SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR II

Rješavanje problema više trgovačkih putnika uz pomoć genetskog algoritma

Bernard Crnković

Voditelj: *Marin Golub*

Zagreb, Svibanj, 2022.

Sadržaj

1. Iskaz problema

Problem trgovačkog putnika jedan je od najistraživanijih optimizacijskih problema. Poznato je da spada u skup NP-potpunih problema sa složenošću čak većom od eksponencijalne (O(n!)). Često se koristi kao demonstrativni primjer za mjerenje performansi raznih metoda optimizacije. Vizualno, radi se o pronalaženju ture u grafu koja uključuje sve čvorove a ima minimalnu duljinu.

Može ga se definirati kao linearni program u kojem je potrebno minimizirati cijenu:

$$\sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$

gdje $x_{ij} = 1$ označava da postoji brid između gradova i i j, a 0 označava da ne postoji. Dodatno, pošto bi minimizacija samo ovakvog problema bila trivijalna ($x_{ij} = 0$ za svaki $i, j \in \{1 \dots n\}$), uvodi se zahtjev da svi čvorovi grafa budu uključeni te da svaki od njih ima točno 1 ulazni:

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \quad \text{za} \quad j = 1, \dots, n$$

te 1 izlazni brid:

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \quad \text{za} \quad i = 1, \dots, n$$

Sada je vidljivo odakle izniče faktorijelna složenost pošto rješenje može se prikazati kao jednu permutaciju gradova. S tim su razlogom razrađene brojne heurističke metode rješavanja ovog problema. Prvi algoritam koji dinamičkim programiranjem rješava ovaj problem jest Held-Karpov čija je složenost O(n^2 * 2 ^ n). Neki drugi pristupi su *B&B* algoritmi (rekurzivni sa ranim povratkom iz podskupova rješenja koja su gora od trenutačno nađenog najboljeg rješenja) i metode linearnog programiranja.

Međutim, više su interesantni heuristički ne-egzaktni algoritmi poput zato što, iako žrtvuju optimalnost, nalaze iznimno dobro rješenje za TSP na vrlo velikim ulaznim primjerima (dobro skaliraju). Jedan takav algoritam je više-fragmentni (MF) algoritam koji istovremeno gradi manje dijelove puta koje naposljetku poveže u jednu zatvorenu šetnju cijelim grafom. To je pohlepna metoda jer uvijek odabire najkraći put između bilo koja dva grada koji zadovoljava jedno od sljedećih svojstava:

- nastaje novi fragment
- proširuje neki od postojećih puteva
- zatvara konačni i jedini ciklus u grafu koji sadrži sve gradove

U praksi se pokazalo da najbolje rezultate za grafove s velikim brojem čvorova postiže upravo s algoritmom kolonije mrava (ACO). Radi na principu populacije

mrava koji posjećuju gradove na temelju privlačnosti (bridovi s više deponiranih feromona imaju veću šansu da budu izabrani) zbog deponiranih feromona drugih uspješnijih članova kolonije. Ključno je da je zasnovan na vjerojatnosti odabira bridova u šetnji jednog mrava, to znači da će mravi i dalje povremeno napustiti trenutno optimalnu turu i istražiti nove ture koje su potencijalno kraće. Feromoni koje najbolji mravi deponiraju isparuju kako bi algoritam postepeno konvergirao prema 'najpopularnijem' rješenju.

Problem više trgovačkih putnika je varijacija originalnog problema gdje imamo $m > 1$ trgovaca te svaki od njih obilazi podskup čvorova grafa, a cilj nam je minimizirati ukupnu cijenu svih puteva od svakog trgovca. Možemo ga formalno definirati kao minimizaciju izraza:

$$\sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$$

no uz dodatan uvjet da svi polaze i završavaju u 1. čvoru:

$$\sum_{j=1}^{n} x_{1j} = m$$

$$\sum_{i=1}^{n} x_{i1} = m$$

I dalje zadržavamo zahtjev da za svaki čvor $v \neq 1$ vrijedi da ima točno dva brida. Također se još uz navedeno postavlja i ograničenje za eliminaciju podtura koje sprečava cikluse (služi za izbjegavanje generiranih nevaljanih rješenja koja sadrže podture koje ne obuhvaćaju ishodišni čvor trgovaca):

$u_i - u_j + P \cdot X_{ij} \leq P - 1$

u_i predstavlja broj gradova od ishodišnog do i-tog grada u turi trgovca.

