PROBLEM NAJBLIŽE NISKE

(Closest String Problem)

Seminarski rad u okviru kursa *Računarska inteligencija*
Matematički fakultet
Univerzitet u Beogradu

Aleksa Kojadinović
130/2017
avgust 2021.

Sadržaj

Uvod

Problem najbliže niske *(Closest String Problem – CSP)* formulisan je na sledeći način.

Dato je n niski $s_1, s_2, \ldots, s_n$ dužine m. Naći nisku s dužine m koja minimizuje d gde je $d = \max\{d_H(s, s_i) | i = 1, 2 \ldots, n\}$. Sa d_H obeležavamo Hamingovu distancu između dve niske, tj.:

$$d_H(s_1, s_2) = \sum_{j=1}^{m} \chi(s_1[j], s_2[j])$$

gde je

$$\chi(a, b) = \begin{cases} 0 & a = b \\ 1 & a \neq b \end{cases}$$

i sa $s[j]$ obeležavamo slovo na j-toj poziciji niske s.

Intuitivno, traži se niska koja je po Hamingovoj distanci najbliža celom skupu zadatih niski, gde kao meru bliskosti celom skupu niski uzimamo udaljenost od one koja joj je najudaljenija.

Ubraja se u NP teške probleme i predmet je mnogih istraživanja, posebno u oblasti bioinformatike.

Najpre predstavljamo proste algoritme pretrage grubom silom, s jednim potencijalnim poboljšanjem. Zatim sagledamo rešenja metaheurističkim algoritmima. Na kraju prikazujemo jednu polinomijalnu aproksimaciju.

Rešavanje problema

Pre prikazivanja rešenja uvodimo neku uobičajenu notaciju.

Σ predstavlja konačan skup slova (karaktera) koji nazivamo *azbuka*. Niske najčešće označavamo malim slovima $s_1, s_2, ...$ Pojedinačne karaktere označavamo standardnom notacijom indeksiranja, upotrebom uglastih zagrada: $s[j]$. Tako nisku s dužine m nekad zapisujemo u razvijenom obliku $s[1]s[2]...s[m]$.

Neka je data niska $s = s[1]s[2]...s[m]$ i lista indeksa $I = [j_1, j_2, \ldots, j_r] \subseteq [1, 2, \ldots, m]$ gde su svi elementi I različiti. Sa $s_{|I}$ označavamo nisku dobijenu od niske s korišćenjem samo slova na indeksima iz I, odnosno $s_{|I} = s[j_1]s[j_2]...s[j_r]$.

Rešenja grubom silom

Kao kod svakog kombinatornog problema, jedan očigledan način za rešavanje je pretraga po celom skupu mogućih vrednosti. Skup svih mogućih vrednosti su sve niske dužine m nad azbukom Σ. Slova se mogu proizvoljno ponavljati pa za svaku poziciju imamo $|\Sigma|$ mogućih slova, tako da je kardinalnost skupa mogućih rešenja $|\Sigma|^m$.

Ostaje još pitanje generisanja svih mogućih niski. Jedan način je generisanje svih kombinacija u leksikografskom poretku, međutim ovo rešenje ne daje prostora ni za kakve dodatne optimizacije. Zbog toga u rešenju koristimo pretragu u dubinu (DFS) kako bismo kasnije iskoristili tehniku odsecanja.

Iscrpna pretraga

Ideja DFS-a je da se počne od prazne niske na nultom nivou. Prvi nivo dobijamo tako što na prvoj poziciji u niski dodajemo svako slovo azbuke, zatim drugi nivo tako što na drugoj poziciji uradimo isto, i tako dalje. Ukupan broj nivoa u stablu pretrage biće upravo dužina niske - m.

```
min_hamming = float('inf')
min_string = None
q = ['']
while q:
    curr_string = q.pop()
    curr_string_length = len(curr_string)
    if curr_string_length == m:
        curr_string_score = problem_metric(curr_string, strings)
        if curr_string_score < min_hamming:
            min_hamming = curr_string_score
            min_string = curr_string
        continue
    q += [curr_string + next_letter for next_letter in alphabet]
```

Pretraga sa odsecanjem

Unapređenje prethodnog algoritma dobija se uvođenjem standardne tehnike odsecanja prilikom pretrage.

Naime, primetimo da se proširivanjem niske Hamingova distanca može samo povećati. Neka je s proizvoljna niska. Neka je p niska koja je prefiks niske r. Ako zanemarimo egzaktno indeksiranje i dužine niski (Hamingovu distancu između niski različitih dužina računamo samo na pozicijama kraće niske) važi:

$$d_H(s,p) \leq d_H(s,r)$$

Pošto ovo važi za Hamingovu distancu, samim tim važiće i za maksimum Hamingovih distanci kroz n niski, odnosno upravo za ciljnu funkciju našeg problema.

Ideja algoritma je da, pošto pamtimo trenutno najbolje rešenje, možemo prekinuti pretragu u onom trenutku kada zaključimo da bilo koja niska trenutnog prefiksa ne može dovesti do boljeg rešenja, odnosno onda kada je vrednost funkcije cilja primenjene na trenutni prefiks, i sve prefikse trenutne dužine originalnih niski, veći ili jednak od trenutnog najboljeg rešenja. Preciznije, neka je trenutna nepotpuna niska $s = s[1]s[2]...s[d], d < m,\ D = [1, 2, \ldots, d]$ i b trenutno najbolje rešenje. Pretragu prekidamo ako je ispunjen sledeći uslov:

$$\max\{d_H(s, s_{i|D})|i = 1, 2, ..., n\} >= b$$

Kako god trenutnu nisku proširili, rezultat može postati samo lošiji, pa nema potrebe pretraživati nastavke.

U ovome se ogleda razlog zašto je korišćena baš pretraga u dubinu, a ne pretraga u širinu (BFS). Korišćenjem DFS-a dolazimo do listova što pre, a uslov da uopšte dođe do odsecanja je dolazak do bar jednog lista. Jer u suprotnom najbolja dosadašnja vrednost uvek će biti ∞, pa gornji uslov ni jednom neće biti ispunjen. Ukoliko bismo koristili BFS, listovi bi na red došli poslednji pa bi odsecanja bila skoro trivijalna, dok bi veliki broj unutrašnjih čvorova bio nepotrebno obrađen.

U teoriji, ukoliko nikada ne bi dolazilo do odsecanja, ovaj algoritam bi bio čak i lošije složenosti od prethodnog jer zahteva računanje delimične ciljne funkcije u svakoj iteraciji. U praksi, ipak, odsecanja su česta i algoritam sa odsecanjem ponaša se značajno bolje.

```
q = ['']
min_hamming = float('inf')
min_string = None

while q:
    iterations += 1
    curr_string = q.pop()
    curr_string_length = len(curr_string)
    curr_string_score = problem_metric(curr_string, strings)

    if curr_string_score >= min_hamming:
        continue

    if curr_string_length == m:
        if curr_string_score < min_hamming:
            min_hamming = curr_string_score
            min_string = curr_string
        continue
    q += [curr_string + next_letter for next_letter in alphabet]
```

*Napomena: Hamingova distanca implementirana je korišćenjem python-ovog **zip** generatora. U slučaju da se proslede dve niske nejednake dužine, „višak“ duže niske se ignoriše. Iz tog razloga nije potrebno implementirati specijalnu verziju Hamingove distance niti ciljne funkcije za rad ovog algoritma.*

Poređenje iscrpne pretrage i pretrage sa odsecanjem

Poređenje ova dva algoritma dato je ranije iz razloga što su neuporedivi sa ostalim rešavačima u kontekstu brzine. Svi ostali eksperimentalni nalazi biće dati u pretpostlednjem poglavlju.

