**Shortest common superstring**

Seminarski projekat iz predmeta Računarska inteligencija  
Matematički fakultet  
Univerzitet u Beogradu

Veljko Kučinar  
[mi18144@alas.matf.bg.ac.rs](mailto:mi18144@alas.matf.bg.ac.rs)

**Septembar 2023.**

**Sadržaj**

1. [Uvod](#Uvod)
   1. [Opis problema](#OpisProblema)
   2. [Primer problema](#PrimerProblema)
2. [Pristupi rešavanja problema](#PristupiResavanjaProblema)
   1. [Brute force – DP](#DP)
   2. [Greedy algorithm](#Greedy)
   3. [S metaheuristics – LS, SA, VNS](#sMetaheuristike)
   4. [Genetic algorithms](#GA)
3. [Poređenje svih pristupa](#PoredjenjeSvihPristupa)
4. [Zaključak](#Zakljucak)
5. [Literatura](#Literatura)

**1.** **Uvod**

**1.1.** **Opis problema**

Problem Shortest common superstring (u daljem tekstu SCS) je NP-težak problem pronalaženja najkraće niske koja sadrži sve date niske.   
  
Formalno zapisano:  
Data je konačna abeceda i konačan skup stringova koji pripada abecedi . Treba pronaći najkraću nisku takvu da je svaka niska podniska od .

**1.2.** **Primer problema**

Neka su date niske *“AAB”, “BAA”, “ABA”, “BAB”*. Prvo možemo uočiti da je trivijalno naći bilo kakvu nisku koja sadrži sve date niske. Jedna takva niska se može dobiti prostim “lepljenjem” svih datih niski jednu uz drugu:

*“AABBAAABABAB”* (*AAB*+*BAA*+*ABA*+*BAB*)

Međutim, u problemu SCS ne tražimo bilo kakvu nisku koja sadrži sve date niske, već tražimo najkraću takvu nisku.  
Detaljnom analizom možemo ručno dobiti da je jedno (ne mora nužno biti jedinstveno) rešenje ovog primera niska:

*“BAABAB”*

Pogledajmo sada nešto složeniji primer – date su niske *“bloa”, “bubl”, “gabl”, “abpo”, “ublm”*. U ovom slučaju do rešenja je znatno zahtevnije doći ručno kao pre. Korišćenjem jednog od algoritama dobija se rezultat:

*“bublmgabloabpo”*

Koji nije bio baš očigledan kao u prvom primeru.

U daljem tekstu biće obrađeni upravo ti algoritmi i pristupi raznih složenosti koji imaju za cilj rešavanje ili aproksimaciju rešenja problema SCS.

**2.** **Pristupi rešavanja problema**

**2.1.** **Brute force – DP**

**Ideja:** Problem SCS je moguce svesti na dinamičko programiranje i na određeni potproblem problema Trgovačkog putnika – TSP (Travelling Salesman Problem).  
Za sada je ovo najefikasniji algoritam koji garantovano nalazi rešenje problema SCS ukoliko mu se da dovoljno vremena i resursa. Mana algoritma je to što je veoma asimptotski kompleksan (reda složenosti ).

**Performanse:** Vremenska složenost ovog algoritma je gde je broj datih niski u nizu. Za merenje performansi algoritma korišćeni su najjednostavniji primeri jer u složenijim primerima dolazi do “pucanja” programa usled velike prostorne složenosti, koja je takođe .

**2.2. Greedy algorithm**

**Ideja:** Kao prvi aproksimativan pristup razmatrajmo dosta jednostavniji pohlepni algoritam.  
Pristup se sastoji u tome da od datog skupa niski biramo dve koje imaju najveće poklapanje. Sada za te dve niske tražimo SCS, što je trivijalno. Rešenje će biti nova niska koja će zameniti prethodno razmatrane dve niske. Algoritam ne garantuje pronalaženje optimalnog rešenja i tačnost rešenja zavisi od redosleda niski u datom nizu, a dodatne provere bi narušile jednostavnost i složenost algoritma. Dobra strana je što će algoritam za neke ulaze srednje veličine davati prilično dobru aproksimaciju rešenja.

**Performanse:** Složenost ovog algoritma je gde je broj datih niski u nizu a je maksimalna dužina pojedinačne niske u nizu.

