Projekat iz predmeta Operaciona istraživanja

Smer Informacioni inženjering

PROBLEM RANCA

Dokumentacija

Profesor: Studenti:

Mirna Kapetina Nikolina Radulović IN11/2018

Lana Kovačević IN38/2018

**Sadržaj**

[1. Opis problema 2](#_Toc95690538)

[1.1 Uvod 2](#_Toc95690539)

[1.2 Algoritmi za rešavanje problema ranca 3](#_Toc95690540)

[1.2.1. Metod grana i granica (*Branch and Bound*) 3](#_Toc95690541)

[1.2.2. Genetski algoritam 4](#_Toc95690542)

[2. Evaluacija i performanse algoritama 7](#_Toc95690543)

[3. Literatura 8](#_Toc95690544)

# 1. Opis problema

## 

## 1.1 Uvod

Problem ranca je problem kombinatorne optimizacije: dat je skup objekata gde svaki ima odredjenu težinu i vrednost. Potrebno je odrediti stavke koje će se naći u rancu tako da ukupna težina ne prelazi zadato ograničenje kapaciteta, a da se maksimizuje vrednost stavki. Ovaj problem slikovito je definisan na različite načine. Jedna od verzija odnosi se na pljačkaša koji treba da popuni ranac predmetima u što većoj vrednosti, znajući da svaki od predmeta može da uzme u celosti, ili ostavi. Ideja ovog problema je prilično jednostavna i jasno razumljiva čak i nestručnim osobama koje se ne sreću često sa problemima slične tematike. Ipak, duže od veka *knapsack* problem je čest predmet izučavanja i primenjuje se u mnogim oblastima – kombinatorika, računarstvo, finansije, teorija složenosti, kriptografija, primenjena matematika,... Štaviše, proširenje ranac problema na više dimenzija kako po broju ograničenja tako i po broju ranaca, kao i modifikacija strukture problema koja se tiče raspoloživog skupa artikala dovodi do niza zanimljivih, praktično primenljivih varijacija koje su predmet intenzivnih istraživanja tokom poslednjih nekoliko godina.

Osnovna verzija problema, koja je i osnova ovog projekta, može se formalno formulisati na sledeći način:

Maksimizovati

tako da

- *težina koja odgovara k-tom elementu, za k =1, 2, ...., N   
 - vrednost koja odgovara k-tom elementu, za k =1, 2, ...., N  
c - težinski kapacitet ranca*

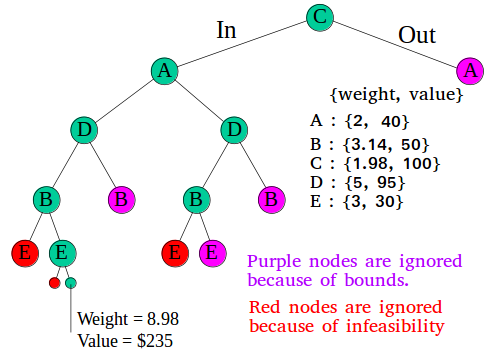
## 

## 1.2 Algoritmi za rešavanje problema ranca

Veći broj različitih algoritama u stanju je da pronadje rešenje ranac problema, izmedju ostalog *brute force* pristup, dinamičko programiranje, memorijske funkcije, pohlepni algoritam, algoritam grana i granica, genetski algoritam,... Izabrati jedan najbolji od navedenih algoritama za rešavanje ovog problema nije moguće, jer performanse uglavnom zavise od samog problema, njegove obimnosti i slozenosti, kompleksnosti implementacije, ali i izbora prioritetnog resursa čiju je upotrebu potrebno minimizovati, kao sto je memorija ili vreme. U okviru ovog projekta implementirali smo metod grana i granica i genetski algoritam, kako bi resili *knapsack* problem.

