Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti: Stojkov Sara, SV38/2023

Milutin Marko, SV40/2023

Predmet: Nelinearno programiranje i evolutivni algoritmi

Broj projektnog zadatka: 3

Tema projektnog zadatka: Genetski algoritam, problem pravljenja rasporeda

# Opis problema

# Problem pravljenja rasporeda nastave je čest izazov različitih organizacija i institucija zbog ograničenog vremena i prostora koje treba što bolje iskoristiti. Ovaj problem svodi se na raspoređivanje predavanja i vežbi po učionicama i radnim danima, u okviru definisanog radnog vremena. Ulazni podaci su imena i trajanja događaja koje treba rasporediti. Raspored podrazumeva dodeljivanje tačne satnice, dana i učionice svakom događaju, uz poštovanje pravila – između aktivnosti u istoj učionici mora biti najmanje 15 minuta pauze, a radni dani su ponedeljak do petak.

# S obzirom na to da ovaj problem spada u NP‑teške probleme, odnosno one za koje ne postoje efikasni polinomijalni algoritmi koji mogu da obrade veliki broj ulaznih podataka, ovo je dobra prilika za korišćenje genetskog algoritma. Takav pristup predstavlja aproksimacionu metodu koja omogućava brzo pronalaženje dovoljno dobrog rešenja, bez iscrpnog ispitivanja svih mogućnosti.

# Uvod

Evolutivno programiranje koristi algoritme koji imaju fiksnu strukturu, dok dozvoljavaju numeričke parametre da *evoluiraju* odnosno menjaju se na osnovu nekih pravila i time se približavaju rešenju. Jedan od glavnih predstavnika ovakvih algoritama je upravo genetski algoritam. Genetski algoritmi su algoritmi za pretragu i optimizaciju koji se baziraju na principima prirodne evolucije. Problemi koji se rešavaju ovim algoritmom bivaju apstrakovani na jedinke i populacije pri čemu je jedinka jedno od potencijalnih rešenja. Ovaj tip algoritama primenjuje strategije optimizacije tako što forsira prirodnu selekciju odnosno opstanak najboljih jedinki. Generalno, ovaj proces se sastoji iz dve faze: prva podrazumeva selekciju jedinki koje će učestvovati u pravljenju sledeće generacije a druga podrazumeva primenu *crossover* operacije odnosno ukrštanja jedinki i mutiranje tih jedinki. Na početku algoritma je uvek potrebno generisati nasumično prvu generaciju koja će ući u pomenuti ciklus.

# Implementacija

Genetski algoritam je na ovom problemu u početku implementiran kroz definisanje njegovih koraka i ulaznih parametara. Osnova ovog algoritma je upravo jedinka – u našem slučaju jedinka jeste jedan raspored časova odnosno način da se ulazni časovi rasporede u *n* dana i *m* učionica. Jedan raspored podeljen je u blokove, pri čemu jedan blok predstavlja jednu učionicu i jedan dan odnosno jednu celinu u kojoj neko predavanje ili vežbe mogu da se odigraju. Prelamanje aktivnosti između blokova je rešeno proverama u kodu. Radi pojednostavljenja logike, vremenski intervali odnosno satnice podeljene su na intervale od po 15 minuta – to je jedinica vremena odnosno trajanje jednog *timeslot* intervala u rasporedima (jedinkama). Jedinka je predstavljena klasom *Schedule* koja kao atribute sadrži:

* *class\_list* – listu koja predstavlja raspored časova po učionicama i danima, pri čemu je implementirano kao matrica (lista listi) gde su redovi dani a kolone učionice
* *mapping* – hashmap strukturu koja služi za bržu pretragu časova, čuva indeks početnog timeslota u listi *class\_list* za svaki od časova iz ulaznih podataka
* *fitness*\_score– fitness jedinke kako bi se izbeglo redudantno računanje (i skupi prolasci kroz listu)
* *num\_blocks* – broj blokova, proizvod broja dana i broja prostorija
* *block\_size* – veličina bloka potrebna za kasnija računanja

Sem implementacija funkcionalnosti, parametri zadati pre pokretanja programa (u *const.py* fajlu) takođe utiču na rezultate algoritma. Parametri obuhvataju: broj radnih dana u nedelji *DAYS\_NUM*, putanju do fajla sa ulaznim podacima *FILE\_PATH*, veličinu populacije (broj jedinki) *POPULATION\_SIZE*, šansu da jedinka mutira *MUTATION\_CHANCE*, deo jedinki iz prethodne generacije koji se čuva pri selekciji *KEEP\_PERCENT*, maksimalan broj generacija *MAX\_GENERATIONS.*

Prilikom pokretanja programa, podaci o časovima se učitavaju iz zadate tekstualne datoteke i dolazi do formiranja prve generacije.

1. **Formiranje prve generacije**
2. **Kriterijum optimalnosti**
3. ***Fitness***
4. **Selekcija**
5. **Ukrštanje**
6. **Mutacije**
7. **Formiranje**

Algoritam se ponavlja dok se ne dođe do maksimalnog broja generacija

# Zaključak

Rešavanje problema aproksimacionim metodama Rešenje koje smo odabrali kao