Naredna tabela prikazuje rezultate uporednog testiranja iscrpne pretrage i pretrage sa odsecanjem. Testiranje je izvršeno na identičnim problemima rastuće složenosti po dužini niske, od $m = 2$ do $m = 20$, za fiksiranu binarnu azbuku $\{0, 1\}$ i fiksiran broj niski $n = 10$. RT (*Running Time*) označava vreme u sekundama zaokruženo na dva decimalna mesta. S (*Score*) označava vrednost funkcije cilja. Oba rešavača daju egzaktna rešenja, tako da je podudarnost u koloni S očekivana. Što se tiče vremena izvršavanja, primećujemo značajnu superiornost algoritma sa odsecanjem prilikom rasta složenosti problema.

	Brute Force Solver		Pruning Solver	
m	RT	S	RT	S
2	0	1	0	1
3	0	2	0	2
4	0	2	0	2
5	0	2	0	2
6	0	3	0	3
7	0	3	0	3
8	0	4	0	4
9	0	4	0	4
10	0.01	4	0	4
11	0.02	5	0	5
12	0.04	5	0	5
13	0.08	5	0.01	5
14	0.16	6	0.02	6
15	0.34	5	0	5
16	0.69	6	0.02	6
17	1.45	7	0.05	7
18	3.39	6	0.03	6
19	6.62	7	0.16	7
20	15.11	8	0.35	8

Metaheuristička rešenja

Metaheuristička rešenja često se prirodno nameću u kombinatornim optimizacionim problemima iz razloga brzine i jednostavnosti. Predstavljamo dva takva rešenja predložena u [1][2][3], sa eventualnim manjim modifikacijama.

Genetski algoritam

Rešenje genetskim algoritmom ne odstupa mnogo od opšte forme genetskog algoritma, tj. nema mnogo prostora za nove ideje. Niske su dobar kandidat za rad sa njima jer je relativno jednostavno definisati operatore ukrštanja i mutacije. Ideja predložena u [3], i ukratko opisana u [1], koristi sledeći pristup.

Selekcija se vrši najpre odabirom $\frac{|P|}{2}$ jedinki, gde je P populacija. Odabir se vrši diskretnom raspodelom verovatnoća koja je obrnuto proporcionalna funkciji cilja (jer je cilj minimizacija), tačnije jedinka p_i se bira sa verovatnoćom (težinom) $m - f(p_i)$ gde je f funkcija cilja.

Ukrštanje se vrši nad nasumično odabranim parovima jedinki iz prethodnog koraka. Konkretan operator ukrštanja realizuje se odabirom nasumične pozicije *j*, na osnovu koje dobijamo dva potomka za roditelje s_1 i s_2:

$$c_1 = s_{1|\{1,2,\ldots j\}} s_{2|\{j+1,j+2,\ldots,m\}}$$
$$c_2 = s_{2|\{1,2,\ldots j\}} s_{1|\{j+1,j+2,\ldots,m\}}$$

Mutacija se sa zadatom verovatnoćom vrši nad svakom novonastalom jedinkom, odabirom nasumične pozicije i menjanjem slova na toj poziciji na nasumično slovo iz azbuke. Pošto je verovatnoća mutacije obično niska (5%), obezbeđuje se da ukoliko se ispuni uslov za mutaciju - da do nje zapravo dodje; odnosno da ne može da dođe do situacije gde se odabrano slovo zameni istim slovom.

Nova generacija se pravi korišćenjem elitističke strategije. Na kraju vršenja svih operacija imamo originalnu generaciju i novu generaciju. Generaciju za sledeću iteraciju biramo odabirom najboljih jedinki od svih (iz obe generacije).

Prikazujemo pseudokod korišćenog genetskog algoritma:

m, n, Σ, s – ulazni problem

N – maksimalan broj iteracija, P – trenutna populacija, γ – stopa mutacija

$b = NULL$ – najbolja jedinka

f – funkcija cilja

Inicijalizuj trenutnu populaciju nasumičnim niskama.

while broj iteracija nije premašen:

1. Izaberi $\frac{|P|}{2}$ jedinki za ukrštanje, jedinka p_i bira se sa verovatnoćom $m - f(p_i)$
2. Za svaki nasumično izabran par jedinki iz jedinki odabranih za ukrštanje primeni operator ukrštanja. Na novodobijene jedinke primeni operator mutacije sa verovatnoćom γ.
3. Neka je S skup novodobijenih jedinki (nova generacija). Trenutnu populaciju zameni biranjem najboljih $|P|$ jedinki iz skupa $P \cup S$
4. Ažuriraj najbolju jedinku b

vrati b

Optimizacija kolonijom mrava

ACO (*Ant Colony Optimization*) jedna je od popularnih metaheurističkih metoda za rešavanje kombinatornih problema ove vrste. Ideja korišćenja ACO za rešavanje problema najbliže niske razmatrana je u [1] i [2]. U eksperimentalnim rezultatima kasnije pokazaće se kao vrlo uspešna.

U opštem slučaju, ACO se koristi za pronalaženje najkraćeg puta u grafu. Zasniva se na ideji puštanja većeg broja agenata (*mrava*) da pretražuju puteve najpre nasumično. Svaki od njih eventualno dolazi do svog rešenja. U okviru jedne iteracije pretrage, najbolje rešenje zovemo *lokalno optimalno*. Svaki mrav na svom putu ostavlja tragove *feromona.* Putevi koji vode ka boljim rešenjima imaće jače tragove feromona, pa će budući mravi sa većom verovatnoćom birati takve puteve. Ukoliko bismo sistem definisali samo na ovaj način postojala bi velika opasnost, kao u mnogim algoritmima optimizacije, od zaglavljivanja u lokalnim optimumima. Zbog toga se vrši i korak *evaporacije* koji kroz iteracije bezuslovno umanjuje tragove feromona, u nadi da će na taj način lokalno optimalna rešenja eventualno biti prevaziđena.

Konkretan model koji predstavljamo zasniva se na korišćenju matrice feromona. To je matrica dimenzija $m \times |\Sigma|$, gde polje matrice na koordinatama j, α gde je $j \in \{1, \ldots, m\}, \alpha \in \Sigma$ sadrži količinu feromona prilikom biranja slova α za poziciju j. Tragovi feromona se inicijalno postavljaju na $\frac{1}{|\Sigma|}$ za svako polje i svako slovo.

primer. Za azbuku $\Sigma = \{a, b, c\}$ i $m = 4$, inicijalna matrica feromona je sledeća.

	a	**b**	**c**
1	0.33	0.33	0.33
2	0.33	0.33	0.33
3	0.33	0.33	0.33
4	0.33	0.33	0.33

Odabir slova vrši se posmatranjem reda u matrici koji odgovara rednom broju slova kao težinsku raspodelu za svako slovo. Većim količinama feromona odgovara veća verovatnoća odabira datog slova. Naravno, neće uvek biti odabrano slovo sa najvećom vrednošću. Ukoliko bismo koristili taj pristup, opasnost od zaglavljivanja u lokalnim optimumima bila bi velika.

