Umelá Inteligencia

Travelling Salesman Problem

Zadanie č.2

Norbert Matuška; xmatuskan@stuba.sk
11-11-2023

Contents

Norbert Matuška
xmatuskan@stuba.sk

Zadanie

Obchodný cestujúci má navštíviť viacero miest. V jeho záujme je minimalizovať cestovné náklady a cena prepravy je úmerná dĺžke cesty, preto snaží sa nájsť najkratšiu možnú cestu tak, aby každé mesto navštívil práve raz. Keďže sa nakoniec musí vrátiť do mesta z ktorého vychádza, jeho cesta je uzavretá krivka.

Je daných aspoň 20 miest (20 – 40) a každé má určené súradnice ako celé čísla X a Y. Tieto súradnice sú náhodne vygenerované. (Rozmer mapy môže byť napríklad 200 * 200 km.) Cena cesty medzi dvoma mestami zodpovedá Euklidovej vzdialenosti – vypočíta sa pomocou Pytagorovej vety. Celková dĺžka trasy je daná nejakou permutáciou (poradím) miest. Cieľom je nájsť takú permutáciu (poradie), ktorá bude mať celkovú vzdialenosť čo najmenšiu.

Riešenia

Genetický algoritmus

Genetická informácia je reprezentovaná vektorom, ktorý obsahuje index každého mesta v nejakom poradí (nejaká permutácia miest). Keďže hľadáme najkratšiu cestu, je najlepšie vyjadriť fitnes jedinca ako prevrátenú hodnotu dĺžky celej cesty.

Jedincov v prvej generácii inicializujeme náhodne – vyberáme im náhodnú permutáciu miest. Jedincov v generácii by malo byť tiež aspoň 20. Je potrebné implementovať aspoň dve metódy výberu rodičov z populácie.

Kríženie je možné robiť viacerými spôsobmi, ale je potrebné zabezpečiť, aby vektor génov potomka bol znovu permutáciou všetkých miest. Často používaný spôsob je podobný dvojbodovému kríženiu. Z prvého rodiča vyberieme úsek cesty medzi dvoma náhodne zvolenými bodmi kríženia a dáme ho do potomka na rovnaké miesto. Z druhého rodiča potom vyberieme zvyšné mestá v tom poradí, ako sa nachádzajú v druhom rodičovi a zaplníme tým ostatné miesta vo vektore.

Mutácie potomka môžu byť jednoduché – výmena dvoch susedných miest alebo zriedkavejšie používaná výmena dvoch náhodných miest. Tá druhá výmena sa používa zriedkavo, lebo môže rozhodiť blízko optimálne riešenie. Často sa však používa obrátenie úseku – znova sa zvolia dva body a cesta medzi nimi sa obráti. Sú možné aj ďalšie mutácie, ako napríklad posun úseku cesty niekam inam.

Simulované žíhanie (simulated annealing)

Simulované žíhanie patrí do skupiny algoritmov, ktoré využívajú na hľadanie riešenia v priestore možných stavov lokálne vylepšovanie. Zároveň sa algoritmus snaží zabrániť uviaznutiu v lokálnom extréme. Z aktuálneho uzla si algoritmus klasicky vytvorí nasledovníkov. Potom si jedného vyberie. Ak má zvolený nasledovník lepšie ohodnotenie, tak doň na 100% prejde. Ak má nasledovník horšie ohodnotenie, môže doň prejsť, ale len s pravdepodobnosťou menšou ako 100%. Ak ho odmietne, tak skúša ďalšieho nasledovníka. Ak sa mu nepodarí prejsť do žiadneho z nich, algoritmus končí a aktuálny uzol je riešením. Pre nájdenie globálneho extrému je dôležitý správny rozvrh zmeny pravdepodobnosti výberu horšieho nasledovníka. Pravdepodobnosť je spočiatku relatívne vysoká a postupne klesá k nule.

Problém je opäť reprezentovaný vektorom, ktorý obsahuje index každého mesta v nejakom poradí (nejaká permutácia miest). Nasledovníci sú vektory, v ktorých je vymenené poradie niektorej dvojice susedných uzlov.

Dôležitým parametrom tohto algoritmu je rozvrh zmeny pravdepodobnosti výberu horšieho nasledovníka. Príliš krátky (rýchly) rozvrh spôsobí, že algoritmus nestihne obísť lokálne extrémy, príliš dlhý rozvrh natiahne čas riešenia, lebo bude obiehať okolo optimálneho riešenia. Je potrebné nájsť vhodný rozvrh.

