Minimalno sortiranje inverzijom intervala (Minimum sorting by reversals)

Naučni rad u okviru kursa
Računarska inteligencija
Matematički fakultet
Univerzitet u Beogradu

Arsenijević Vladimir 39/2017
Vujčić Luka 63/2017

Uvod

Motivacija

Problem preuređenja genoma je jedan od osnovnih problema molekularne biologije.

Preuređenje genoma predstavlja njihovo pomeranje ili kopiranje na neku drugu lokaciju u okviru istog ili drugog hromozoma.

Jedno od najznačanijih preuređenja genoma je inverzija .

Pregled različitih varijanata problema

Problem dolazi u dve varijante: **orijentisane** i **neorijentisane** inverzije.

Ispostavlja se da je problem orijentisanih inverzija značajno lakši od problema neorijentisanih inverzija.

Problem orijentisanih inverzija je rešiv u polinomskom vremenu.

Problem neorijentisanih inverzija je **NP težak** problem.

U daljem nastavku, bavimo se isključivo neorijentisanim slučajem.

Formulacija neorijentisanog slučaja

Postavka:

Data je permutacija elemenata od 1 do n. Potrebno je permutaciju dozvoljenim operacijama dovesti do permutacije identiteta (permutacija sortirana rastuće), tako da broj operacija bude minimalan.

Dozvoljene operacije su operacije invertovanja intervala [i,j], 1<=i,j<=n.

Primer:

Ulazni niz: [1,4,2,3]

Koraci:

[1,4,2,3] → inverujemo [2,3]
[1,2,4,3] → inverujemo [3,4]
[1,2,3,4]

Dobili smo permutaciju identiteta , koristeći dve odgovarajuće operacije.

Možemo se uveriti da je ovo minimalan broj operacija.

Opis rešenja

Za rešavanje ovog problema ovde koristimo genetske algoritme sa promenljivom dužinom jedinke.

Kodiranje rešenja

Kako je potrebno proivoljnu permutaciju dovesti do identične permutacije, potreban nam je niz koraka.

Taj niz koraka je sekvenca operacija inverzije intervala koji invertujemo.

Naš hromozom je oblika: $[a_1,b_1,a_2,b_2,...,a_n,b_n]$.

Ovakav hromozom kodira inverzije: $[a_1,b_1],[a_2,b_2],...,[a_n,b_n]$

Kako broj potrebnih operacija nije fiksiran i poznat za sve ulaze unapred (ne može ni da bude poznat unapred, jer upravo minimalan broj operacija želimo da izračunamo) iz ovog razloga koristimo promenljive dužine hromozoma

Komentar: Primetimo da je hromozom uvek parne dužine

Fitnes funkcija

Da bismo uveli fitnes funkciju najpre uvedimo prelomne tačke (breakpoints).

Neka je niz $a=[a_1,a_2,...,a_n]$.

Kažemo da između a_i i a_j postoji prelomna tačka ako važi $|a_i - a_j|>1$.

Broj prelomnih tačaka niza jedak je sumi svih prelomnih tačaka između susednih članova niza.

Kako utvrđujemo koliko je naš hromozom dobar?

Naš hromozom je dobar onoliko koliko je dobra permutacija koja se dobije kad se hromozom primeni na početnu permutaciju. Tako dobijenu permutaciju u daljem tekstu zovemo završna permutacija.

Napomena: Završna permutacija je u potpunosti određena početnim nizom i hromozomom

Takođe želimo da imamo što kraći hromozom.

Tako da fitnes našeg hromozoma definišemo kao:

Težina1*dužina(hromozoma)+težina2*brojPrelomnihTačaka(završna permutacija)

Težina1 i težina2 su parametri algoritma, određuju uticaj broja prelomnih tačaka i dužine, određujemo empirijski.

Operator ukrštanja hromozoma

Razlikujemo slučajeve kada su dva hromozoma jednakih ili različitih dužina

1. Jednake dužine

    Koristimo standardno ukrštanje sa dve fiksne tačke.

2. Različite dužine
    Koristimo ,,Apsorpciju”

Apsorpcija: Funkcionise na principu ukrstanja sa 2 tacke gde se pocetna pozicija veceg hromozoma odabere nasumicno i onda se ceo manji hromozom stavi na pozicije nakon nje

Primer:
    Roditelj 1: 0,1,2,3,4,5
    Roditelj 2: 1,5,2,1
    Dete 1: 0,1,5,2,1
    Dete 2: 1,2,3,4

Operatori mutacije

Postoje 4 operatora mutacije

1. Uzmemo 2 inverzije (a b) I (c d) iz niza inverzija. Obrisemo b i c i dobijemo novu jedinku
2. Izaberemo 2 inverzije (a b) I (c d) I na pozicije nakon b I pre c umetnemo 2 random broja cime dobijamo duzu jedinku.
3. Izmena nasumicnog gena iz hromozoma
4. Izaberemo 2 pozicije iz niza inverzija i umetnemo random broj na tu poziciju cime dobijamo duzu jedinku.

