Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti: Popović Milica, SW11/2019

Radešić Nevena, SW12/2019

Predmet: Nelinearno programiranje i evolutivni algoritmi

Broj projektnog zadatka: 2

Tema projektnog zadatka: Genetski algoritam, problem ranca

# Opis problema

“Dat je skup predmeta. Svaki predmet je opisan težinom i vrednošću. Dat je ranac u koji je moguće stavljati predmete ukoliko njihova ukupna težina ne prelazi zadati kapacitet ranca wmax. Koju kombinaciju predmeta staviti u ranac tako da ukupna vrednost predmeta u rancu bude najveća moguća, a da ukupna težina predmeta ne prelazi maksimalnu dozvoljenu težinu wmax? Svaki predmet se može staviti u ranac maksimalno jednom(duplikati nisu dozvoljeni).”

U ovom projektu bavile smo se problemom ranca upotrebom genetskog algoritma (GA).

Odnosno, naš zadatak je da, na osnovu zadatog kapaciteta ranca, kao i liste predmeta, od kojih je svaki opisan svojom masom i vrijednošću, odredimo najveću vrijednost koja se može staviti u ranac tako da ukupna masa predmeta ne prevađe kapacitet samog ranca.

# Uvod

Da bismo primijenili GA, koristili smo binarno kodiranje. Svaka kombinacija predmeta u rancu predstavljena je kao niz nula i jedinica. Jedinica na *i*-toj poziciji znači da se u rancu nalazi *i*-ti element iz liste, dok nula znači da se ne nalazi. Svaka takva kombinacija predstavlja jedinku. Ako je ukupna težina jedinke veća od zadatog kapaciteta ranca, vrijednost jedinke se postavlja na 0, a u suprotnom se sabiraju vrijednosti pojedinačnih predmeta koji se u tom nizu nalaze.

# Implementacija

Strukturu programa čine četiri klase:

* *Population* – populacija
* *Chromosome* – jedinka (hromozom)
* *Item* – stavka u rancu
* *InputData* – početni podaci dati u fajlu

Klasa *Population* sadrži privatne i javne metode za generisanje inicijalne i nove populacije, rangiranje jedinki, odabir roditelja, ukrštanje, odabir jedinke sa najvećom vrijednošču.

Klasa *Chromosome* sadrži metode za računanje sopstvene vrijednosti i težine kao i metode za mutiranje tj. invertovanje bita.

*InputData* je po svojoj suštini struktura podataka i jedina njena svrha je da prenosi podatke iz fajlova potrebne za dalje računanje u programu.

*Item* je najjednostavnija klasa. Takođe je struktura podataka i pomaže u organizaciji i prenošenju podataka i lakšem pristupu.

Kriterijum optimalnosti

Kriterijum optimalnosti je zbir vrijednosti izabranih stavki, a ograničenje je ukupna težina koja mora biti manja ili jednaka zadatom kapacitetu ranca. Međutim, nakon uvođenja pravila da se svakoj kombinaciji, čija težina prelazi kapacitet ranca dodjeljuje vrijednost 0, ograničenje se može zamenariti, pa posmatramo samo kriterijum optimalnosti.

Strategije odabira jedinki za ukrštanje

Da bismo izbjegli pojavu super-jedinki, odlučili smo se da izbjegnemo upotrebu ruletske i sličnih selekcija, pa smo koristili metode rangiranja.

Odabir jedinki za ukrštanje (roditelja) započinje određivanjem vrijednosti (*fitness*) svake jedinke. Nakon toga se svakoj jedinki, u zavisnosti od *fitness*-a dodijeli rang i pomnoži nasumičnim brojem. Dobijeni proizvod predstavlja rezultat (*score*) jedinke. Na kraju je samo potrebno za roditelje odrediti jedinke sa najvećim rezultatom.

Ukrštanje i mutacija

Kada odredimo roditelje, možemo pristupiti samom ukrštanju. Najprije se generiše nasumičan prirodan broj, “*index*” u opsegu [0, broj bita koji kodira jednu jedinku - 1]. Sada se prvo dijete kreira tako što se prvih *index* bita preuzme od prvog roditelja, a ostali od drugog, dok se drugo dijete kreira simetrično: prvih *index* bita od drugog, ostali biti od prvog roditelja.

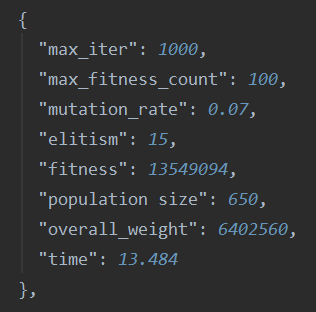
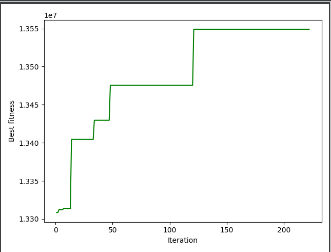
Da bismo izbjegli prerano sužavanje prostora pretrage, uveli smo mutaciju. Ona je implementirana tako što se, nakon kreiranja svakog djeteta, nad njim pozove funkcija “*mutate*”. Tada se za svaki bit jedinke generiše nasumičan broj i ako je on manji od određene vrijednosti, taj bit se invertuje.

