksom i a završava sa indeksom j je označen sa $s[i : j]$; ako je $i = j$ riječ je o karakteru $s[i]$, a ako je $j < i$ onda je $s = \varepsilon$ (ε označava prazan karakter). Broj pojavljivanja karaktera $a \in \Sigma$ u sekvenci s je označen sa $|s|_a$, dok je broj pojavljivanja podskupa karaktera $\Sigma' \subseteq \Sigma$ u sekvenci s označen sa $|s|_{\Sigma'} = \sum_{\sigma \in \Sigma'} |s|_\sigma$. Sa s^{rev} označimo reverznu sekvencu sekvence s. Sa $S = \{s_1, \ldots, s_m\}$ označimo skup ulaznih sekvenci, a dužinu najduže sekvence u S označavamo sa n, ukoliko to nije drugačije naznačeno u tekstu.

Glava 3

LCS problem sa varijabilnim razmacima

Bazni problem zajedničke podsekvence sa varijabilnim razmacima (eng. *the variable gapped longest common subsequence* (VGLCS)) je definisan ulazom koji se sastoji od dvije sekvence s_1 i s_2, nad azbukom Σ i dvije funkcije razmaka (eng. *gap constraints*):

$$G_{s_1}(i) : \{1, \ldots, |s_1|\} \to \mathbb{N}, \quad G_{s_2}(j) : \{1, \ldots, |s_2|\} \to \mathbb{N}$$

Spisak parova indeksa poravnanja dat sa sljedećim nizom

$$(i_1, j_1), (i_2, j_2), \ldots, (i_p, j_p)$$

predstavlja **podsekvencu koja zadovoljava ograničenja razmaka** akko su ispunjeni sljedeći uslovi:

1. *Poklapanje karaktera:*

$$s_1[i_x] = s_2[j_x], \quad \text{za } 1 \leq x \leq p$$

2. *Rastući redoslijed indeksa:*

$$1 \leq i_1 < i_2 < \cdots < i_p \leq |s_1| \quad \text{i} \quad 1 \leq j_1 < j_2 < \cdots < j_p \leq |s_2|$$

3. *Ograničenja razmaka:*

$$i_x - i_{x-1} \leq G_{s_1}(i_x) + 1 \quad \text{i} \quad j_x - j_{x-1} \leq G_{s_2}(j_x) + 1, \quad \text{za } 2 \leq x \leq p$$

Cilj problema je pronaći najdužu moguću podsekvencu koja zadovoljava sva tri uslova, odnosno *maksimizovati* vrijednost varijable p. Da bi problem bio jasniji,

potkrijepimo strogu definiciju sa jednim konkretnim primjerom.

Primjer. Pretpostavimo da su dati sljedeći ulazni podaci (instanca problema):

- Sekvenca $s_1 = \texttt{ABCDEFG}$
- Sekvenca $s_2 = \texttt{AXBDYFG}$
- Funkcije razmaka:

$$G_{s_1}(i) = 2 \quad \text{za sve } i = 1, \ldots, 8, \qquad G_{s_2}(j) = 1 \quad \text{za sve } j = 1, \ldots, 7.$$

Jedna moguća podsekvenca koja zadovoljava uslove date razmacima je sastavljena od sljedećih parova (karaktera):

$$(s_1[1], s_2[1]) = (\texttt{A}, \texttt{A}), (s_1[2], s_2[3]) = (\texttt{B}, \texttt{B}), (s_1[4], s_2[4]) = (\texttt{D}, \texttt{D}),$$
$$(s_1[6], s_2[6]) = (\texttt{F}, \texttt{F}), (s_1[7], s_2[7]) = (\texttt{G}, \texttt{G})$$

Indeksi rastu za obe koordinate, karakteri se poklapaju, a razmaci između pozicija su u skladu sa dozvoljenim vrijednostima, tj. vrijedi:

$$\begin{aligned} 2 - 1 = 1 &\leq G_{s_1}(2) + 1 = 3 \\ 4 - 2 = 2 &\leq 3 \\ 6 - 4 = 2 &\leq 3 \\ 7 - 6 = 1 &\leq 3 \end{aligned} \qquad \text{dok isto važi i za } s_2.$$

Dakle, ovo je validna promjenljiva podsekvenca koja poštuje ograničenja data funkcijama razmaka. Može se provjeriti da je uistinu $p = 5$.

Zašto varijabilni razmaci? Ograničenja varijabilnog tipa pojavljuju se u mnogim problemima iz stvarnog svijeta kao što su

- *Računarska biologija* – podudaranje proteina ili DNK motiva sa razmakom specifičnim za domene.
- *Iskopavanje teksta* (eng. *data mining*) – podudaranje ključnih riječi koje se moraju pojaviti u fleksibilnim okvirima.
- *Analiza vremenskih serija* (eng. *time series analysis*) – događaji koji se moraju dogoditi unutar određenih vremenskih kašnjenja.

Specijalni slučajevi ovog problema uključuju:

- Ako je $G_s(t) = +\infty$, za sve t, tada ne postoje ograničenja razmaka koja treba da budu zadovoljena – problem se svodi na klasični LCS problem [6];
- Ako je $G_s(t) = d$ (konstanta) za sve t, problem se svodi na LCS sa *fiksnim* razmakom. Ovaj konkretan problem je predložen u [18].

3.1 Dinamičko programiranje za VGCS problem (dvije niske)

Definišimo strukture podataka (tabele) koje su neophodne za računanje VGLCS na prefiksima ulaznih sekvenci s_1 i s_2:

- $F[i, j]$: dužina VGLCS za prefiksne nizove $s_1[1, i]$ i $s_2[1, j]$ pod uslovom da se posljednji zajednički karakter koji se uključuje u rješenje odgovara pozicijama poklapanja $s_1[i]$ i $s_2[j]$.
- $V[i, j]$: dužina VGLCS za prefiksne nizove $s_1[1, i]$ i $s_2[1, j]$, bez ograničenja da se poklapanje mora završiti u (i, j).

Označimo sa $I_G(i, j)$ skup svih parova prethodnih pozicija koji su u dozvoljenim granicama razmaka:

$$I_G(i,j) = \left\{ (i',j') \;\middle|\; \begin{array}{l} i' < i,\ j' < j, \\ i - i' \leq G_{s_1}[i] + 1, \\ j - j' \leq G_{s_2}[j] + 1 \end{array} \right\}$$

Skup prethodnih validnih poklapanja definisan je kao:

$$M_G(i,j) = \{(i',j') \in I_G(i,j) \mid s_1[i'] = s_2[j']\}$$

Bazični slučajevi.

$$F[0,j] = 0 = F[i,0], \quad V[0,j] = 0 = V[i,0], \quad \text{za sve } i, j$$

Rekurzivne relacije. Za tabelu F:

$$F[i,j] = \begin{cases} 1 + \max\limits_{(i',j') \in M_G(i,j)} F[i',j'], & \text{ako } s_1[i] = s_2[j] \\ 0, & \text{inače} \end{cases}$$

Za tabelu V:

$$V[i,j] = \begin{cases} \max\{F[i,j],\ V[i-1,j],\ V[i,j-1]\}, & \text{ako } s_1[i] = s_2[j] \\ \max\{V[i-1,j],\ V[i,j-1]\}, & \text{inače} \end{cases}$$

Tabela F se koristi za praćenje "lokalno aktivnih" poklapanja sa zadovoljenim razmacima, dok tabela V održava globalno optimalnu dužinu VGCS na nivou prefiksa.

Konačno rješenje problema nalazi se na poziciji $V[|s_1|, |s_2|]$, dok se konkretno poklapanje može rekonstruisati primjenom algoritma za unazadnu pretragu (eng. *backtracking*).

Pseudokod ovog pristupa dat je u Algoritmu 2.

Algoritam 2 Računanje VGLCS koristeći V i F strukure ([35])

Ulaz: Sekvence $s_1[1..|s_1|]$, $s_2[1..|s_2|]$, funkcije razmaka $G_{s_1}[1..|s_1|]$, $G_{s_2}[1..|s_2|]$
Izlaz: Dužina LGCS za dvije sekvence s_1 i s_2
Inicijalizacija $V[0..|s_1|][0..|s_2|] \leftarrow 0$, $F[0..|s_1|][0..|s_2|] \leftarrow 0$
for $i = 1$ to $|s_1|$ **do**
    for $j = 1$ to $|s_2|$ **do**
        if $s_1[i] = s_2[j]$ **then**
            $F[i][j] \leftarrow 0$
            for $i' = i - G_{s_1}[i] - 1$ to $i - 1$ **do**
                for $j' = j - G_{s_2}[j] - 1$ to $j - 1$ **do**
                    if $s_1[i'] = s_2[j']$ **then**
                        $F[i][j] \leftarrow \max(F[i][j], F[i'][j'])$
                    end if
                end for
            end for
            $F[i][j] \leftarrow F[i][j] + 1$ ▷ dodatno uparivanje
            $V[i][j] \leftarrow \max(F[i][j], V[i-1][j], V[i][j-1])$
        else
            $F[i][j] \leftarrow 0$
            $V[i][j] \leftarrow \max(V[i-1][j], V[i][j-1])$
        end if
    end for
end for
return $V[|s_1|][s_2]$

3.2 Napredni egzaktni metodi za $m = 2$

Prethodni pristup dinamičkim programiranjem zahtjeva $O(|s_1|^2 \cdot |s_2|^2)$ vremena za izvršavanje. Korišćenjem prethodno izračunatih maksimuma u određenim pravougaonim oblastima strukture F, u [35] uz pomoć pametnih struktura pokazano je da se kompleksnost može smanjiti. Za brzo računanje maksimuma u pravougaonim oblastima

matrice F, definišemo dodatne pomoćne matrice:

- Col$[i][j]$: maksimum u vertikalnom segmentu kolone j, tj.

$$\text{Col}[i][j] = \max_{k \in [i - G_{s_1}(i) - 1, i-1]} F[k][j]$$

- All$[i][j]$: maksimum u pravougaonom regionu dimenzije $(G_{s_1}(i)+1) \times (G_{s_2}(j)+1)$ definisan odgovarajućim razmacima pozicija i i j, tj.

$$\text{All}[i][j] = \max_{\substack{k \in [i - G_{s_1}(i) - 1, i-1] \\ l \in [j - G_{s_2}(j) - 1, j-1]}} F[k][l]$$

koji se na efikasniji način računa koristeći vrijednosti iz matrice `Col`.

Uz ove optimizovane podatke, rekurzivne relacije iz prethodne sekcije se transformišu na sljedeće slučajeve:

- Ako je $s_1[i] = s_2[j]$, tada:

$$F[i][j] = 1 + \text{All}[i][j]$$

$$V[i][j] = \max\left(F[i][j],\ V[i-1][j],\ V[i][j-1]\right)$$

- Inače:

$$F[i][j] = 0$$

$$V[i][j] = \max\left(V[i-1][j],\ V[i][j-1]\right)$$

Za sve nevalidne indekse $i = 0$ ili $j = 0$, postavljamo:

$$F[i][j] = V[i][j] = \text{Col}[i][j] = \text{All}[i][j] = 0$$

Prethodno opisani kod je predstavljen Algoritmom 3.