Riešenia

Genetický algoritmus

Princíp genetického algoritmu

Genetický algoritmus je heuristické riešenie, ktoré napodobňuje proces prirodzeného výberu jedincov na generovanie kvalitných výsledkov pri hľadaní. Ako to funguje v mojom kóde si povieme v nasledovných odstavcoch.

Použitím mechanizmov inšpirovaných biologickou evolúciou, ako je selekcia, kríženie a mutácia, genetický algoritmus v mojom kóde efektívne hľadá optimálne alebo takmer optimálne riešenie pre TSP. Tento prístup je užitočný najmä pri problémoch, kde je vyhľadávací priestor veľký a zložitý, ako v prípade problému cestujúceho obchodníka.

Inicializácia

Moje riešenie najprv vygeneruje množinu miest, každé s náhodnými súradnicami. Tieto mestá sú reprezentované ako body v dvojrozmernom priestore o veľkosti 200x200km. Následne sa vytvorí populácia možných riešení (trás). Každý jednotlivec v tejto populácii predstavuje kompletnú trasu, ktorá raz navštívi každé mesto. Trasa je znázornená ako postupnosť indexov miest.

Fitness funkcia

Výpočet vzdialenosti: Celková vzdialenosť trasy sa vypočíta sčítaním euklidovských vzdialeností medzi po sebe nasledujúcimi mestami na trase. Táto vzdialenosť slúži ako miera kvality riešenia – čím kratšia vzdialenosť, tým lepšie riešenie.

Fitness skóre: Fitness jednotlivca (trasa) sa vypočíta ako prevrátená hodnota celkovej vzdialenosti. Vyššie hodnoty fitness zodpovedajú kratším trasám.

```
def euclidean_distance(city1, city2):
    return ((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1]) ** 2) ** 0.5

def calculate_total_distance(route):
    total_distance = 0
    num_cities = len(route)

    for i in range(num_cities):
        current_city = cities[route[i]]
        next_city = cities[route[(i + 1) % num_cities]]
        distance = euclidean_distance(current_city, next_city)
        total_distance += distance

    return total_distance

def calculate_fitness(route):
    return 1 / calculate_total_distance(route)
```

Výber rodičov

Roulette wheel metóda

Najprv sa vypočíta celková fitness populácie a následne sa predelí fitness každého jedinca celkovou fitness populácie, podľa čoho sa budú vyberať rodičia. Jedinci s väčšou fitness majú väčšiu šancu pri výbere.

```
def roulette_wheel(fitness_scores):
    total_fitness = sum(fitness_scores)
    selection_probs = [f / total_fitness for f in fitness_scores]
    selected_parents = random.choices(population, weights=selection_probs,
k=2)
    return selected_parents
```

Tournament metóda

Náhodný výber *„k“* jedincov z ktorých sa vyberie ten s najväčšou fitness. Prakticky sa „pobijú“ a ten najsilnejší vyhráva.

```
def tournament_selection(fitness_scores, k=5):
    winners = []
    while len(winners) < 2:
        selected_indices = random.sample(range(len(population)), k)
        selected_fitness = [fitness_scores[i] for i in selected_indices]
        best_index =
selected_indices[selected_fitness.index(max(selected_fitness))]
        if population[best_index] not in winners:
            winners.append(population[best_index])
    return winners
```

Kríženie

Na pároch vybraných rodičov sa vykoná kríženie s cieľom splodiť potomka. Segment trasy sa skopíruje od jedného rodiča a zvyšok miest sa vyplní od druhého rodiča, pričom sa zachová poradie miest a zabezpečí sa, že každé mesto sa zobrazí práve jeden krát.

```
def crossover(parent1, parent2):
    size = len(parent1)
    child = [-1] * size
    start, end = sorted(random.sample(range(size), 2))
    child[start:end] = parent1[start:end]
    child_pos = end
    for gene in parent2[end:] + parent2[:end]:
        if gene not in child:
            if child_pos == size:
                child_pos = 0
            child[child_pos] = gene
            child_pos += 1
    return child
```

