Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti:	Bogdan Ljubinković, SV02/2023 Andjela Broćeta, SV75/2023
Predmet:	Nelinearno programiranje i evolutivni algoritmi
Broj projektnog zadatka:	1
Tema projektnog zadatka:	Genetski algoritam, problem pravljenja rasporeda

Uvod

U ovom projektu razvijen je genetski algoritam za optimalno raspoređivanje predavanja i vežbi na fakultetu. Ulazni podaci čine skup događaja (naziv i trajanje) i spisak učionica, a raspored se formira za radne dane (pon–pet) u vremenskom prozoru od 07:00 do 19:00. Između svakog para termina u istoj učionici obavezna je pauza od najmanje 15 minuta.

Fitness funkcija maksimizuje zbir proizvoda dužine pauze pre prvog i posle poslednjeg termina po učionici i danu. Algoritam radi u iteracijama: inicijalizuje populaciju, potom u svakoj generaciji primenjuje selekciju, jednopoint crossover i mutaciju, sve dok se ne zadovolji kriterijum konvergencije.

Struktura programa

Program je organizovan u devet nezavisnih modula. U **main.py** se pokreće čitav proces: poziva se read_rooms() i read_lectures() iz data_handler.py, zatim map_lectures() i generate_population() iz utils.py, nakon čega se prosleđuje inicijalna populacija u funkciju genetic_algorithm() iz genetic_algorithm.py i ispisuje rezultat (best_chromosome).

U **constants.py** su definisani svi parametri algoritma: **POPULATION_SIZE, MAX_GENERATIONS, ELITISM_RATE, MUTATION_RANGE, MUTATION_RATE_PER_CHROMOSOME, STAGNATION_THRESHOLD, SHAKEUP_RATIO, NUMBER_OF_DAYS, NUMBER_OF_CLASSROOMS, MAX_TIME_IN_CLASSROOM, MIN_PAUSE_TIME, MIN_NUMBER_OF_LECTURES, AVERAGE_LECTURE_DURATION, MAX_TIME_BEFORE_FIRST_CLASS, MAX_ATTEMPTS, CROSSOVER_ATTEMPTS, MUTATION_ATTEMPTS, DEDUCTED_FITNESS i NUM_OF_LECTURES.**

U **data_handler.py** se nalaze read_rooms() koje učitava nazive učionica iz data_timetable.txt i read_lectures() koje čita nazive i trajanja predavanja.

U **utils.py** su pomoćne funkcije: map_lectures(lectures) vraća mapu i listu predavanja, generate_population(mapped_lectures) kreira početnu populaciju odgovarajuće veličine i find_lecture_duration(index) vraća trajanje predavanja po indeksu.

Klasa **Chromosome** u chromosome.py definiše atribut genes (lista ID-jeva učionica po rasporedu) i metodu calculate_fitness() koja sabira proizvode pauza pre i posle terminskih blokova i oduzima DEDUCTED_FITNESS za svako kršenje MIN_PAUSE_TIME.

U **selection.py** su implementirane dve strategije izbora: rank_selection(population) koja sortira jedinke po fitness-u i dodeljuje rank-score, i tournament_selection(population, k=10) koja bira najboljeg od deset nasumično odabranih.

U **crossover.py** funkcija crossover(parent1, parent2) pokušava do CROSSOVER_ATTEMPTS da zameni po dve stavke između roditelja i koriguje nevažeće gene,

dok u **mutation.py** funkcija mutation(chromosome) vrši mutacije unutar MUTATION_RANGE, ponavljajući do MUTATION_ATTEMPTS.

Na kraju, **genetic_algorithm.py** u funkciji genetic_algorithm(population) radi inicijalnu evaluaciju svih hromozoma, a zatim svake generacije primenjuje rank selection sa elitizmom (ELITISM_RATE), shakeup po STAGNATION_THRESHOLD i SHAKEUP_RATIO, crossover za svaki par roditelja i mutaciju potomaka, prateći niz best_fitness i vraćajući najbolji hromozom nakon završenih generacija.

Ograničenja problema

Ograničenja problema obuhvataju fiksnu mrežu od pet radnih dana (NUMBER_OF_DAYS = 5) i pet učionica (NUMBER_OF_CLASSROOMS = 5), pri čemu se nastava odvija u vremenskom prozoru od 07:00 do 19:00 (720 minuta, MAX_TIME_IN_CLASSROOM) . Svako predavanje mora početi i završiti unutar tog intervala, a između bilo koja dva susedna termina u istoj učionici mora postojati pauza od najmanje 15 minuta (MIN_PAUSE_TIME) . Ukupan broj događaja definisan je sa 60 predavanja (NUM_OF_LECTURES = 60), a za svaku učionicu je obavezno zakazati minimum dva predavanja dnevno (MIN_NUMBER_OF_LECTURES = 2) .