Ovime se značajno poboljšava efikasnost algoritma, jer se za svaku poziciju u matrici koristi prethodno akumulirana informacija o maksimumima iz lokalno dozvoljenih oblasti. Održavanje maksimuma u pokretnim prozorima može se dodatno ubrzati korišćenjem monotonih redova (*deque* strukture) ili union-find strukture podataka, što omogućava implementaciju algoritma na efikasniji način.

U [35], autori su koristili pristup baziran na strukturi **union-find** koja pronalazi za svaku poziciju (i, j) maksimalan element u podmatrici (pravougaonoom oblasti) $F[i - 1 - G_{s_1}[i], i - 1][j - 1 - G_{s_2}[j], j - 1]$ koje se inkrementalno popunjavaju iterišući kroz rastuće indekse (pozicije) i i j. Posmatrajući opisan pristup, naišli smo

Algoritam 3 Optimizovani DP za VGLCS sa dodatnim matricama `Col` i `All`

Input: Sekvence $s_1[1..|s_1|]$, $s_1[1..|s_1|]$, gap funkcije $G_{s_1}[1..n]$, $G_{s_2}[1..|s_2|]$
Output: Dužina VGLCS između s_1 and s_2
Inicijalizacija $V[0..|s_1|][0..|s_2|] \leftarrow 0$, $F[0..|s_1|][0..|s_2|] \leftarrow 0$
Inicijalizacija $Col[0..|s_1|][0..|s_2|] \leftarrow 0$, $All[0..|s_1|][0..|s_2|] \leftarrow 0$
for $i = 1$ to $|s_1|$ **do**
    for $j = 1$ to $|s_2|$ **do**
        $start\_row \leftarrow \max(1, i - G_{s_1}[i] - 1)$
        $start\_col \leftarrow \max(1, j - G_{s_2}[j] - 1)$
        $Col[i][j] \leftarrow \max_{i' \in [start\_row, i-1]}(F[i'][j])$
        $All[i][j] \leftarrow \max_{j' \in [start\_col, j-1]} Col[i][j']$
        if $s_1[i] = s_2[j]$ **then**
            $F[i][j] \leftarrow All[i][j] + 1$
            $V[i][j] \leftarrow \max(F[i][j], V[i-1][j], V[i][j-1])$
        else
            $F[i][j] \leftarrow 0$
            $V[i][j] \leftarrow \max(V[i-1][j], V[i][j-1])$
        end if
    end for
end for
return $V[|s_1|][|s_2|]$

na poteškoće u generisanju i primjeni union-find strukture. Međutim, prilagodili smo pristup korištenjem drugih naprednih struktura podataka (monotonih redova – *dequeue*) i dobili pristup koji se izvršava u sličnoj kompleksnosti kao i napredni pristup opisan u [35]. U nastavku objašnjavamo prilagođavanje.

U originalnom dinamičkom programiranju za problem najduže zajedničke podsekvence sa promenljivim razmacima (VGLCS), koriste se pomoćne matrice `Col` i `All` za efikasno računanje maksimuma u zadatim pravougaonim regionima matrice F. Tačnije, posmatrajući Algoritam 3, vidimo da

- `Col[i][j]` čuva maksimum vrednosti $F[i'][j]$ za $i' \in [start\_row, i-1]$
- `All[i][j]` čuva maksimum svih `Col[i][j']` za $j' \in [start\_col, j-1]$

Cilj optimizacije je eliminacija ovih matrica radi uštede memorije i postizanja konstantnog (amortizovanog) vremena po ćeliji (i, j). Korišćenjem *monotonih redova* omogućava se da se maksimum u prozoru dimenzije k ($start\_row$ i $start\_col$ varijable) ažurira i očitava u $\mathcal{O}(1)$ vremenu za svaki element. Pseudokod algoritma predstavljen je Algoritmom 4.

Na taj način se vrednosti `Col[i][j]` i `All[i][j]` računaju u letu, bez potrebe

da se eksplicitno čuvaju cijele matrice.

Vremenska složenost pristupa. Za svaki par (i, j), algoritam posmatra pravougaoni region dimenzija do $(G_{s_1}[i]+1) \times (G_{s_2}[j]+1)$ i izračunava maksimum po redovima korišćenjem monotonih struktura *dequeue* (efikasno za 1D maksimum).

U najgorem slučaju, vijeme izvršavanja algoritma je

$$T(n, m) = \mathcal{O}(n \cdot m \cdot G_{s_1}^{\max} \cdot G_{s_2}^{\max}),$$

gdje je $\cdot G_s^{\max} = \max\{G_s[i] \colon i = 1, \ldots, |s|\}$ *Prostorna složenost algoritma je* $\mathcal{O}(|s_1| \cdot |s_2|)$ potrebno za čuvanje matrica F i V.

Ova optimizacija čini algoritam efikasnijim i pogodnijim za implementaciju na većim instancama problema, naročito kada je memorija ograničena.

Algoritam 4 Optimizovani DP za VGLCS koji koristi monotoni red Dequeue

Inicijalizuj matrice $F[0..|s_1|][0..|s_2|] \leftarrow 0$, $V[0..||s_1][0..|s_2|] \leftarrow 0$
for $i \leftarrow 1$ to n **do** $g_a \leftarrow G_{s_1}[i]$
  for $j \leftarrow 1$ to m **do**
    $g_b \leftarrow G_{s_2}[j]$
    $row\_start \leftarrow \max(1, i - g_a - 1)$
    $col\_start \leftarrow \max(1, j - g_b - 1)$
    $max\_val \leftarrow 0$
    for $x \leftarrow row\_start$ to $i - 1$ **do**
      max_deq $\leftarrow$ Koristi monotoni red **deque** za pronalazak max $F[x][col\_start..j-1]$
      $max\_val \leftarrow \max(max\_val, \text{max\_deq})$
    end for
    if $s_1[i] = s_2[j]$ **then**
      $F[i][j] \leftarrow 1 + max\_val$
    else
      $F[i][j] \leftarrow 0$
    end if
    $V[i][j] \leftarrow \max(F[i][j], V[i-1][j], V[i][j-1])$
  end for
end for
return $V[n][m]$

3.3 ILP model

Kada je u pitanju VGLCS problem sa dvije sekvence u ulazu, u ovoj sekciji predlažemo model cjelobrojnog programiranja, metodološki drugačiji pristup od ostalih pristupa u literaturi uglavnom baziranih na DP-u.

U tu svrhu, neka je:

- $M = \{(i, j) \mid s_1[i] = s_2[j]\}$ skup svih pozicija na kojima se karakteri poklapaju.

Za svako $(i, j) \in M$ definišemo binarnu promjenljivu:

$$x_{i,j} = \begin{cases} 1, & \text{ako par } (i,j) \text{ učestvuje u kreiranju rješenja,} \\ 0, & \text{inače.} \end{cases}$$

Uvedimo dodatne binarne promjenljive $s_{i,j}$ koje označavaju da li par (i, j) predstavlja početak podsekvence.

Ciljna funkcija. Maksimizujemo broj izabranih dopustivih poklapanja:

$$\max \sum_{(i,j)\in M} x_{i,j}$$

Ograničenja. Sljedeća ograničenja se uvode:

1. **Prethodnici (varijabilni razmak)**. Za svaki $(i, j) \in M$, definišemo skup validnih prethodnika:

   $$\texttt{Pred}(i, j) = \{(i', j') \in M \mid i' < i, j' < j, i - i' \leq G_{s_1}[i] + 1, j - j' \leq G_{s_2}[j] + 1\}$$

   Ograničenje aktivacije je predstavljeno sa:

   $$x_{i,j} \leq \sum_{(i',j')\in \texttt{Pred}(i,j)} x_{i',j'} + s_{i,j}$$

2. **Početna pozicija**. Dozvoljen je najviše jedan početni par:

   $$\sum_{(i,j)\in M} s_{i,j} \leq 1$$

3. **Konfliktna ograničenja (redoslijed karaktera)**. Dva različita para (i, j) i (i', j') su u konfliktu ako narušavaju redoslijed karaktera:

   $$\text{Ako } (i \leq i' \text{ i } j \geq j') \text{ ili } (i \geq i' \text{ i } j \leq j'), \text{ onda: } \quad x_{i,j} + x_{i',j'} \leq 1$$

Domen promjenljivih je dat sa

$$x_{i,j}, s_{i,j} \in \{0, 1\}, \quad \forall (i, j) \in M$$

Model je formiran tako da maksimizira broj validnih poklapanja koji formiraju zajedničku podsekvencu, pri čemu se poštuju redoslijed elemenata i maksimalni dozvoljeni razmaci između uzastopnih elemenata u obje sekvence.

Glava 4

Generalizovani problem za poizvoljan broj ulaznih sekvenci (MVGLCS)

Problem najduže zajedničke podsekvence sa varijabilnim razmacima može se proširiti na više od dvije sekvence. Prošireni problem, skraćeno nazvan **MVGLCS**, formalno je definisan u obliku ulaz–izlaz na sljedeći način:

Ulaz. Data je kolekcija od m sekvenci

$$s_1, s_2, \ldots, s_m.$$

Za svaku sekvencu s_t pridružena je funkcija razmaka (tj. gap funkcija)

$$G_{s_t}(i) : \{1, \ldots, |s_t|\} \to \mathbb{N}, \quad \text{za } t = 1, 2, \ldots, m.$$

Izlaz. Potrebno je naći niz indeksa dužine p:

$$(i_1^{(1)}, i_1^{(2)}, \ldots, i_1^{(m)}),\ (i_2^{(1)}, i_2^{(2)}, \ldots, i_2^{(m)}),\ \ldots,\ (i_p^{(1)}, i_p^{(2)}, \ldots, i_p^{(m)})$$

koji predstavlja zajedničku podsekvencu, uz zadovoljenje sljedećih uslova:

1. *Poklapanje karaktera:*

$$s_1[i_x^{(1)}] = s_2[i_x^{(2)}] = \cdots = s_m[i_x^{(m)}], \quad \text{za } x = 1, \ldots, p.$$

2. *Rastući indeksi:*

$$i_1^{(t)} < i_2^{(t)} < \cdots < i_p^{(t)}, \quad \text{za svaki } t = 1, \ldots, m.$$

3. *Ograničenja razmaka:*

$$i_x^{(t)} - i_{x-1}^{(t)} \leq G_{s_t}(i_x^{(t)}) + 1, \quad \text{za sve } x = 2, \ldots, p \text{ i } t = 1, \ldots, m.$$

Cilj. Maksimizovati dužinu podsekvence p, tj. pronaći najdužu zajedničku podsekvencu koja poštuje ograničenja razmaka između svih m sekvenci.