Mutácia

S malou pravdepodobnosťou môže potomok zmutovať. Mutácia sa vykonáva zámenou dvoch miest na trase, čo prináša do populácie variabilitu.

```
def mutate(route):
    idx1, idx2 = random.sample(range(len(route)), 2)
    route[idx1], route[idx2] = route[idx2], route[idx1]
    return route
```

Vytváranie novej populácie

Nová populácia pre ďalšiu generáciu sa vytvára prostredníctvom série krokov zahŕňajúcich selekciu, kríženie a mutáciu. Tento proces je ústredným prvkom GA, pretože jeho cieľom je vytvoriť nový súbor jednotlivcov (trás), ktoré sú, dúfajme, bližšie k optimálnemu riešeniu. Tento proces selekcie, kríženia a

mutácie sa opakuje, kým sa nenaplní nová populácia. Avšak, veľkosť novej populácie je o jedno menšia, keďže sa uplatňuje elitizmus. Aby sa zabezpečilo, že kvalita riešení neklesá z generácie na generáciu, najlepší jedinec zo súčasnej generácie sa vždy prenáša do ďalšej generácie. Elitizmus pomáha udržiavať dobrú úroveň fitness v populácii tým, že zachováva najlepšie doteraz nájdené riešenie.

```
for generation in range(1000):  # Number of generations
    fitness_scores = [calculate_fitness(individual) for individual in
population]

    avg_fitness = sum(fitness_scores) / len(fitness_scores)
    average_fitness.append(avg_fitness)
    for i, individual in enumerate(population):
        if fitness_scores[i] > best_fitness:
            best_fitness = fitness_scores[i]
            best_individual = individual

    new_population = []
    for _ in range(population_size - 1):
        if str(selection_method) == "1":
            parent1, parent2 = roulette_wheel(fitness_scores)
            child = crossover(parent1, parent2)
        elif str(selection_method) == "2":
            parent1, parent2 = tournament_selection(fitness_scores)
            child = crossover(parent1, parent2)
        else:
            print("Invalid selection method")
            break
        if random.random() < 0.1:  # Mutation probability
            child = mutate(child)
        new_population.append(child)

    new_population.append(best_individual)

    population = new_population

    highest_fitness_value = max(fitness_scores)
    highest_fitness.append(highest_fitness_value)
```

Konvergencia

Počas iterácií sa sleduje najlepšie nájdené riešenie a jeho skóre fitness. Algoritmus zvyčajne konverguje k riešeniu, čo je najkratšia nájdená cesta. Túto konvergenciu je následne možné vidieť na grafoch pri vizualizácii

Simulované žíhanie (Simulated annealing)

Princíp simulovaného žíhania

Časť kódu simulovaného žíhania (SA) poskytuje riešenie problému cestujúceho obchodníka (TSP) pomocou odlišného prístupu v porovnaní s genetickým algoritmom. Simulované žíhanie je optimalizačná technika, ktorá napodobňuje proces žíhania v metalurgii, techniku zahŕňajúcu zahrievanie a riadené chladenie materiálu na zväčšenie veľkosti jeho kryštálov a zníženie ich defektov.

Simulované žíhanie v tomto kóde efektívne skúma priestor riešení vyvážením prieskumu (prijímanie horších riešení) a využívania (zameriava sa na zlepšovanie súčasných riešení) na základe riadeného plánu

chladenia. Tento prístup je užitočný najmä pri problémoch s drsným prostredím riešení, pretože umožňuje algoritmu uniknúť z lokálneho optima a potenciálne nájsť globálne optimálne alebo takmer optimálne riešenie.

Výpočet vzdialenosti a fitness

Algoritmus vypočíta celkovú vzdialenosť aktuálnej trasy súčtom euklidovských vzdialeností medzi po sebe nasledujúcimi mestami. Táto vzdialenosť sa následne používa ako porovnanie lepších a horších riešení. Fitness vypočíta ako prevrátenú hodnotu vzdialenosti, ale tento výpočet používa len pre vizualizáciu v grafoch.

```
def calculate_fitness_from_distance(distance):
    return 1 / distance
```

Teplota a ochladzovanie

Algoritmus inicializuje premennú „teploty“. Táto teplota riadi pravdepodobnosť prijatia horších riešení v priebehu algoritmu, čím umožňuje prieskum priestoru riešenia. Teplota sa postupne znižuje pri každej iterácii na základe rýchlosti ochladzovania. Toto postupné znižovanie simuluje proces žíhania, čo umožňuje algoritmu pomaly sa sústrediť na určitú oblasť priestoru riešenia.