**2.3. S metaheurustics – LS, SA, VNS**

**Ideja:** Pomoću S metaheuristika (LS, SA, VNS – Local search, Simulated annealing, Variable neighborhood search) naći bar nekakvo rešenje vrlo brzo. Osnovna ideja u sva tri pristupa je da je k-ti “sused” našeg niza niz u kome je izvršeno k swap-ova elemenata. Rešenje SCS se dobija lepljenjem susednih niski u nizu bez preklapanja – ovo će biti detaljnije opisano u sledećem poglavlju.  
Ove heuristike nisu baš pogodne za rešavanje SCS problema ali su svakako uzete u obzir čisto radi merenja performansi i kao moguća nadogradnja genetskog algoritma.

**Performanse:** Sve tri metaheuristike izbacuju rešenja u vrlo kratkom vremenu – ali ta rešenja nisu baš nešto mnogo značajna jer se sva tri pristupa vrlo brzo zaglave na nekom ekstremumu. Broj iteracija je postavljen na 100 jer veći broj iteracija ne utiče mnogo na kvalitet rešenja.

**2.4.** **Genetic algorithm**

**Ideja:** Kao jedinku za genetski algoritam možemo razmatrati jednu permutaciju niza polaznih niski.  
Prilagođenost (“fitness”) jedinke bi bila dužina niske koju bismo dobili malo jednostavnijim pristupom koji je sličan pohlepnom algoritmu – “lepljenjem” susednih niski u nizu bez preklapanja. Za razliku od pohlepnog algoritma, ovde bismo za svaki uzastopni par niski odredili najveći sufiks niske koji je isti kao prefiks niske . Kada su takvi sufiks i prefiks nađeni onda formiramo nisku koja ne sadrži nađeni prefiks. Konačno, spajamo te dve niske u i nastavljamo proces dok ne dođemo do kraja niza. Najbolju prilagođenost će imati jedinke koje imaju najkraću nisku .  
Jedinke se biraju pomoću turnirske selekcije. Razmatrana je i ruletska selekcija, ali je ona u svim mogućim testiranjima dovodila do lošijih rezultata.  
Ukrštanje se vrši pomoću ukrštanja prvog reda.  
Mutacija se svodi na swap između dve niske u nizu koji pripada jedinci. Razmatrana je i mutacija u kojoj bi se odabrala 2 indeksa u nizu i onda bi se sve niske unutar ta dva indeksa izmešale. Ovakav pristup je, u najčešćem slučaju, veoma loše uticao na prilagođenost jedinki pa je konačna odluka bila da se koristi mutacija bazirana na swap-u.  
Na kraju svake iteracije se pamti određen broj najboljih jedinki – elitizam.  
Parametri za veličinu populacije, turnira, mutaciju, elitizam i broj iteracija su odabrani po uzoru na [[2](#Dva)] uz blage izmene do kojih je došlo eksperimentacijom.

**Performanse:** Algoritam je uglavnom namenjen kao aproksimacija većih primera i ideja je da se ti veći primeri izvrše u donekle prihvatljivom vremenu. Genetski algoritam daje malo lošija rešenja od pohlepnog, ali će za veoma velike instance problema davati bar nekakvu aproksimaciju posle par minuta, dok bi se pohlepni algoritam izvršavao nekoliko sati.

**3.** **Poređenje svih pristupa**

Pre nego što pogledamo tabelu sa rešenjima posvetićemo malo pažnju primerima koji su generisani radi testiranja algoritama. U tabeli ispod |N| označava broj niski u datom nizu, |s| označava dužinu jednog elementa niza, tj. niske, s je prikaz jedne nasumične niske koja pripada nizu niza.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | |N| | |s| | | s |  |  |
| Primer 1 | 4 | 3 | AAB | |  |  |
| Primer 2 | 5 | 4 | bloa | |  |  |
| Primer 3 | 20 | 4 | wobj | |  |  |
| Primer 4 | 200 | 8 | upzishtr | |  |  |
| Primer 5 | 300 | 32 | sxiippjlmtopzleipjnbwptxszcdlulh | |  |  |
| Primer 6 | 500 | 10 | tgtcctgata | |  |  |

Primer 6 je veoma važan za primenu algoritama jer se rešenje SCS problema najčešće traži u sekvencama DNK, a u ovom primeru su generisane sekvence koje podsećaju na nasumične sekvence DNK (jedina dozvoljena slova su “t”, “g”, “c” i “a”).  
Manji primeri su pravljeni ručno, a veći primeri su generisani pomoću <https://www.random.org/strings/>. Za generisanje nasumičnih DNK sekvenci je korišćen <https://birc.au.dk/~palle/php/fabox/random_sequence_generator.php>.