Vreme pre prvog predavanja u učionici nasumično se bira iz opsega [0, MAX_TIME_BEFORE_FIRST_CLASS), gde je:

$$\text{MAX\_TIME\_BEFORE\_FIRST\_CLASS} = \text{MAX\_TIME\_IN\_CLASSROOM} - \text{MIN\_NUMBER\_OF\_LECTURES} \times \text{AVERAGE\_LECTURE\_DURATION} + \text{MIN\_NUMBER\_OF\_LECTURES} - 1) \times \text{MIN\_PAUSE\_TIME})$$

(AVERAGE_LECTURE_DURATION se računa nad svim ulaznim trajanjem predavanja) . Pritom se pri inicijalizaciji i mutaciji ograničava broj pokušaja (MAX_ATTEMPTS = 100) kako bi se izbeglo beskonačno petljanje prilikom raspoređivanja ili izmene termina.

Način implementiranja operatora mutacije i ukrštanja

Operator mutacije je definisan u funkciji mutation(chromosome) u mutation.py i vrši se dok nije postignut broj uspešnih mutacija jednak MUTATION_RATE_PER_CHROMOSOME ili dok max_attempts ne dostigne MAX_ATTEMPTS.

Svaki pokušaj započinje kopijom trenutnih gena u temp = [list(gene) for gene in chromosome.genes], pa se nasumično biraju dva različita indeksa predavanja (lecture_index1, lecture_index2) iz opsega [1, NUM_OF_LECTURES). Proverava se da se ne nalaze u istoj učionici (repeat_mutation), a zatim se u temp lociraju pozicije oba tuple-a.

Za svako od izabranih predavanja poziva se find_lecture_duration(lecture_index) kako bi se dobila originalna dužina, računa se razlika između stare i nove vrednosti trajanja, a susedne

pauze (celobrojni elementi) se povećavaju ili smanjuju za iznos razlike sve dok nijedna ne postane negativna. Kada su oba ažuriranja uspešno izvršena (successful_mutation == 2), chromosome.genes se zameni sa temp, counter se uveća, a temp se resetuje iz ažuriranog chromosome.genes. Nakon završetka petlje poziva se chromosome.calculate_fitness() i vraća izmenjeni hromozom.

Operator ukrštanja je realizovan funkcijom crossover(parent1, parent2) iz crossover.py. Najpre se proverava da oba roditelja budu instance klase Chromosome i da imaju istu dužinu liste gena, a zatim se kreiraju child1 i child2 dubokim kopijama lista roditeljskih gena.

U okviru petlje koja se izvršava do CROSSOVER_ATTEMPTS puta nasumično se biraju dve različite učionice (indeksi u child1.genes) i po jedan termin u svakoj, definišući donor_lecture1 i donor_lecture2.

Zatim se u listi child2.genes pronalaze odgovarajući termini (recipient_lecture1 i recipient_lecture2) iteracijom kroz sve učionice i termine. Kad su sva četiri predavanja identifikovana, prekopiraju se privremene liste gena (temp_c1g i temp_c2g), u njih se na mesto pomenutih indeksa swapuju tuplovi predavanja, a potom se izračunavaju razlike u trajanjima (donor_time – recipient_time) za oba predavanja i prilagođavaju susedne pauze tako da nijedna ne postane negativna. Ako nijedna provera ne padne, child1.genes i child2.genes se ažuriraju iz temp_c1g i temp_c2g i petlja se prekida; u suprotnom, pokušaj se odbacuje i prelazi na sledeći dok ne istekne broj pokušaja. Funkcija vraća child1 i child2 čak i ako nijedna razmena nije uspela.

Strategiju odabira jedinki za ukrštanje

Selektovanje roditelja za ukrštanje obavlja se jednom od standardnih rank-selection metoda iz selection.py. Na početku svake generacije poziva se funkcija rank_selection(population), koja prvo sortira populaciju po silaznom fitness-u, zatim svakoj jedinki dodeli slučajan skor jednak proizvodu nasumične vrednosti i njenog trenutnog ranga, pa nove liste sortira po tom skoru.

Time se postiže da jedinke sa boljim fitness-om imaju veću šansu da se nađu među prvim mestima u izmenjenom nizu. Nakon toga se za svaki indeks i formira par roditelja: jedinka na poziciji i i jedinka na poziciji i+1 (zadnja se uduplava ako nema para), i upravo te susedne jedinke ulaze u crossover(parent1, parent2) kako bi proizvele potomke.