Dodatno, često se postavljaju zahtjevi i na minimizaciju razlike između najduljeg i najkraćeg puta kako bi trgovci proveli što manje vremena putujući. Tada to postaje višekriterijska optimizacija gdje ne se ne traži samo najmanja cijena, već i ujednačavanje opterećenja trgovaca što je u stvarnim primjenama često krucijalno pošto je vrijeme skupo pa se zadaci pokušavaju paralelizirati. Kod višekriterijske optimizacije pamte se sva nedominirana najbolja rješenja po nekom od kriterija (ona čine *Paretto* frontu). Primjer takvog algoritma je NSGA-II koji grupira rješenja u skupove iste dominantnosti.

2. Genetski algoritam

Najmanje istražena metoda za rješavanje problema više trgovačkih putnika su upravo genetski algoritmi gdje postoji još uvijek prostora za napredak. Genetski algoritam je metaheuristički algoritam inspiriran prirodnom selekcijom te spada u širu skupinu evolucijskih algoritama. Radi na način da populacija jedinki (fenotipi) koje se sastoje od kromosoma (opisnik rješenja, genotip) koji se križaju i mutiraju iterativnim postupkom selekcije. Inicijalna se populacija jedinki može stvoriti nasumično ili heuristikom, ovisno o domeni problema. Jedinke su određena reprezentacija gena koji se najčešće prikazuje kao kodirani vektor cijelih ili realnih brojeva (ili znakova). Značenje određenog elementa vektora je proizvoljno te treba težiti prema memorijski efikasnim reprezentacijama sa što manje redundantnih informacija kako bi populacija mogla biti što veća.

Vrste genetskih algoritama:

- *Generational* (GGA) - sve jedinke generacije se zamjenjuju novom generacijom
- *Steady-state* (SSGA) - najgora jedinka se zamjenjuje novonastalom djecom
- *Steady-generational* (SGGA) - zamjenjuje se nasumična (ne najbolja) jedinka populacije sa nastalim potomkom
- $(\mu + \mu)$-GA - stvara potomstvo 2-turnirskom selekcijom dok veličina nove populacije nije jednaka veličini roditeljske populacije te potom spoji te dvije populacije te filtrira pola najgorih jedinki

Nema strogih pravila oko odabira jednog od navedenih vrsta algoritama. Ponekad se bolja rješenja mogu dobiti kombinacijom ideja iz više algoritama što je upravo kod ovog problema i slučaj. Empirijskim je testiranjem utvrđeno da generacijski algoritam s k-elitizmom (k najboljih rješenja uvijek prelazi u novu generaziju) daje dobra rješenja.

Druga važna komponenta genetskog algoritma je evaluacijska funkcija koja određuje kvalitetu svake pojedine jedinke populacije. Temeljem njenih povratnih vrijednosti algoritam donosi odluku o odabiru jedinka koje će sudjelovati u stvaranju nove populacije za sljedeću iteraciju evolucijske petlje. Neki od poznatih strategija selekcije su:

- *roulette wheel selection* - vjerojatnost odabira je proporcionalna s dobrotom jedinke
- *k-tournament selection* - odabir temeljem usporedbe s k drugih nasumično izvučenih jedinki iz populacije (može biti s ponavljanjem i bez ponavljanja)
- *rank selection* - selekcija proporcionalna

Algoritam evaluacijsku funkciju koristi kao crnu kutiju te istražuje prostor rješenja negradijentnim metodama za razliku od strojnog učenja. *Fitness function* može imati skalarnu povratnu vrijednost ili vektorsku (tada govorimo o višekriterijskoj optimizaciji gdje se primjenjuje *paretto* fronta kod traženja najboljih rješenja).