Iteratívne zlepšovanie

Generovanie nových riešení: Pri každej iterácii algoritmus mierne upraví aktuálne riešenie, aby vytvoril nové riešenie. Táto úprava je zvyčajne malá zmena: výmena dvoch miest na trase.

Koncept energie

V kontexte SA sa pojem „energia“ často používa na opis kvality riešenia. V prípade problému cestujúceho obchodníka možno celkovú vzdialenosť trasy považovať za energiu tohto riešenia – čím kratšia vzdialenosť, tým nižšia energia.

Pravdepodobnosť prijatia

Keď sa nájde nové riešenie, SA vypočíta zmenu energie (ΔE) medzi novým riešením a súčasným riešením. Ak je nové riešenie lepšie (menej energie), je vždy akceptované. Ak je však nové riešenie horšie (vyššia energia), pravdepodobnosť jeho prijatia sa vypočíta pomocou Boltzmannovej distribúcie, zvyčajne vyjadreného ako $e^{-\Delta E/T}$, kde T je aktuálna teplota.

```
math.exp((current_distance - new_distance) / temperature
```

Únik z miestneho optima

Vyhýbanie sa miestnym minimám: Prijatím horších riešení môže SA uniknúť miestnym minimám. Lokálne minimum je riešenie, ktoré je lepšie ako jeho bezprostrední susedia, ale nie také dobré ako globálne minimum. Ak algoritmus akceptuje iba vylepšenia, môže sa zaseknúť v lokálnom minime. Akceptovanie horších riešení umožňuje SA skúmať aj mimo týchto lokálnych miním pri hľadaní lepších, potenciálne globálnych miním.

Náhodná prechádzka: Toto akceptovanie horších riešení zavádza prvok náhodnej prechádzky do procesu vyhľadávania. Aj keď sa algoritmus ocitne v nízkoenergetickom (dobrom) stave, nezostane tam uväznený, ak existuje šanca nájsť ešte lepšie riešenie dočasným prechodom do vysokoenergetických (horších) stavov.