Napomena o računskoj složenosti uopštenog problema. Proširenjem na više dimenzija (sekvenci) eksponencijalno raste broj stanja. Ipak, u praksi poput bioinformatike koriste se heuristike i metaheuristike za rješavanje instanci srednje i velike dimenzije. U slučaju da se primijeni metod dinamičkog programiranja za opšti slučaj, to bi zahtijevalo memorisanje m-dimenzionalnog cjelobrojnog vektora kao ključa tabele (heš mape). U osnovi broj takvih ključeva bi bio reda veličine $O(n^m)$ gdje je n dužina najveće sekvence u ulazu, što pristup čini teško primjenjivim već za slučaj $m \geq 3$. Između ostalog, poznato je da je LCS, specijalan slučaj problema kojeg ovdje rješavamo, NP-težak za proizvoljni skup sekvenci S u ulazu.

4.1 Graf stanja i pretaga bima za MVGLCS problem

Za rješavanje MVGLCS problema uz pomoć heurističkog algoritma pretrage tzv. *pretrage bima*, potrebno je definisati graf stanja kroz koji algoritam ima sposobnost da na metodološki način pretražuje najperspektivnije puteve u tom grafu koji odgovaraju konstrukciji rješenja posmatranog problema.

Definicija grafa stanja

Ako je u ulazu dato m sekvenci, svako stanje (čvor) u grafu je definisano m-torkom indeksa:

$$v = (i^{(1)}, i^{(2)}, \ldots, i^{(m)})$$

Gde je $i^{(t)}$ trenutna pozicija u t-toj sekvenci.

- **Početno stanje.** Vještačko (korjeno) stanje $\mathbf{r}$=$(1, 1, \ldots, 1)$, prije nego što je izabran bilo koji karakter. U osnovi to znači da se cio problem $S[\mathbf{r}]$ uzima kao relevantan za proširenje početnog, praznog rješenja ε.
- **Sljedeće stanje.** Iz trenutnog stanja $(i^{(1)}, \ldots, i^{(m)})$ moguće je preći u stanje $(i'^{(1)}, \ldots, i'^{(m)})$ ako su zadovoljeni sljedeći uslovi:

1. Svi karakteri na tim pozicijama se poklapaju:

$$s_1[i'^{(1)}] = s_2[i'^{(2)}] = \cdots = s_m[i'^{(m)}]$$

2. Indeksi rastu:

$$i'^{(t)} > i^{(t)} \quad \text{za sve } t \in \{1, \ldots, m\}$$

3. Poštovana su ograničenja razmaka u svakoj poziciji (gap):

$$i'^{(t)} - i^{(t)} \leq G_t(i'^{(t)}) + 1$$

Izrazimo prethodno rezonovanje u uopštenoj formu, preko grafa stanja za MVGLCS problem kao kompaktnog prostora pretraživanja ovog problema. U osnovni, graf stanja je predstavljen usmjerenim, acikličnim grafom stanja $G = (V, E)$.

Kažemo da parcijalno rješenje s^p *indukuje* čvor $v = ((p_1, \ldots, p_m), l_v)$ akko

- $|s^p| = l_v$;
- $s_i[1 : p_i - 1]$ je prefiks minimalne dužine koji sadrži s^p za sve $i = 1, \ldots, m$;
- Za sve odabrane pozicije poklapanja koje generišu karaktere parcijalnog rješenja s^p u svakoj od sekvenci, zadovoljenja su ograničenja koja se odnose na razmake.

Generalni korjeni čvor je definisan sa $r = ((1, \ldots, 1), 0)$ i odgovara praznom (parcijalnom) rješenju ε. Nadalje, *ciljni* čvorovi su predstavljeni listovima grafa, tj. onim čvorovima koji nemaju potomke. Kažemo da postoji grana (tranzicija) $e = (v_1, v_2)$ između dva čvorova $v_1 = ((p_1^{v_1}, \ldots, p_m^{v_1}), l_{v_1})$ i $v_2 = ((p_1^{v_2}, \ldots, p_m^{v_2}), l_{v_2})$ (čvor v_2 je potomak čvora v_1), gdje je $l(e) = a, a \in \Sigma$, akko je ispunjeno sljedeće:

- Parcijalno rješenje s^{v_2} je dobijeno konkatenacijom simbola $a \in \Sigma$ parcijalnom rješenju koje indukuje čvor s^{v_1}, tj. $s^{v_2} = s^{v_1} \cdot a$ kao i $l_{v_2} = l_{v_1} + 1$.
- $s_1[p_1^{v_2} - 1] = \ldots = s_m[p_m^{v_2} - 1] = a$.
- s^{v_2} zadovoljava sve uslove date ograničenjima razmaka, tj. vrijedi $p_i^{v_2} - 1 - (p_i^{v_1} - 1) = p_i^{v_2} - p_i^{v_1} \leq G_i(p_i^{v_2} - 1) + 1$.

Ciljne tranzicije. Svaki čvor (stanje) koje se može dostići nizom validnih prelaza predstavlja potencijalno rješenje. Cilj je pronaći najduži put u ovom grafu, koji odgovara najdužoj zajedničkoj podsekvenci poštujući praznine. Svaka tranzicija ima težinu 1 (parcijalno rješenje potomaka je prošireno na dopustiv način za jedan karakter u u odnosu na njegove roditeljske čvorove).

Ekstenzija čvora v (generisanje potomaka čvora). Neka je $v = ((p_1^v, \ldots, p_m^v), l_v)$. Neka je sa $\mathtt{a} \in \Sigma^v$ označen skup svih simbola koji se pojavljuju u svakom od sufiksa $s_i[p_i, |s_i|], i = 1, \ldots, m$ a pri tome su zadovoljena ograničenja razmaka. Za dva simbola $\mathtt{a}, \mathtt{b} \in \Sigma^v$ kažemo da $\mathtt{a}$ *dominira* nad $\mathtt{b}$ akko vrijedi

$$p_i^{\mathtt{a},v} \leq p_i^{\mathtt{b},v}, \text{ za sve } i = 1, \ldots, m,$$

gdje $p_i^{a,v}$ predstavlja minimalni indeks $i^* \geq p_i^v$ tako da je $s_i[i^*] = \mathtt{a}$, u suprotnom $i^* = -1$, uz dodatni uslov da je

$$G_i(p_i^{b,v}) + p_i^{a,v} + 1 \geq p_i^{b,v}, \forall i \tag{4.1}$$

U tom slučaju simbol $\mathtt{b}$ se moze eliminisati iz Σ^v, dobijajući skup nedominirajućih ekstenzija Σ_v^{nd} za proširenje čvora v. Generisanje potomaka čvora v je sada predstavljeno procedurom sa sljedećim koracima.

- Izračunati skup nedominirajućih ekstenzija čvora v, tj. skup Σ_v^{nd};
- Za sve simbole $a \in \Sigma_v^{nd}$:
  - Izračunati pozicije prvog pojavljivanja karaktera a u rezidualima, tj. $p^{a,v} = (p_i^{a,v})_{i=1}^m$ (ovo se može izračunati u $O(m)$ vremenu, zahvaljujući *preprocesiranju*); prethodno provjeriti da li je $p_i^{a,v} - p_i^v \leq G_{s_i}(p_i^{a,v})$ za sve i; ako jeste, ova ekstenzija je validna shodno razmacima;
  - Generiši potomak $w = (p^{a,v} + \mathbf{1}, l_v + 1)$
  - Generisati usmjerenu granu iz u prema w, labelirati sa $l(uw) = a$.

Primijetimo da je ovakav način definisanja proširenja predstavlja relaksaciju prostora pretrage, jer treba da ispitamo i one čvorove do kojih se ne može stići iz korjenog čvora $r = ((1, 1), 0)$ da bismo kompletirali pretragu. Dakle, možda se iz takvih čvorova može stići u ciljni čvor preko dužeg puta nego iz r. O ovoj situaciji će biti riječ u nastavku rada.

Proceduru ekstenzije nekog čvora v ćemo označiti sa $\texttt{Extend}(v)$, za koju se podrazumijeva da vraća skup potomaka čvora v i sa tim značenjem ćemo je koristiti u nastavku teksta. U slučaju da je skup ekstenzija čvora v prazan, za v kažemo da je *kompletan*, odnosno predstavljen je kompletnim parcijalnim rješenjem. Te čvorove ćemo zvati *cljinim* čvorovima.

O korjenim čvorovima. U slučaju iscrpnih algoritama pretrage, graf stanja je potrebno proširiti na kompletan prostor pretrage jer se dešava da postoji više korjenih čvorova, osim baznog korjenog čvora r — svaki par poklapanja (i, j) za koja

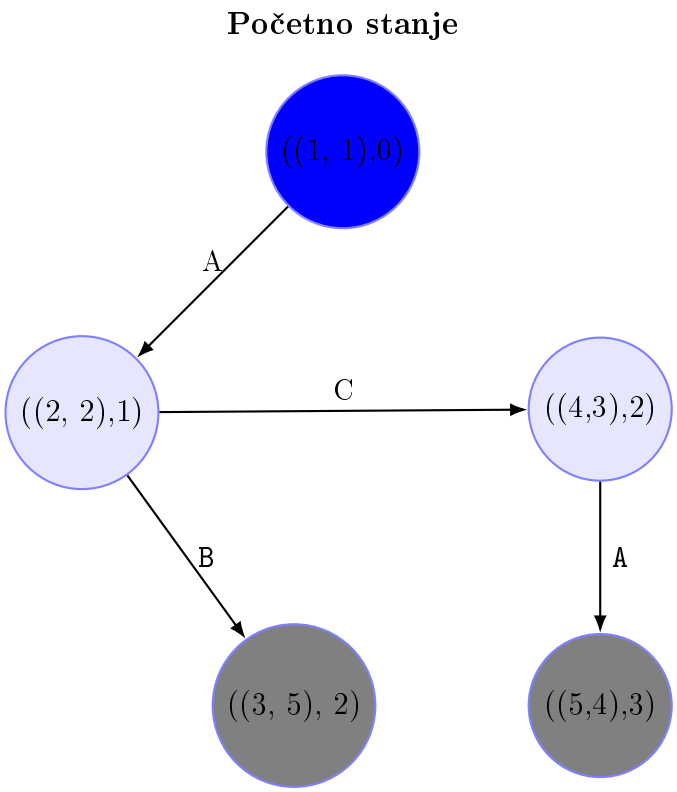
ne postoji niti jedan prethodni par $(i', j') \in M_G(i, j)$ se smatra korjenim. Neka je skup takvih parova označen sa R, kojem pripada i čvor $r = (\mathbf{1}, 0)$ (nemaju roditelj čvora i odgovaraju parcijalnom rješenju ε). Ovakvi čvorovi bi se mogli preprocesirati za mali broj ulaznih sekvenci m, ali veće m su potrebni veliki vremenski resursi, veličine $\mathcal{O}(n^m)$. Dodavanje korjenih čvorova u prostor pretrage će biti heuristički određeno, što ćemo objasniti u nastavku.

Napomena. Broj stanja u grafu stanja generalno iznosi $\mathcal{O}(|s_1| \cdot |s_2| \cdots |s_m|)$, što eksponencijalno raste sa porastom dimnezije m, pa se zbog toga u praksi koriste informisane (i heurističke) metode pretrage, npr. pretraga bima, druge metaheuristike ili informisane egzakne metodologije kao što je A* pretraga.

