Minimum relevant variable in linear system

Marijana Čupovic

Septembar 2024

Sadržaj

1 Uvod

Ovaj rad se bavi rešavanjem problema minimalizacije broja promenljivih koje su potrebne za zadovoljenje sistema $Ax = b$. Problem je definisan na sledeći način: Neka je $A \in \mathbf{Z}_{m \times n}$ i $b \in \mathbf{Z}_m$. Tada treba naći vektor $x \in \mathbf{Q}_n$ takav da je $A * x = b$ i da je broj ne-nula vrednosti što manji mogući [4].

Za ovaj problem se zna da je NP težak, ali se ne zna mnogo o njegovim svojstvima aproksimacije, ali se zna da je najteže aproksimirati x kada njegovi elementi uzimaju vrednosti iz diskretnog skupa, posebno kada su u pitanju binarne vrednosti [1].

Posebni slučajevi ovog problema su izučavani u kontekstu mašinskog učenja i statistike, gde se problem poredio sa problemima smanjenja broja potrebnih parametara za korektno klasifikovanje instanci jer se, kako navode, pokazalo da broj ne-nula vrednosti ima jak uticaj na učinak klasifikatora kada se primenjuju na neviđene podatke[1].

Pored navedenog oblika problema, u radovima se pominju i njegove verzije u kojima vrednosti matrice A, vektora b i x mogu uzimati i realne brojeve, i opšti oblik problema je $A * x \ \mathcal{R} \ b$, gde je $\mathcal{R} = \{<, \leq, =, >, \geq\}$.

2 Gruba sila

Algoritam grube sile se zasniva na potpunoj enumeraciji prostora rešenja. Ideja je da se kreira svaki mogući podskup kolona matrice A i da se za njega proveri da li je sistem tada rešiv. Odabir kolona matrice je ekvivalentan postavljanju određenih elemenata vektora x na nula vrednosti. Kako se iterativno povećava broj kolona koje mogu učestovati u rešenju, možemo odraditi rano odsecanje i prekinuti petlju u trenutku kada se naiđe na prvo prihvatljivo rešenje.

Prolazak kroz sve moguće kombinacije kolona koje učestvuju u aproksimaciji vektora b je složenosti $O(2^n)$, gde je n broj kolona početne matrice A. Pošto je složenost algoritma eksponencijalna, za relativno mali broj kolona izvršavanje algotitma traje veoma dugo. Test instance na koje je primenjen algoritam grube sile su bile dimenzija 7×5, 10×7, 15×10, 15×12 i 20×15. Ovo su, na prvi pogled, veoma mali podaci, međutim, povećanje broja kolona dovodi do toga da se algoritam izvršava predugo.

Eksperimentalno je pokrenut algoritam grube sile za prvi test primer velikih dimenzija koji je kreiran za testiranje optimizacionih algoritama. Njegov broj kolona je 30 i izvršavanje je trajalo skoro $3.5h$ pre nego što je prekinuto. Kako je složenost algoritma eksponencijalna, dužina izvršavanja je očekivano velika i nedopustiva ukoliko želimo da radimo sa sistemima koji će imati stotine kolona matrice A.

3 Genetski algoritam

3.1 Opis osnovnog algoritma

Genetski algoritam je optimizaciona tehnika koja se zasniva na kodiranju potenciajlnog rešenja i računanju njegove podobnosti i primenom operacija nad njime kako bi se podobnost poboljšala. Tako predstavljeno potencijalno rešenje se naziva hromozom, dok funkcija podobnosti treba da oceni koliko je hromozom kvalitetno rešenje za problem. Opšti algoritam se sastoji iz narednih koraka:

- kreira se populacija hromozoma
- skup operacija se primenjuje na populaciju kako bi se prosečna podobnost populacije povećala

Period u toku koga se prmenjuju operacije na staru populaciju se naziva generacija. Osnovni genetski algoritam koristi barem operacije reprodukcije, ukrštanja i mutacije nad hromozomima [3]. Ceo ovaj postupak treba da simulira prirodnu selekciju među živim bićima koja se pokazala veoma uspešna u očuvanju najpodobnijih jedinki.

3.2 Korišćeni algoritam

Najbitnije karakteristike svakog genetskog algoritma su način kodiranja rešenja, procenjivanje podobnosti rešenja i operacije koje se primenjuju nad hromozomima populacije. Implementacija koje je kreirana za potrebe ovog rada je pored osnovnih operacija nad hromozomima je koristi i elitizam. Elitizam služi za zadržavanje najboljih jedinki stare populacije u novoj, što je postignuto jednostavnim kopiranjem hromozoma koji su bili najpogodniji u novu populaciju.