Evolucijska petlja može se opisati sljedećim pseudokodom:

```
for i in n_generations:
    newPop = []
    newPop.add(best(population)) // elitism
    while newPop.size( ) < population.size( ):
        p1, p2 = select(2, pop)
        child  = crossover(p1, p2)
        if uniform_rand(0, 1) < mutation_prob:
            child = mutate(child)
        newPop.add(child)
```

Nakon koraka selekcije roditelja iz populacije, treba odrediti strategiju križanja (*crossover*) kako bi se rekombinirao kvalitetan genetski materijal te stvorilo novo rješenje koje potencijalno sadrži bolje dijelove iz oba pretka. Neke od poznatijih metoda križanja su:

- *single-point crossover* - određuje se alel u kromosomu te se dio prije te točke nasljeđuje iz prvog roditelja, a drugi iz drugog roditelja (nastaju 2 potomka)
- *two-point crossover* - slično kao i jedna točka prekida, samo se naizmjenično uzmu segmenti kromosoma roditeljskih jedinki
- *exponential crossover* - nasumično se odabere točka te duljina segmenta (*burst*) koji slijedi prati eksponencijalnu distribuciju (kraći nizovi su eksponencijalno vjerojatniji)
- *uniform crossover* - jednaka je vjerojatnost za svaki alel da bude odabran iz bilo kojeg od roditelja
- *OX crossover* - korišten u rješavanju problema više trgovačkih putnika, uzima dio alela iz poretka permutacije u kromosomu prvog roditelja te iz drugog roditelja uzima alele koji se ne nalaze u tom skupu prvog roditelja s idejom da se očuva poredak jer on kodira kvalitetu rješenja.

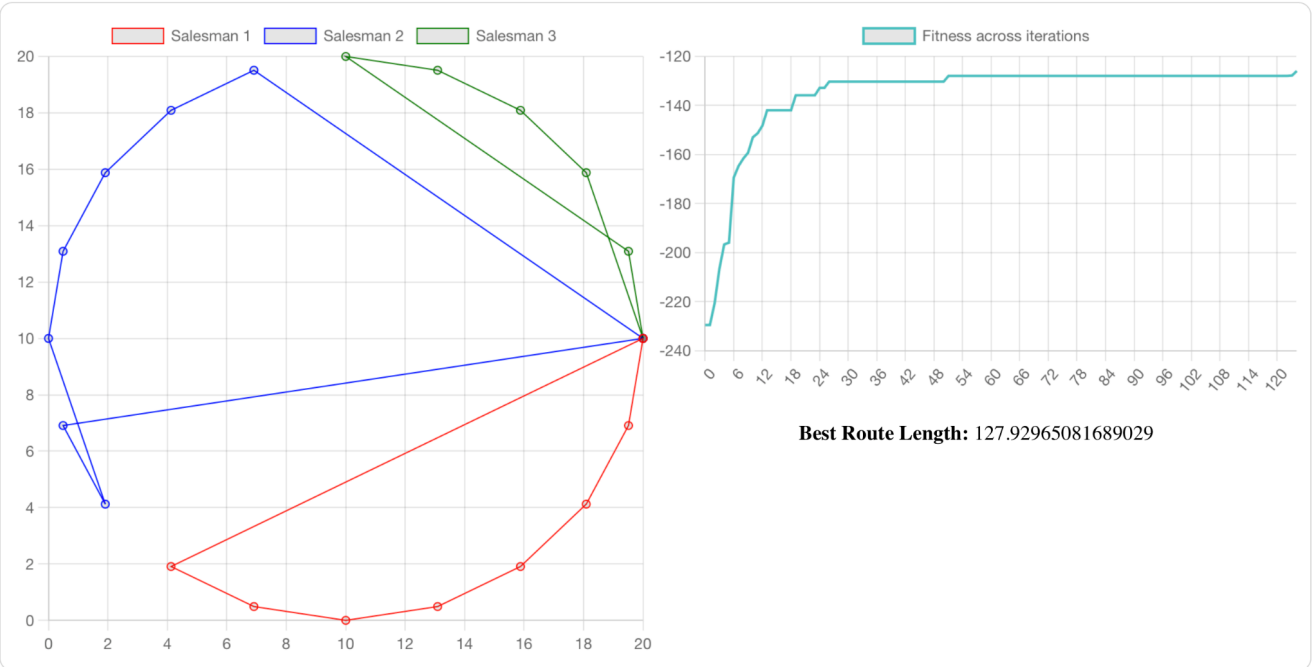
Konkretno, možemo se koristiti inicijalizacijom populacije na redosljed gradova *sweep-line* metodom koja se rotira oko ishodišnog čvora te svaki čvor ima određenu vjerojatnost da pri presjeku s linijom bude uključen u turu koja se gradi. Heuristike za za generiranje početne populacije treba vrlo rezervirano koristiti da se ne smanji genetska raznolikost (sužen potencijalni prostor pretraživanja u potrazi za globalnim optimumom u prostoru rješenja). Stoga, za očuvanje raznolikosti, pola populacije se generira potpuno nasumičnom permutacijom i podjelom gradova, a polovica sa *sweep-line* metodom.