Odabir parametara algoritma

Parametri genetskog algoritma su odabrani na osnovu kompromisa između kvaliteta rešenja i performansi, prateći opšte smernice za GA i empirijskim tuniranjem na test primerima. Veličina populacije (POPULATION_SIZE = 100) je dovoljna da obezbedi raznovrsnost kromozoma, a da ne uspori previše svaku generaciju . Maksimalan broj generacija (MAX_GENERATIONS = 600) omogućava algoritmu dovoljno iteracija za konvergenciju, ali je

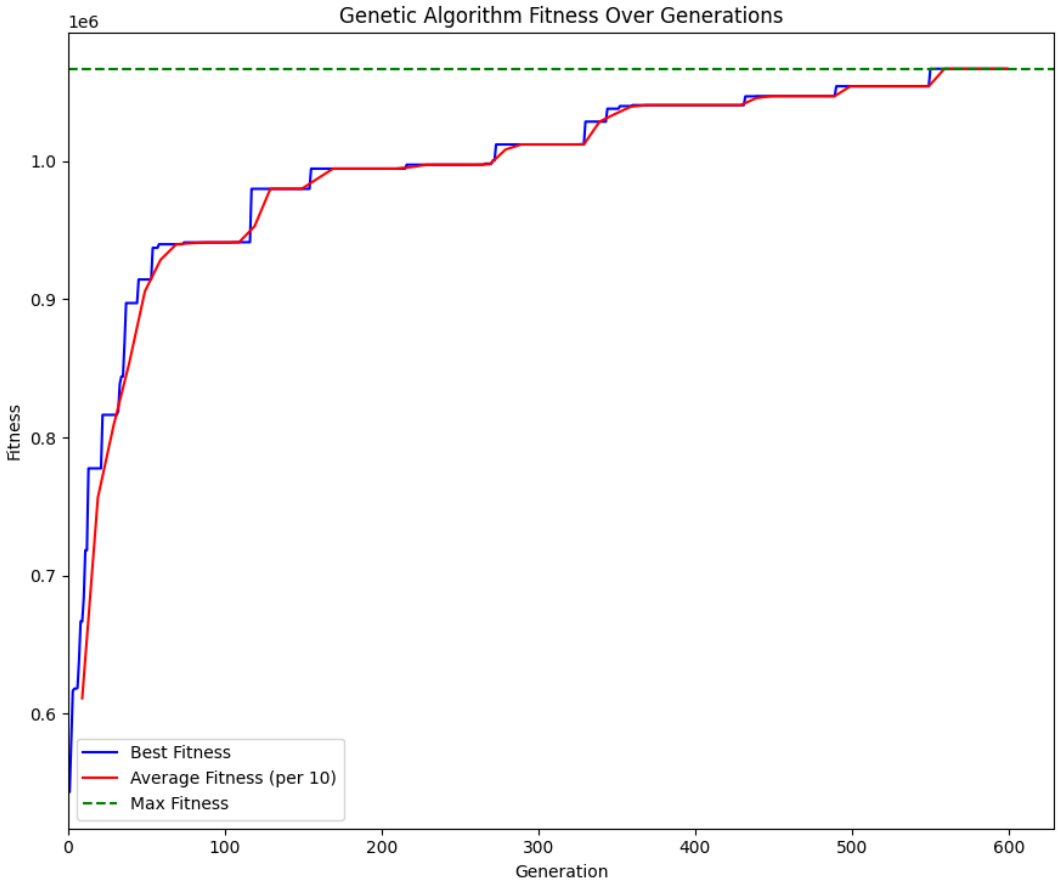
istovremeno ograničen da se izbegne predugo izvođenje . Elitizam je postavljen na 10 jedinki (ELITISM_RATE = 10), što garantuje da se najbolji rasporedi očuvaju u svakoj iteraciji .

Mutacioni i crossover parametri balansiraju eksploataciju i istraživanje prostora rešenja. Verovatnoća crossover-a je modelovana kroz prag pokušaja (CROSSOVER_ATTEMPTS = 50), što omogućava do 50 pokušaja uspešnog ukrštanja pre nego se odustane . Mutacija se ograničava na najviše 6 promena po hromozomu (MUTATION_RATE_PER_CHROMOSOME = 6) unutar maksimalno 15 pokušaja (MUTATION_ATTEMPTS = 15), kako bi se sprečilo da mutacije razore dobro prilagođene jedinke . Za sprečavanje zaglavljenja u lokalnim optimumima koristi se prag stagnacije od 10 generacija (STAGNATION_THRESHOLD = 10) i shakeup faktor 0.9 (SHAKEUP_RATIO = 0.9), koji nasumično mutira do 90 % populacije nakon stagnacije .

Rezultat algoritma

Algoritam je u 597. generaciji dostigao konačni fitness 1 066 940 i od tada više nije bilo dodatnih poboljšanja. Prvih 200 generacija donelo je oko 99 % maksimalne vrednosti, nakon čega je usledio sporiji, stepenoviti rast sve do oko 580. generacije, kada je shakeup mehanizam uspešno probio lokalni optimum.

Smanjenjem broja generacija na 300–400 i ranijim pokretanjem shakeupa (prag stagnacije 5 generacija) znatno se skraćuje vreme izvršavanja, dok blago povećanje stope mutacije ili smanjenje elitizma unosi dodatnu raznovrsnost populacije i otvara prostor za eventualno dalja poboljšanja.


Genetic Algorithm Fitness Over Generations
1e6
1.0
0.9
0.8
0.7
0.6
Fitness
0
100
200
300
400
500
600
Generation
Best Fitness
Average Fitness (per 10)
Max Fitness