Rešenje je kodirano korišćenjem dva vektora, $cols$ koji čuva indekse kolona koje su trenutno izabrane za izdvajanje iz matrice A i vektora x koji je predstavljao konkretno rešenje sistema $A * x = b$. Prilikom incijalizacije hromozoma, vektor $cols$ dobija nasumični vektor nasumične dužine. Ukoliko se desi da za takav izbor vrenosti sistem nije rešiv, pokušavamo ponovo.

Podobnost ili $fitness$ hromozoma predstavlja broj kolona koje učestvuju u aproksimaciji vektora b, ukoliko je za baš taj izbor sistem dovoljno dobro aproksimirao b, a inače se penalizuje takvo rešenje, postavljanjem vrednosti $fitness$ na beskonačno. Da li razmatrano rešenje dovoljno dobro aproksimira b se proverava upoređivanjem srednje kvadratne greške koja nastaje aproksimacijom i predhodno zadate tolerancije. Tolerancija se zadavala na osnovu najmanje greške koju proizvodi funkcija $np.linalg.lstsq$ kada se primeni na celu matricu A i vektor b, tako što je tolerisano odsupanje od prave vrednosti u iznosu od 101% predhodno dobijene greške. Korišćen je ovaj pristup jer se za manje vred-

nosti od reziduala koji daje funkcija *np.linalg.lstsq* pokazalo da algoritam ne uspeva da nađe ni jedno podobno rešenje. S time u vidu, jedina opcija je bila da se tolerancija na grešku poveća u odnosu na slučaj kada se koriste sve kolone matrice A da bi se aproksimiralo b.

Selekcija hromozoma koji će učestovati u kreaciji novih jedinki je bila turnirska, gde se uzimao hromzom koji je imao najmanji broj kolona koje učestvuju u aproksimaciji iz nasumično odabranog uzorka.

Proces ukštanja je bio jednostavan i podrazumevao je kreiranje unije kolona roditelja. Ovaj pristup može da kreira jedinku koja je manje podobna od roditelja, međutim, kako se u algoritmu koristi elitizam i selekcija se vrši na osnovu podobnosti, ovakav hromozom će živeti mali broj generacija.

```
def crossover(parent1, parent2, child, A, b):
    child.cols = list(set(parent1.cols) | set(parent2.cols))
    child.update_x(A, b)
```

Slika 1: Algoritam ukštanja

Mutacija hromozoma je podrazumevala zamenu kolone nekom nasumičnom kolonom. Kako može da se desi da imamo višestruko pojavljivanje neke kolone, moramo kreirati skup kolona i od njega kreirati listu kako bi se kolone pojavljivale tačno jednom.

```
def mutation(individual, mutation_prob, A, b):
    n = A.shape[1]
    for i in range(len(individual.cols)):
        if random.random() < mutation_prob:
            individual.cols[i] = random.randint(0, n-1)
    individual.cols = list(set(individual.cols))
    individual.update_x(A, b)
```

Slika 2: Algoritam mutacije

Sam genetski algoritam je pratio klasičnu formu koja se može naći u literaturi, sa dodatkom elitizma. Pokretan je sa 170 jedinica hromozoma i 250 generacija što je duplo više od standarda koji je pronađen u literaturi [3]. Verovatnoća mutacije je bia 0.002, što je preuzeto iz literature [3], dok je veličina elitističke populacije bila 10.

4 Simulirano kaljenje

4.1 Osnovni algoritam

Simuirano kaljenje pripada familiji optimizacionih metoda u kojima veštački parametar, nazivan temperatura, kontroliše način istraživaja prostora pretrage uz održavanje konvergencije ka globalnom minimumu. Osnovni doprinos ove vrste optimizacije je što se trudi da izbegne lokalne minimume prilikom traženja rešenja. Lokalna pretraga uvek ide niz gradijentni spust čime kreira veliki rizik da, umesto globalnog, nađe lokalni minimum. Simulirano kaljenje pak daje mogućnost da se trenutno izme rešenje koje je lošije kako bi se istražio ostatak prostora pretrage. Ovaj, donekle stohastički momenat u simuliranom kaljenju mu daje mogućnost da izbegne zaglavljivanje u lokalnom minimumu. Iako se lako implementira i teoretski je smisleno, ipak ima manu, a to je da konvergira jako sporo [2].