### 1.2.1. Metod grana i granica (*Branch and Bound*)

Metod grana i granica, kao jedna od tehnika za rešavanje optimizacionih problema, bazira se na konstrukciji binarnog stabla, gde svako potencijalno resenje predstavlja put od korena ka listu. Ovo nije jedini pristup koji razvija binarno stablo, ali je karakterističan po tome sto konstruiše i evaluira potencijalno rešenje do koga vodi dalje razvijanje podstabla od posmatranog čvora. Ako vrednosti preostalih komponenti ne mogu dati bolje resenje od trenutno najboljeg, onda se podstablo posmatrane komponente neće ni generisati, što smanjuje kompleksnost i daje bolje performanse u slučaju obimnijih kombinatornih problema.

****Svaki čvor stabla nosi sledeće informacije:

• nivo čvora – na kojoj dubini stabla se čvor nalazi

• vrednost – kumulativna vre- dnost svih elemenata na grani

• tezina – kumulativna težina svih elemenata na grani

Koren stabla kao inicijalni čvor ima vrednost i težinu 0, i nalazi se na nultom nivou. Grana stabla koja vodi levo od čvora podrazumeva da je naredni element izabran, dok desna ukazuje na njegovo isključenje. U narednom koraku potrebno je izračunati gornju granicu za svaki od čvorova- sabraćemo kumulativnu vrednost do sada odabranih elemenata na grani – v sa proizvodom preostalog kapaciteta ranca (W-w) i najboljim mogućim rezultatom koji potencijalno daju preostali elementi ( /)

,gde je W ukupan kapacitet ranca. Na osnovu dobijene vrednosti jasno je da li ima svrhe razvijati podstablo posmatranog čvora tj. da li je uključivanjem tog elementa moguće dobiti bolji rezultat od trenutnog, pa se u zavisnosti od gornje granice odredjeni čvorovi odbacuju.

*PriorityQueue <Node> pq*

*Initialize the root*

*pq.enqueue(root)*

*max\_value = value(root)*

*while(pq is not empty)*

*current = pq.getMax()*

*if(current.bound > maxvalue)*

*Set left child of the current node to include next item*

*if(left child. value > maxvalue)*

*update bestitems*

*update maxvalue*

*if(left child.bound > maxvalue)*

*pq.enqueue(left child)*

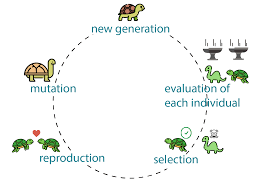
*Set right child of the current node not to include next item*

*pq.enqueue(right child)*

*return the best solution and its maximum value*

### 1.2.2. Genetski algoritam

Genetski algoritam je kompjuterski algoritam koji traži dobra rešenja problema između velikog broja mogućih rešenja koja prolaze kroz mnogo generacija. Razvoj ovog algoritma je počeo 60-ih godina prošlog veka od strane Džona Holanda, njegovih studenata i kolega na Univerzitetu u Michigan. Ove računske paradigme su inspirisane prirodnom evolucijom, uključujući opstanak najsposobnijih i preživelih jedinki, koje prolaze kroz niz koraka kao što su mutacija i reprodukcija.

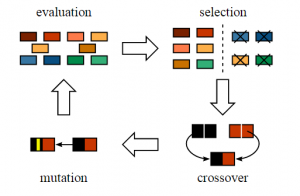


Slika 2- Prirodni proces genetskog algoritma

Algoritam na samom početku formira populaciju koja se sastoji iz jedinki (hromozoma), da bi na kraju ovog algoritma nova populacija bila stvorena od rešenja stare populacije u nadi da će dati optimalnije rešenje, odnosno biti bolja. Izabrana najoptimalnija rešenja će činiti potomak. Što su rešenja prikladnija, to su veće šanse da moraju da se reprodukuju. Ovaj proces se ponavlja sve dok se ne ispuni određeni uslov.

Neke primene genetskog algoritma su u optimizacijske, uključuju automatsko programiranje, mašinsko učenje, javljaju se u oblasti ekonomije, oblasti populacione genetike.