Populacija

Populacija je veličine n*log(n), gde je n dužina niza koji se sortira.
Inicijalizuje se jedinkama koje imaju hromozome čiji broj inverzija ide od broja tačaka preloma do 3n.

Takođe dužina niza koji se invertuje ne može preći 10% dužine niza koji se sortira.

Pri generisanju nove populacije koristi se turnirska selekcija i dodatno se bira bolji od roditelja i deteta.

Ideja za ovako definisan algoritam
Neki osnovi ideja su preuzeti iz rada (1).
Šta smo koristili iz rada?

- Veličinu početne populacije
- Operatore ukrštanja
- Prva dva navedena operatora mutacije
- Dužina niza koja se invertuje ne prelazi 10% niza koji se sortira

Sve ostalo je nastalo po našoj ideji.

Po pretpostavkom da su navedeni podaci u radu tačni, rešenje koje je predloženo u radu je bolje od našeg rešenja (ima bržu konvergenciju i bliže je tačnom rešenju).

Zašto smo modifikovali rešenje koje je dato u radu?

Na samom početku implementacije algoritma, prvo smo pokušali da reprodukujemo rešenje iz rada.

Problem 1:

Nakon velikog broja iteracija, utvrdili smo da pristup iz rada dovodi do toga da naš algoritam konvergira ka delu prostora u kojem se rešenje nalazi.

Primer takvog ponašanja je kada najduži hromozom u populaciji bude kraći od neophodne dužine (ako znamo da se optimalno rešenje dobija u n koraka, a naš najduži hromozom ima dužinu m, pri čemu je m<n). Ovaj problem je rešen mutacijom 4.

Problem 2:

Često se dešavalo da imamo situaciju da je niz dosta dobro sortiran tj potrebno je uraditi jednu ili dve inverzije za potpuno sortiranje. Algoritam iz rada nema rešenje za ovaj problem. Zato smo dodali mutaciju 3.

Šta smo jos promenili u odnosu na standardne genetske algoritme?

Kod tradicionalnih genetskih algoritama, jednom inicijalizujemo populaciju vršimo selekciju, ukrštanja i mutacije i to sve ponavljamo određen broj generacija.

U našem pristupu nakon odrećenog broja generacija zaustavljamo algoritam, uzimamo najbolji hromozom, na ulazni niz primenjujemo hromozom i dobijamo izlazni niz.

Čest slucaj je da dobijeni izlazni niz nije sortiran, zato za dati niz pokrećemo ponovo naš algoritam. Sveukupno pokretanje algoritma odgovara uzastopnim pokretanjem genetskog algoritma.

Kod iz svakog pokretanja genetskog algoritma nadovezujemo na prethodno dobijeni kod.

Eksperimentalni rezultati

Poređenje sa brute-force rešenjem

Zbog velike složenosti algoritma grube sile, pokretaćemo samo za nizove do dužine 5.

Generisaćemo pseudoslucajni niz dužina od 2 do 5.

Dužina početnog niza	Početni niz	Broj inverzija algoritmom grube sile	Broj inverzija našom metodom
2	[2,1]	1	1
3	[2,1,3]	1	1
4	[4,1,3,2]	2	2
5	[3,2,1,5,4]	2	2
5	[1, 4, 2, 5, 3]	3	3

Tabela 1-Poređenje metoda grube sile i našeg genetskog algoritma

Napomena: Kako se radi o genetskom algoritmu, dobijeni rezultat nije deterministički, stoga za iste ulazne podatke, možemo dobiti drugačije izlaze

Takođe treba napomenuti da se već za dužinu niza 5 naš algoritam značajno brže izvršava od algoritma grube sile.

U narednoj tabeli možemo možemo uporediti prosečan izlaz dobijen u 10 pokretanja programa za svaku dužinu niza od 5 do 20 (upitanju su nasumično generisani nizovi) kao i prosečan broj prelomnih tačaka.

Dužina početnog niza	Prosečan izlaz iz našeg programa za datu dužinu niza	Prosečan broj prelomnih tačaka za datu dužinu niza
5	2.8	2.5
6	4.5	3.5
7	5.4	4.1
8	7.1	5.7
9	7.6	6.2
10	7.3	7.4
11	9.4	8.4
12	9.7	9.5
13	13.7	10.1
14	12	11.2
15	13.5	11.8
16	15.5	13.4
17	16.3	14.1
18	21.4	15.2
19	20.5	16.3

20	20.3	16.7

Tabela 2-Prosečan broj inverzija koji program ispisuje za obim uzorka 10

Broj pokretanja za svaku dužinu niza je 10.