Rozvrh chladenia

Postupné znižovanie teploty je kľúčovým aspektom SA. V počiatočných štádiách, keď je teplota vyššia, je algoritmus viac prieskumný a môže voľnejšie prijímať horšie riešenia. S klesajúcou teplotou sa zameranie presúva na zdokonaľovanie a zlepšovanie existujúcich dobrých riešení s klesajúcou pravdepodobnosťou prijatia horších riešení.

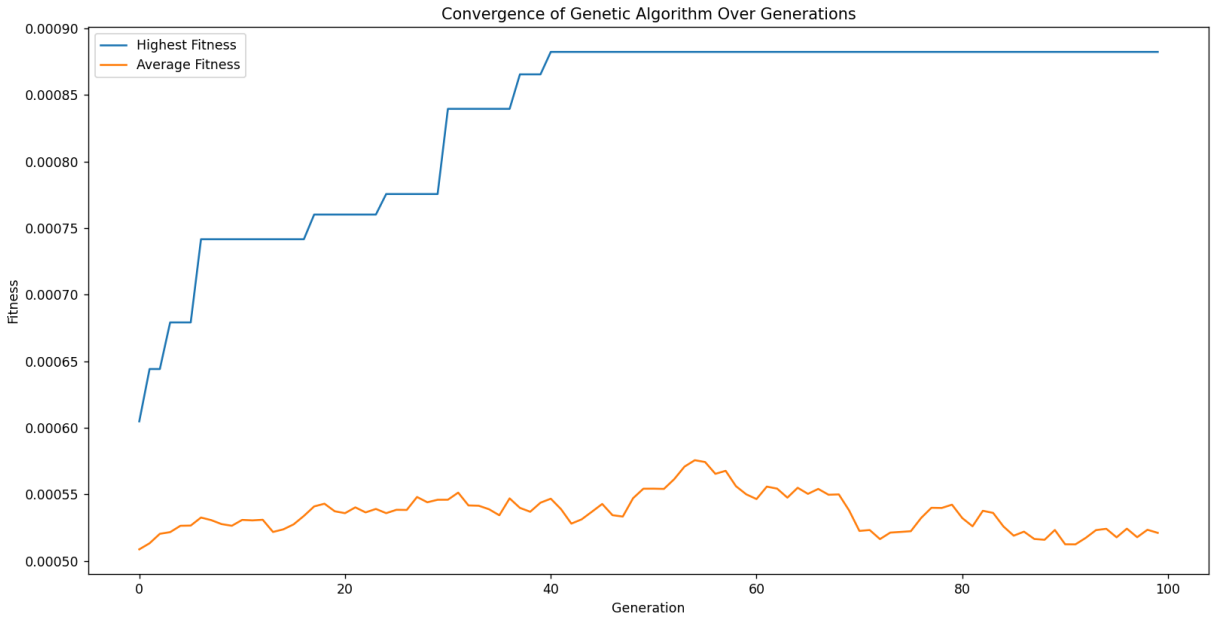
```
def simulated_annealing(cities):
    current_solution = random.sample(range(number_of_cities),
number_of_cities)
    current_distance = calculate_total_distance(current_solution)
    sa_fitness = [calculate_fitness_from_distance(current_distance)]
    temperature = 1.0
    cooling_rate = 0.0003

    while temperature > 0.0001:
        new_solution = current_solution[:]
        idx1, idx2 = random.sample(range(len(cities)), 2)
        new_solution[idx1], new_solution[idx2] = new_solution[idx2],
new_solution[idx1]

        new_distance = calculate_total_distance(new_solution)
        if new_distance < current_distance:
            current_solution, current_distance = new_solution, new_distance

sa_fitness.append(calculate_fitness_from_distance(current_distance))
        else:
            if random.random() < math.exp((current_distance - new_distance) /
temperature):
                current_solution, current_distance = new_solution,
new_distance

sa_fitness.append(calculate_fitness_from_distance(current_distance))

        temperature *= 1 - cooling_rate

    return current_solution, current_distance, sa_fitness
```

