
Univerzitet u Novom Sadu
Fakultet tehničkih nauka


# Dokumentacija za projektni zadatak

Studenti: Bajić Vedran, SV10/2023
Čvorović Nikša, SV14/2023

Predmet: Nelinearno programiranje i evolutivni algoritmi

Broj projektnog zadatka: 1

Tema projektnog zadatka: Genetski algoritam, problem 2D rezanja

## Opis i analiza problema

Problem 2D rezanja je optimizacioni problem sa širokom primenom u industrijama koje zahtevaju sečenje malih pravougaonika iz velikih pravougaonih tabli (npr. industrija papira, industrija lima, industrija stakla...).

Postavka problema je sledeća: dato nam je $n$ vrsti pravougaonika kojih moramo iseći u $l_i$ , $1 \leq i \leq n$ primeraka od velikih tabli dimenzija $a$ i $b$. Kako iseći sve pravougaonike iz određenog broja tabli, a da pritom bacimo što manje materijala?

Neka svaki mali pravougaonik ima površinu $S_i$, $1 \leq i \leq n$ i neka je prilikom sečenja upotrebljeno $p$ tabli. Ukupnu upotrebljenost mterijala možemo definisati kao odnos zbira površina pravougaonika koje treba iseći i ukupne površine svih tabli koje smo upotrebili za sečenje:

$$\frac{\sum_{i=1}^{n} S_i l_i}{pab}$$

Gornji izraz će nam biti kriterijum optimalnosti koji je potrebno maksimizovati. Primetimo da je ukupna površina pravougaonika i površina jedne table konstantna, tako da ceo kriterijum optimalnosti zavisi samo od vrednosti $\frac{1}{p}$. Dakle, da bismo maksimizovali kriterijum optimalnosti, moramo minimizovati vrednost promenljive $p$, odnosno, da bismo upotrebili materijal što efikasnije, potrebno je upotrebiti što manje tabli za sečenje.

Naravno, minimizacija promenljive $p$ nije dovoljna – raspored sečenja mora da zadovolji uslov i da su je tačan broj traženih pravougaonika isečen iz datih tabli. Neka je $k_{ij}$ broj $i$-tih pravougaonika isečenih iz $j$-te table ($1 \leq i \leq n \wedge 1 \leq j \leq p$). Svako sečenje mora da zadovolji $n$ ograničenja:

$$\sum_{j=1}^{p} k_{1j} = l_1 \,;\; \sum_{j=1}^{p} k_{2j} = l_2 \,;\; \ldots \sum_{j=1}^{p} k_{nj} = l_n$$

tj. na $p$ tabli mora biti raspoređeno tačno $l_i$ pravougaonika $i$-te vrste.

Poslednje ograničenje je fizičko – dimenzije i površine pravougonika isečenih sa jedne table ne mogu prevazilaziti dimenzije i površinu date table.

# Implementacija

Traženje optimalnog rešenja je realizovano genetskim algoritmom u programskom jeziku Python.

Implementacija genetskog algoritma je sadržana u `ga.py` fajlu

Prvi korak genetskog algoritma je generisanje početne populacije „hromozoma“. U našem slučaju, jedan hromozom će predstavljati jedan nasumični raspored svih traženih pravougaonika preko više tabli.

Generisanje se vrši u okviru funkcije `create_init_population(pieces, piece_counts, stock_width, stock_height)`, koja kao parametre prima, redom, vrste pravougaonika, broj svake vrste pravougaonika i visinu i širinu table sa koje sečemo pravougaonike. Od ulaznih parametara se formira niz svih pravougaonika, on se permutuje u nasumičnom redosledu i redom se svaki pravougaonik postavlja na tablu što je više moguće levo. Kada sledeći pravougaonik u permutovanom nizu ne može da stane na tablu, dodajemo novu praznu tablu i nastavljamo daređamo pravougaonike. Kao rezultat jednog ovog postupka dobijamo niz tabli koji ispunjava tražena ograničenja – što nam čini jedan hromozom. Dati postupak se ponavlja dok ne dobijemo dovoljno hromozoma za početnu populaciju

Nakon generisanja početne populacije – genetski algoritam iterativno vrši sledeći niz operacija:

1. Odabir roditelja – od trenutne populacije veličine $2x$ na osnovu određenih parametara (od kojih su neki nasumični) izdvaja se $x$ jedinki (bira se polovina populacije)
2. Krosing over – iz selektovane polovine se $x$ puta nasumično biraju dva hromozoma i ubacuju se u funkciju koja kao izlaz daje još dva hromozoma – kao rezultat dobijamo $4x$ hromozoma