Primjer. Neka su data dva niza:

- s_1=`ABCA`
- s_2=`ACAB`

sa varijabilnim razmacima dužine 2, tj. $G_i(\cdot) = 2$, za sve $i = 1, \ldots, m$. Graf stanja ove instance je dat Slikom 4.1.



Slika 4.1: Graf stanja za MVGLCSP između sekvenci `ABCA` i `ACAB`. Radi interpretabilnosti dva završna stanja su predstavljena (čvorovi obojeni sivom bojom).

Napomenimo da se MVGLCS problem može posmatrati i sa stanovišta problema sa reverzibilnom sekvencama s_i^{rev}, $i = 1, \ldots, m$, gdje se dobija prirodniji izgled problem u smislu konstrukcije ektenzija (potomaka) čvora u grafu stanja, a ne ispušta se krajnje rješenje originalnog problema. Dakle, skup ulaznih sekvenci u odgovarajućem problemu bi bio dat sa $S^{rev} = \{s_1^{rev}, \ldots, s_n^{rev}\}$, dok bi funkcije razmaka ostale iste. U tom slučaju, dodavanje ekstenzija, tj. proširivanje postojećih parcijalnih rješenja bi išlo u odnosu na pozicije koje se pomijeraju sa desna na lijevo, prema početku svakog od ulaznih sekvenci iz skupa S. Na taj način, za potomak w roditelja $u, u \neq r$ bilo bi potrebno provjeriti da li vrijedi $p_i^u - p_i^w \leq G_i(p_i^w) + 1$, za sve $i = 1, \ldots, m$.

Preprocesiranje. `Succ` predstavlja strukturu koja se kreira prije izvršavanja algoritma, a služi u svrsi efikasnije ekspanzije čvorova, tj. generisanje potomaka u grafu pretrage. U detaljima, `Succ`$[i][j][a]$ gdje su $1 \leq i \leq m, 1 \leq j \leq |s_i|, a \in \Sigma$, smješta minimalni indeks i^* za koji je $s_i[i^*] = a, i^* \geq j$ i za koji je $i^* - (j-1) \leq G_{s_i}(i^*)$ – dakle i^* se uklapa u ograničenja razmacima za datu poziciju. U slučaju da takav indeks ne postoji, dodjeljuje se vrijednost -1.

4.2 Pretraga bima

Pseudokod ovog pristupa dat je Algoritmom 5, uz pretpostavku da je jedini korjeni čvor upravo r, tj. $R = \{r\}$. Uopštenje metoda biće objašnjeno u Sekciji 4.5 Ova poznata metaheuristika predstavlja informisanu pretragu, jer svaki čvor u ovoj pretrazi koristi evaluaciju za procjenu "dobrote", poznatu pod nazivom *heuristika* pretrage. Algoritam pretrage bima započinje od korjenog čvora/stanja $r = (\mathbf{1}, 0)$ i inicijalnog najboljeg rješenja *best*. Na početku, bim sadrži jedino korjeni čvor r. Pretraga se izvršava inkrementalno sve dok bim ne postane prazan. U svakoj iteraciji, za svaki čvor iz bima generišu se njegovi potomci uz pomoć procedure `Extend`(), koji se dodaju u skup proširenja *Candidates*. Ukoliko ovaj skup nije prazan, tada se svi kandidati sortiraju po heurističkoj vrijednosti h, gdje se bolji čvorovi smiještaju na početak liste, a vrijednost *best* se inkrementira za 1. Nakon sortiranja, najviše $\beta > 0$ najboljih čvorova iz skupa *Candidates* se zadržava u narednoj iteraciji, kreirajući novi bim. Pri završetku, kada je bim prazan, algoritam vraća vrijednost najboljeg pronađenog rješenja *best*.

Ovaj algoritam se može posmatrati kao ograničena pretraga u širinu, gdje je širina kontrolisana parametrima veličine bima β. Ova veličina daje balans između pohlepnog pristupa ($\beta = 1$, potpuno pohlepan) i kompletne pretrage ($\beta = +\infty$, što odgovara algoritmu pretrage u širinu–BFS).

Ovaj algoritam jednostavno nazovimo `BS`(r), gdje je r korjeni čvor pretrage bima.

Algoritam 5 Pretraga bima za MVGLCS problem

Ulaz: Sekvence $s_1, s_2, \ldots, s_m$, funkcije praznina $G_{s_1}, \ldots, G_{s_m}$, veličina bima $\beta > 0$

Izlaz: Dužina zajedničke podsekvence uz poštovanje ograničenja razmaka

```
best ← 0
Inicijalizuj beam B ← {r}                          ▷ Početno stanje r
while B ≠ ∅ do
    Candidates ← ∅
    for all v ∈ B do
        N_v ← Extend(v)
        Candidates ← Candidates ∪ N_v
    end for
    if |Candidates| > 0 then
        best ← best + 1
        Sort(Candidates, h)
    end if
    B ← najboljih β iz Candidates
end while
return best
```

4.3 Heuristike za evaluaciju čvorova u algoritmu pretrage bima

U cilju efikasnog rangiranja čvorova, odnosno procjenjivanja njihove "dobrote" u pretrazi, razmatrati se mogu sljedeće heurističke funkcije ocjenjivanja čvora $\mathbf{v} = ((p_1^v, p_2^v, \ldots, p_m^v), l_v)$:

1. Dužina trenutne sekvence (greedy pristup):

$$L(\mathbf{v}) = l_v \tag{4.2}$$

   Ova heuristika favorizuje stanja sa dužim sekvencama.

2. "Pogled unaprijed" za preostalu dužinu:

$$UB_1(\mathbf{v}) = \min_{i=1,\ldots,m} (|s_i| - p_i^v + 1) \tag{4.3}$$

   Procjena potencijalnog produženja sekvence do kraja najmanje sekvence.

3. Frekvencijska usklađenost karaktera:

$$UB_2(\mathbf{v}) = l_v + \sum_{\sigma \in \Sigma} \min(\text{freq}_{s_1}(\sigma, p_1^v), \ldots, \text{freq}_{s_m}(\sigma, p_m^v)) \tag{4.4}$$

gdje je $\text{freq}_{s_i}(\sigma, p_i^v) = |s_i[p_i^v, |s_1|]|_\sigma$.

U osnovi, ova heuristika broji koliko maksimalno karaktera se može poklopiti u (optimalnom) rješenju na osnovu frekvencije svakog od karaktera. Ovaj skor se može izračunati na efikasan način u $O(m)$ vremenu uz pomoć preprocesiranja (kreiranja pametnih struktura podataka).

4. *Musavijeve matrice.* Jedna vrlo korisna tehnika koja služi za dizajniranje heuristika efektivne pretrage baznog problema najdužeg zajedničkog podniza su tzv. Musavijeve matrice, koje aproksimiraju vrijednost vjerovatnoće događaja da je sekvenca s podniz neke slučajne sekvence s_1 nad abecedom Σ.

   Izvođenje ovih struktura ide za LCS problem [28] je izvedeno na sljedeći način. Pretpostavimo da imamo dvije niske s i s_1, dužine k i n, respektivno. Neka je s_1 uniformno generisana niska, dok je s proizvoljna. Razlikujemo dva slučaja.

   (a) Ako je $s_1[n] = s[k]$, onda je $Prob(s \prec s_1) = \frac{1}{|\Sigma|} Prob(s[1, k-1] \prec s_1[1, n-1])$, gdje "$\prec$" predstavlja relaciju *biti podniz*.

   (b) Ako je $s_1[n] \neq s[k]$, onda je $Prob(s \prec s_1) = \frac{|\Sigma|-1}{|\Sigma|} Prob(s \prec s_1[1 : n-1])$

   Prema tome, dobijamo

$$Prob(s \prec s_1) = \frac{1}{|\Sigma|} Prob(s[1 : k-1] \prec s_1[1, n-1]) + \frac{|\Sigma| - 1}{|\Sigma|} Prob(s \prec s_1[1 : n-1]) \tag{4.5}$$

   Kako jednakost (4.5) predstavlja invarijantu u odnosu na obje niske, glavnu ulogu igraju njihove dužine, prethodna formula se može izraziti na sljedeći način koristeći konvenciju $Prob(s \prec s_1) := P(|s_1|, |s|) = P(n, k)$:

$$P(n, k) = \frac{1}{|\Sigma|} P(n-1, k-1) + \frac{|\Sigma| - 1}{|\Sigma|} P(n-1, k), \tag{4.6}$$

   gdje $P(i, 0) = 1$ za svaki $i = 0, \ldots, n$, i $P(0, j) = 0$, za svaki $j = 1, \ldots, k$. Matrica P može da bude implementirana u $O(n^2)$ vremenu uz pomoć metode dinamičkog programiranja. Na ovaj način, aproksimirana vrijednost vjerovatnoće da je proizvoljna niska dužine j podsekvenca uniformne niske dužine i nalazi na poziciji $P(i, j)$.

   Dalje, da bi se heuristika primijenila na svaki čvor u pretrazi bima, neophodno je pretpostaviti da je svaka od niski iz S nezavisna jedna od druge u konstruisanju, kao i da su generisane na slučajni način. Tada se može zaključiti da za

proizvoljnu nisku $s, |s| = k$ vrijedi

$$Prob(s \prec S) = Prob(s \prec s_1) \cdots Prob(s \prec s_1) = \prod_{i=1}^{n} Prob(s \prec s_i) = \prod_{i=1}^{n} P(|s_i|, k) \tag{4.7}$$

Dakle, za heuristiku h i proizvoljan čvor v u pretrazi vrijedi ocjena (dobrote)

$$h(v) = \prod_{i=1}^{n} P(|s_i| - p_i^v + 1, k) \tag{4.8}$$

Ostalo je odrediti vrijednost k, kao i prilagoditi ovu derivaciju za MVGLCS problem.
U tu svrhu, označimo sa $\overline{G_{s_i[p:q]}} = \frac{1}{n} \sum_{j=p}^{q} G_{s_i}(j)$. Primijetimo da u najboljem slučaju imamo da je $\overline{G_{s_i}}$ jednako $|s_i|$. U osnovi, što su kraći razmaci, vjerovatnoća da je s podsekvenca neke niske s_i postaje manja. Prema tome, ima smisla množiti težinski faktor za svaki term pod proizvodom u (4.8), dobijajući

$$h_{prob}(v) = \prod_{i=1}^{n} P(|s_i| - p_i^v + 1, k) \cdot \frac{\overline{G_{s_i[p_i^v:|s_i|]}}}{|s_i| - p_i^v + 1} \tag{4.9}$$

Za izbor vrijednosti k, primijenjujemo istu vrijednost kakva je izabrana za bazni LCS problem i pretragu bima na proizvoljnom nivou pretrage, i to na sljedeći način

$$k = \max \left\{ 1, \left\lfloor \min_{v \in Candidates} \min_{i=1,\ldots,n} \frac{|s_i| - p_i^v + 1}{|\Sigma|} \right\rfloor \right\} \tag{4.10}$$

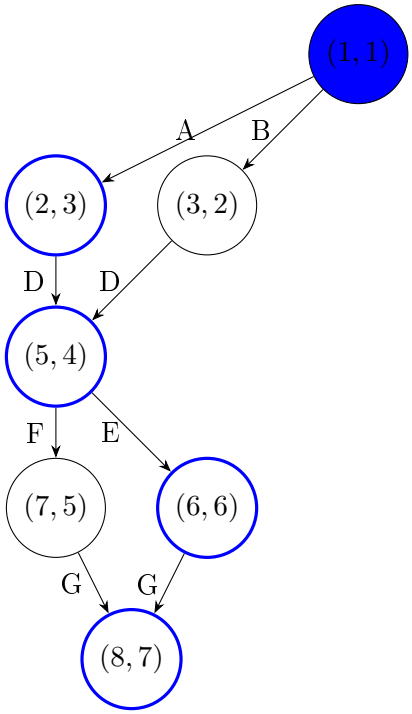
Ova vrijednost se, kako smo rekli, računa za svaki novi nivo u pretrazi bima.