Ostavljanje tragova/ažuriranje feromona vrši se, ponovo, elitističkom strategijom - samo najbolji mrav iz trenutne iteracije ima pravo da ažurira svoj trag feromona. Neka je m dužina niski i lb vrednost funkcije cilja najboljeg mrava iz trenutne iteracije. Za svako polje matrice kojim je najbolji mrav „prošao“ ažuriranje se vrši po sledećoj formuli:

$$M_{j,a}^{\text{new}} = M_{j,a}^{\text{old}} + (1 - \frac{lb}{m})$$

Primetimo da što je rešenje bolje, to je lb manje, samim tim mera $(1 - \frac{lb}{m})$ raste, pa se vrednost feromona zapravo povećava.

Evaporacija feromona vrši se zadavanjem fiksnog metaparametra $\rho \in (0, 1)$ koji nazivamo *stopa evaporacije* po formuli:

$$M_{j,a}^{\text{new}} = M_{j,a}^{\text{old}} * (1 - \rho)$$

Prikazujemo pseudokod korišćenog ACO algoritma.

m, n, Σ, s – ulazni problem

N – maksimalan broj iteracija, A – broj mrava, ρ – stopa evaporacije

$b = NULL$ – najbolji mrav

M – matrica feromona

Inicijalizuj matricu feromona tako da je svaka vrednost $\frac{1}{|\Sigma|}$

while broj iteracija nije premašen:

1. Inicijalizuj $lb = NULL$ kao lokalnog najboljeg mrava
2. Ponovi A puta:
   i. Konstruiši mrava biranjem za svaku poziciju $j = 1, \ldots, m$ slovo α sa verovatnoćom $\frac{M_{j,\alpha}}{\sum_{\beta \in \Sigma} M_{j,\beta}}$
   ii. Proveri da li je trenutni mrav bolji od lb i ažuriraj lb po potrebi
3. Izvrši evaporaciju feromona za sve elemente matrice
4. Izvrši ažuriranje feromona po tragovima lokalnog najboljeg mrava lb
5. Proveri da li je lb bolji od b i ažuriraj b po potrebi

vrati b

Polinomijalne aproksimacije

Razvijeno je nekoliko PTAS (*Polynomial Time Approximation Scheme*) algoritama za aproksimaciju problema najbliže niske u polinomijalnom vremenu ([4][5][6]). Ipak, eksperimentalni rezultati će pokazati kasnije, njihova primena i dalje ostaje najviše teorijska.

Polinomijalne aproksimacije koje su razvijene zasnivaju se na sličnoj ideji, dok su razlike u samim algoritmima male. Osnovna ideja je uzimanje nekog potproblema iz početnog problema (recimo samo određenih pozicija u niskama, ili samo određenog skupa ulaznih niski) i njegovo formulisanje kao problema celobrojnog programiranja. Zapravo, kao i svaki optimizacioni problem na diskretnom i konačnom skupu, ceo problem najbliže niske može se opisati problemom celobrojnog programiranja, konkretnije problemom 0-1 programiranja. Problem je u tome što je obim takvog problema često vrlo veliki i nepraktičan za rešavanje postojećim metodama rešavanja celobrojnog programiranja. Naime, problem možemo formulisati na sledeći način.

$$
\begin{aligned}
&(\min) \quad d \\
&\sum_{\alpha \in \Sigma} x_{j,a} = 1, \quad j = 1, 2, \dots m \\
&\sum_{1 \le j \le m} \sum_{\alpha \in \Sigma} \chi(s_i[j], \alpha) x_{j,\alpha} \le d, \quad i = 1, \dots n \\
&x_{j,\alpha} \in \{0, 1\}, \quad j = 1, \dots, m, \alpha \in \Sigma
\end{aligned}
$$

$x_{j,\alpha}$ je indikatorska promenljiva koja ima vrednost 1 ako je slovo na poziciji j baš α. Prvi skup uslova nam govori da se na jednoj poziciji može nalaziti tačno jedno slovo. Drugi skup nam govori da Hamingova distanca do svake ulazne niske može biti najviše d. Postavljanjem kao cilja minimizaciju d dobijamo upravo problem najbliže niske.

U konkretnim aproksimacijama, naravno, nećemo formulisati ceo problem kao problem celobrojnog programiranja, već samo neki njegov potproblem. Strategija za rešavanje se svakako ne menja. Efikasno rešavanje problema je nemoguće, tako da se primenjuje sledeća strategija.

- Ukoliko je (po nekom kriterijumu) problem malog obima onda ga rešiti grubom silom, proverom svih $|\Sigma|^m$ mogućih niski.
- Ukoliko je (po nekom kriterijumu) problem velikog obima onda napraviti njegovu LP relaksaciju odbacivanjem uslova celobrojnosti, tj. postaviti $x_{j,\alpha} \in [0, 1]$. Na taj način možemo proceniti rešenje originalnog problema tretirajući, za svaku poziciju j, vektor $(x_{j,\alpha_1}, x_{j,\alpha_2}, \dots, x_{j,\alpha_{|\Sigma|}})$ kao raspodelu verovatnoće za odabir slova na poziciji j.

PTAS korišćenjem parova niski

Naredna jednostavna aproksimacija iz [4] bira dve niske iz skupa ulaznih niski $s_1, s_2 \in S$ takve da je $s_1, s_2 = \text{argmax}_{i \neq j} d_H(s_i, s_j)$, odnosno dve niske koje su međusobno najudaljenije. Neka je $Q = [j \in \{1, \ldots, m\} | s_1[j] = s_2[j]]$ uređena lista pozicija gde se s_1 i s_2 slažu i, slično, P uređena lista pozicija gde se ne slažu. Obzirom na to da su odabrane niske najudaljenije, a ipak se slažu na $|Q|$ pozicija, razumno je u rezultujućoj niski na sve pozicije iz Q ostaviti upravo ta slova. Za ostale pozicije formuliše se potproblem celobrojnog programiranja:

$$
\begin{aligned}
&(\min) \quad d \\
&\sum_{\alpha \in \Sigma} x_{j,a} = 1, \quad j = 1, 2, \ldots |P| \\
&\sum_{1 \leq j \leq |P|} \sum_{\alpha \in \Sigma} \chi(s_i[P[j]], \alpha) x_{j,\alpha} \leq d, \quad i = 1, \ldots n \\
&x_{j,\alpha} \in \{0, 1\}, \quad j = 1, \ldots, |P|, \alpha \in \Sigma
\end{aligned}
$$

U ovom slučaju se i značenje promenljive $x_{j,\alpha}$ menja (zbog indeksiranja), tj. sada će ta promenljiva predstavljati indikator da se na poziciji $P[j]$ nalazi slovo α.

Problem rešavamo opisanom strategijom, koristeći neki kriterijum obimnosti problema. Kriterijum iskorišćen u originalnom radu je sledeći:

- Ako je $|P| \leq \frac{6 \ln(\sigma m)}{\epsilon^2}$ problem rešiti grubom silom
- Inače problem rešiti LP relaksacijom

za neke fiksirane $\epsilon \in (0, 1), \sigma > 1$.

Nakon rešavanja problema dobijamo novu nisku x za koju očekujemo da je bolje rešenje od s_1 ili s_2. Ipak, nije nam garantovano, tako da kao konačno rešenje uzimamo bolju nisku između x i s_1.