3. Vizualizacija rješenja kroz web sučelje

Implementirano je sučelje koje uživo prati statistike algoritma koji se izvodi u backend-u. Korišten je Javascript Websocket API koji je podržan u većini modernih web preglednika. Taj API podržava bidirekcijsku vezu (TCP socket) između klijenta i poslužitelja te koristi poseban ws:// protokol definiran RFC 6455 kao proširenje HTTP protokola koji se naznačuje za zaglavljem `Upgrade: websocket`., a poslužitelj (ako podržava taj protokol) odgovara sa statusom `101` i porukom `Switching protocols`. Kao takav, ubraja se u protokol 7. sloja OSI modela.

Razlog za korištenje WebSocket sučelja je učinkovitost u razmjeni podataka u stvarnom vremenu. Alternativa korištenju WebSocket sučelja bila bi periodičko poliranje dodatnim upitima s klijenta što je iznimno neučinkovito i nepouzdano za *real-time* podatke. Valja napomenuti da postoji i SSE mehanizam (Server Sent Events), no on nažalost nije ostvario toliku podržanost u svim web preglednicima. Zanimljivost WebSocket sučelja je i da je moguće razmjenjivati i binarne poruke umjesto tekstualnih što može biti važno za neke aplikacije.

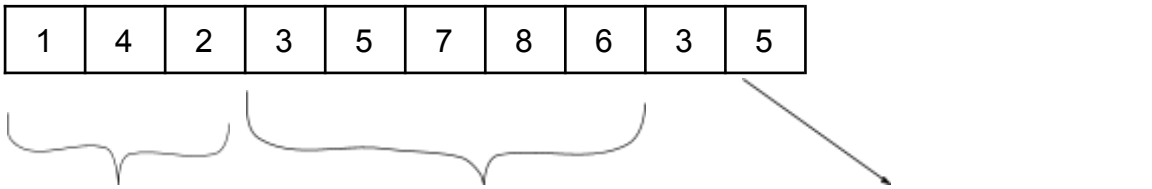
Backend poslužitelja implementiran je u programskom jeziku Java koristeći minimalni radni okvir Javalin koji ima jednostavnu podršku za WebSocket-e. Pri povezivanju, klijenti šalju parametre za pokretanje mTSP algoritma. mTSP algoritam implementiran je kroz obrazac *Observable*. Moguće je pretplatiti slušatelja (*Consumer<T>*) na evolucijsku petlju te slušati na događaj završetka jedne generacije s potrošačem koji je zapravo spomenuta krajnja točka Javalin web poslužitelja. S obzirom na to da je izvođenje petlje algoritma iznimno brzo, potrošač rezultata je konfigurabilan te može birati frekvenciju primanja novog najboljeg rješenja evolucijske petlje. To je implementirano da se izbjegne opterećenje klijenta s prečestim podacima i zasićenje mreže.