Primjer. Pretpostavimo da su date ulazne sekvence

- s_1=`ABCDEFG`
- s_2=`BADFEGC`

sa funkcijama razmaka $G_1(i) = G_2(i) = 1$.

Na Slici 4.2 je dato BS stablo instance za $\beta = 1$. Primijetimo da se u dopustivim ekstenzijama korjenog čvora nalaze i dva dominirajuća karaktera, a to su `E` i `F` i `G`. Naime, `E` i `F` dominirani od strane karaktera `D`, dok je `G` dominirano od strane karaktera `F`. Prikazane su samo one grane koje zadovoljavaju uslove MVGLCS problema. Rješenje dobijeno pretragom bima (pohlepna strategija sa $\beta = 1$ je `ADEG` sa dužinom rješenja od 4 karaktera.



Slika 4.2: Pohlepna strategija: pretraga bima za $\beta = 1$. Druga koordinata čvorova je izostavljena jer je to jasno na osnovu dubine u kojoj se svaki od čvorova nalazi (čvorovi na istoj dubini su indukovani parcijalnim rješenjima iste dužine). Krajnje rješenje nađeno ovim pristupom je `ADEG`.

4.4 Preprocesiranje

Za efikasno izračunavanje heurističke funkcije UB_2, neophodno je uvesti sljedeću pomoćnu strukturu $C[i][j][a], 1 \leq i \leq m, a \in \Sigma$ koja računa frekvenciju (broj) pojavljivanja karaktera a u relevantnom sufiksu $s_i[j, |s_i|]$. Na taj način, UB_2 se može izračunati u linearnom vremenu $O(|\Sigma| \cdot m) = O(m)$ kao

$$UB_2(\mathbf{v}) = \sum_{\sigma \in \Sigma} \min_{i=1,\ldots,m} C[i][p_i^v][\sigma]$$

4.5 Višestruko-izvorišna pretraga bima

Prethodno opisana pretraga bima može da "ispusti" optimalno rješenje bez obzira ako se pristupom garantuje kompletnost (usljed dovoljno velikog bima β). Dakle, uz pretpostavku da ne razmatramo ograničenja sa razmacima, polazi se od čvora $(1, 1)$ i to je jedini pravi korijen grafa. Međutim, uz prisustvo ograničenja, može se desiti da neki čvor (i, j) nije dostižan od pozicije $(1, 1)$, ali ipak može biti početak validne podsekvence. Na primjer, posmatrajmo dvije sekvence u ulazu, $s_1 =$ `ATGGAAAA` i $s_2 =$ `ATCCAAAA`, sa ograničenjima razmaka $G_{s_1}(\cdot) = G_{s_2}(\cdot) = 1$, za sve pozicije obe

sekvence. Ako bi pretraga bima krenula od pozicije $(1,1)$ (kao korjenim čvorom), zaustavila bi se u poziciji $(2+1, 2+1) = (3,3)$ vraćajući rješenje $s = \texttt{AT}$. Očigledno je da ovo nije optimalno rješenje, jer u slučaju da pokrenemo pretragu bima od pozicije $(3,3)$ kao korjenim čvorom, došlo bi se do boljeg rješenja. U tom slučaju, ekstenzije bi mogle voditi redom na pozicije (resp. čvorove) $(5,5), (6,6), (7,7), (8,8)$, pronalazeći rješenje `AAAA`, što je definitivno duže rješenje od gorepomenutog (s). Ovo se dešava jer postoji više korjenih čvorova osim inicijalno (podrazumijevanog) korjenog čvora $(\mathbf{1}, 0)$ koji segmentiraju prostor pretrage na dva dijelom disjunktna (pod)grafa. Zbog toga, neophodno je dodatno prilagoditi pretragu bima, opisanu Algoritmom 5 da bi bio kompetitivan i u ovakvim slučajevima. Ovo znači da graf stanja ne mora biti povezan i da je neophodno pretragu inicirati iz svih izvornih čvorova kako bi se pokrile sve moguće nedominirajuće komponente. U tu svrhu, za čvor kažemo da je izvorišni ako se ne može "uhvatiti" kao produžetak nekog ranije generisanog čvora. Kako izvorišnih čvorova može biti eksponencijalno mnogo, te to zavisi od distribucije veličine razmaka, u praksi nije smisleno generisati sve takve čvorove prije pokretanja pretrage bima. Npr. možemo ih generisati heuristički, birajući one više obećavajuće u odnosu na veličinu razmaka (pozicije sa manjim razmakom imaju veći šansu da budu dio komponenti nekog korjenog čvora).

Inicijalizujmo skup korjenih čvorova R date instance problema sa $r = (\mathbf{1}, 0) \in R$; pokrenimo pretragu bima gdje je r inicijalni korjen, dakle $\texttt{BS}(r)$, te izbacimo r iz R. Za svaki čvor v koji je kompletan/ciljni u pretrazi bima (skup potomaka je prazan), generišimo skup (nedominirajućih) čvorova $r^w = (p^w, 0)$, gdje p_i^w odgovara minimalnoj poziciji p_i^w koja je veća ili jednaka $p_i^v, i = 1, \ldots, m$ i pri čemu je $s_i[p_i^w] = a, a \in \Sigma$, za sve $i = 1, \ldots, m$. Potom, ubacimo ovakve čvorove u skup R. Nakon završetka pretrage bima, najbolje rješenje se čuva u s_{best}, potom se uzima novi čvor $r^w = (p^w, 0) \in R$, te se ispituje da li vrijedi $|s_{best}| < \min_{i=1,\ldots,m}\{|s_i| - p_i^w + 1\}$). Ako vrijedi, pokrenemo novu pretragu bima $\texttt{BS}(r^w)$ ponavljajući prethodne korake, te po potrebi ažurirajući najbolje rješenje. U protivnom, uzimamo novi korjeni čvor iz R sve dok postoji barem jedan neselektovani čvor u tom skupu ili je izvršen određeni broj BS iteracija.

Dakle, skup izvorišnih čvorova R u startu nije "poplavljen" ogromnim brojem korjenih čvorova, nego se oni dinamički dopunjuju tokom pretrage.

Ukratko, ova vrsta tzv. iterativne pretrage bima sa višestrukim izvorišnim čvorovima (eng. *iterativnog multi-source beam search* (IMSBS)), generiše izvorišne čvorove "na zahtjev". To je u praktičnom scenariju korisno jer spaja lokalno pretraživanje i globalnu pokrivenost prostora rješenja.

Prednosti IMSBS algoritma su:

- Izbjegava se eksplozija izvorišnih čvorova.
- Heuristički balans između pretrage bima (lokalno pretraživanje obećavajućih puteva) i generisanja novih izvora (globalno pokrivanje prostora).
- Dominacija filterisanje: smanjuje nepotrebne čvorove, ubacuje odbacijanja dominirajućih puteva, tzv. *pruning*.

Skup R se ovdje prioritizuje – uzimamo one kojene čvorove koji su više obećavajući shodno UB_2 heuristici. Koliko je takvih iteracija kompletnog BS-a inicijalizovano čvorovima iz skupa R je parametrizovano parametrom $beam\_iters$.

Algoritam 6 Iterativna pretraga bima sa višestrukim izvorišnim čvorovima (IMSBS) za MVGLCS

Require: Sekvence $s_1, \ldots, s_m$; funkcije razmaka $G_{s_i}(\cdot)$; veličina bima B; $number\_sources\_from\_R$; $beam\_iters$
Ensure: Aproksimirana zajednička podsekvenca s_{best}

Inicijalizuj $R \leftarrow \{(\mathbf{1}, 0)\}$ ▷ Skup korjenih čvorova
$iter \leftarrow 0$
$s_{\text{best}} \leftarrow \varepsilon$
while $R \neq \emptyset \wedge iter < beam\_iters$ **do**
    Izaberi $S \subseteq R$ $number\_sources\_from\_R$ čvorova, ukloni ih iz R ▷ R moze biti i sortiran npr. u odnosu na h_5
    $B \leftarrow S$
    Pokreni beam search $BS(B)$
    for all kompletan čvor v iz $BS(B)$ **do**
        for all simbol $a \in \Sigma$ **do**
            $p^w \leftarrow$ minimalne pozicije u sekvencama $\geq p^v$ gdje $s_i[p_i^w] = a$
            if $|s_{\text{best}}| \leq \min_{i=1,\ldots,m}\{|s_i| - p_i^w + 1\}$ **then**
                Dodaj $r^w = (p^w, 0)$ u R
            end if
        end for
    end for
    Ažuriraj s_{best} ako je pronađena duža podsekvenca u $BS(B)$
    $iter \leftarrow iter + 1$
end while
return s_{best}

Glava 5

Eksperimentalna analiza

U ovoj sekciji izlažemo rezultate svakog od algoritama predstavljenih u prethodnim sekcijama. U tu svrhu, prvo opisujemo način na koji su instance generisane, a potom obrađujemo rezultate numerički.

Svi algoritmi su testirani na mašini sa AMD Ryzen 9 5900HS sa brzinom procesora od 3.30GHz i memorijom od 16GB. Svi eksperimenti su puštani na jednoj niti (eng. single-threaded).

5.1 Generisanje instanci MVGLCS problema

Za generisanje ulaznih instanci MVGLCS problema potrebno je kreirati sljedeće komponente:

- Skup sekvenci $s_1, s_2, \ldots, s_m$, nad abecedom Σ;
- Funkcije dozvoljenog razmaka (gap funkcije) $G_k : \{1, \ldots, |s_k|\} \to \mathbb{N}$, $k = 1, \ldots, m$.

Tipično se koristi konačna abeceda, kao što je $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, a sekvence se generišu nasumičnim izborom karaktera iz abecede, generišući slučajne sekvence. Funkcije razmaka mogu biti uniformne (svi karakteri imaju isti maksimalni razmak) ili neuniformne, generisane slučajnim cjelobrojnim vrijednostima.