Pokazano je da ovakva PTAS aproksimira problem u vremenu $O\left(\frac{4}{3}(1+\epsilon)d\right)$, gde je d optimalno rešenje. Pseudokod izostavljamo jer je praktično ceo algoritam opisan gornjim postupkom.

Eksperimentalni rezultati

U testiranju izostavljamo rešavače grubom silom iz razloga pomenutih ranije.

Ponašanje rešavača pri povećanju obima ulaza

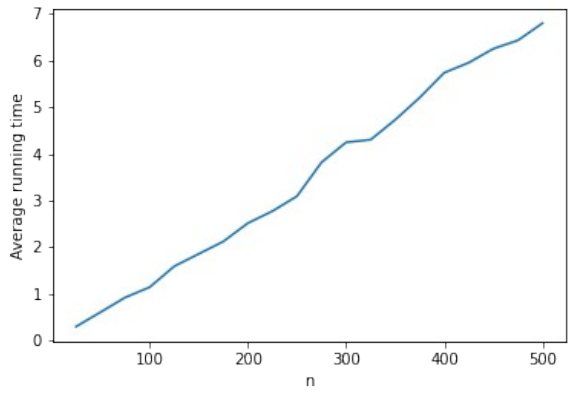
U ovom odeljku testiramo svaki rešavač nezavisno i posmatramo ponašanje pri povećanju svakog od ulaznih parametara. Za svaku vrednost ulaznog parametra koji se testira, rešavač je pokrenut 5 puta i prosečne vrednosti su korišćene.

Od interesa su nam dve mere: *prosečno vreme izvršavanja (Average running time)* i *prosečan kvalitet rešenja (Average quality)*. Vreme merimo u sekundama, dok se kvalitet rešenja računa u procentima kao $100 * \frac{(m-r)}{m}$ gde je r rešenje a m dužina niske. Usled činjenice da je za velike probleme nemoguće dobiti egzaktna rešenja, nema načina da objektivno procenimo kvalitet rešenja, pa koristimo ovu meru. Ipak, svi rešavači se testiraju na istom skupu problema, pa je ipak moguće uvideti odnose između njih.

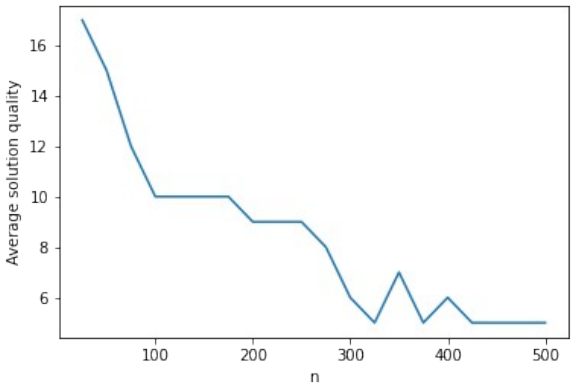
Povećavanje broja ulaznih niski

U duhu primarne primene ovog problema, korišćena je fiksna azbuka azotnih baza $\Sigma = \{A, C, T, G\}$. Dužina niski fiksirana je na $m = 20$, dok n varira po vrednostima $n \in \{25, 50, 75, 100, \ldots, 500\}$.

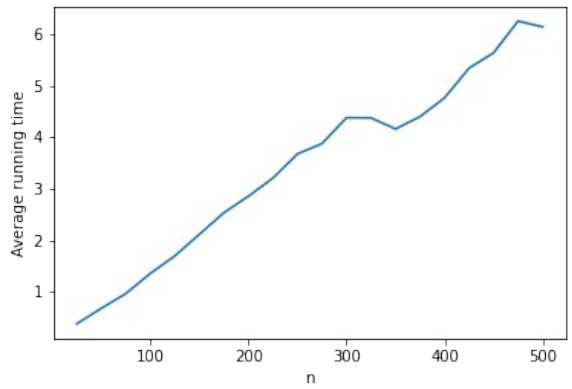
Genetski algoritam konfigurisan je na 500 iteracija i veličinu populacije 10, sa stopom mutacije od 5%. Slično, kolonija mrava konfigurisana je na 500 iteracija i veličinu kolonije 10, sa stopom evaporacije 10%.



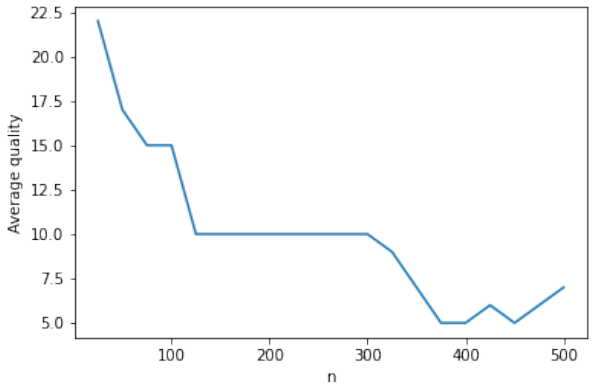
Genetski algoritam *– vreme izvršavanja*
Uočavamo linearan rast prosečnog vremena izvršavanja u odnosu na broj ulaznih niski.



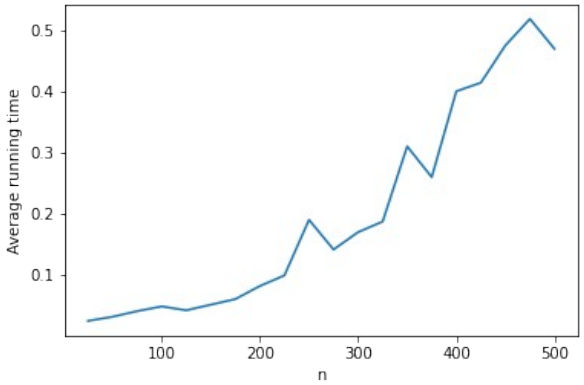
Genetski algoritam *– kvalitet rešenja*
Kvalitet rešenja opada, međutim to je i očekivano s porastom broja niski.



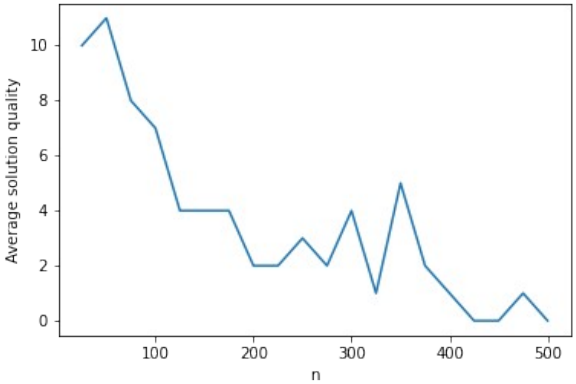
***ACO** – vreme izvršavanja*
Uočavamo linearan rast prosečnog vremena izvršavanja u odnosu na broj ulaznih niski.



***ACO** – kvalitet rešenja*
Kvalitet rešenja opada, međutim to je i očekivano s porastom broja niski.



***PTAS** – vreme izvršavanja*
PTAS se izvršava veoma brzo, ali razlog za to je što je svaki put iskorišćeno linearno programiranje, pa mu je složenost ekvivalentna složenosti rešavanja problema linearnog programiranja.