3. Mutacije – nad nasumično odabranim jedinkama se primenjuje funkcija koja prima jedan hromozom kao ulaz i vrši izmene nad njim
4. Selekcija – uklanjamo $2x$ jedinki od trenutnog skupa $4x$ hromozoma – kao rezultat dobijamo sledeću generaciju populacije sa $2x$ hromozoma – isto kao i na početku postupka

Dati postupak se može ponavljati unapred zadati broj puta ili može biti vremenski ograničen (što je slučaj u ovoj implementaciji algoritma)

Genetski algoritam je obmotan u okviru funkcije `ga(population, constraints, stock_width, stock_height)` koja kao parametre prima trenutnu populaciju, ograničenja na broj pravougaonika u hromozomu i dimenzije table.

Za ocenjivanje hromozoma koristi se broj tabli koje hromozom sadrži

Odabiri roditelja se vrše po principu turnirske selekcije. Turnirska selekcija je algoritam selekcije u kojem svakoj jedinci dodeli numerička ocena, pa se zatim kroz više iteracija formira skup odabranih jedinki – tako što se pri svakoj iteraciji nasumično bira mali podskup jedinki – koji nazivamo turnir, od njih se bira ona sa najboljom ocenom, uklanja se iz skupa svih jedinki i stavlja se u skup odbranih jedinki. Ovim postupkom u skup roditelja sledeće generacije biramo jedinke sa visokim ocenama, ali unosimo i varijabilnost (ukoliko sve jedinke u turniru imaju nisku ocenu – biće odabrana jedinka koja je najmanje „loša“). U okviru ove selekcije postoji elitizam – prvih nekoliko najboljih jedinki se odmah ubacuju u skup roditelja

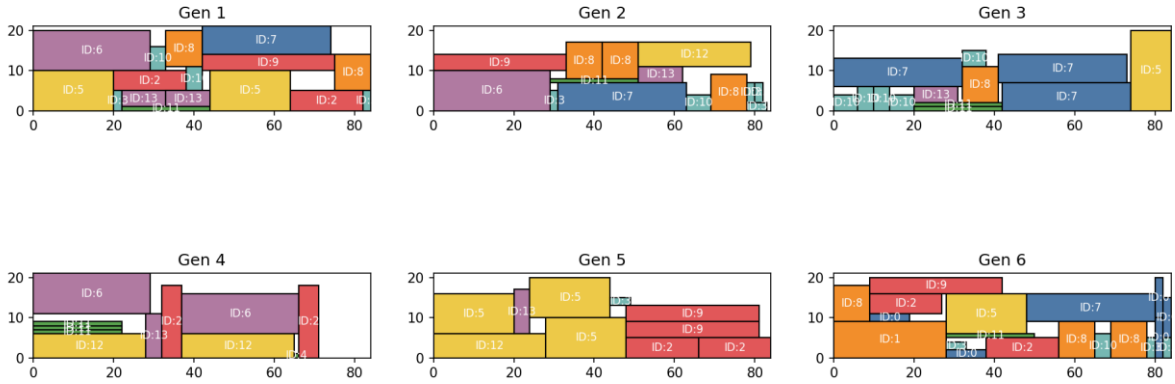
Za krosing over se koriste dve funkcije – `one_point_crossover(first, second)` i `two_point_crossover(first, second)`. Krosing over se realizuje tako što odaberemo dve jedinke i u zavisnosti od funkcije im zamenimo jednu tablu ili više uzastopnih tabli. Postoji jednaka šansa da će prilikom krosing overa jedna od ove dve funkcije biti izabrana.

Tokom ovog postupka se može desiti da novodobijene jedinke ne ispunjuju ograničenja, stoga se nad svakim dobijenim hromozomom primenjuje funkcija `check_chromosomes(constraints, chromosome)`, koja kao ulaz prima ograničenja i hromozom. Funkcija vraća vrednost `True` ukoliko hromozom ispunjava ograničenja, specijalno, ukoliko hromozom prevazilazi ograničenja i moguće je izbaciti jednu tablu tako da ih ispunjuje – ta tabla se izbacuje i vraća se `True`. U suprotnom, funkcija vraća `False` i hromozom se uklanja iz populacije.