Format ulazne instance: Svaka instanca zapisana je u sljedećem formatu:

1. Prvi red: broj sekvenci m
2. Zatim, za svaku sekvencu:
    - Red sa karakterima sekvence;

- Red sa odgovarajućom funkcijom razmaka sekvence zapisane u prethodnom redu (razmak za svaki broj koji pripada kodomenu funkcije poštujući redoslijed indeksa).

Primjer. Jedna takva instanca je data u nastavku.

```
3
ACGTAGCTAG
1 2 1 2 1 2 1 2 1 2
TGCTAGCATA
2 1 2 1 2 1 2 1 2 1
CGATCGTACG
1 1 1 1 1 1 1 1 1 1
```

Pretpostavimo da su sekvence koje generišemo jednake dužine (n). Za svaku kombinaciju parametara instance:

- $n \in \{50, 100, 200\}$
- $m \in \{2, 3, 5, 10\}$
- $|\Sigma| \in \{2, 4\}$

10 instaci se generišu na sa slučajnim sekvencama. Prema tome, ukupno 180 instanci problema je generisano. Funkcije razmaka za svaku od instanci predstavljene su slučajnim brojevima pri čemu je zadovoljeno $G_{s_i}(\cdot) \in \{\lfloor 0.5 \cdot |\Sigma| \rfloor, \ldots, \lfloor 1.5 \cdot |\Sigma| \rfloor\}$ za sve $i = 1, \ldots, m$.

Sve instance skupa podataka RANDOM se mogu naći na linku `https://github.com/markodjukanovic90/NikolaBalabanDiplomski/tree/main/src/instance\_i\_generator`.

5.2 Numerički rezultati egzaktnih algoritama za $m = 2$

U ovoj sekciji poredimo rezultate pristupa iz literature i to redom

- Osnovni algoritam dinamičkog programiranja, pogledati Algoritam 2, označen sa DP;
- Napredni algoritam dinamičkog programiranja sa matricama `Col` i `All`, označen sa DP-1, pogledati Algoritam 3

- Napredni algoritam dinamičkog programiranja sa strukturom *dequeue*, označen sa DP-2, pogledati Algoritam 4;
- Metod cjelobrojnog linearnog programiranja, označen sa ILP, čiji je model predstavljen u Sekciji 3.3.

Svakom algoritmu je dato pola sata (30 minuta) za izvršavanje. ILP model je rješavan pomoći generalnog rješavača CPLEX-a verzije 12.8.0.

Tabela 5.1 je organizovana na sljedeći način. Prve tri kolone daju karakteristike grupa instanci nad kojima je uprosječen rezultat (10 instanci u svakoj grupi). Dalje, naredna četiri bloka daju rezultate četiri algoritma: DP, DP-1, DP-2 i ILP pristupa, ovim redom. Nad 10 instanci svake grupe prikazan je prosječni kvalitet rješenja svakog od algoritama zajedno sa prosjekom vremena izvršavanja.

Tabela 5.1: Rezultati na skupu podataka RANDOM za slučaj $m = 2$: egzaktni pristupi iz literature.

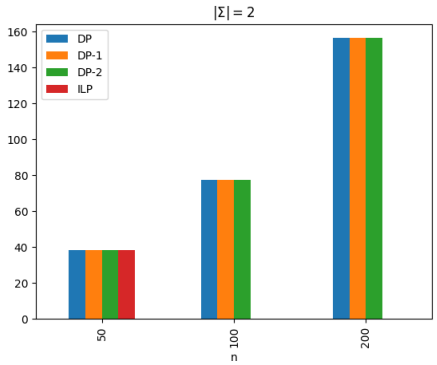
Inst.			DP		DP-1		DP-2		ILP	
m	n	$\lvert\Sigma\rvert$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$
2	50	2	38.10	**0.01**	38.10	**0.01**	38.10	**0.01**	38.10	79.42
2	50	4	30.30	**0.01**	30.30	0.02	30.30	0.02	30.30	18.0
2	100	2	77.40	0.09	77.40	**0.03**	77.40	0.04	0.0	–
2	100	4	62.30	0.07	62.30	**0.06**	62.30	0.08	0.00	1952.0
2	200	2	156.40	0.60	156.40	**0.13**	156.40	0.16	0.0	–
2	200	4	127.20	0.46	127.20	**0.24**	127.20	0.31	0.0	–

Na osnovu numeričkih rezultata iz Tabele 5.1, možemo zaključiti sljedeće:

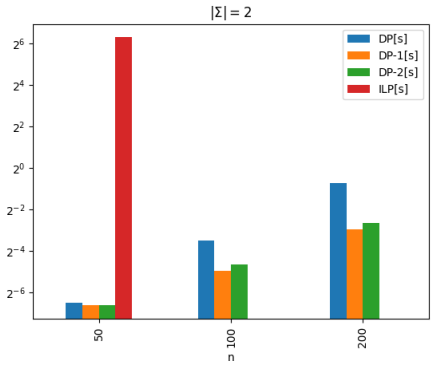
- Sva tri pristupa dinamičkog programiranja su ekstremno efikasna i sposobna su nađu optimalna rješenja za svih 60 instanci.
- Vremena koja su potrebna pristupima DP-a da nađu optimalna rješenja su veoma kratka, ispod 1 sekunde.
- Pristup ILP je sposoban da riješi veoma male instance sa $n = 50$ (dakle, 60 instanci) ali je za to potrebno dva reda veličine veće vrijeme od dinamičkog programiranja. Za ostale instance, a posebno za instance sa $n = 200$, ILP nije sposoban riješiti ni početnu (korjenu) relaksaciju.

5.3 Numerički rezultati heurističkih algoritama

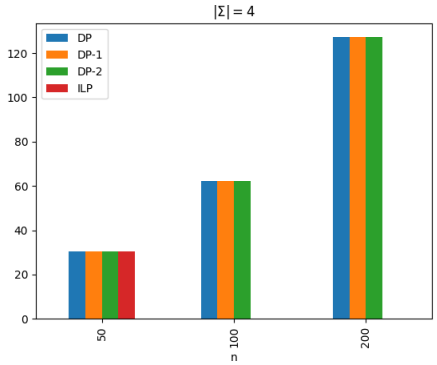
U ovoj sekciji poredimo rezultate heurističkih pristupa dizajnirani za slučaj generalizovanog problema za proizvoljno $m \geq 2$ i to



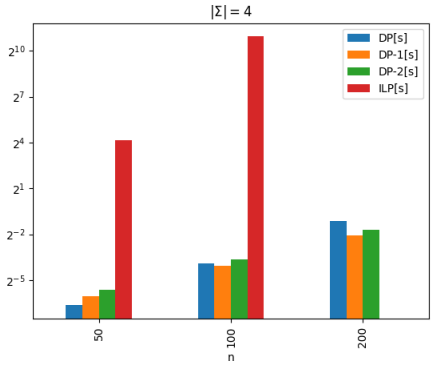
(a) Instance sa $|\Sigma| = 2$: kvalitet rješenja



(b) Instance sa $|\Sigma| = 2$: vremena



(c) Instance sa $|\Sigma| = 4$: kvalitet rješenja

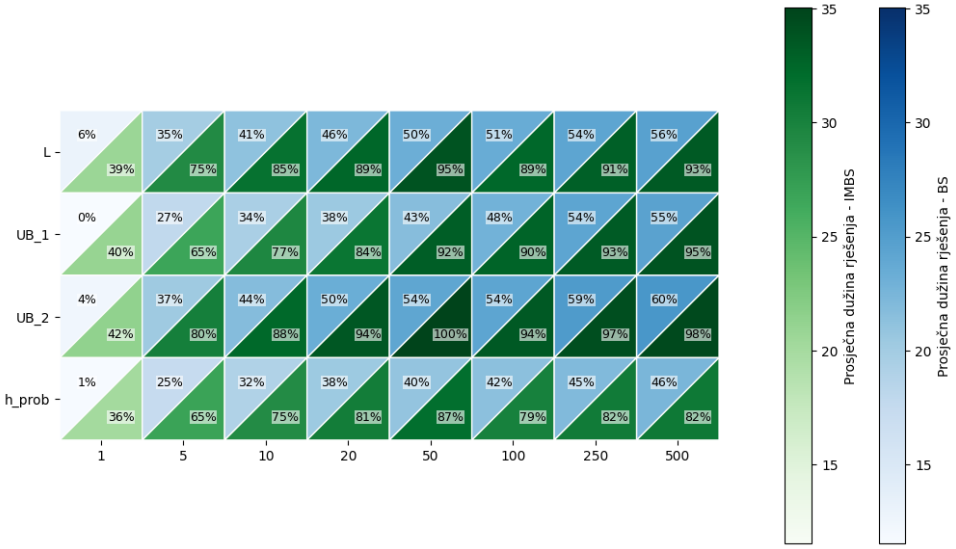


(d) Instance sa $|\Sigma| = 4$: vremena

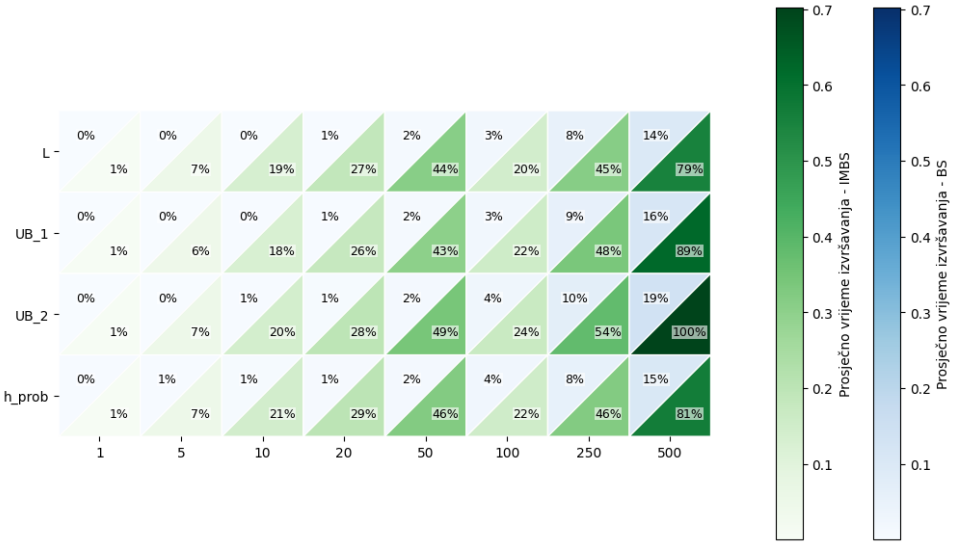
Slika 5.1: Kvalitet rješenja i vremena za 4 pristupa u slučaju instanci za $m = 2$.

- Bazni pristup pretrage bima, dat Algoritmom 5, u oznaci Bs sa veličinom bima $\beta = 10000$.
- Iterativna pretraga bima sa višestrukim izvorišnim čvorovima, dat Algoritmom 6, u oznaci IMSBS sa veličinom bima $\beta = 500$.
- Pohlepni pristup, koji odgovara baznom pristupu pretrage bima u konfiguraciji $\beta = 1$, u oznaci Greedy.