Konačno, za gore pomenute primere možemo izmeriti rad algoritama i uporediti njihova rešenja:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Brute force | Greedy | LS avg | SA avg | VNS avg | Genetic avg |
| Primer 1 | 6 | 6 | 6 | 6 | 6 | 6 |
| Primer 2 | 14 | 14 | 14.3 | 14.3 | 14 | 14.3 |
| Primer 3 | 76 | 77 | 78 | 78 | 77 | 76.2 |
| Primer 4 | N/A | 1401 | 1589.7 | 1588 | 1583.3 | 1488 |
| Primer 5 | N/A | 9231 | 9584.3 | 9586 | 9579.7 | 9439.3 |
| Primer 6 | N/A | 2540 | 4642.3 | 4627 | 4616.3 | 4048.8 |

Najpre možemo videti da algoritam grube sile nije mogao da dođe do rešenja u većim primerima – razmatran je i primer koji sadrži niz od 25 niski dužine 4, u njemu algoritam grube sile ne bi izbacivao rešenje zbog svoje velike prostorne složenosti.  
Algoritmi bazirani na S metaheuristikama su davali rešenja koja se zaglave na nekakvom ekstremumu, ali se do tih rešenja dolazilo veoma brzo.  
Genetski algoritam se uglavnom dobro snalazio u malim primerima i često izbacivao tačna rešenja. Kod srednjih i velikih primera su rešenja bila lošijeg kvaliteta od pohlepnog, ali ta rešenja nikad ne bi bila više od 7% lošija osim u poslednjem primeru.

Obratimo pažnju na poslednji primer – primer 6. U njemu dolazi do mnogo većeg broja preklapanja između niski u odnosu na ostale primere. Tu se može videti da pohlepni algoritam izbacuje daleko bolja rešenja od svih ostalih algoritama, ali je ujedno i najsporiji.

Pogledajmo brzine rada algoritama na istim primerima:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Brute force | Greedy | LS avg | SA avg | VNS avg | Genetic avg |
| Primer 1 | <0.0001 | <0.0001 | 0.0004 | 0.0004 | 0.0025 | 0.167 |
| Primer 2 | 0.0001 | <0.0001 | 0.0005 | 0.0005 | 0.0031 | 0.18 |
| Primer 3 | 38.921 | 0.002 | 0.0017 | 0.0019 | 0.0083 | 0.369 |
| Primer 4 | N/A | 2.692 | 0.025 | 0.025 | 0.081 | 5.21 |
| Primer 5 | N/A | 27.69 | 0.096 | 0.093 | 0.297 | 14.162 |
| Primer 6 | N/A | 51.472 | 0.07 | 0.066 | 0.261 | 21.734 |

Algoritam grube sile već počinje da troši dosta vremena kod trećeg primera, dok ostali algoritmi dolaze do skoro optimalnog rešenja za manje od jedne sekunde.  
S metaheuristike i za najveće primere dolaze do rešenja vrlo brzo – to im je bila i svrha s obzirom na to da nisu baš najpogodnije metode za rešavanje ovog problema.  
Pohlepni algoritam male i srednje velike primere rešava veoma brzo, ali vreme izvršavanja počinje jako brzo da raste uvođenjem većih primera.  
Genetski algoritam najjednostavnije primere izvršava malo duže nego ostali algoritmi, ali su i dalje u pitanju vremena ispod jedne sekunde. Kod srednjih i velikih primera vreme izvršavanja raste, ali ne toliko naglo kao kod pohlepnog algoritma.