Mutacije se vrše nad odabranim podskupom populacije. Tokom mutacije se odabranim hromozomima nasumično biraju dve table, prazne se, i njihov sadržaj se opet raspoređuje po njima. Ukoliko nakon raspoređivanja jedna tabla ostane prazna – ona se izbacuje. Specijalno – ukoliko se detektuje da je dvema tablama okupirano manje od polovine ukupne površine – njih dve će biti izabrane za mutaciju.

Selekcija sledeće generacije se vrši algoritmom ruletske selekcije. Svakom hromozomu dodeljujemo numeričku ocenu, koja se množi sa nasumičnim brojem. Sve jedinke se sortiraju i izbacuju se najgore ocenjene sve dok u populaciji ne ostane isto jedinki koliko je bilo pre krosing overa. U opštem slučaju, od populacije sa $4x$ jedinki bismo ih tokom selekcije uklonili $2x$, ali pošto se uklanjanje vrši i nakon krosovera sa jedinkama koje ne ispunjavaju uslove, u ovom koraku skoro uvek uklanjamo manje od $2x$ jedinki. U okviru ove selekcije postoji elitizam – pre množenja ocena nasumičnim brojem određeni broj najbolje ocenjenih sigurno prelazi u novu generaciju.

Nakon izvršavanja svih iteracija genetskog algoritma – konačna populacija se sortira po oceni i prikazuje se prvi hromozom u nizu, koji je najbliži optimumu. Vizuelizacija je izvršena korišćenjem `matplotlib` bibloteke i njen kod se nalazi u `graphics.py` fajlu.




Slika 1. Primer vizuelizacije jednog rešenja

# Diskusija

Genetski algoritam je po svojoj prirodi neefikasan i uključuje mnoštvo pseudoslučajnih brojeva, ali je ipak višestruko efikasniji za pronalaženje rešenja približnog optimalnom od analitičkog rešavanja za slučajeve visoke dimenzionalnosti.

Naša implemetacija genetskog algoritma je vremenski ograničena – izvršava se 20 sekundi, nakon kojih vraća najbolje pronađeno rešenje. U okviru tog vremena se u zavisnosti od parametara prolazi kroz varijabilni broj generacija.

Svi benchmarkovi su rađeni nad fajlom `in2.txt` u test primerima

Testovi su pokazali da je najskuplja operacija u okviru implementacije mutacija elemenata nakon krosovera. U sledećoj tabeli je data zavisnost broja generacija od broja mutacija u jednoj generaciji i broj tabli u rezultujućem rešenju. Pri svakom testu je za elitizam nad roditeljima i sledećoj generaciji odabran da bude 20.

| Broj mutacija u jednoj generaciji | Broj generacija | Broj tabli u najboljem pronađenom rešenju |
|---|---|---|
| 0 | 30779 | 10 |
| 10 | 2745 | 6 |
| 20 | 1439 | 7 |
| 30 | 955 | 7 |
| 40 | 527 | 7 |
| 50 | 553 | 6 |
| 60 | 466 | 6 |

Primetimo da sa porastom broja mutacija po generaciji do određene tačke broj značajno opada dok kvalitet rešenja ostaje približno isti.

Druga interesantna pojava je uticaj elitizma na kvalitet rešenja. Naime, pri svakom odabiru roditelja i sledeće generacije određeni broj najbolje ocenjenih jedinki garaantovano ulazi u skup roditelja / sledeću generaciju. Sledeća tabela prikazuje uticaj elitizma na konačno rešenje. Za broj mutacija u svakoj generaciji odabrana je vrednost 20. Algoritam je izvršavan 20 sekundi

| Broj jedinki zagarantovano odbranih za roditelje | Broj jedinki zagarantovano odabranih za sledeću generaciju | Broj tabli u najboljem pronađenom rešenju |
|---|---|---|
| 0 | 0 | 6 |
| 10 | 0 | 7 |
| 0 | 10 | 7 |
| 10 | 10 | 7 |
| 20 | 20 | 7 |
| 20 | 0 | 6 |
| 0 | 20 | 7 |

Možemo zaključiti da elitizam ne igra značajnu ulogu u našoj implementaciji i da je rešenje približno isto bez obzira na elitizam.

# Zaključak

Genetski algoritam se može uspešno primeniti na rešavanje problema 2D sečenja, iako je u zavisnosti od pseudoslučajnih brojeva moguče dobiti gore rešenje. Sami parametri algoritma imaju malu ulogu na kvalitet konačnog rešenja.