Složenost genetskog algoritma zavisi od broja stavki (N) i od broj hromozoma u svakoj generaciji (Size). To je **O (Size\*N)**.



***Hromozomi*** u genetskom algoritmu predstavljaju prostor kandidata za rešenja. Moguća kodiranja hromozoma su *binarna, permutacija, vrednost i kodiranja stabla*. Problem ranca, koristi binarno kodiranje, gde je svaki hromozom niz bitova, 0 ili 1.

Prvo što radimo jeste **inicijalizacija** svih promenljivih koje ćemo koristiti tokom narednih koraka (funkcija *init*).

Zatim se inicijalizuje populacija na random način. Ako zamislimo da imamo 4 objekta, mi ćemo praviti različite permutacije vektora od 4 bita sačinjenog od 0 ili 1. I to će činiti polaznu populaciju(ako stavimo npr 0110- to znači da će 2. i 3. objekat upasti u ranac, računa se njihov profit i ako je <= od zadatog kapaciteta ranca, onda ulazi u populaciju i koristi se u narednim koracima).

1. *Start: Randomly generate a population of N chromosomes.*
2. *Fitness: Calculate the fitness of all chromosomes.*
3. *Create a new population:*
4. *Selection: Randomly select 2 chromosomes from the population.*
5. *Crossover: Perform crossover on the 2 chromosomes selected.*
6. *Mutation: Perform mutation on the chromosomes obtained.*
7. *Replace: Replace the current population with the new population.*
8. *Test: Test wether the end condition is satisfied. If so, stop. If not, return the best solution in current population and go to Step 2*

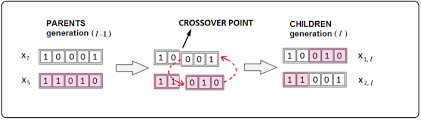
Genetski algoritam zahteva funkciju **fitnesa** koja dodeljuje rezultat svakom hromozomu u trenutnoj populaciji. Kada se formira populacija sačinjena od random sastavljnjnih jedinki, sledeći korak je da se izračunaju vrednosti koristi i težina svih jedinki pojedinačno. Ako izračunata težina jedinke je veća od kapaciteta ranca, ona se odbacuje, dok sve ostale prelaze u naredni korak.

U funkciji evaluation računamo fitness svakog roditelja i sortiramo vrednosti od najbolje ka najlošijoj, zatim smeštamo u vektor.

Proces ***selekcije***- jedinke koje se vrednuju sa višim vrednosti će najverovatnije biti izabrane za reprodukciju, dok će one sa niskim vrednostima biće odbačene. Međutim, najprikladnije jedinke mogu biti izabrane nekoliko puta,dok broj hromozoma odabranih za reprodukciju jednak je veličini populacije, održavajući veličinu konstantnom za svaku generaciju. Ova faza ima element slučajnosti baš kao i opstanak organizama u prirodi.

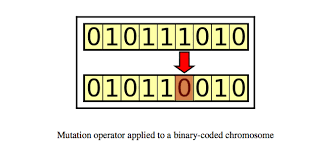
***Ukrštanje*** je proces kombinovanja bitova jednog hromozoma sa delovima drugog. Ovo je stvaranje potomstva za sledeću generaciju koje nasleđuje osobine oba roditelja.

Kada radimo ukrštanje kod binarnih jedinki, u slučaju kada smo uzeli dva roditelja, sačinjena od po 4 bita, od kojih će nastati dva detetato se radi tako što- prva dva bita prvog roditelja i poslednja dva bita drugog roditelja će činiti 1. dete, dok prva dva bita drugog roditelja i poslednja dva bita prvog će činiti drugo dete . Od n roditelja treba da nastane 2n potomaka.