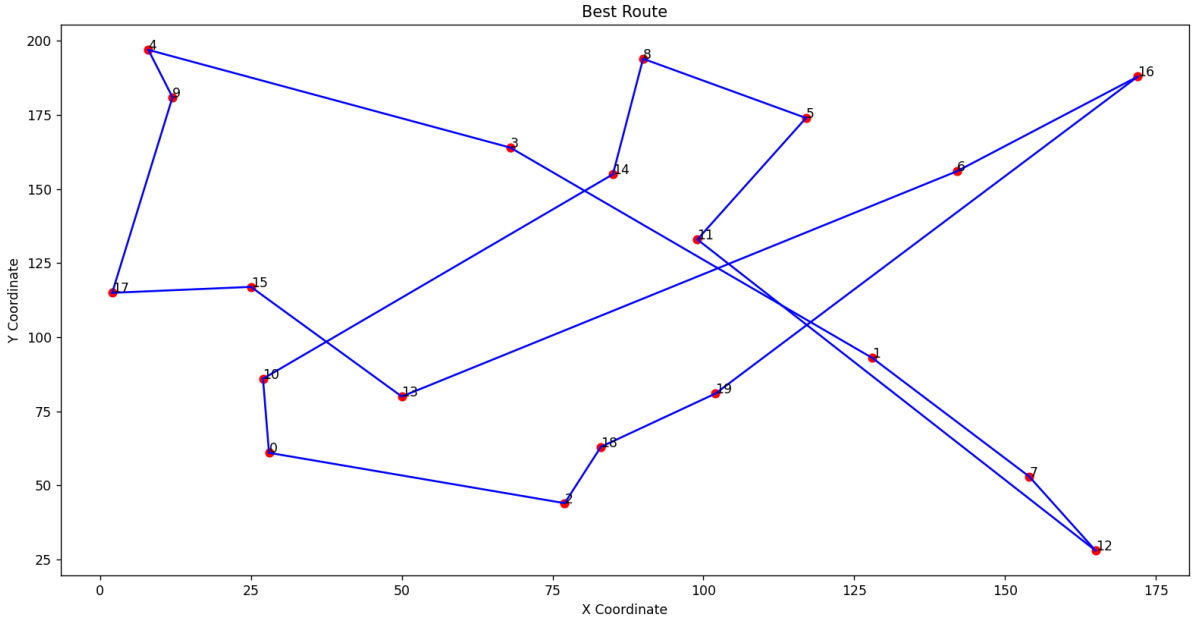
Testovanie a porovnanie

Testovanie genetického algoritmu

Roulette wheel selection 100

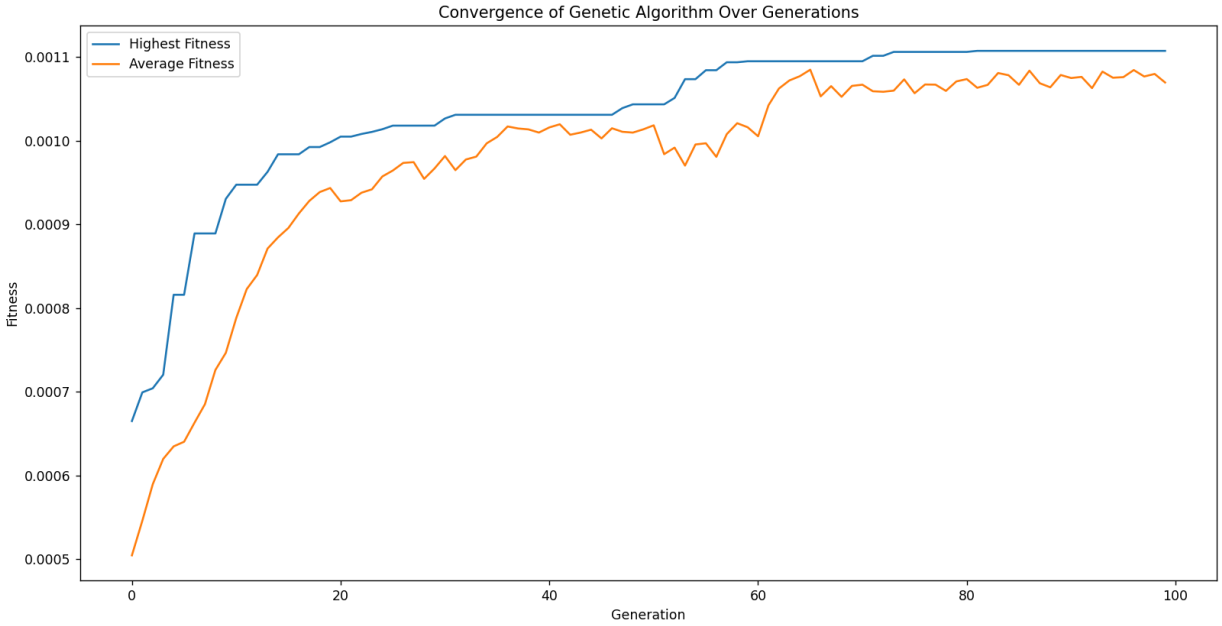
Populácia – 100, Generácie – 100, mutácia – 10%, 20 miest