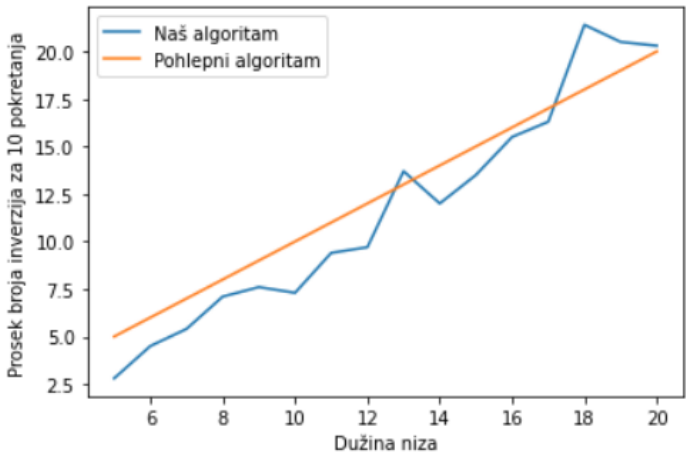
Iz praktičnih razloga obim uzorka nije veći od 10. Za veći uzorak nam je potrebno dosta više vremena. Iz istih razloga smo išli samo do dužine 20.

Iz tabele vidimo da je broj prelomnih tačaka relativno visok u odnosu na dužinu niza.

Nizovi koji imaju veliki broj prelomnih tačaka su teži za obradu od nizova koji imaju manji broj prelomnih tačaka.

Nizovi u Tabeli 2 predstavljaju teške instance problema (imamo veliki broj prelomnih tačaka).

U praksi, ako niz nema preveliki broj prelomnih tačaka ponašanje algoritma će biti značajno bolje.



Grafik 1: Grafik poređenja našeg algoritma i pohlepnog algoritma zasnovan na tabeli 2

Okruženje za testiranje

Razvojno okruženje	**Jupyter Notebook**
Verzija interpretera	**Python 3.6.9**
Operativni sistem	**Ubuntu 18.04**
Procesor:	**AMD Ryzen 5 2600**

Tabela prikaza okruženja za testiranje

Zaključak

Pregled urađenog

U navedenom radu dali smo ideju za rešavanje problema minimalnog sortiranja inverzijom intervala.

Prikazano rešenje je realizovano upotrebom genetskog algoritma sa promenjivom duzinom hromozoma.

Prikazanim rešenjem ne garantujemo da je problem rešen optimalno.

Algoritam na kraju vraća broj inverzija kao i listu inverzija koju je potrebno izvršiti da bi se dobio sortiran niz.

Možemo više puta pokrenuti program za istu instancu problema (isti ulazni niz) i na taj način možemo potencijalno dobiti različite rezultate. Najmanji broj među dobijenim rezultatima uzimamo kao najbolje rešenje za koje smo saznali.

Za dobijeni rezultat nije moguće utvrditi koliko je daleko od optimalno rešenje.

Zapravo tačno određivanje udaljenosti našeg rešenja od optimalnog se svodi na određivanje optimalnog rešenja.

Objašnjenje:
Ako nam program vrati kao rešenje broj inverzija n, i pri tome na neki način znamo da je naše rešenje za k udaljeno od optimalnog rešenja, onda odmah znamo da je optimalno rešenje n-k.

Kako znamo da rešenje od n+k inverzija nije optimalno rešenje (jer je i ono udaljeno takođe za k)?

Mi već imamo rešenje od n inverzija, tako da rešenje od n+k inverzija sigurno nije optimalno, tj rešenje od n inverzija je ,,optimalnije” od rešenja od n+k inverzija pa samim tim rešenje od n+k inverzija ne može biti optimalno.

Pravci daljeg unapređenja

U daljem razvoj algoritma, mogli bismo da razmotrimo drugačije operatore mutacije i ukrštanja. Potencijalno ako bismo smislili bolje ove operacije dobili bismo bolji algoritam.

Takođe, mogli bismo testirati neku drugu vrstu selekcije (npr selekciju rangiranjem).

Ako nam je vreme izvršavanja kritično, mogli bismo isti algoritam uraditi u nekom drugom programskom jeziku koji se brže izvršava npr C ili C++. Naravno na ovaj način ne možemo promeniti asimptotsku složenost predloženog rešenja!

Literatura

(1)
Sorting Unsigned Permutations by Reversals using Multi-Objective EvolutionaryAlgorithms with Variable Size Individuals
Ahmadreza Ghaffarizadeh, Kamilia Ahmadi and Nicholas S. Flann
Computer Science Department, Utah State University, Logan Utah 84322-4205

Link:
https://www.researchgate.net/publication/221009015_Sorting_unsigned_permutations_by_reversals_using_multi-objective_evolutionary_algorithms_with_variable_size_individuals