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 logički je podeljen 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, dok je sama jedinka implementirana kao jednodimenziona lista. 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, dužina liste jeste

12 (sati) \* 4 (četvrtine sata) \* 5 (dana) \* 5 (učionica) = 1200 *timeslot*-ova

* *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

Populacija je implementirana kao lista jedinki odnosno objekata *Schedule*.

**Formiranje prve generacije** vrši se tako što se funkcija *set\_random\_classes* poziva nad praznim inicijalizovanim rasporedima. Ova funkcija prolazi kroz sve časove zadate ulaznim fajlom i smešta svaku od aktivnosti u nasumični blok. Pokušava da smesti časove tako da nema preklapanja odnosno *overlap* ukoliko je to moguće. Svakoj jedinki se pri kreiranju računa i *fitness score*.

**Kriterijum optimalnosti** ove metode optimizacije je već unapred zadat u specifikaciji problema. Cilj je da prosečno vreme početka nastave za svaku učionicu i svaki dan bude što kasnije, a prosečno vreme završetka što kasnije. Ukoliko obeležimo broj učionica sa , broj radnih dana sa tada će broj kombinacija učionica i dana biti i to možemo nazvati broj blokova. Takođe, za svaki od ovih blokova može se definisati odnosno vreme od 07:00 do početka prvog časa, kao i vreme od kraja poslednjeg časa do 19:00. Ako ovaj kriterijum formalizujemo to je:

***Fitness* funkcija** služi da odredimo koje jedinke su bolje od drugih. U našoj implementaciji, fitness funkcija se sastoji od 2 dela: prvi deo proverava validnost rasporeda i boduje ga u skladu sa tim, a drugi deo služi da vrednuje učionice u kojima časovi počinju kasnije i završavaju se ranije kao bolje. Konačan rezultat je kombinacija ova dva bodovanja.

Ovde možda formula

**Selekcija** odnosno izbor jedinki koje čine sledeću generaciju implementirana je kroz elitizam. Elitizam podrazumeva da postoji podela između roditeljske populacije i populacije dece. Iz roditeljske populacije u sledeću prelazi samo *n* najboljih jedinki po fitnesu (elita), dok se ostatak nove populacije popunjava najboljim jedinkama dece.

**Ukrštanje (*crossover*)** implementirano je tako da dva roditelja daju dvoje dece. Izbor roditelja vrši se ruletskom selekcijom gde se favorizuju jedinke sa boljim fitnesom ali ipak se unosi i stepen nasumičnosti. Za rangiranje jedinki se ne koristi njihov fitnes već rang njihovog fitnesa radi skaliranja i efikasnije randomizacije roditelja. Ovaj način odabira je bitan kako se ne bi došlo do jednolikosti populacije, a opet se smanjuje šansa da manje adaptirane jedinke daju potomstvo. Samo ukrštanje je implementirano preko *three-way crossover* tehnike – odaberu se 2 tačke odnosno indeksa u listi časova koje dele oba roditelja na po 3 segmenta časova. Ta 3 segmenta se kombinuju kao 1-2-1 i 2-1-2 čime se dobijaju 2 nove jedinke. Postoji pokušaj korekcije ako dođe do preklapanja pri ovakvim rotacijama časova. Nakon što se formiraju nove jedinke, one potencijalno prolaze kroz mutacije.

**Mutacije** služe da unesu varijabilnost u populaciju i pomognu da se prođe neki lokalni ekstrem. Zato je i u ovoj implementaciji funkcija mutacije u formi randomizaciji odnosno *shuffle* časova – časovi se sa trenutne pozicije nasumično premeštaju na druge pozicije. Broj časova koji se ovako premešta je takođe varijabilan i to nasumičan između i ukupnog broja časova. Šansa da se mutacija desi je varijabilna tokom izvršavanja algoritma i sa brojem generacija opada.

Algoritam se ponavlja dok se ne dođe do maksimalnog broja generacija ili dovoljno blizu rešenja (vrednosti *fitness score*-a najbolje jedinke u populaciji) za koje se smatra da je optimalno.

# Zaključak

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.* Ovde ćemo diskutovati koje vrednosti ovih parametara su izabrane kao optimalne i zašto.

Parametri *DAYS\_NUM* i *FILE\_PATH* zadati su specifikacijom i iznose 5 i *„data\_timetable.txt“.*

Parametar *POPULATION\_SIZE* postavljen je na 100 jedinki jer se pri povećavanju populacije ne dolazi do boljeg rešenja, već algoritam samo postaje značajno sporiji.

Parametar *KEEP\_PERCENT* koji se kristi pri selekciji (elitizmu) postavljen je na 0.2 jer se pri manjim procentima dešava da algoritam ne uspeva da nađe dovoljno dobro rešenje, a ukoliko se čuva više jedinki iz roditeljke populacije, sam fitnes brzo konvergira i nestane varijabilnost u populacijama.

Parametar *MAX\_GENERATIONS* je testiran više puta, ali pokazalo se da nakon 2000 generacija algoritam stagnira, tako da uglavnom nema potrebe za daljim izvršavanjem.

Što se tiče *MUTATIONS\_CHANCE* parametra, on sadrži 3 vrednosti i to [0.4, 0.3, 0.2]. Prva vrednost 0.4 se primenjuje na prvoj polovini maksimalnog broja generacija, druga vrednost 0.3 na sledećoj polovini (ukupno četvrtina) generacija, a 0.2 do kraja. Za varijabilnu šansu mutacije odlučili smo se jer se pokazalo da je u prvim iteracijama mnogo bitnije da mutacije unesu diverzitet u populaciju, dok kasnije poboljšanje fitnesa potiče uglavnom od ukrštanja.

Rezultat algoritma ogleda se u fitnesu najbolje jedinke u poslednjoj generaciji odnosno pri izlasku iz petlje i uglavnom iznosi (fitnes skorovi). TODO

Najbolje rešenje do kog smo došli korišćenjem gore navedenih parametara je \_\_\_\_. TODO

U skladu sa tim da su rešenja prilično slična i uravnotežena, nameće se zaključak da je implementacija genetskog algoritma na ovaj problem dala prihvatljivo i brzo rešenje.