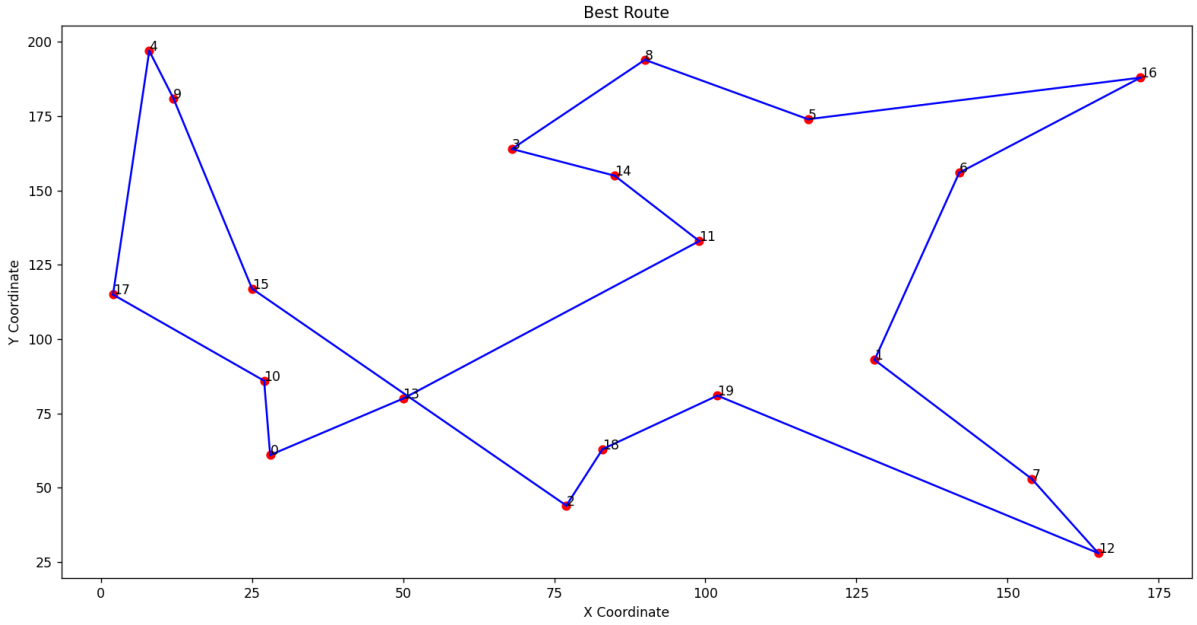




Tournament selection 100

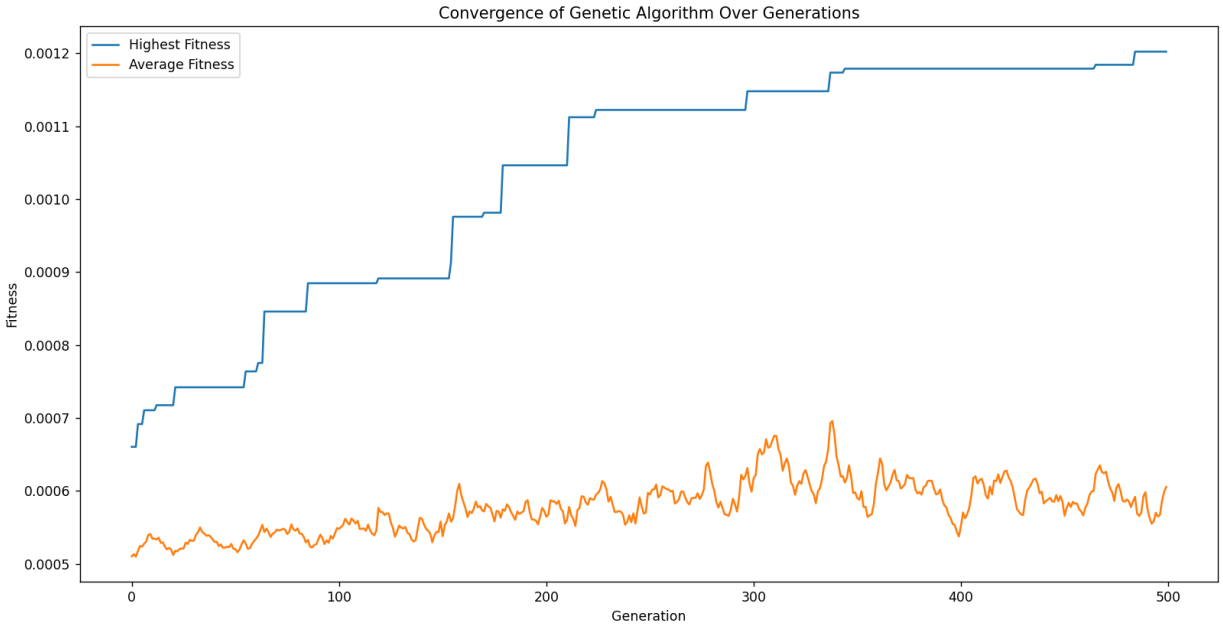
Populácia – 100, Generácie – 100, mutácia – 10%, 20 miest

