**PRIRODOSLOVNO-MATEMATIČKI FAKULTET   
SVEUČILIŠTE U SPLITU**

**(GENETSKI ALGORITAM & ALGORITAM SIMULIRANOG KALJENJA)**

**PROBLEM TRGOVAČKOG PUTNIKA**

Matea Vučemilović-Vranjić

**Studijska grupa:** Informatika

**Predmet**: Algoritmi u primjeni

**Ak. god.**: 2022./2023.

**Nastavnik**: prof.dr.sc. Divna Krpan

Split, lipanj 2023.

Sadržaj

[1. UVOD 1](#_Toc140103604)

[2. PRIMJENA 1](#_Toc140103605)

[3. GENETSKI ALGORITAM 2](#_Toc140103606)

[3.1. IMPLEMENTACIJA U C# 3](#_Toc140103607)

[4. ALGORITAM SIMULIRANOG KALJENJA 6](#_Toc140103608)

[4.1. IMPLEMENTACIJA U C# 6](#_Toc140103609)

[5. USPOREDBA ALGORITAMA 9](#_Toc140103610)

[6. ZAKLJUČAK 11](#_Toc140103611)

[7. LITERATURA 11](#_Toc140103612)

# UVOD

Kroz ovaj projekt bavit ću se tzv. **Problemom trgovačkog putnika** (engl. Traveling Salesman Problem, TSP), opisat ću dva algoritma za rješavanje ovog problema te ih implementirati u C#-u. Također ću na pojedine načine usporediti njihovo djelovanje i optimalnost. Navedeni algoritmi biti će Genetski algoritam (engl. **Genetic Algorithm**, GA) i algoritam Simuliranog kaljenja (engl. **Simulated Annealing**, SA).

Traveling Salesman Problem je matematički problem koji se javlja u području kombinatorne optimizacije. Ako imate popis gradova koje želite posjetit, osnovna ideja problema je pronaći najkraći mogući put koji će vas voditi kroz sve gradove jednom, prije nego se vratite na početnu točku.

Formalna definicija iz kombinatorike:

*Neka imamo skup od n gradova i matricu udaljenosti koja prikazuje udaljenost između svakog para gradova. Cilj je pronaći Hamiltonov ciklus, tj. zatvoreni put koji prolazi kroz svaki grad točno jednom, s minimalnom ukupnom udaljenosti.*

TSP je poznat kao NP-težak problem, što znači da ne postoji poznati algoritam koji će uvijek pronaći najkraći put u razumnom vremenskom roku za sve ulaze. Međutim, postoji nekoliko algoritama koji daju dobra rješenja za mnoge primjene TSP-a. Neki od takvih algoritama osim navedenih su: Ant colony optimization algorithms, Brute force, Nearest Neighbor…

TSP ima mnoge primjene u stvarnom svijetu, poput optimizacije rute u dostavi, raspoređivanju vozila, projektiranju tiskanih pločica, raspoređivanju zrakoplova i mnogim drugim područjima gdje je potrebno pronaći najkraći put koji prolazi kroz skup lokacija.

# PRIMJENA

Neki od primjera primjene Traveling Salesman Problema u svakodnevnom životu su:

* **Rutiranje dostave**

Kompanije koje se bave dostavom često se suočavaju s izazovom optimiziranja ruta svojih vozača kako bi minimalizirale ukupno vreme i troškove. TSP se može primijeniti kako bi se pronašao najkraći put za svakog vozača koji obuhvaća sve točke isporuke.

* **Planiranje putovanja**

Kada planirate višednevno putovanje i imate listu destinacija koje želite posjetiti, TSP može vam pomoći da organizirate redoslijed posjeta na način koji minimizira ukupnu udaljenost i optimizira vaše vrijeme putovanja.

* **Planiranje rute za prodaju**

Prodavači koji imaju više sastanaka s klijentima u različitim gradovima ili područjima mogu koristiti TSP za planiranje najbolje rute koja minimizira putne troškove i vrijeme provedeno na putu.

* **Rutiranje vozila u logistici**

Kompanije koje upravljaju flotom vozila, kao što su kurirske službe ili prijevoznici, mogu koristiti TSP za optimalno rutiranje vozila. Cilj je minimizirati ukupnu udaljenost koju vozila trebaju prijeći kako bi ispunila sve potrebe dostave.

* **Elektroničko projektiranje integriranih krugova**

U području dizajna IC-a, TSP se može koristiti za optimalno rutiranje veza između komponenata na čipu. Cilj je minimizirati ukupnu duljinu veza kako bi se poboljšala brzina i performanse IC-a.