Za kraj, uporedimo malo detaljnije rešenja pohlepnog algoritma sa genetskim algoritmom:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Greedy | Genetic avg | Genetic best | Genetic worst |
| Primer 1 | 6 | 6 | 6 | 6 |
| Primer 2 | 14 | 14.3 | 14 | 16 |
| Primer 3 | 77 | 76.2 | 76 | 77 |
| Primer 4 | 1401 | 1488 | 1483 | 1492 |
| Primer 5 | 9231 | 9439.3 | 9434 | 9447 |
| Primer 6 | 2540 | 4048.8 | 4003 | 4089 |

Vidimo da genetski algoritam u malim i srednje malim primerima u najboljem slučaju može doći i do optimalnog rešenja, dok je za veće primere bolja rešenja izbacivao pohlepni algoritam.   
Prednost genetskog algoritma u odnosu na pohlepni je to što brže dolazi do neke svoje aproksimacije kod velikih primera.  
Zanimljivo je da u veoma velikom primeru (5000 DNK sekvenci dužine 50) genetski algoritam uspeva da završi sa radom nakon skoro 30 minuta, dok pohlepni algoritam nije uspeo da dođe do rešenja ni nakon više od 8 sati rada. Kvalitet dobijenog rešenja, nažalost, nije bilo moguće uporediti ni sa kojim algoritmom.

**4.** **Zaključak**

Od svih razmatranih algoritama može se uočiti da pohlepni algoritam izbacuje najbolja rešenja za srednje velike i velike probleme. Jedina mana mu je što je dosta sporiji od ostalih pristupa kako povećavamo veličinu primera.

S metaheuristike kao jedinu vrlinu imaju ekstremno brzo vreme izvršavanja – ono proizilazi iz toga da im nije neophodan preveliki broj iteracija jer se algoritmi vrlo lako zaglavljuju na nekom ekstremumu.

Genetski algoritam se najbolje pokazao u odnosu na pohlepni algoritam – za male i srednje primere je izbacivao skoro optimalna rešenja, dok je kod većih primera davao rešenja koja su do najviše 7% lošija od pohlepnog algoritma ukoliko u primeru nema mnogo preklapanja. Dodatna prednost genetskog algoritma je što za veće primere skalira daleko bolje od pohlepnog algoritma i završava sa radom posle desetak minuta, dok bi pohlepnom algoritmu trebalo nekoliko sati.

Iz ovih rezultata može se zaključiti da bi za neke određene probleme koji nisu prevelike dužine bilo moguće kombinovati sva tri pristupa u jedan:

1. Koristiti algoritam pohlepne pretrage da bi se našla najbolja aproksimacija rešenja i kako bi se našao niz koji formira to rešenje  
2. Dobijeni niz provući kroz neku od S metaheuristika radi dobijanja potencijalnog boljeg rešenja i boljeg rasporeda elemenata u nizu (ovaj korak jedini nije nužno raditi jer je vrlo mala verovatnoća da će se zapravo dobiti bolje rešenje od onog koji daje pohlepan algoritam)  
3. Konačno, novodobijeni niz koristiti kao “seed” za originalni genetski algoritam ili za potencijalno modifikovani genetski algoritam koji će prioritizovati ukrštanje ostalih jedinki sa “seed” jedinkom u ranijim iteracijama (dodatno, moguće je ubaciti i neki algoritam iz S metaheuristika da “popravlja” jedinke s vremena na vreme, ali eksperimentalno se to pokazalo kao neefikasan pristup)

Vreme izvršavanja ovakvog kombinovanog pristupa za dati primer bi bilo manje ili jednako dvostrukom vremenu izvršavanja pohlepnog algoritma za isti taj primer.

Ovaj zaključak je inspirisan naučnim radom [[2](#Dva)] – ključna razlika je u tome što je ovde opisani genetski algoritam vremenski manje zahtevan od pohlepnog i daje malo lošija rešenja, dok je u pomenutoj literaturi genetski algoritam više vremenski zahtevan od pohlepnog algoritma ali se oslanja na neke druge pristupe koji omogućavaju tačnija rešenja.

**5. Literatura**

[1] Jonathan S. Turner, „Approximation Algorithms for the Shortest Common Superstring Problem“, <https://www.sciencedirect.com/science/article/pii/0890540189900448>, 1989.  
[2] Tyler Giallanza, „Novel Applications of Stochastic Global Optimization Algorithms to the Shortest Common Superstring Problem“, <https://www.nshss.org/media/29819/giallanza.pdf>, 2016.  
[3] Xuan Liu, Ondrej Sýkora, „Sequential and Parallel Algorithms for the Shortest Common Superstring Problem“,<https://citeseerx.ist.psu.edu/document?doi=d406c2ac168e4af40e8380b85d28fec5de6959e9>, 2005.