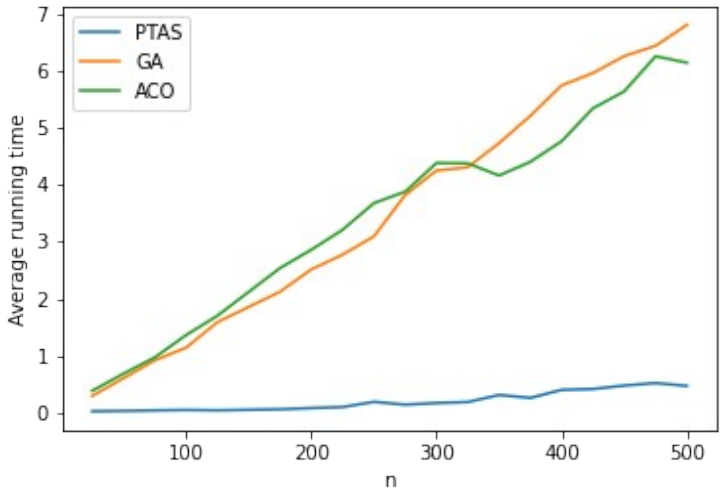


***PTAS** – kvalitet rešenja*
Problem s PTASom je kvalitet rešenja zbog korišćenja linearnog programiranja za svaki problem.

Poređenje rezultata

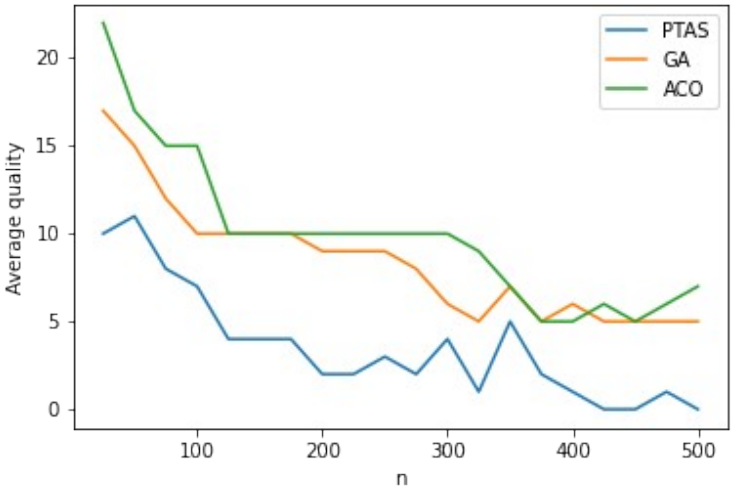
U narednim tabelama i graficima razmatramo koji rešavač je najbolji u kom aspektu na zadatim problemima. Podebljanim brojevima naznačavamo najbolji rezultat. PTAS se pokazao kao najbrži, ali na uštrb kvaliteta. Ukoliko bismo poredili samo ACO i genetski algoritam, ACO se čini kao bolji izbor jer pruža i kraće vreme i bolji kvalitet.

n	GA	ACO	PTAS
25	0.30	0.38	**0.02**
50	0.61	0.68	**0.03**
75	0.92	0.97	**0.04**
100	1.14	1.36	**0.05**
125	1.59	1.70	**0.04**
175	2.12	2.54	**0.06**
200	2.51	2.86	**0.08**
225	2.77	3.21	**0.10**
250	3.09	3.68	**0.19**
275	3.82	3.88	**0.14**
300	4.25	4.38	**0.17**
325	4.30	4.37	**0.19**
350	4.73	4.16	**0.31**
375	5.21	4.40	**0.26**
400	5.74	4.76	**0.40**
425	5.96	5.34	**0.42**
450	6.26	5.64	**0.48**
475	6.43	6.25	**0.52**
500	6.80	6.14	**0.47**



Poređenje vremena izvršavanja

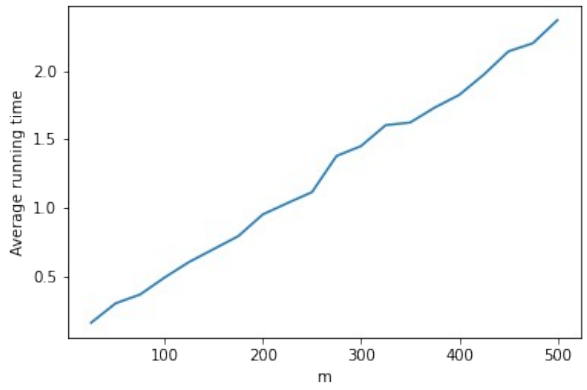
n	GA	ACO	PTAS
25	17	**22**	10
50	15	**17**	11
75	12	**15**	8
100	10	**15**	7
125	**10**	**10**	4
175	**10**	**10**	4
200	9	**10**	2
225	9	**10**	2
250	9	**10**	3
275	8	**10**	2
300	6	**10**	4
325	5	**9**	1
350	**7**	**7**	5
375	**5**	**5**	2
400	**6**	5	1
425	5	**6**	0
450	**5**	**5**	0
475	5	**6**	1
500	5	**7**	0



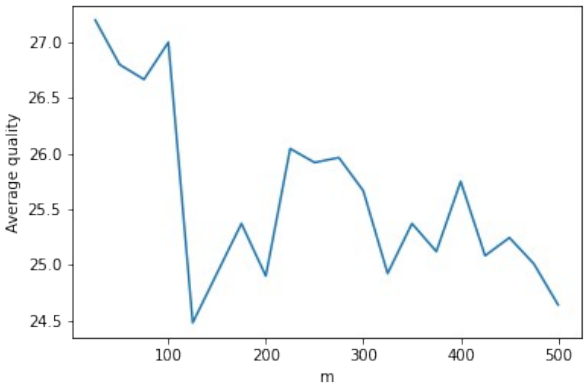
Poređenje kvaliteta rešenja

Povećavanje dužine niski

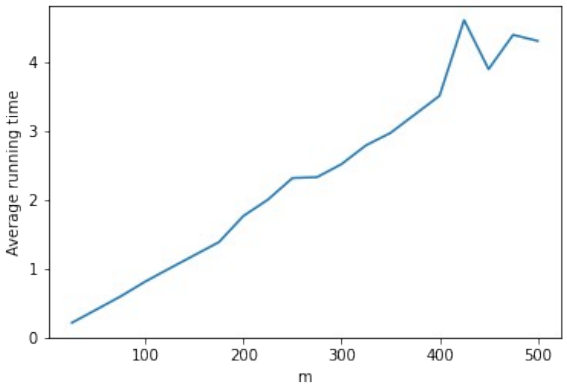
Azbuka je ponovo fiksirana na $\Sigma = \{A, C, T, G\}$. Broj niski fiksiran je na $n = 10$, dok m varira po vrednostima $m \in \{25, 50, 75, 100, \ldots, 500\}$. Metaheuristički rešavači su ponovo konfigurisani na isti način.



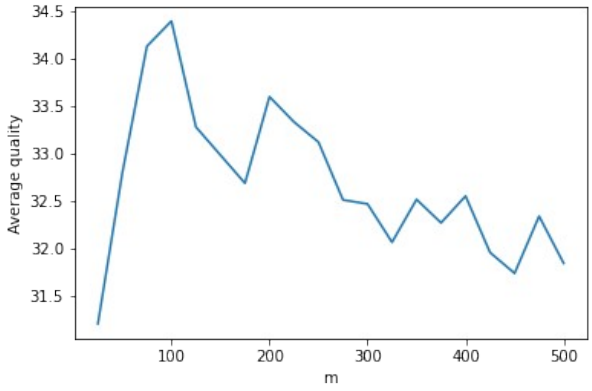
Genetski algoritam *– vreme izvršavanja*
Uočavamo linearan rast prosečnog vremena izvršavanja u odnosu na dužine ulaznih niski.