Slika 4- Prikaz binarnog ukrštanja

***Mutacija*** se vrši nakon ukrštanja da bi se sprečilo opadnje rešenja populacije u lokalni optimum problema. Mutacija menja novo potomstvo okretanjem bitova od 1 do 0 ili od 0 do 1.

Kada imamo binarni genetski algoritam mutacija se može vršiti inverzijom ili rotacijom. Kod mutcije rotacijom se bira 1 tačka i invertuje se u obrnuti bit. A kod mutacije inverѕijom izaberemo 1, 2 ili više tačaka i rotiramo genetski materijal. Ovde je tipično zadata vrednost koja se zove mutation rate i prolaskom kroz petlju gleda se da li je vrednost trenutnog bita veća od tog rejta, i ako jeste 0 okrenemo u 1 i obrnuto.

Slika 5-Prikaz mutacije kod binarnog GA

Svaka iteracija ovog procesa naziva se **generacija**. Čitav skup generacija naziva se a *run*.

Da bismo došli do najoptimalnijeg rešenja, možemo preći i na poslednji optimalni korak koji se zove ***Elitizam*** u kome stavljamo n najboljih jedinki, u kojoj se moraju naći i jedinke roditelja i jedinke dece. Postoje dva razloga zašto je to tako. Ako bismo samo izabrali roditelje, nikad ne bismo konvergirali i došli do rešenja, a ako bismo izabrali samo decu, mi biso konvergirali ali sva rešenja bi bila slična i ne bismo dobili optimalno.

# 2. Evaluacija i performanse algoritama

Najpre smo primenili oba algoritma nad prostim primerom od 4 elementa:

|  |  |
| --- | --- |
| broj elemenata | 4 |
| vrednosti | [40, 30, 50, 10] |
| težine | [2, 5, 10, 5] |
| kapacitet ranca | 16 |

Oba algoritma jednostavno dolaze do očiglednog rešenja, koje podrazumeva izbor prvog i tećeg elementa sa vrednostima 40 i 50 i tezinama 2 i 10 tj. maksimalna vrednost elemenata u rancu je 90, gde zbir težina koji iznosi 12 ne premašuje ograničenje kapaciteta ranca od 16 jedinica. Da bi došao do ovog rešenja, *Branch and Bound* algoritam generisao je 11 čvorova, dok je genetskom algoritmu bilo potrebno da kreira samo jednu populaciju.

U narednoj tabeli prikazani su početni parametri u vidu vektora tezina, vrednosti, kapaciteta ranca za skup od 10 mogućih elemenata:

|  |  |
| --- | --- |
| broj elemenata | 10 |
| vrednosti | [40, 30, 50, 10, 12, 60, 30, 24, 55, 45] |
| težine | [2, 50, 10, 5, 6, 30, 6, 4, 5, 9] |
| kapacitet ranca | 32 |

Branch and bound sada generise 25 čvorova kako bi došao do najbolje kombinacije kojom dobija vrednost 220, ispunjavajući pun kapacitet ranca – 32. Genetski algoritam daje istu kombinaciju nakon sto izgeneriše 70 populacija, i prezentuje rešenje kao jedinku tj. vektor sačinjen od nula i jedinica:

[1, 0, 1, 0, 0, 0, 1, 0, 1, 1]

što znači da je potrebno izabrati prvu, treću, sedmu, devetu i desetu stavku.

# 

# 3. Literatura

1. <http://micsymposium.org/mics_2005/papers/paper102.pdf?fbclid=IwAR0vhTqiIoSp2gN-TEe8J17-sWC1JFsjzdo6a0SPiA4RlfkeD4c5gRBkTFM>
2. <http://www.sc.ehu.es/ccwbayes/docencia/kzmm/files/AG-knapsack.pdf?fbclid=IwAR2U7lW1fIIbtPnWmfnTvxfaa3Ysf8Z3d_z47xx4YBSbQ37jwKMEoI0Lexk>
3. <https://personal.utdallas.edu/~scniu/OPRE-6201/documents/DP3-Knapsack.pdf>