Pri izvođenju eksperimenata su za IMSBS bili postavljeni parametri $beam\_iters = 50$ za $\beta < 100$, $beam\_iters = 10$ za $\beta \geq 100$ i $number\_sources\_from\_R = 10$. Ove vrijednosti su izabrane da bi se uspostavila ravnoteža između dozvoljenih vremenskih resursa u odnosu na vremena baznog BS pristupa, te krajnjeg kvaliteta rješenja. Kvaliteti rješenja i vremena izvršavanja algoritama u zavisnosti od korištene heuristike i veličine bima su prikazani na Slici 4 i Slici 5.



Slika 5.2: Kvalitet rješenja BS i IMSBS u zavisnosti od veličine bima i heuristike.



Slika 5.3: Vrijeme izvršavanja BS i IMSBS u zavisnosti od veličine bima i heuristike.

Tabela 5.2: Rezultati na skupu podataka RANDOM za slučaj $m \geq 2$: heuristički pristupi

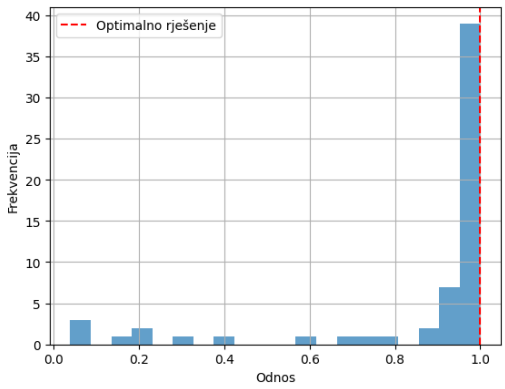
Inst.			GREEDY		BS		IMSBS	
m	n	$\|\Sigma\|$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$
2	50	2	19.45	**0.00**	33.60	0.01	**34.98**	0.03
2	50	4	23.02	**0.00**	**29.65**	0.48	28.18	0.07
2	100	2	20.08	**0.00**	48.15	1.09	**63.72**	0.32
2	100	4	46.50	**0.00**	**55.38**	5.48	54.48	0.39
2	200	2	21.85	**0.00**	95.55	9.28	**121.58**	2.46
2	200	4	60.48	**0.00**	**109.35**	19.73	108.85	2.51
3	50	2	12.32	**0.00**	17.50	0.01	**25.80**	0.04
3	50	4	12.50	**0.00**	**21.65**	0.09	21.32	0.06
3	100	2	10.55	**0.00**	19.70	0.03	**45.95**	0.42
3	100	4	11.22	**0.00**	33.40	1.24	**39.28**	0.99
3	200	2	7.22	**0.00**	15.20	0.07	**59.82**	0.89
3	200	4	17.15	**0.00**	77.82	13.02	**83.88**	6.06
5	50	2	2.95	**0.00**	4.80	**0.00**	**14.40**	0.02
5	50	4	3.80	**0.00**	8.90	**0.00**	**11.70**	0.04
5	100	2	4.45	**0.00**	6.30	**0.00**	**10.90**	0.02
5	100	4	2.17	**0.00**	5.30	**0.00**	**17.80**	0.06
5	200	2	3.80	**0.00**	5.30	**0.00**	**13.40**	0.05
5	200	4	3.72	**0.00**	6.40	0.01	**17.40**	0.09
10	50	2	1.42	**0.00**	1.70	**0.00**	**4.50**	0.01
10	50	4	1.35	**0.00**	1.90	**0.00**	**3.40**	0.02
10	100	2	1.05	**0.00**	1.10	**0.00**	**4.10**	0.02
10	100	4	1.35	**0.00**	2.20	**0.00**	**3.20**	0.03
10	200	2	1.65	**0.00**	2.50	**0.00**	**5.40**	0.04
10	200	4	1.42	**0.00**	2.20	**0.00**	**3.20**	0.04

Na osnovu numeričkih rezultata iz Tabele 5.2, možemo zaključiti sljedeće.

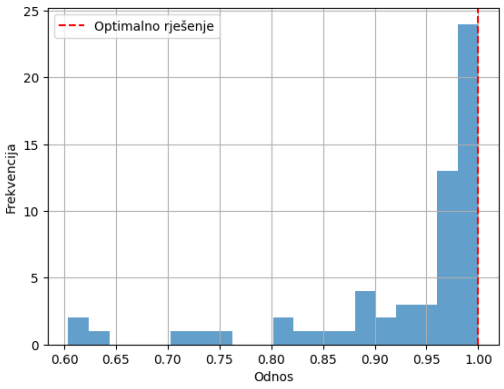
- IMSBS pronalazi najkvalitetnija rješenja u 20 od 24 grupe i u prosjeku nadmašuje BS za oko 8 karaktera, ili u relativnom procentu od 31.66%. Bazni pristup je marginalno bolji na 4 grupe, za manje m i $|\Sigma| = 4$. Prednost IMSBS nad BS raste sa porastom veličine ulaznih niski n, a naročito za $n = 200$ dostiže preko 12 karaktera u prosjeku, ili u relativnom procentu od 31.56%. Najveća prednost IMSBS nad BS je preko 44 karaktera za grupu $m = 3, n = 200$ i $|\Sigma| = 2$.

- Pohlepni pristup je očekivano po konstrukciji brži u odnosu na ostale pristupe, ali donosi najlošiji kvalitet rješenja. BS ima duže prosječno vrijeme izvršavanja (2.11s) u odnosu na IMSBS (0.61s), iako IMSBS ima kraće vrijeme za 8 od 24 grupe. Međutim, u najvećem broju slučajeva, razlika je neosjetna i bliska nuli (pogledati grupe instanci sa $m \in \{5, 10\}$). Bazni pristup na pojedinim grupama ima znatno natprosječna vremena izvršavanja sa ekstremom u $m = 2, n = 200, |\Sigma| = 4$, gdje je značajnije sporiji od IMSBS. To se može objasniti time da je za manje m početni izbor korjenog čvora $r = (1, \ldots, 1)$ dobar izbor, te BS napreduje u dubini i pronalaska visoko-kvalitetnih rješenja. Sa porastom m, šansa da je r dobar izbor za kojeni čvor dramatično opada, a time i kvalitet BS-a zajedno sa vremenom izvršavanja. Tu dolazimo do zaključka da je IMSBS ipak znatno robusniji i otporniji na promjenu broja ulaznih niski instance.

- Kao što je prikazano na Slici 7, BS i IMSBS u mnogim slučajevima dostižu približno ili tačno optimalna rješenja za manje n i veće $|\Sigma|$. Kvalitet rješenja u odnosu na optimalna u slučaju BS pristupa za najveći broj instanci od 65% i više, dok su to za IMSBS od 80% i više. Za oko 50-ak instanci (od ukupno 60) kvaliteta od 90% kvalitete optimalnih rješenja je obezbijeđeno sa ova dva pristupa, što se može vidjeti na Slici 6.

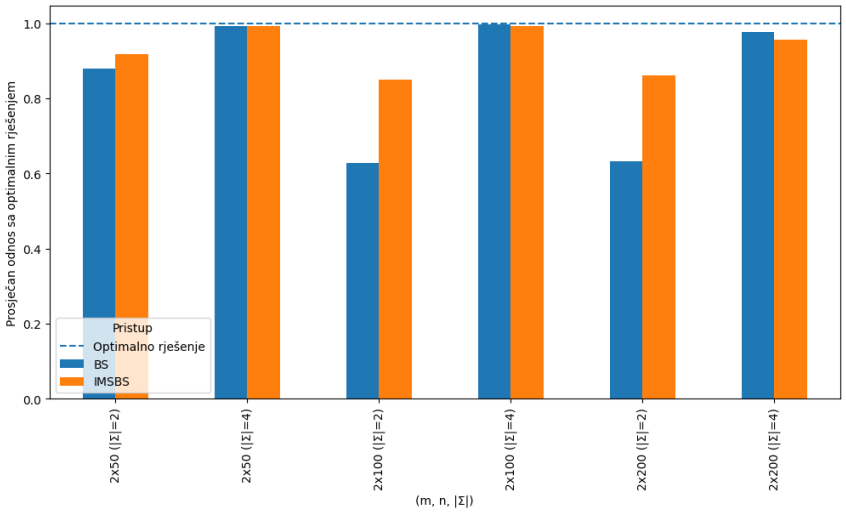


(a) Frekvencije kvaliteta rješenja - BS



(b) Frekvencije kvaliteta rješenja - IMSBS

Slika 5.4: Frekvencije kvaliteta BS i IMSBS rješenja u odnosu na optimalna za $m = 2$.



Slika 5.5: Kvaliteti BS i IMSBS rješenja u odnosu na optimalna za $m = 2$.

Glava 6

Zaključak

U ovom radu smo razmatrali generalizovanu verziju problema najdužeg zajedničkog podniza (LCS) sa proizvoljnim brojem niski, koji ima niz primjena u bioinformatici i strukturalnom poređenju DNK i RNK molekula. Problem najdužeg zajedničkog podniza sa varijabilnim razmacima, rješavan ovdje, unosi dodatnu dimenziju u osnovni LCS problem, jer ograničava rastojanja između karaktera koji pripadaju rezultujućem zajedničkom podnizu, što odgovara realnom scenarijumu da nukleotidi u molekulama koji su bliži jedni drugima interaguju više nego oni udaljeniji. Dati problem je riješen pomoću pretrage bima, kao jedne limitirane varijante pretrage u širinu koja je usmjerena određenim heuristikama. Prvo je definisan generalni prostor pretrage preko grafa stanja i konstruisana bazna pretraga bima, koja kreće od samog početka svake od sekvenci. Uvidjeći da postoji više korjenih čvorova u pretrazi koji dijele prostor na podregione koji ne mogu svi da se dohvate idući iz početne pozicije svake od sekvenci, problem se sveo na nalazak obećavajućih korjenih čvorova—kao globalni aspekt našeg algoritma—a potom nalaska najboljeg rješenja problema iz tog korjenog čvora—utičući na lokalni akspekt problema. Lokalni aspekt problema je riješen pokretanjem bazne pretrage bima sa odabranim korjenim čvorom. Globalni aspekt je dizajniram iterativnim puštanjem pretrage bima na pažljivo odabrane korjene čvorove, koji su prikupljeni u toku ranije pokretanih pretraga bima, sortirani heurističkom funkcijom favorizući one najbolje. Ovaj algoritam je nazvan iterativna pretraga bima sa višestrukim korjenim čvorovima (IMSBS).