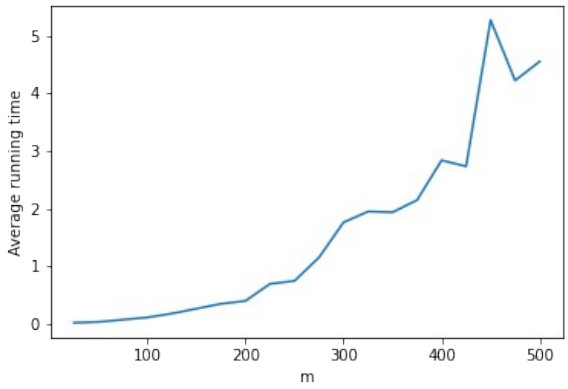
Genetski algoritam *– kvalitet rešenja*
Kvalitet rešenja ponaša se delimično haotično, ali sa opštim opadajućih trendom.



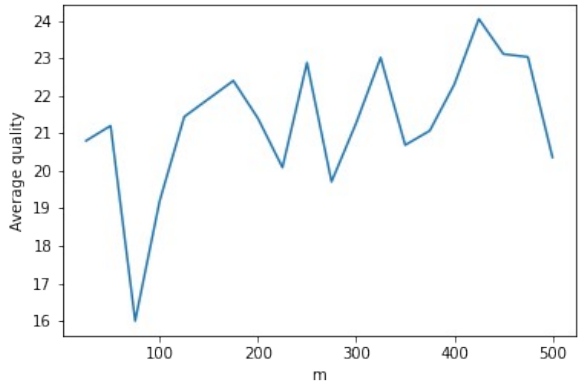
ACO *– vreme izvršavanja*
Uočavamo linearan rast prosečnog vremena izvršavanja u odnosu na dužinu ulaznih niski.



ACO *– kvalitet rešenja*
Slično kao u genetskom algoritmu. Za početni rast utvrđeno je da je slučajan ponovljenim testiranjem. Svakako nije od značaja u poređenju koje sledi.



***PTAS** – vreme izvršavanja*
Uočava se naizgled eksponencijalni rast.



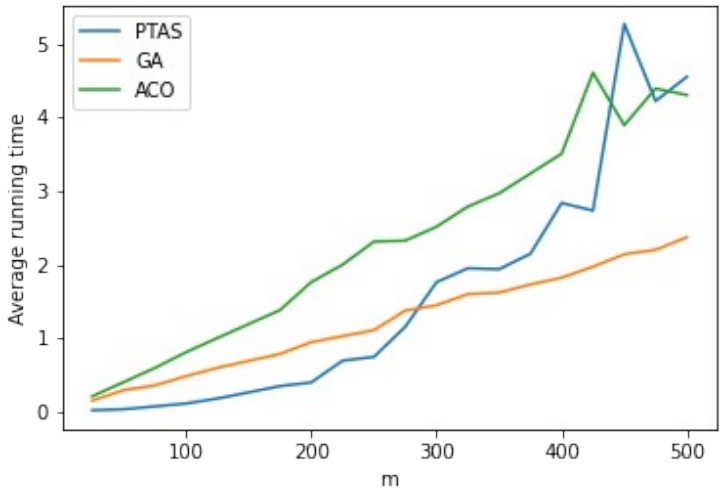
***PTAS** – kvalitet rešenja*
Kvalitet rešenja generalno visok, slično kao u genetskom algoritmu.

Poređenje rezultata

Iz ugla vremena, može se reći da se najbolje ponaša genetski algoritam. Za manje obime kraće vreme trošio je PTAS, ali nakon određenog trenutka genetski algoritam postaje bolji, što je očekivano zbog njegovog linearnog trenda.

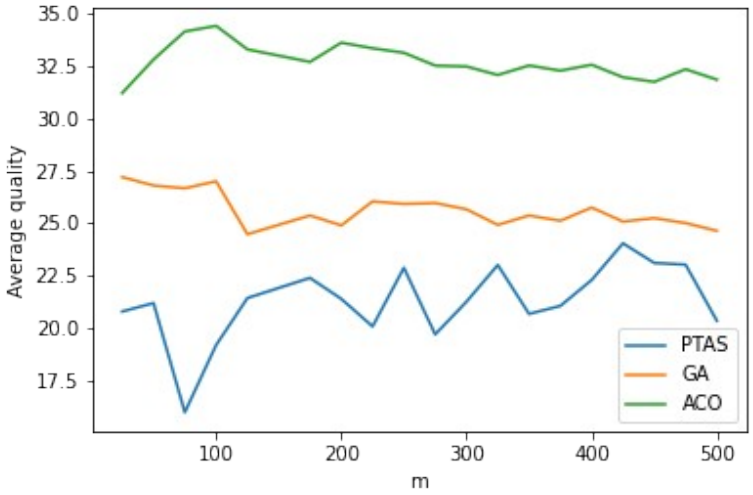
Iz ugla kvaliteta rešenja i dalje prednjači ACO. PTAS ponovo tehnički daje najlošija rešenja, ali ovaj put je vrlo blizu genetskog algoritma.

m	GA	ACO	PTAS
25	0.16	0.22	**0.03**
50	0.30	0.41	**0.04**
75	0.36	0.60	**0.08**
100	0.49	0.82	**0.12**
125	0.60	1.01	**0.19**
175	0.79	1.39	**0.35**
200	0.95	1.77	**0.40**
225	1.03	2.00	**0.70**
250	1.11	2.31	**0.75**
275	1.38	2.33	**1.16**
300	**1.45**	2.52	1.76
325	**1.60**	2.79	1.95
350	**1.62**	2.97	1.94
375	**1.73**	3.24	2.15
400	**1.83**	3.51	2.84
425	**1.97**	4.60	2.74
450	**2.14**	3.89	5.27
475	**2.20**	4.39	4.22
500	**2.37**	4.30	4.55



Poređenje vremena izvršavanja

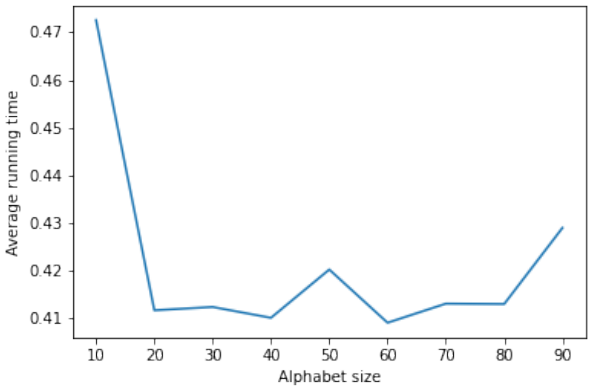
m	GA	ACO	PTAS
25	27.20	**31.20**	20.80
50	26.80	**32.80**	21.20
75	26.67	**34.13**	16.00
100	27.00	**34.40**	19.20
125	24.48	**33.28**	21.44
175	25.37	**32.69**	22.40
200	24.90	**33.60**	21.40
225	26.04	**33.33**	20.09
250	25.92	**33.12**	22.88
275	25.96	**32.51**	19.71
300	25.67	**32.47**	21.27
325	24.92	**32.06**	23.02
350	25.37	**32.51**	20.69
375	25.12	**32.27**	21.07
400	25.75	**32.55**	22.30
425	25.08	**31.95**	24.05
450	25.24	**31.73**	23.11
475	25.01	**32.34**	23.03
500	24.64	**31.84**	20.36