# GENETSKI ALGORITAM

Genetski algoritam je metaheuristička tehnika inspirirana principima evolucije, prirodne selekcije i genetike koja se često koristi za rješavanje kombinatornih optimizacijskih problema poput TSP-a.

Osnovni koraci:

1. Inicijalizacija populacije – sastoji se od jedinki ili rješenja problema, svaka jedina je predstavljena genetski kodom
2. Evaluacija fitnessa – svaka jedinka u populaciji ocjenjuje se na temelju njezine kvalitete, što se naziva fitness, mjeri koliko dobro jedinka rješava problem
3. Selekcija – odabire roditelje iz populacije koji će se koristiti za stvaranje nove generacije
4. Križanje – proces kombiniranja genetskog materijala roditelja kako bi se stvorila nova potomstva
5. Mutacija – slučajna promjena u genetskom kodu jedinke, služi za održavanje raznolikosti populacije
6. Zamjena – nova potomstva zamjenjuju lošije jedinke iz prethodne generacije, osiguravajući kontinuiranu evoluciju
7. Iteracija – koraci 2-6 ponavljaju se kroz više generacija dok se ne postigne određeni uvjet zaustavljanja (npr. maksimalni broj generacija, vremenski uvjet…)

Genetski algoritam koristi prirodni proces selekcije, križanja i mutacije kako bi pretražio prostor rješenja TSP-a (ili nekog drugog problema) i postupno konvergirao prema boljim rješenjima. Kroz iteracije, populacija se poboljšava sve dok se ne dobije zadovoljavajuće rješenje ili dok ne bude dostignut zadani kriterij zaustavljanja. Važno je napomenuti da genetski algoritmi nisu zajamčeni da će pronaći globalno optimalno rješenje TSP-a, ali mogu pružiti dobra približna rješenja u razumnom vremenskom okviru. Osim toga, genetskim algoritmom možemo riješiti neke od sljedećih problema: optimizacija rasporeda (raspored ispita, smjena i slično), rutiranje vozila, planiranje resursa, strojno učenje, financijsko modeliranje (upravljanje rizikom, optimizacija ulaganja)…

## IMPLEMENTACIJA U C#

* Prvo sam definirala klasu *Grad* (koju ćemo koristiti za oba algoritma, ona predstavlja gradove sa svojim koordinatama i nekim indeksom)

Slika na kojoj se prikazuje tekst, snimka zaslona, softver, Multimedijski softver

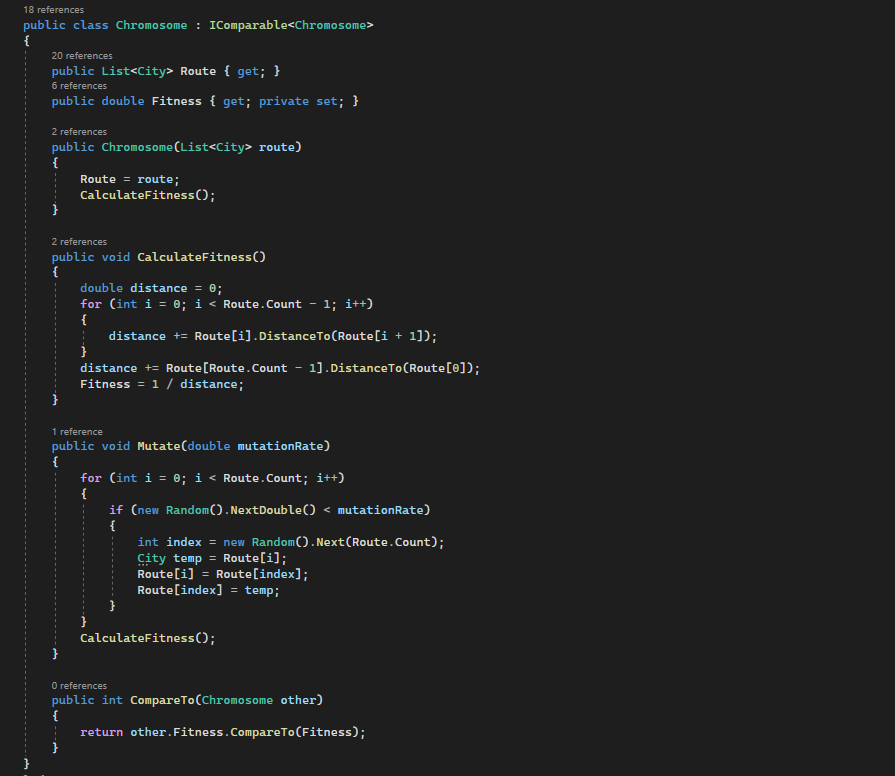
Opis je automatski generiran

* Definiramo klasu *TSP* koja definira sami problem

Slika na kojoj se prikazuje tekst, snimka zaslona, Font

Opis je automatski generiran

* Definiramo klasu *Chromosome* koja predstavlja moguće rješenje problema, sadrži listu *Route* gradova koji predstavljaju rutu i metode *CalcuateFitness()* koja računa fitness koji je objašnjen gore i metodu *Mutate()* za izvođenje mutacije