4. Rješavanje problema genetskim algoritmom

Ova je implementacija genetskog algoritmima hibridni je pristup algoritama iz [1] i [2]. Genotip je kodiran pomoću jednog kromosoma koji sadrži informaciju o permutaciji gradova iza kojih slijedi m brojeva. Za stvaranje inicijalne populacije korišten je potpuno nasumičan generator permutacija. Pomoću **`Collections.shuffle()`**. Za raspoređivanje gradova trgovcima trebalo bi rastaviti broj na m nasumičnih sumanada čija je suma jednaka broju gradova. Dodatno, potrebno je pri rastavi paziti na ograničenja **`minCities`** i **`maxCities`** jer svaki od trgovaca mora posjetiti broj gradova između ta dva broja.

Kromosom je kodiran kao što je prikazano na skici:



Iako autori navode da je korištenje *sweep-line* heuristike, korišteno je potpuno nasumično generiranje permutacija u genotipu inicijalne populacije.
Algoritam je generacijski, odnosno, potomstvo se ne miješa s roditeljskom populacijom već se sve jedinke zamjene s novima u sljedećoj generaciji.

Korišten je ranije spomenuti operatori križanja *ordered crossover* čiji algoritam detaljnije možemo opisati sljedećim pseudokodom:

- Odaberi nasumičan i, j tako da 0 < i < j < n kromosoma prvog roditelja
- stvori set S = kromosom[i:j]
- za svaki alel A (oznaka grada) drugog roditelja
    - ako je u A element iz S
        - nastavi petlju
    - inače
        - kopiraj alel A u prvu neokupiranu ćeliju potomka

Ideja iza korištenja poredanog križanja jest djelomično očuvanje poretka alela iz oba roditelja što u slučaju preuzimanja lokalno boljih dijelova (kraćih dijelova ruta) iz svakog od njih dovodi do potomstva s još većom dobrotom.

Izvorni autori algoritma navode čak dvije kritične heuristike koje daju rješenje iznimno blizu optimalnom (često čak i optimalno): 2-opt algoritam za razrješavanje bridova s presjecištima te sweep-line za generiranje inicijalne populacije, oni nisu uobičajene prakse kod genetskih algoritama. Lako je vidljivo zašto su u ovom problemu od velike pomoći. Važno je napomenuti da implementirani algoritam nema ugrađenu heuristiku koja bi ga navodila da pretražuje “kružna rješenja” već je funkcija dobrote za njega crna kutija te ju poziva samo kako bi donio odluku o sljedećem području rješenja koje će istražiti.

5. Kvaliteta rješenja

S obzirom na stohastičnost evolucijskih algoritama, za očekivati je da će se rješenje razlikovati između pokretanja. Testiranjem je utvrđeno da je algoritam gotovo uvijek pronalazio optimum za 10 gradova, dok je za ~80 gradova nalazio vrlo dobro rješenje. U tablici su navedene neke izmjerene omjeri duljina pronađene ture i optimalne ture. Pošto je problem konstruiran tako da je analitički vrlo lako odrediti globalno optimalni raspored gradova po trgovcima, korištena je ta prednost za računanje sljedećih omjera:

m / n	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**
1	1.0	1.48	1.39	1.15	1.2	1.77	1.91	1.42
2	1.0	1.98	2.39	2.55	2.07	3.184	3.5	3.5
3	1.0	2.4	2.55	3.18	2.87	3.18	3.82	3.66

U početnim iteracijama vrlo brzo supstituira duge bridove s kraćim perifernim bridovima te vrlo duge tetive brzo nestaju iz populacije. Algoritam nema eksplicitno implementiranu metodu filtriranja (ne poznaje koncept udaljenosti između gradova, samo ukupni *fitness*) loših konfiguracija te su stoga ovi rezultati pokazuju skalabilnost genetskih algoritama čak i na velike ulazne primjere NP-potpunih problema.