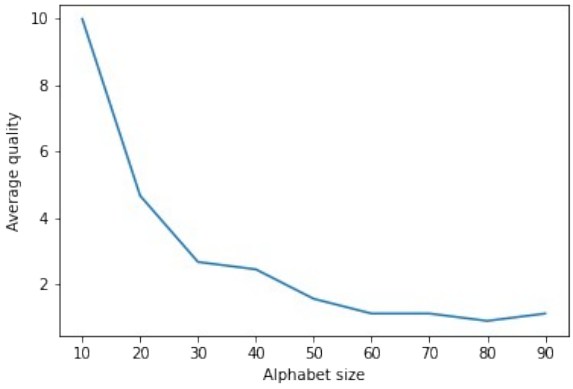
Poređenje kvaliteta rešenja

Povećavanje veličine azbuke

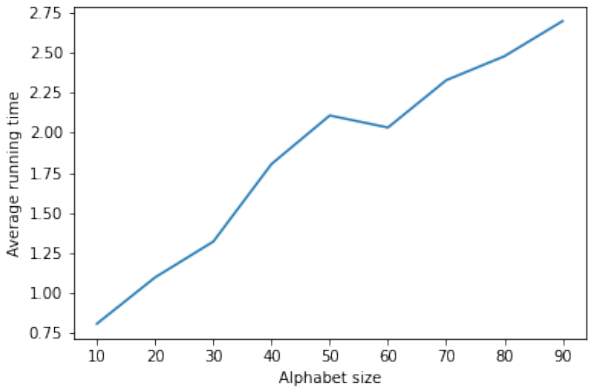
Broj niski fiksiran je na $n = 10$. Problem se javlja kod povećanja dužine azbuke jer usput moramo povećati i dužinu niski ako očekujemo pristojne rezultate. Razlog je to što ako je dužina niski mala a veličina azbuke velika, sami problemi će imati loša rešenja jer će niske biti vrlo međusobno različite. Iz tog razloga fiksiramo dužinu niski na $m = 90$ dok je veličina azbuke iz skupa $|\Sigma| \in \{10, 20, \ldots, 90\}$.



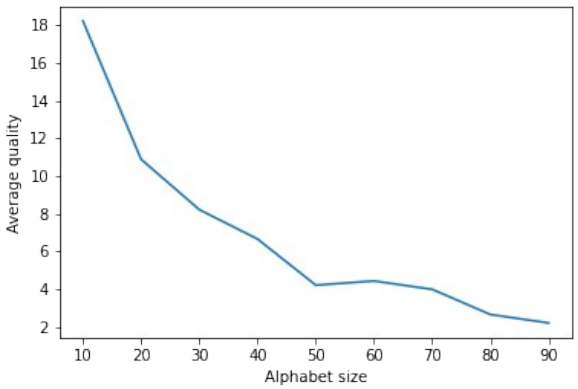
Genetski algoritam *– vreme izvršavanja*
Vreme izvršavanja ne zavisi od veličine azbuke.



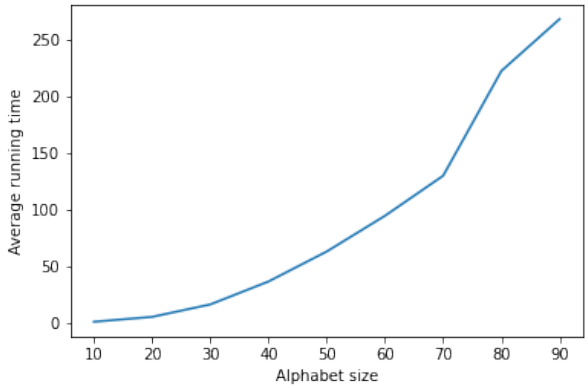
Genetski algoritam *– kvalitet rešenja*
Kvalitet opada s povećanjem veličine azbuke.



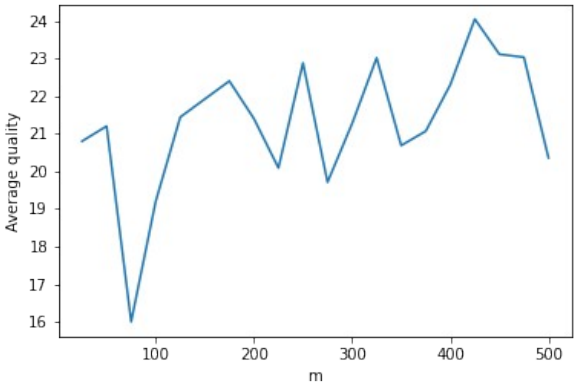
***ACO** – vreme izvršavanja*
Vreme linearno raste s povećanjem azbuke.



***ACO** – kvalitet rešenja*
Kvalitet opada s povećanjem azbuke.



***PTAS** – vreme izvršavanja*
Relativno stabilan rast, ali jako velike vrednosti.



***PTAS** – kvalitet rešenja*
Kvalitet relativno visok.

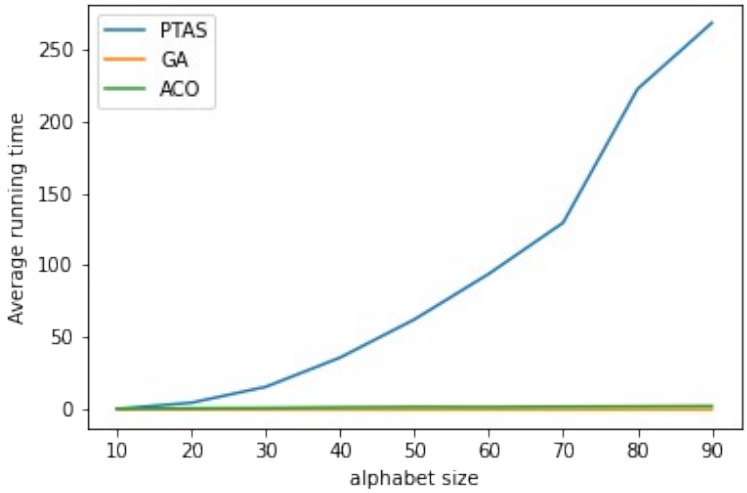
Poređenje rezultata

Prilikom testiranja rešavača na velikim azbukama dolazimo do relativno neuobičajenih rezultata u poređenju sa prethodnim analizama.

Vreme – Genetski algoritam nije osetljiv na veličinu azbuke iz razloga što on jedino koristi azbuku prilikom kreiranja početne populacije i prilikom mutacija, a to su sve brze operacije. ACO pokazuje linearan rast – porastom veličine azbuke povećava se matrica feromona koju koristi pa samim tim operacije nad matricom postaju skuplje. PTAS pokazuje jako loše vreme izvršavanja za velike ulaze. Razlog počiva u činjenici da je i $|\Sigma|$ i m veliko. U slučaju kada koristi iscrpnu pretragu potrebno je pretražiti (u najgorem slučaju) $|\Sigma|^m$ niski, dok ukoliko koristi linearno programiranje matrica može dostići dimenzije do $(m+n) \times (|\Sigma|m)$. U oba slučaja dobijaju se jako složeni problemi.