* Sada implementiramo genetski algoritam u klasi GA, koja će se baviti populacijom, selekcijom, križanjem i mutacijom

Slika na kojoj se prikazuje tekst, snimka zaslona

Opis je automatski generiran

Slika na kojoj se prikazuje tekst, snimka zaslona, softver

Opis je automatski generiran

Slika na kojoj se prikazuje tekst, snimka zaslona

Opis je automatski generiran

# ALGORITAM SIMULIRANOG KALJENJA

SA je podvrsta stohastičkih optimizacijskih algoritama. Algoritam kreće od jednog početnog rješenja te postojeće rješenje zamjenjuje boljim, ali ga može zamijeniti i sa lošijim, uz određenu vjerojatnost prihvaćanja. Vjerojatnost prihvaćanja lošijeg rješenja opada kako algoritam napreduje. Koristi se uobičajeno u problemima gdje je prostor pretraživanja diskretan. Inspiracija za ime dolazi od kaljenja u metalurgiji. Tehnika je to koja koristi zagrijavanje te kontrolirano hlađenje materijala kako bi se povećali njegovi kristali i smanjili njihovi nedostatci. Za analogiju rada algoritma s fizikalnim zakonima trebamo potrebno je staviti stvari u ispravan kontekst. Svaki korak rada simuliranog kaljenja zamjenjuje trenutno rješenje za nekim, njemu „bliskim“, slučajno odabranim rješenjem. Vjerojatnost odabira zamjenskog rješenja ovisi o tome kolika je razlika u energetskoj razini (prikladnosti, dobroti) rješenja kandidata i trenutačnog rješenja. Također ta vjerojatnost ovisi i o globalnom parametru T, koji označava temperaturu i postepeno se smanjuje tijekom rada algoritma kroz višestruke iteracije. Ovisnost je takva da se trenutačno rješenje zamjenjuje gotovo uvijek kada je T velik, neovisno je li zamjena pozitivno ili negativno utjecala na kvalitetu rješenja. Kada je T malen i približava se nuli, tada se pri zamjenama sve više inzistira da zamjensko rješenje bude uvijek bolje od trenutačnog rješenja. Početno dopuštanje uzlaznog trenda dobrote rješenja sprječava zaglavljivanje rješenja u lokalnom minimumu što je glavni problem kod pohlepnih algoritama.

## IMPLEMENTACIJA U C#

* U klasi *SimuliranoKaljenje* definiramo konstruktor koji prima tri parametra: listu gradova, početnu temperaturu, stopu hlađenja i instancu klase Random

Slika na kojoj se prikazuje tekst, snimka zaslona

Opis je automatski generiran

* Metoda *PronadiNajkraciPut* pokreće algoritam i vraća rezultirajuću putanju

Slika na kojoj se prikazuje tekst, snimka zaslona

Opis je automatski generiran

* Metoda *GenerateRandomPath* stvara početnu putanju tako što kopira gradovi u novu listu i miješa redoslijed gradova pomoću metode *Shuffle*.
* Metoda *GenerateNeighborPath* stvara susjednu putanju tako što kopira trenutnu putanju (path) u novu listu, a zatim zamjenjuje dva slučajno odabrana grada u toj putanji.

Slika na kojoj se prikazuje tekst, snimka zaslona, softver

Opis je automatski generiran

* Metoda *CalculateDistance* računa ukupnu udaljenost putanje tako što iterativno zbraja udaljenosti između svakog para gradova u putanji. Dodaje se i udaljenost između posljednjeg grada i prvog grada da bi se dobila zatvorena putanja.
* Metoda *CalculateTotalDistance* je slična metodi *CalculateDistance*, ali prima putanju kao argument umjesto da radi na trenutnoj putanji.