Eksperimentalna analiza je poredila tri vrste heurističkih algoritama sa četiri vrste heuristika, gdje se frekvencijska heuristika pokazala kao najbolja. Tri algoritma koja su poređena su: (*i*) pohlepni pristup koji kreće od početka svake od sekvenci i bira samo jedan karakter za proširenje u svakom koraku; (*ii*) bazna pretraga bima sa parametrizovanom veličinom bima, koja kreće od početne pozicije svake od sekvenci; (*iii*) IMSBS. Generisane su sintetičke instance na kojima su testirani svi pristupi. Što se tiče heurističkih algoritama, najbolja se pokazala pretraga IMSBS, koja za

slično ili manje vremena u odnosu na baznu pretragu bima je nalazila rješenja znatno boljeg kvaliteta. Nadalje, obična pretraga bima se pokazala korisnom za slučaj sa manjim brojem ulaznih sekvenci (m) i alfabetom kardinalnosti 4. U ovom radu je specijalno obrađen slučaj instanci sa $m = 2$, u kojem su pored heurističkih pristupa, upoređena i tri egzaktna pristupa iz literature, te ILP pristupa, specijalno dizajniran ovdje za slučaj $m = 2$. Napredni egzaktni pristup dinamičkog programiranja (bez nepotrebnih rekalkulacija) se pokazao kao najbolji, rješavajući sve instance ovog tipa, kao i ostala dva pristupa dinamičkog programiranja. ILP pristup se pokazao limitirajući, rješavajući samo male instance sa $n = 50$. Heuristički pristupi bazirani na pretragi bima su se pokazali efikasni, za većinu instanci nađen je kvalitet rješenja od 90% kvaliteta optimalnog rješenja. Međutim, heuristićki pristupi dolaze sa prednošću da mogu biti iskorišćeni za rješavanje instanci sa proizvoljnim m.

Što se tiče budućeg istraživanja i narednih koraka, ideja je da se razviju naprednije heuristike koje na pametniji način uzimaju u obzir i razmake na svakoj od pozicija. Npr. uticaj razmaka bi se direktno mogao inkorporirati u Musavijeve matrice, kroz rekurziju računanja vrijednosti vjerovatnoća. Dalje, IMSBS algoritam bi se mogao dodatno optimizovati, da se svaka iteracija izvršava u sklopu već dijelom "iscrtanog" grafa stanja. Dakle, ukoliko bismo naišli na već ispitan čvor u ranijim iteracijama, u okviru njega bismo mogli da čuvamo informaciju koliko duboko u prostor pretrage se otišlo prije prateći taj čvor. Tako bi se spriječio ulazak u već ispitane regione pretrage za koje znamo da ne doprinose nalasku bolejg rješenja, bez da region lokalno ispitujemo. Dodatno, trebalo bi se razmisliti o generisanju realnih instanci i studiji koja bi pokazala benefit ovih algoritama u praktičnim situacijama. Još, trebalo bi povećati dimenziju samih sekvenci, da se provjeri skalabilnost algoritama za znatno veće sekvence. Posljednje, ali ne manje bitno, je korištenje alata za automatsko hiper-parametarsko podešavanje pogodnijih konfiguracija svakog od algoritama, kao što su irace [25] ili Optuna [1] koji bi mogli dodatno poboljšali kvalitet rješenja.

Literatura

[1] T. Agrawal. "Optuna and autoML" in *Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient*. Springer, 2020, pp. 109–129.

[2] H.-Y. Ann, C.-B. Yang, and C.-T. Tseng. "Efficient polynomial-time algorithms for the constrained LCS problem with strings exclusion". *Journal of Combinatorial Optimization* 28(4) (2014), pp. 800–813.

[3] A. Apostolico. "Improving the worst-case performance of the Hunt-Szymanski strategy for the longest common subsequence of two strings". *Information Processing Letters* 23(2) (1986), pp. 63–69.

[4] A. Apostolico and C. Guerra. "The longest common subsequence problem revisited". *Algorithmica* 2(1) (1987), pp. 315–336.

[5] A. N. Arslan and Ö. Eğecioğlu. "Algorithms for the constrained longest common subsequence problems". *International Journal of Foundations of Computer Science* 16(06) (2005), pp. 1099–1109.

[6] L. Bergroth, H. Hakonen, and T. Raita. "A survey of longest common subsequence algorithms" in *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000.* IEEE. 2000, pp. 39–48.

[7] C. Blum and M. J. Blesa. "A comprehensive comparison of metaheuristics for the repetition-free longest common subsequence problem". *Journal of Heuristics* 24(3) (2018), pp. 551–579.

[8] C. Blum, M. J. Blesa, and B. Calvo. "Beam-ACO for the repetition-free longest common subsequence problem" in *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer. 2013, pp. 79–90.

[9] C. Blum, M. J. Blesa, and M. Lopez-Ibanez. "Beam search for the longest common subsequence problem". *Computers & Operations Research* 36(12) (2009), pp. 3178–3186.

[10] A. Chaudhuri. "A dynamic algorithm for the longest common subsequence problem using ant colony optimization technique". *arXiv preprint arXiv:1307.1905* (2013).

[11] S. Deorowicz and J. Obstój. "Constrained longest common subsequence computing algorithms in practice". *Computing and Informatics* 29(3) (2010), pp. 427–445.

[12] M. Djukanovic, G. R. Raidl, and C. Blum. "A beam search for the longest common subsequence problem guided by a novel approximate expected length calculation" in *International conference on machine learning, optimization, and data science*. Springer. 2019, pp. 154–167.

[13] M. Djukanovic, G. R. Raidl, and C. Blum. "Finding longest common subsequences: new anytime A* search results". *Applied Soft Computing* 95 (2020), p. 106499.

[14] C. G. Fernandes, C. E. Ferreira, C. Tjandraatmadja, and Y. Wakabayashi. "A polyhedral investigation of the LCS problem and a repetition-free variant" in *Latin American Symposium on Theoretical Informatics*. Springer. 2008, pp. 329–338.

[15] D. S. Hirschberg. "Algorithms for the longest common subsequence problem". *Journal of the ACM (JACM)* 24(4) (1977), pp. 664–675.

[16] M. Horn, G. Raidl, and C. Blum. "Job sequencing with one common and multiple secondary resources: An A∗/Beam Search based anytime algorithm". *Artificial Intelligence* 277 (2019), p. 103173.

[17] J. W. Hunt and T. G. Szymanski. "A fast algorithm for computing longest common subsequences". *Communications of the ACM* 20(5) (1977), pp. 350–353.

[18] C. S. Iliopoulos and M. S. Rahman. "Algorithms for computing variants of the longest common subsequence problem". *Theoretical Computer Science* 395(2-3) (2008), pp. 255–267.

[19] P. Janičić and M. Nikolić. "Veštačka inteligencija" (2021).

[20] T. Jiang and M. Li. "On the approximation of shortest common supersequences and longest common subsequences". *SIAM Journal on Computing* 24(5) (1995), pp. 1122–1139.

[21] T. Jiang, G.-H. Lin, B. Ma, and K. Zhang. "The longest common subsequence problem for arc-annotated sequences" in *Annual Symposium on Combinatorial Pattern Matching*. Springer. 2000, pp. 154–165.

[22] T. Jiang, G. Lin, B. Ma, and K. Zhang. "The longest common subsequence problem for arc-annotated sequences". *Journal of Discrete Algorithms* 2(2) (2004), pp. 257–270.

[23] M. Li, X. Chen, X. Li, B. Ma, and P. M. Vitányi. "The similarity metric". *IEEE transactions on Information Theory* 50(12) (2004), pp. 3250–3264.

[24] D. Liben-Nowell, E. Vee, and A. Zhu. "Finding longest increasing and common subsequences in streaming data". *Journal of Combinatorial Optimization* 11(2) (2006), pp. 155–175.

[25] M. López-Ibáñez, L. P. Cáceres, J. Dubois-Lacoste, T. G. Stützle, and M. Birattari. *The irace package: User guide*. IRIDIA, Institut de Recherches Interdisciplinaires et de Développements en ..., 2016.

[26] C. Meister, T. Vieira, and R. Cotterell. "Best-First Beam Search". *Transactions of the Association for Computational Linguistics* 8 (Dec. 2020), pp. 795–809.

[27] R. S. Mincu and A. Popa. "Better heuristic algorithms for the repetition free LCS and other variants" in *International Symposium on String Processing and Information Retrieval*. Springer. 2018, pp. 297–310.

[28] S. R. Mousavi and F. Tabataba. "An improved algorithm for the longest common subsequence problem". *Computers & Operations Research* 39(3) (2012), pp. 512–520.

[29] N. Nakatsu, Y. Kambayashi, and S. Yajima. "A longest common subsequence algorithm suitable for similar text strings". *Acta Informatica* 18(2) (1982), pp. 171–179.

[30] H. Ney, D. Mergel, A. Noll, and A. Paeseler. "A data-driven organization of the dynamic programming beam search for continuous speech recognition" in *ICASSP '87. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 12. 1987, pp. 833–836.

[31] B. Nikolic, A. Kartelj, M. Djukanovic, M. Grbic, C. Blum, and G. Raidl. "Solving the longest common subsequence problem concerning non-uniform distributions of letters in input strings". *Mathematics* 9(13) (2021), p. 1515.

[32] P. S. OW and T. E. MORTON. "Filtered beam search in scheduling†". *International Journal of Production Research* 26(1) (1988), pp. 35–62.

[33] M. Paterson and V. Dančík. "Longest common subsequences" in *International symposium on mathematical foundations of computer science*. Springer. 1994, pp. 127–142.

[34] Y.-H. Peng and C.-B. Yang. "Finding the gapped longest common subsequence by incremental suffix maximum queries". *Information and Computation* 237 (2014), pp. 95–100.

[35] Y.-H. Penga and C.-B. Yangb. "The Longest Common Subsequence Problem with Variable Gapped Constraints" in *Proceedings of the 28th Workshop on Combinatorial Mathematics and Computation Theory, Penghu, Taiwan.* Citeseer. 2011, pp. 17–23.

[36] M. Sazvar, M. Naghibzadeh, and N. Saadati. "Quick-MLCS: A new algorithm for the multiple longest common subsequence problem" in *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering.* 2012, pp. 61–66.

[37] B. Taboada, C. Verde, and E. Merino. "High accuracy operon prediction method based on STRING database scores". *Nucleic acids research* 38(12) (2010), e130–e130.

[38] Y.-T. Tsai. "The constrained longest common subsequence problem". *Information Processing Letters* 88(4) (2003), pp. 173–176.

[39] R. A. Wagner and M. J. Fischer. "The string-to-string correction problem". *Journal of the ACM (JACM)* 21(1) (1974), pp. 168–173.

[40] L. Wang, X. Wang, Y. Wu, and D. Zhu. "An Efficient Dynamic Programming Algorithm for the Generalized LCS Problem with Multiple Substring Exclusion Constrains". *arXiv preprint arXiv:1303.1872* (2013).

[41] J. Yang, Y. Xu, G. Sun, and Y. Shang. "A new progressive algorithm for a multiple longest common subsequences problem and its efficient parallelization". *IEEE Transactions on Parallel and Distributed Systems* 24(5) (2012), pp. 862–870.

[42] M. Yu, G. Li, D. Deng, and J. Feng. "String similarity search and join: a survey". *Frontiers of Computer Science* 10(3) (2016), pp. 399–417.

[43] D. Zhu and X. Wang. "A simple algorithm for solving for the generalized longest common subsequence (LCS) problem with a substring exclusion constraint". *Algorithms* 6(3) (2013), pp. 485–493.