4.2 Korišćeni algoritam

Na samom početku algoritma je potrebno inicijalizovati neko rešenje. Ovo je odrađeno postavljanjem 0 i 1 u vektoru dužine n, gde je n broj kolona matrice A. Ideja iza ovoga je da rešenje čuva indikatore da li je kolona učestvovala ili ne u aproksimaciji vektora b. Verovatnoća inicijalnog uzimanja kolone je 66.6% što se čini veliko ali se kod predhodno pokretanih algoritama pokazalo da je zapravo potrebno uključiti većinu kolona prilikom određivanja vektora b. Prilikom inicijalizacije i manipulacije vektorom koji predstavlja izabrane kolone, izbegavalo se jedino rešenje koje je popunjeno 0 vrednostima jer oni nije validno u smislu da se mora izabrati barem jedna kolona matrice A.

Prilikom izmene vektora koji sibolizuje rešenje, birao se nasumični indeks kome će se promeniti vrednost iz 0 u 1 ili iz 1 u 0, pri čemu se ponovo pazilo na to da postoji barem jedna ne-nula vrednost.

```
def small_change(solution, n):
    idx = random.randint(0, n-1)
    solution[idx] = abs(solution[idx]-1)
    if not any(solution):
        idx = random.randint(0, n-1)
        solution[idx] = 1
    return solution
```

Slika 3: Algoritam izmene rešenja

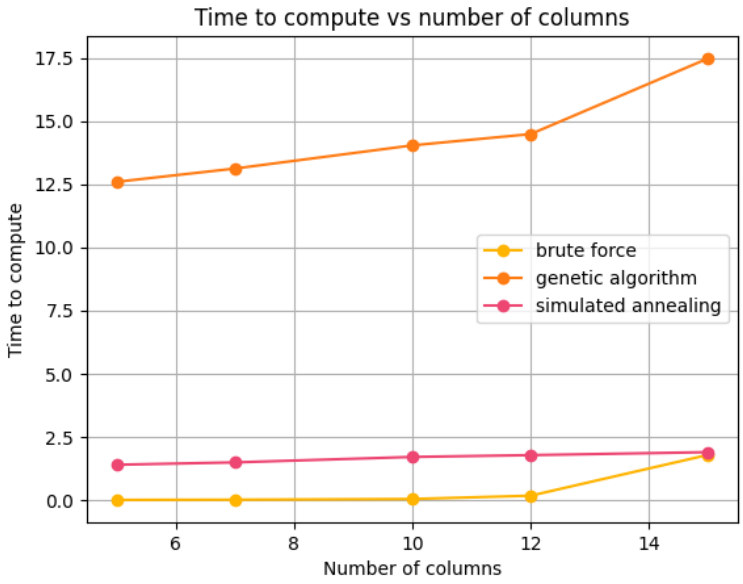
Računanje vrednosti trenutnog rešenja, kao i provera da li je rešenje dopustivo se vršilo na isti način kao i u genetskom algoritmu.

5 Poređenje rešenja

Za algoritam grube sile znamo da je eksponencijalne složenosti ali je dovoljno efikasan za manje dimenzije kao kontolne grupe. On je pokretan kako bi se našla rešenja i uporedila sa rešenjima koja optimizacione metode nalaze ali i da bi se vremena izvršavanja uporedila.

Genetski algoritam nalazi optimalna rešenja koja je algoritam grube sile našao za manje dimenzije problema ali u znatno većem vremenu.

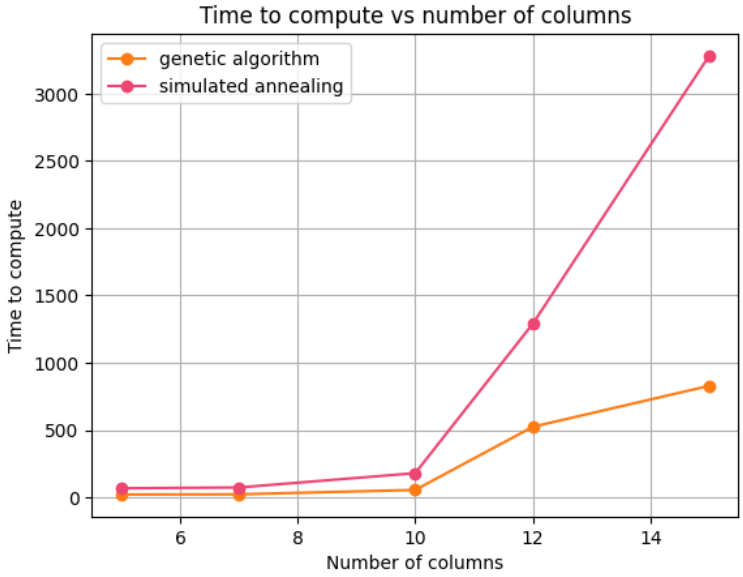
Simulirano kaljenje je našlo rešenja koja je našao algoritam grube sile i to u vremenu koje je znatno manje od onog koje je bilo potrebno genetskom algoritmu.