Slika na kojoj se prikazuje tekst, snimka zaslona, softver

Opis je automatski generiran

* Metoda *AcceptanceProbability* izračunava vjerojatnost prihvaćanja nove putanje na temelju trenutne udaljenosti (currentDistance), nove udaljenosti (newDistance) i temperature (temperature).
* Ako je nova udaljenost manja od trenutne udaljenosti, vraca se 1, što znači da je nova putanja bolja i da ju treba prihvatiti.
* Inače, izračunava se vjerojatnost prihvaćanja na temelju razlike u udaljenostima i temperature koristeći izraz iz algoritma simuliranog kaljenja.
* Metoda Shuffle služi za miješanje elemenata u listi. Koristi se algoritam Fisher-Yates za miješanje elemenata. Svaki element na poziciji i zamjenjuje se s nasumično odabranim elementom iz preostalih elemenata.

Slika na kojoj se prikazuje tekst, snimka zaslona, softver

Opis je automatski generiran

# USPOREDBA ALGORITAMA

Kako bih odredila koji je od navedenih algoritama „bolji“ ispitala sam ih na temelju dvije liste gradova. Algoritmi su dali svoju optimalnu rutu obilaska svih gradova te prijeđenu udaljenost. Također sam ispitala i brzinu izvođenja algoritama. Smatra se da je algoritam to bolji što mu je brže izvođenje i što pokazuje kraću udaljenost među gradovima. Rezultati su sljedeći:

1. Slika na kojoj se prikazuje tekst, snimka zaslona, Font

   Opis je automatski generiran

Ispis:

Slika na kojoj se prikazuje tekst, elektronika, snimka zaslona, računalo

Opis je automatski generiran

Oba algoritma pokazala su nam istu rutu obilaska gradova (3, 1, 2, 4, 5), što znači da je i udaljenost ista (17,6632…). Međutim, kao što vidimo na slici, algoritam simuliranog kaljenja brže se izvršio.

1. Slika na kojoj se prikazuje tekst, snimka zaslona, Font

   Opis je automatski generiran

Ispis:

Slika na kojoj se prikazuje tekst, snimka zaslona, Font, dizajn

Opis je automatski generiran

Rezultati nakon drugog pokretanja programa:

Slika na kojoj se prikazuje tekst, snimka zaslona

Opis je automatski generiran

Kod drugog primjera htjela sam istaknuti kako algoritmi ne daju uvijek isto rješenje te kako sami algoritmi mijenjaju svoja rješenja kada ponovo pokrenemo program.

Evo nekoliko razloga zašto algoritmi mogu davati različita rješenja:

* Mnogi optimizacijski algoritmi koriste elemente nasumičnosti, poput slučajnog odabira, križanja, mutacije ili inicijalizacije. Ovi slučajni elementi mogu dovesti do različitih rezultata prilikom izvođenja algoritma, čak i ako koristimo iste ulazne podatke.
* Početno stanje ili inicijalno rješenje koje se koristi u algoritmu može utjecati na konačno rješenje. Različiti početni uvjeti mogu dovesti do različitih putanja pretraživanja prostora rješenja i rezultirati različitim konačnim rješenjima
* Parametri koji se koriste u algoritmu, poput veličine populacije, vjerojatnosti križanja ili mutacije, stope hlađenja, mogu imati utjecaj na ponašanje algoritma i rezultirati različitim rješenjima. Promjene u parametrima mogu dovesti do različitih razina istraživanja i iskorištavanja prostora rješenja
* Stohastički algoritmi: Neki algoritmi, poput genetskog algoritma ili simuliranog kaljenja, koriste stohastičke procese tijekom izvođenja. To znači da će svako izvođenje algoritma dati malo drugačije rezultate. Više iteracija ili ponavljanja algoritma može dati različite rezultate.

# ZAKLJUČAK

Simulirano kaljenje (SA) i Genetski algoritam (GA) su heuristički algoritmi koji koriste različite strategije za rješavanje optimizacijskih problema poput problema putujućeg trgovca (TSP). Oba algoritma imaju svoje prednosti i mane te performanse mogu varirati ovisno o karakteristikama problema. Važno je napomenuti da je rezultat algoritama uvijek specifičan za konkretni problem i konkretnu konfiguraciju algoritama. Izvođenje više eksperimenata s različitim parametrima i ulaznim podacima može pružiti bolji uvid u brzinu izvršavanja i performanse SA i GA algoritama za određeni problem.

# LITERATURA

<https://visualstudiomagazine.com/articles/2022/12/20/traveling-salesman-problem.aspx>

<https://bilenmehmet.com/gonderi/traveling_salesman_genetic_algorithms_cSharp>

<https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7>

<https://jamesmccaffrey.wordpress.com/2021/12/01/the-traveling-salesman-problem-using-simulated-annealing-in-csharp/>