NASLOVNA STRANA

– skloni numeraciju s naslovne i sledece

Table of Contents

Uvod....3
Rešavanje problema....4
  Rešenja grubom silom....4
    Iscrpna pretraga....4
    Pretraga sa odsecanjem....5
    Poređenje iscrpne pretrage i pretrage sa odsecanjem....6
  Metaheuristička rešenja....8
    Genetski algoritam....8
    Optimizacija kolonijom mrava....9
  Polinomijalne aproksimacije....10
Eksperimentalni rezultati....11
Zaključak....12

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 $d = \max\{d_H(s, s_i) | i = 1, \ldots n\}$. 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 iz [?] [?]. Na kraju, razmatramo dve polinomijalne aproksimacije date u [?] [?].

Rešavanje problema

Pre prikazivanja rešenja uvodimo neke uobičajene notacije.

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 $I = [j_1, j_2, \dots j_r] \subseteq [1, 2, \dots, m]$ 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 kakave dodatne optimizacije. Zbog toga u rešenju koristimo DFS pretragu 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]
```

// vidi moze li kod nekako bolje kod da se stavi u tekst

Računanje vrednosti funkcije cilja zahteva poređenje na m pozicija za svaku od n niski, tako da je složenosti mn. Na nivoima $1, 2, \ldots, m-1$ ne računamo funkciju cilja. Čvorova u tim nivoima ima $\frac{1-|\Sigma|^m}{1-|\Sigma|}$, i u svakom prolazimo kroz azbuku tako da složenost završno sa pretposlednjim nivoom je $\frac{1-|\Sigma|^m}{1-|\Sigma|}|\Sigma|$. Na poslednjem nivou, koji se sastoji od tačno $|\Sigma|^m$ čvorova, računamo funkciju cilja, tako da je složenost na tom nivou $|\Sigma|^m mn$. Ukupna složenost algoritma predstavljala bi zbir prethodna dva dela:

$$\frac{1-|\Sigma|^m}{1-|\Sigma|}|\Sigma| + |\Sigma|^m mn$$

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 su s_1 i s_2 niske dužina d_1 i d_2, gde je $d_1 > d_2$ i s_2 je prefiks od s_1. Tada važi:

$$d_H(s_{|\{1,\ldots d_1\}}, s_1) \geq d_H(s_{|\{1,\ldots,d_2\}}, s_2).$$

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:

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

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

U ovome se ogleda razlog zašto je korišćen baš DFS a ne 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 će najbolja dosadašnja vrednost uvek 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 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 [] i [], 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 [] 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 $m - f(p_i)$ 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 uobičajeno niska (5%) obezbeđuje se da ukoliko dolazi do mutacije - da do nje zapravo dodje; odnosno da ne može da dođe do situacije da 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 [] [] []. U eksperimentalnim rezultatima kasnije pokazaće se kao vrlo uspešna.

U opštem slučaju, ACO koristi se 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 *lokalnog optimalnog rešenja*. 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 $m \times |\Sigma|$. Polje matrice na koordinatama j, α gde je $j \in \{1, \ldots, m\}, \alpha \in \Sigma$ pokazuje 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, ali naravno to nije garantovano.

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 parametra $\rho \in (0, 1)$ 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

Eksperimentalni rezultati

Zaključak