Slika 4: Vreme izvršavanja u odnosu na broj kolona

Može se reći da su gruba sila i simulirano kaljenje dovoljno brzi kada su manje dimenzije problema u pitanju ali kada pređemo na malo veće dimezije situacija se menja.

Algoritam grube sile ne uspeva da nađe rešenje za najmanju instancu u skupu velikih podataka u roku od $3.5h$ nakon čega biva prekinut. Sa druge strane se nalazi algoritam simuliranog kaljenja koji se izvršavao nekoliko sati i nije uspeo da nađe dopustivo rešenje ni za jednu instancu sem za prvu. U poređenju sa njima, genetski algoritam nalazi rešenja koja se čine ispravnim i koja jesu dopustiva za najviše $15min$ i to za instancu koja ima 150 kolona.



Slika 5: Vreme izvršavanja u poređenju sa brojem kolona

Ukoliko bi smo posmatrali samo ponašanje na malim podacima, i algoritam grube sile i simulirano kaljenje bi se mogli smatrati dovoljno efikasnim ali optimizacioni algoritmi služe da omoguće rešavanje problema za velike dimenzije u realnom vremenu. Iz te perspektive, moj pokušaj rešavanja problema simuliranim kaljenjem nije ni dovoljno efikasno niti daje ispravna rešenja, dok se za genetski algoritam može smatrati da je pristojan.

6 Zaključak

Prilikom izrade ovog projekta moglo se videti da je algoritam grube sile eksponencijalne složenosti, pa je samim time i neupotrebjiv za veće dimenzije. Iz literature se moglo saznati da ovaj problem nije moguće aproksimirati unutar faktora $2^{log^{1-\epsilon}n}$, za bilo koje $\epsilon > 0$, gde je n broj promenljivih [1]. Zato je potrebno primeniti optimizacione metode koje će se izvršavati u razumnom vremenu čak i za dimenzije problema koje su veće.

Genetski algoritam se pokazao sporijim za male ulazne podatke od grube sile ali je to očekivano s obzirom na veličinu populacije i broj generacija koji su korišćeni. Bez obzira na to, algoritam se pokazao upotrebljivim za znatno veće dimenzije, koje algoritam grube sile ne bi mogao da izvrši u razumnom vremenu. Postoji prostor za poboljšanje pristupa, na primer ispitivanjem novih načina kodiranja rešenja. Postoje različiti primeri u literaturu koji se mogu isprobati kako bi se utvrdilo da li će oni naći pogodnija rešenja i to brže.

Simulirano kaljenje je dalo tačne rezultate u skoro istom vremenu za sve dimenzije manjih ulaza, međutim, za veće dienzije nije uspevalo da nađe ni jedno dopustivo rešenje. Razlog ovome može biti način kodiranja rešenja i metoda izmene trenutnog rešenja. Može se poraditi na različitim načinima reprezentacije problema u cilju postizanja boljih rezulatata kada su veće dimenzije u pitanju.

Literatura

[1] Edoardo Amaldi i Viggo Kann. "On the Approximability of Minimizing Nonzero Variables or Unsatisfied Relations in Linear Systems". *Theor. Comput. Sci.* 209 (1998.), str. 237–260. URL: https://api.semanticscholar.org/CorpusID:14385658.

[2] Thomas Guilmeau, Émilie Chouzenoux i Víctor Elvira. "Simulated Annealing: a Review and a New Scheme". *2021 IEEE Statistical Signal Processing Workshop (SSP)* (2021.), str. 101–105. URL: https://api.semanticscholar.org/CorpusID:237245864.

[3] Aaron H. Konstam. "Linear discriminant analysis using genetic algorithms". *ACM Symposium on Applied Computing*. 1993. URL: https://api.semanticscholar.org/CorpusID:17523239.

[4] D.S. Johnson M.R. Garey. *Computers and intractability : a guide to the theory of NP-completeness*. W.H. Freeman i Company, 1979.