Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Dokumentacija za projektni zadatak

Studenti:	Dejan Lisica, SV49/2022 Aleksa Ćup, SV27/2022
Predmet:	Nelinearno programiranje i evolutivni algoritmi
Broj projektnog zadatka:	5
Tema projektnog zadatka:	Genetski algoritam, Flexible Job Shop Problem (FJSP)

Opis problema

Projektni zadatak se odnosi na optimizacioni problem poznat kao "Flexible Job Shop Problem" (FJSP). Ovaj problem uključuje dodeljivanje poslova različitim mašinama i određivanje vremena za njihovo izvođenje. Postoje određeni poslovi koji se sastoje od uzastopnih operacija, i svaka operacija se može obaviti na jednom ili više različitih mašina. Glavni cilj je minimizirati ukupno vreme potrebno za završetak svih operacija.

Zadatak uključuje formiranje optimizacionog problema, implementaciju rešenja koristeći genetski algoritam, i grafičku prezentaciju dobijenih rezultata. Genetski algoritam treba da odredi redosled operacija na mašinama, kao i vreme početka i završetka svake operacije.

Uvod

Flexible Job Shop Problem (FJSP) je napredna verzija klasičnog problema rasporeda poslova (Job Shop Problem). U ovom problemu, skup poslova treba da se obradi na skupu mašina. Svaki posao se sastoji od serije operacija koje treba izvršiti u određenom redosledu. Glavna razlika u odnosu na tradicionalni Job Shop Problem je fleksibilnost u obradi: svaka operacija se može obaviti na više različitih mašina.

Kriterijum optimalnosti

U okviru ovog projekta, kriterijum optimalnosti za FJSP definisan je kao minimizacija najdužeg trajanja operacije na bilo kojoj mašini, poznatog kao 'makespan'. Ovaj pristup se fokusira na smanjenje ukupnog vremena potrebnog za završetak svih poslova, što je ključno za povećanje produktivnosti i efikasnosti proizvodnih procesa. Da bismo izračunali 'makespan', prvo pretvaramo genetski kod jedinke u konkretno rešenje ('solution'), a zatim analiziramo svaku operaciju unutar tog rešenja da bismo identifikovali operaciju s najvećim vremenom završetka. Ova vrednost 'makespan'-a služi kao osnovni indikator efikasnosti rasporeda i centralni je element u oceni i selekciji rešenja unutar genetskog algoritma.

Implementacija

Implementacija genetskog algoritma za rešavanje Flexible Job Shop Problem-a (FJSP) je podeljena na zasebne module, svaki sa svojom specifičnom ulogom. Ovo poglavlje detaljno opisuje svaku komponentu, kako one funkcionišu i kako su međusobno povezane u rešavanju problema.

1. config.py

Ovaj modul sadrži konfiguracione parametre za genetski algoritam koji su esencijalni za podešavanje algoritma i imaju direktan uticaj na njegovu efikasnost i efektivnost.

***svi konfiguracioni parametri u config fajlu su utvrđeni empirijski.**

- **population_size (200)**: Veličina populacije, odnosno broj različitih rešenja koja će algoritam istovremeno razmatrati.
- **number_of_generations (300)**: Broj generacija, tj. broj iteracija algoritma u procesu traženja optimalnog rešenja.
- **crossover_rate (0.9)**: Stopa krosiranja, verovatnoća kombinovanja gena dve jedinke da bi se stvorila nova.
- **mutation_rate (0.05)**: Stopa mutacije, verovatnoća nasumičnih promena u genima jedinke.
- **elitism_rate (0.2)**: Stopa elitizma, procenat najboljih jedinki koje se direktno prenose u sledeću generaciju.
- **mutation_rate_change (0.9)**: Faktor promene stope mutacije, koristi se za dinamičko prilagođavanje stope mutacije tokom algoritma.
- **mutation_rate_change_interval (100)**: Interval (u generacijama) na koji se menja stopa mutacije.
- **elitism_rate_change (0.05)**: Promena stope elitizma, koristi se za postepeno povećanje stope elitizma tokom algoritma.
- **elitism_rate_max (0.3)**: Maksimalna stopa elitizma koja može biti postignuta tokom algoritma.
- **elitism_rate_increase_interval (100)**: Interval (u generacijama) na koji se povećava stopa elitizma.
- **termination_variancy (20)**: Kriterijum za terminaciju algoritma zasnovan na varijaciji fitnessa među generacijama.
- **termination_variancy_len (20)**: Broj poslednjih generacija koje se uzimaju u obzir za izračunavanje varijacije fitnessa.
- **min_gen_number (100)**: Minimalan broj generacija koje algoritam mora izvršiti pre nego što može biti terminiran.