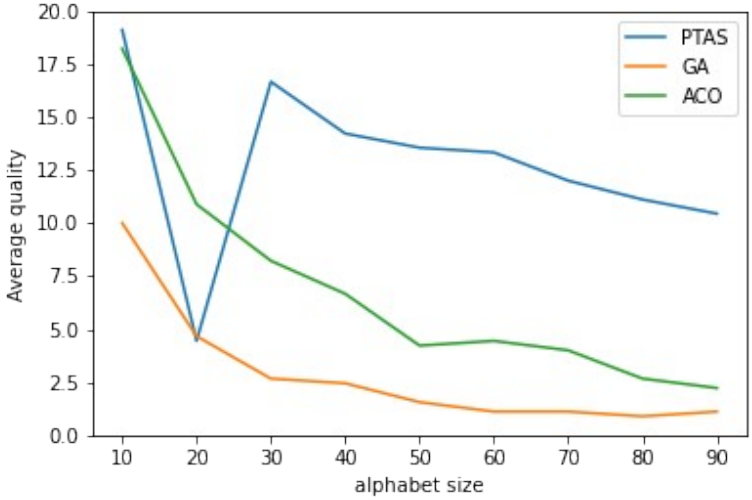
Kvalitet – Značajne razlike između genetskog algoritma i ACO ne postoje, dok ubedljivo rešenja najboljeg kvaliteta daje PTAS.

as	GA	ACO	PTAS
10	**0.47**	0.80	0.69
20	**0.41**	1.10	4.97
30	**0.41**	1.32	15.98
40	**0.41**	1.81	36.22
50	**0.42**	2.11	62.66
60	**0.41**	2.03	94.17
70	**0.41**	2.33	129.55
80	**0.41**	2.48	221.98
90	**0.43**	2.70	267.77



Poređenje vremena izvršavanja

as	GA	ACO	PTAS
10	10.00	18.22	**19.11**
20	4.67	**10.89**	4.44
30	2.67	8.22	**16.67**
40	2.44	6.67	**14.22**
50	1.56	4.22	**13.56**
60	1.11	4.44	**13.33**
70	1.11	4.00	**12.00**
80	0.89	2.67	**11.11**
90	1.11	2.22	**10.44**



Poređenje kvalitet rešenja

Zaključak

Algoritmi grube sile predstavljeni su radi kompletnosti, dok njihova praktična primena doseže samo do problema malih obima.

Polinomijalna aproksimacija koja je razmatrana, iako jednostavna, pokazuje lošije rezultate od ostalih rešavača u praksi. Naime, kako god konfigurisali algoritam, za velike vrednosti veličine azbuke i dužine niske dolazimo do jednog od dva problema:

- Ukoliko algoritam zaključi da treba koristiti iscrpnu pretragu moguća su dva slučaja:
    1. Veličina prostora pretrage je zaista mala – za veće probleme to se u praksi vrlo retko dešava.
    2. Veličina prostora pretrage je mala u odnosu na celokupan problem, ali i dalje je objektivno velika – u tom slučaju imamo problem sa vremenom izvršavanja.
- Ukoliko algoritam zaključi da treba koristiti LP relaksaciju može doći do problema sa brzinom izvršavanja. U implementaciji korišćena je biblioteka *scipy* za rešavanje LP problema. Brži komercijalni rešavači postoje, tako da postoji prostor za poboljšanje brzine. Međutim, čak i ako se rešenje brzo dobije, u nekim situacijama kvalitet LP rešenja jednostavno nije dobar.

Buduće analize bi uključivale testiranje ponašanja ostalih (složenijih) polinomijalnih aproksimacija koje ovde nisu obrađene. Jedna takva može se naći u [6], gde je ideja generalizovana tako da se problem redukuje dodatno na neki podskup niski početnog problema fiksne dužine.

Metaheuristički algoritmi su se pokazali kao najbolje rešenje. Njihova prednost u vidu vremena je predvidivost – najviše zavisi od same konfiguracije algoritma, dok je vremenski rast sa složenošću problema pretežno linearan. Rezultati potvrđuju deo zaključaka izvedenih u [1] – ACO se pokazao kao bolja opcija od genetskog algoritma.

Buduće analize uključivale bi detaljno testiranje rešavača sa raznim kombinacijama metaparametara. To je delimično i odrađeno, međutim značajne razlike nisu uočene, stoga nije pomenuto. [1] takođe razmatra mogućnost korišćenja algoritma *simuliranog kaljenja*, međutim zaključak je da se ponaša najlošije od ostale dve metaheurističke metode.

Kratak opis projekta

Prateći projekat nalazi se na repozitorijumu: https://github.com/aleksakojadinovic/RI-nearest-string

Projekat je pisan u *python*-u koristeći razvojno okruženje *PyCharm*. Od relevantnih biblioteka, ne računajući sistemske, korišćeni su:

- numpy
- pandas
- scipy
- tqdm

Struktura projekta

bps – *s*kraćeno od **b**enchmark **p**acket**s**. Sadrži direktorijume koji predstavljaju skupove problema za testiranje

docs – sadrži ovaj dokument u *pdf* i *odt* formatu

src – sav izvorni kod projekta

- *abstractions.py*
    - sadrži klase *CSPProblem*, *CSPSolution*, *AbstractSolver* za rad sa problemima, rešenjima i rešavačima na višem nivou. Svaki konkretan rešavač nasleđuje *AbstractSolver* i implementira apstraktnu metodu *solve_*
- *utils.py*
    - nesvrstane funkcije koje se koriste na mnogim mestima u projektu
- **generators**
    - *random_generator.py*
        - funkcije za generisanje nasumičnih problema za testiranje
- **solvers**
    - **exact**
        - *brute_force_search.py*
            - rešavač grubom silom
        - *pruning_search.py*
            - rešavač sa odsecanjem

    - **nature**
        - *ant.py*
            - ACO rešavač
        - *genetic.py*
            - rešavač genetskim algoritmom
    - **ptas**
        - *simple_ptas.py*
            - PTAS korišćen u radu
- **stats**
    - *experimental.py*
            - funkcije za testiranje

Izostavljeni su fajlovi koji nisu korišćeni u finalnoj analizi.

Literatura

[1] Faro, Simone & Pappalardo, Elisa. (2010). *Ant-CSP: An Ant Colony Optimization Algorithm for the Closest String Problem*. 370-381. 10.1007/978-3-642-11266-9_31.

[2] Bahredar, Faranak et al. *"A Meta Heuristic Solution for Closest String Problem Using Ant Colony System."* DCAI (2010).

[3] Xuan Liu, Hongmei He, and Ondrej Sýkora. 2005. *Parallel genetic algorithm and parallel simulated annealing algorithm for the closest string problem*. Springer-Verlag, Berlin, Heidelberg, 591–597. DOI: https://doi.org/10.1007/11527503_70

[4] J. Kevin Lanctot, Ming Li, Bin Ma, Shaojiu Wang, Louxin Zhang. *Distinguishing string selection problems*, Information and Computation, Volume 185, Issue 1, 2003, ISSN 0890-5401

[5] Li, Ming & ma, Bin & Wang, Lusheng. (2000). *Finding Similar Regions In Many Strings*. Conference Proceedings of the Annual ACM Symposium on Theory of Computing. 63. 10.1145/301250.301376.

[6] Li, Ming & ma, Bin & Wang, Lusheng. (2002). *On The Closest String and Substring Problems*. Journal of the ACM. 49. 10.1145/506147.506150.