Zanimljivo je spomenuti da kod optimalnog rješenja za problem s gradovima na obodu kružnice očekujemo da će gradovi raspoređeni po trgovcima biti klasterirani u kontinuiranim krivuljama na obodu kružnice bez presjecanja bridova. I općenitije, pravilo o nepresjecanju uvijek vrijedi za optimalno rješenje bilo kakvog rasporeda čvorova u grafu te se može koristiti kao ranije spomenuta 2-opt heuristika. Loša strana 2-opt heuristike je nepredvidiva gornja ograda složenosti $$O(n^3)$$ složenost no u praksi završava s otpetljanom turom u puno manje koraka.

Zbog toga što se koristi OX križanje, ono se može zamisliti kao da se odabire način da se za ovaj specifičan specifičan graf u obliku kružnice podijeli na dva segmenta te se uzme jedan segment iz prvog a drugi iz drugog roditelja. To dovodi do stagnacije u lokalnim optimumima gdje populacija ne nalazi rješenje za problem kada svi gradovi trgovca nisu klasterirani u kontinuirani dio oboda kružnice, dok, naspram tome, vrlo lako optimira raspored gradova unutar ture jednog trgovca kada su klasterirani.

Uzmemo li u obzir dodatno ograničenje da optimalno rješenje sigurno nema presjecišta između bridova, jer se tada uvijek može konstruirati kraći put prema nejednakosti trokuta. Algoritam bi se mnogo lakše naveo na optimalno rješenje “genetskim inženjeringom”.

6. Zaključak i daljnji rad

Genetski algoritmi su se pokazali kao izvrsna metaheuristika za rješavanje problema za koje ne postoje efikasne determinističke metode. Zbog svoje općenitosti (može se kodirati bilo kakav gen kromosomima), primjenjivi su u širokom skupu domena. Vrlo su slabo istraženo područje zbog samog spleta opcija manjih dijelova koji čine cjelokupni algoritam. Prema *no free lunch teoremu*, najbolji izbor strategija evolucijskog algoritma domenski je specifičan i mnogo pristupa u evolucijskom računarstvu su otvoreno pitanje. Komplementarni je pristup sa strojnim učenjem i dubokim neuronskim mrežama te se zajedno nadomještaju slabosti svakog pojedinačnog pristupa.

U planu je implementirati nekolicinu drugih genetskih algoritama koji se mogu parametrizirano izvoditi te vizualno pratiti kvaliteta rješenja i/ili brzina konvergencije za različite strategije. Projekt se potencijalno može pretvoriti u edukativnu platformu za izučavanje ponašanja genetskih algoritama.

Literatura:

1. Lo, Kin-Ming, et al. ‘A Genetic Algorithm with New Local Operators for Multiple Traveling Salesman Problems’: *International Journal of Computational Intelligence Systems*, vol. 11, no. 1, 2018, p. 692. *DOI.org (Crossref)*, https://doi.org/10.2991/ijcis.11.1.53.
2. 2. Jenkins, Alison, et al. ‘Variations of Genetic Algorithms’. *ArXiv:1911.00490 [Cs]*, Nov. 2019. *arXiv.org*, http://arxiv.org/abs/1911.00490.
3. Englert, Matthias, et al. ‘Worst Case and Probabilistic Analysis of the 2-Opt Algorithm for the TSP’. *Algorithmica*, vol. 68, no. 1, Jan. 2014, pp. 190–264. *DOI.org (Crossref)*, https://doi.org/10.1007/s00453-013-9801-4.
4. https://jenetics.io/manual/manual-7.0.0.pdf