2. encoding.py

Ovaj modul je zadužen za enkodiranje i dekodiranje gena, inicijalizaciju populacije, i evaluaciju rešenja u okviru genetskog algoritma.

- **jobs_to_genes(jobs)**: Ova funkcija enkodira poslove u gene. Za svaki posao u listi, odabira se nasumična mašina iz dostupnih opcija. Takođe, formira se i niz operacija gde su operacije raspoređene nasumično. Rezultat je dva niza: jedan sa odabranim mašinama za svaku operaciju i drugi sa redosledom operacija.

- **initialize_population(jobs, population_size)**: Funkcija kreira inicijalnu populaciju za genetski algoritam. Za to koristi funkciju jobs_to_genes da generiše pojedinačne članove populacije (individualne rešenja problema) do željene veličine populacije.
- **__is_free(tab, start, duration)**: Pomoćna funkcija koja proverava da li je mašina slobodna u određenom vremenskom intervalu.
- **__find_first_available_place(start_cstr, prcTime, param)**: Funkcija traži prvo dostupno mesto na mašini koje zadovoljava vremenska ograničenja. Uzima u obzir već zauzeta mesta na mašinama i traži prvo slobodno mesto na kojem može da započne operacija.
- **genes_to_solution(genes, jobs, machine_number)**: Ova funkcija dekodira gene u rešenje problema. Na osnovu genetskog koda (mašine i redosled operacija), funkcija određuje kako će operacije biti raspoređene po mašinama.
- **get_time_span(solution)**: Funkcija računa ukupno vreme potrebno za izvršavanje svih operacija u rešenju. To je vreme završetka poslednje operacije u rešenju, i predstavlja ključni kriterijum za ocenu kvaliteta rešenja u genetskom algoritmu.
- **translate_decoded_to_gantt(machine_operations)**: Ova funkcija pretvara dekodirane podatke o operacijama u format pogodan za prikazivanje na Ganttovom dijagramu. Za svaku mašinu, funkcija pravi listu operacija sa njihovim početnim i završnim vremenima, što omogućava vizuelni prikaz rasporeda rada na mašinama.

3. gantt.py

Modul **gantt.py** je zadužen za vizualizaciju rešenja problema Flexible Job Shop Problem (FJSP) u obliku Ganttovog dijagrama. Sadrži funkciju draw_chart, koja crta Ganttov dijagram kao vizualizaciju rešenja FJSP. Ova funkcija koristi podatke o operacijama raspoređenim po mašinama da nacrta horizontalne barove na dijagramu, pri čemu svaki bar predstavlja određenu operaciju. Dužina i položaj bara odgovaraju trajanju i vremenu početka operacije. Različite boje se koriste za razlikovanje pojedinačnih poslova, a svaki bar je označen sa informacijama o operaciji.

4. genetic.py

U ovom modulu implementiramo genetski algoritam za rešavanje optimizacionog problema.

initialize_population: Inicijalizuje početnu populaciju potencijalnih rešenja koristeći funkciju iz modula encoding. Svako rešenje je predstavljeno genetskim kodom koji odražava raspored poslova i izbor mašina.

calculate_fitness: Računa fitnes vrednost jedinke, što je ključni deo evaluacije rešenja. Fitnes se bazira na ukupnom vremenu završetka svih poslova, tj. na efikasnosti rasporeda.

selection: Selekcija jedinki za reprodukciju. Ova funkcija pravi kopiju populacije i rangira trenutnu populaciju na osnovu njihovih fitnes vrednosti i bira najbolje jedinke za ukrštanje.

crossover: Ukrštanje (rekombinacija) dve jedinke (roditelja) da bi se stvorile nove (deca). Za svaki par gena se stvara privremeni niz koji sadrži gene oba roditelja. Taj niz se nasumično meša tako da se redosled gena promeni. Ovako se omogućava da svako dete nasleđuje deo genetskog materijala od oba roditelja, ali na nasumičan način. Ukrštanje kombinuje 'gene' roditelja, čime se stvaraju nove kombinacije rasporeda poslova i izbora mašina.

mutate: Funkcija **mutate** nasumično modifikuje genetski kod jedinke kako bi unela varijabilnost populacije, što je ključno za izbegavanje lokalnih optimuma i istraživanje šireg prostora rešenja.

Mutacija se dešava pod sledećim uslovima:

- **Mutacija mašinskog stringa (ms)**: Za svaku operaciju u svakom poslu, funkcija prolazi kroz ms (mašinski string) jedinke i odlučuje da li će doći do mutacije na osnovu nasumično generisanog broja i zadate stope mutacije (mutation_rate). Ako se mutacija desi (tj., ako je nasumični broj manji od mutation_rate), bira se novi nasumični broj koji predstavlja indeks mašine za tu operaciju, čime se menja originalni izbor mašine za datu operaciju.

