# Shortest common superstring

Veljko Kučinar, 144/2018

# Opis problema

Problem Shortest common superstring (u daljem tekstu SCS) je NP-težak problem pronalaženja najkraće niske koja sadrži sve niske iz nekog datog niza niski.
Generisanje bilo kakve niske koja sadrži sve date niske je trivijalno - možemo samo "zalepiti" sve niske iz niza jednu uz drugu i dobiti nisku koja ih sadrži, problem postaje NP-težak tek kada tražimo **najkraću** takvu nisku.

Primer 1:
SCS("AAB", "BAA", "ABA", "BAB") = "BAABAB"
Ovde nije toliko teško naći rešenje problema SCS ručno.

Primer 2:
SCS("bloa", "bubl", "gabl", "abpo") = "bublmgabloabpo"
Već u ovom primeru je malo teže naći rešenje ručno, a primer nije mnogo teži od prethodnog.

Algoritmi koji su razmatrani za ovaj problem: algoritam grube sile (DP), pohlepni algoritam, S metaheuristički algoritmi (LS, SA, VNS) i genetski algoritam.

# Algoritam grube sile (DP)

SCS možemo posmatrati kao podvrstu problema trgovačkog putnika - TSP (Travelling Salesman Problem).
Najefikasniji optimalni pristup bi se sveo na rešavanje modelovanog TSP problema pomoću dinamičkog programiranja (DP) gde bi se u tabeli pamtili i ažurirali odnosi svake niske sa svakom – što je neprihvatljivo zahtevno već za neke srednje-lake primere.

Algoritam garantuje optimalnost, ali već za srednje-lake primere ne može da dođe do rešenja zbog svoje ogromne prostorne i vremenske složenosti.

Složenost: $O(2^n * n^2)$ gde je $n$ broj datih niski u nizu.

# Pohlepni algoritam

Pseudokod:
1. Definisati niz *S* koji sadrži *n* niski *s*
2. Pronaći dve niske $s_i$ i $s_j$ koje imaju najveće poklapanje sufiksa i prefiksa i spojiti te dve niske u jednu, smanjujući veličinu polaznog niza *S* za 1
3. Ponavljati 2. sve dok ne dobijemo jednu nisku – ta niska će biti rešenje tj. pronađeni SCS

Primer 3:
Greedy("GGGH", "HGHH", "GHHG") = Greedy("GGGH", "HGHHG") = Greedy("GGGHGHHG")=
= "GGGHGHHG"

Ovaj algoritam je dosta vremenski i memorijski efikasniji od algoritma grube sile.
Mana mu je što nije optimalan – možemo to videti u jednostavnim naštimovanim primerima (algoritam za niz ["bp", "pb", "np"] daje nisku "bpbnp" dužine 5, a optimalno rešenje je niska dužine 4, "npbp").

Složenost: $O(n^3*m)$ – *n* je broj datih niski u nizu a *m* je maksimalna dužina jedne od niski iz niza.

# S metaheurustike – LS, SA, VNS

Kratko su razmatrana tri algoritma S metaheuristike – algoritam lokalne pretrage (LS), simulirano kaljenje (SA) i VNS.

Oni nisu baš pogodni za rešavanje ovog problema ali bar mogu ponuditi nekakvo rešenje za ekstremno brzo vreme (za manje od jedne sekunde). To znači da bi njihova potencijalna uloga mogla biti kao neka vrsta brze optimizacije nekog rešenja koji je dobijen drugim algoritmom.

Slično kao u pohlepnom algoritmu – gledaćemo susedne elemente i njihova preklapanja. Rešenje će biti raspored niski u nizu koje lepljenjem suseda daju najkraću nađenu nisku.
Broj iteracija algoritma je stavljen eksperimentalno na 100 jer se ne dobijaju bolja rešenja sa većim brojem iteracija, a znatno se utiče na njihovo vreme izvršavanja.

# Genetski algoritam

Jedinka – jedna nasumična permutacija polaznog niza.
Fitness – dužina niske koja se dobija spajanjem svih susednih elemenata uz preklapanje.
Selection – turnirska selekcija.
Mutation – swap dva elementa jedinke tj. dveju niski.
Crossover – ukrštanje prvog reda.
Parametri algoritma su odabrani po uzoru na [2] uz male eksperimentalne modifikacije.

Primer 5:
Za niz ["AAB", "BAA", "ABA", "BAB"] jedinka bi npr. bila ["BAB", "BAA", "ABA","AAB", ] , fitness ove jedinke bi bio 9 (duzina niske "BABAABAAB"), ukoliko bi data jedinka trebalo da prođe kroz mutaciju, onda bi se nasumično odabrale 2 pozicije u nizu i izvršio swap elemenata:
Mut(["BAB", "BAA", "ABA","AAB"]) = ["ABA" , "BAA", "BAB","AAB"].

Algoritam kod malih i srednje velikih primera vrlo često dolazi do optimalnog rešenja, kod srednje složenih primera izbacuje malo lošije rešenje blisko optimalnom, a kod većih primera pravi veće greške, ali te greške ne odstupaju više od 7% u poređenju sa pohlepnim algoritmom za isti primer. Genetski algoritam za jako velike primere mnogo brže radi od pohlepnog.

# Poređenje pristupa

| REZULTATI | 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 |

| VREME | 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 |

# Zaključak

Pohlepni algoritam daje daleko najbolje rezultate kada imamo veliki broj preklapanja u datom nizu niski. Kada nema mnogo preklapanja i dalje je najbolji ali ne i ubedljivo najbolji.

S metaheuristike nisu baš pogodne kao pristup za rešavanje problema SCS, ali potencijalno mogu da posluže kao efikasan način da se malo poboljša već dobijeno rešenje.

Genetski algoritam radi malo lošije u poređenju sa pohlepnim algoritmom ako nema prevelikog broja preklapanja između niski – kod primera sa manjim brojem preklapanja genetski algoritam dobija rešenja koja su do 7% lošija od rešenja pohlepnog algoritma.
Dodatno, genetski algoritam daleko brže rešava veoma velike probleme od pohlepnog algoritma – za jedan isti veliki problem genetski algoritam je izbacio rešenje za malo manje od 30 minuta dok pohlepni algoritam ni nakon 8 sati rada nije uspeo da izbaci rešenje.

Za neke srednje velike i velike primere vrlo je moguće koristiti pohlepni algoritam za "seed-ovanje" genetskog algoritma uz dodatno povremeno ubacivanje nekog od algoritama S metaheuristike radi mogućeg poboljšanja rešenja.

# 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.