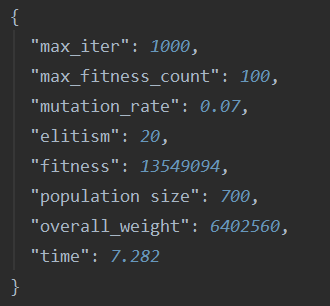
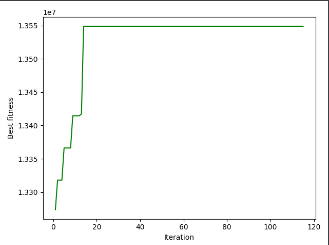
Kriterijumi zaustavljanja

Da bi došlo do zaustavaljanja, potrebno je da se ispuni jedan od dva definisana kriterijuma. Prvi je unaprijed određen broj iteracija koji kada se dostigne, algoritam se prekida (algoritam nije iskonvergirao). Drugi je uslovan, i ispuniće se ako se desi da se najbolja jedinka u populaciji ponavlja u zadatom broju generacija (algoritam je iskonvergirao).

Odabir parametara algoritma i rezultati u zavisnosti od njih

Parametri su definisani u fajlu *config.py*. Parametre smo odabrali u procesu testiranja. U zavisnosti od vremena izvršavanja i rezultata koje nam je genetski algoritam davao podešavali smo parametre tako da dobijemo najbolje rezultate. Parametri i rezultati svakog pokretanja programa su sačuvani u zasebnom fajlu kako bismo ih mogli analizirati. Takođe je dodat grafički prikaz napretka algoritma.

Najbolje rezultate smo dobili sa sljedećim parametrima:



*max\_iter* – Maksimalan broj generacija (kriterijum zaustavljanja)

*max\_fitness\_count* – Maksimalan broj ponavljanja najbolje jedinke (kriterijum zaustavljanja)

*mutation\_rate* – Stopa mutacija

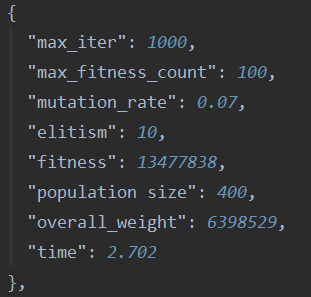
*elitism* – Elitizam

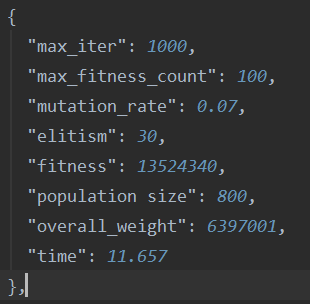
*fitness* – Ukupna vrijednost predmeta u rancu (nula ako ukupna težina predmeta predmeta prevazilazi maksimalni kapacitet)

*population\_size* – Veličina populacije

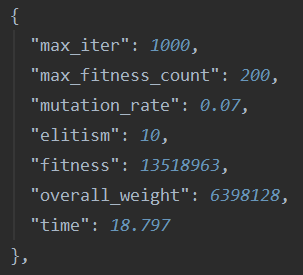
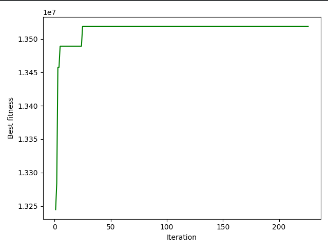
*overall\_weight* – Ukupna težina predmeta u rancu

*time* – Vrijeme izvršavanja programa u sekundima

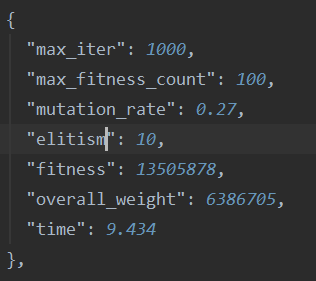
Nešto lošije rezultate smo dobili sa sljedećim parametrima:



Povećavanjem populacije na 800 i elitizma na 30 dobili smo sličan rezultat, ali se povećalo vrijeme izvršavanja. Smanjivanjem populacije na 400 i elitizma na 10 znatno se smanjilo vrijeme izvršavanja programa, ali je zato rezultat vidno lošiji.



Udvostručavanjem maksimalnog broja ponavljanja najbolje jedinke nismo dobili bolji rezultat, a pri tome se i program dugo izvršavao. Sa grafika se vidi da je nepotreban ovoliki broj ponavljanja.



Povećanjem stope mutacije nismo uspjeli dobiti bolji rezultat od najboljeg dobijenog.

# Zaključak

Ovime smo pokazali naše rješenje popularnog problema ranca za koje već postoje efikasnija rješenja i algoritmi. Ipak, na osnovu dobijenih rezultata, možemo zaključiti da je genetski algoritam dao veoma dobre rezulate. Da smo koristili grubu silu, bilo bi potrebno generisati i ispitati, u našem slučaju, ukupno 224 kombinacija, što bi bilo vremenski mnogo zahtjevnije. Uprkos tome što naš GA ne daje uvijek najbolje rješenje, smatramo da je, uz ograničenja koja imamo, naše rešenje dovoljno kvalitetno.