- **Mutacija operativnog stringa (os)**: Slično kao i za mutaciju mašinskog stringa, funkcija odlučuje da li će doći do mutacije operativnog stringa os, koji predstavlja redosled operacija. Ako se mutacija desi (opet na osnovu mutation_rate), funkcija određuje nasumičan broj permutacija koje će se desiti u os. Za svaku permutaciju, bira se dva nasumična indeksa u os i vrši zamena vrednosti na tim pozicijama. Ovo menja redosled izvršenja operacija.

run: Glavna funkcija algoritma koja upravlja celokupnim procesom. Uključuje inicijalizaciju populacije, selekciju, ukrštanje, mutaciju, i evaluaciju fitnesa. Takođe prati promene u fitnesu kroz generacije i može prekinuti izvršavanje ako je ispunjen kriterijum terminacije.

5. **parser.py**

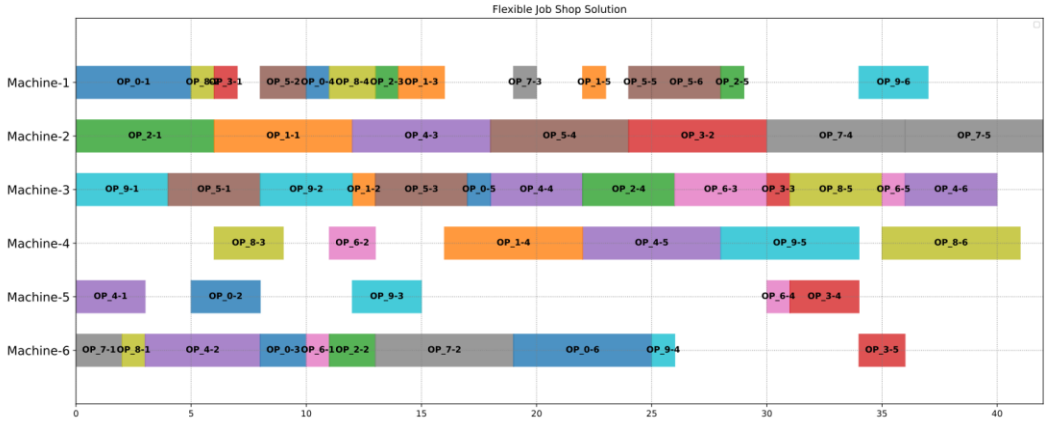
Modul **parser.py** sadrži funkciju **job_parser** koja je zadužena za parsiranje podataka iz datoteke specifično formatirane za FJSP. Ova funkcija čita datoteku koja sadrži informacije o poslovima i mašinama, te operacijama koje treba izvršiti.

Kada se pozove sa putanjom do datoteke, **job_parser** prvo čita zaglavlje datoteke kako bi odredio broj poslova **(job_number)** i broj mašina **(machine_number)**. Nakon toga, funkcija prolazi kroz svaki posao u datoteci, čitajući broj operacija za svaki posao i mašine koje mogu obaviti te operacije, kao i vreme potrebno za izvršenje svake operacije na svakoj mašini.

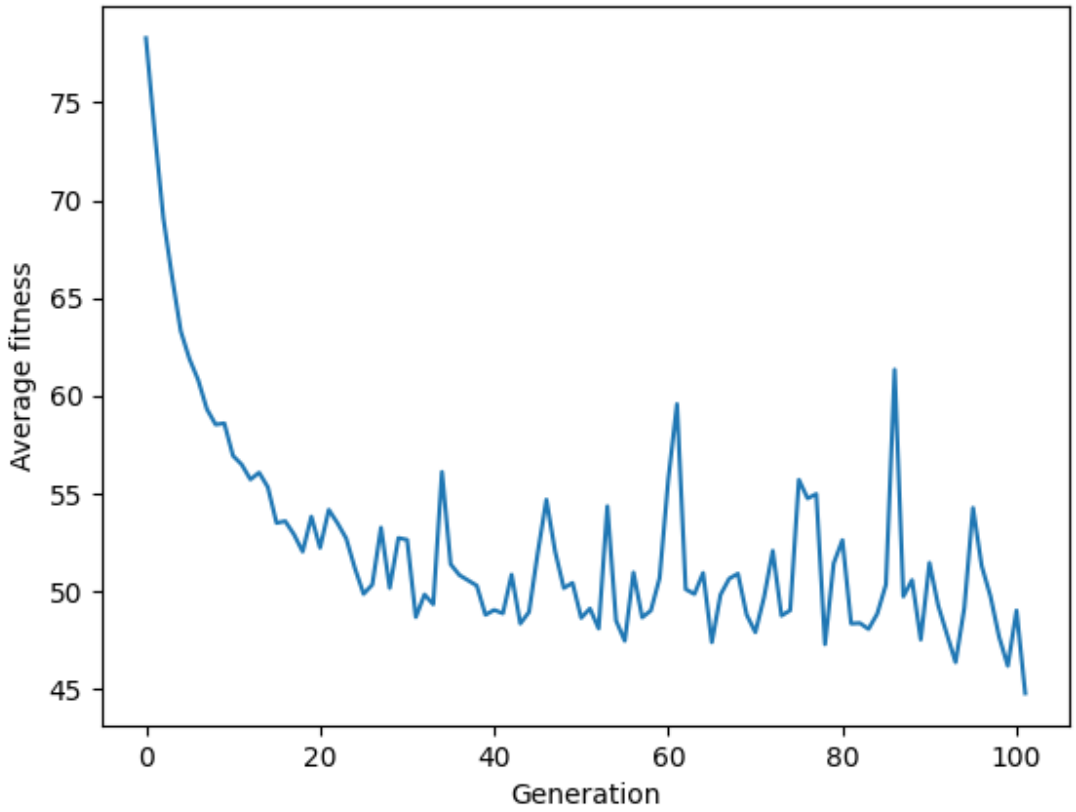
Analiza i rezultat

Važno je istaći da se u okviru ove implementacije genetskog algoritma stepen mutacije i elitizma dinamički menja tokom vremena, što dodatno doprinosi efikasnosti i prilagodljivosti algoritma. Konkretno, stopa mutacije se postepeno smanjuje nakon određenog broja generacija, što je definisano u konfiguracionom fajlu. Ovo smanjenje stope mutacije omogućava algoritmu da u početnim fazama istraživanja prostora rešenja ima veću varijabilnost, dok u kasnijim fazama, sa smanjenom stopom mutacije, omogućava finije podešavanje i optimizaciju već dobrih rešenja. Paralelno s tim, stepen elitizma se postepeno povećava tokom vremena, takođe definisan u konfiguraciji. Ovaj pristup osigurava da se kvalitetne jedinke iz ranijih generacija zadrže i potencijalno koriste u kreiranju novih rešenja. Povećanje elitizma pomaže u očuvanju i akumulaciji dobrih genetskih osobina kroz generacije, što vodi ka efikasnijem i ciljanom pristupu u pronalaženju optimalnog rešenja.

Na sledećoj slici nalazi se Ganttov dijagram koji grafički prikazuje raspoređenost poslova po mašinama, kao rezultat primene genetskog algoritma na Fleksibilni Problem Raspoređivanja Poslova (FJSP). Ovaj dijagram vizuelno demonstrira konačni raspored operacija, pri čemu različite boje predstavljaju različite poslove, a dužina i položaj svake barske linije odražavaju trajanje i vremenski redosled svake operacije na odgovarajućoj mašini. Dijagram omogućava jasno uočavanje ključnih aspekata rasporeda, uključujući efikasnost raspoređivanja, iskorišćenost mašina i ukupno vreme završetka svih operacija.



Sa druge strane, prosečan fitnes pokazuje opadajući trend u korelaciji sa povećanjem broja generacija, sugerisajući da se kvalitet rešenja poboljšava kako algoritam napreduje kroz generacije, što se jasno može videti na grafikonu prikazanom ispod.



Opadajući trend fitnesa u odnosu na broj generacija

Zaključak

U zaključku ove dokumentacije, posebno ističemo da je struktura hromozoma korišćena u našoj implementaciji genetskog algoritma za rešavanje ovog optimizacionog problema direktno inspirisana iz naučne studije koja je priložena u literaturi. Ova studija je pružila ključne uvide i teorijsku osnovu za način na koji smo pristupili modeliranju problema i razvoju genetskog koda, što je bilo suštinski važno za efikasnost i efektivnost algoritma.

Primena genetskog algoritma na optimizacionom problemu Flexible Job Shop Problem (FJSP) pokazala značajan potencijal u efikasnom rešavanju kompleksnih problema raspoređivanja u proizvodnim i operativnim okruženjima. Kroz implementaciju i analizu genetskog algoritma, demonstrirana je sposobnost algoritma da generiše optimalna ili sub-optimalna rešenja, uzimajući u obzir različite operativne zahteve i ograničenja.

Ganttovi dijagrami koji su proizvedeni kao rezultat algoritma omogućili su jasnu vizualizaciju rasporeda i efikasnosti planiranja, nudeći korisne uvide u proces raspoređivanja. Pored toga, prilagodljivost genetskog algoritma u pogledu njegovih parametara, kao što su stopa mutacije i ukrštanja, pokazala je da algoritam može biti finije podešen za specifične potrebe i scenarije.

Literatura

- An Effective Chromosome Representation for Evolving Flexible Job Shop Schedules
- A Genetic Algorithm for Flexible Job Shop Scheduling