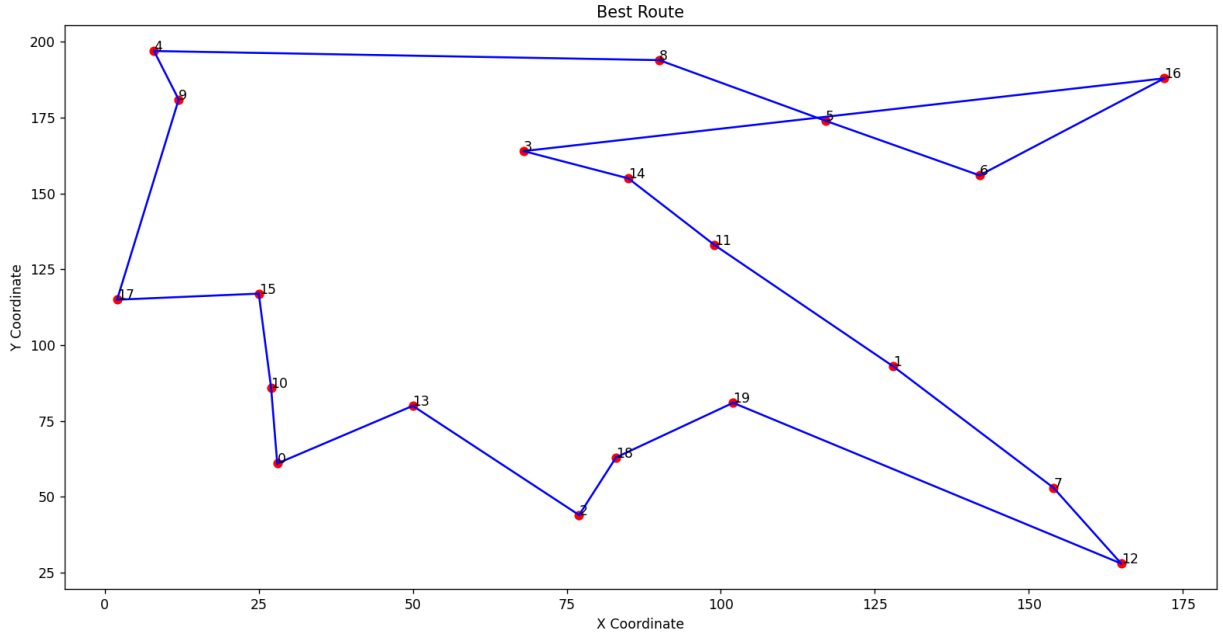




Roulette wheel selection 500

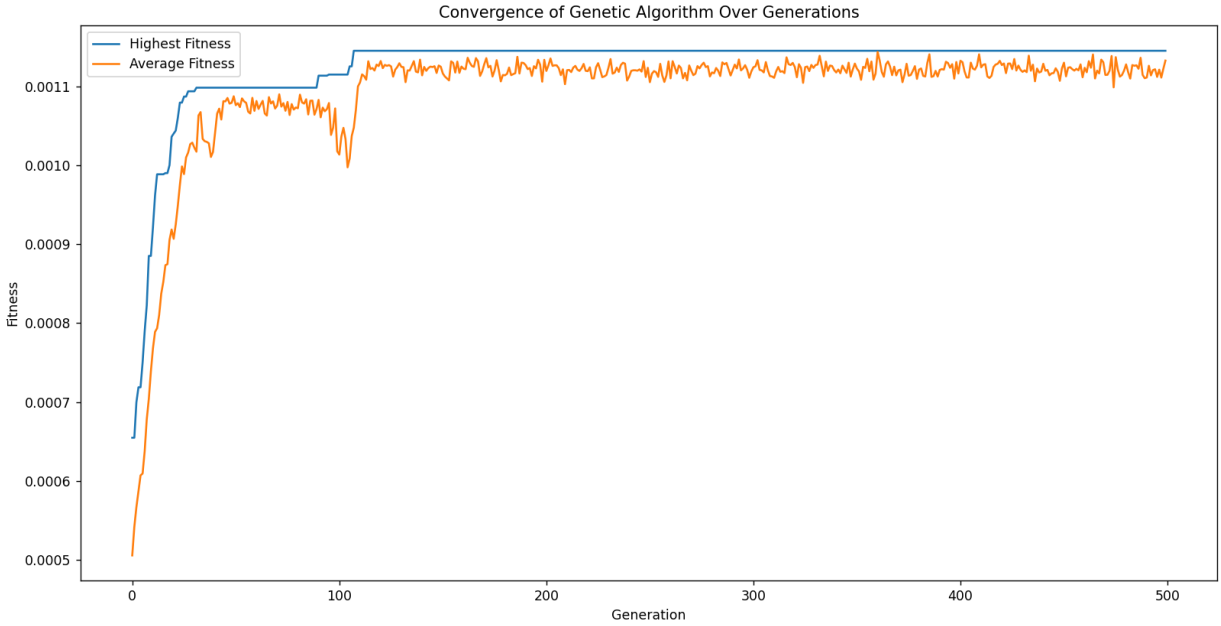
Populácia 100, Generácie – 500, mutácia – 10%, 20 miest

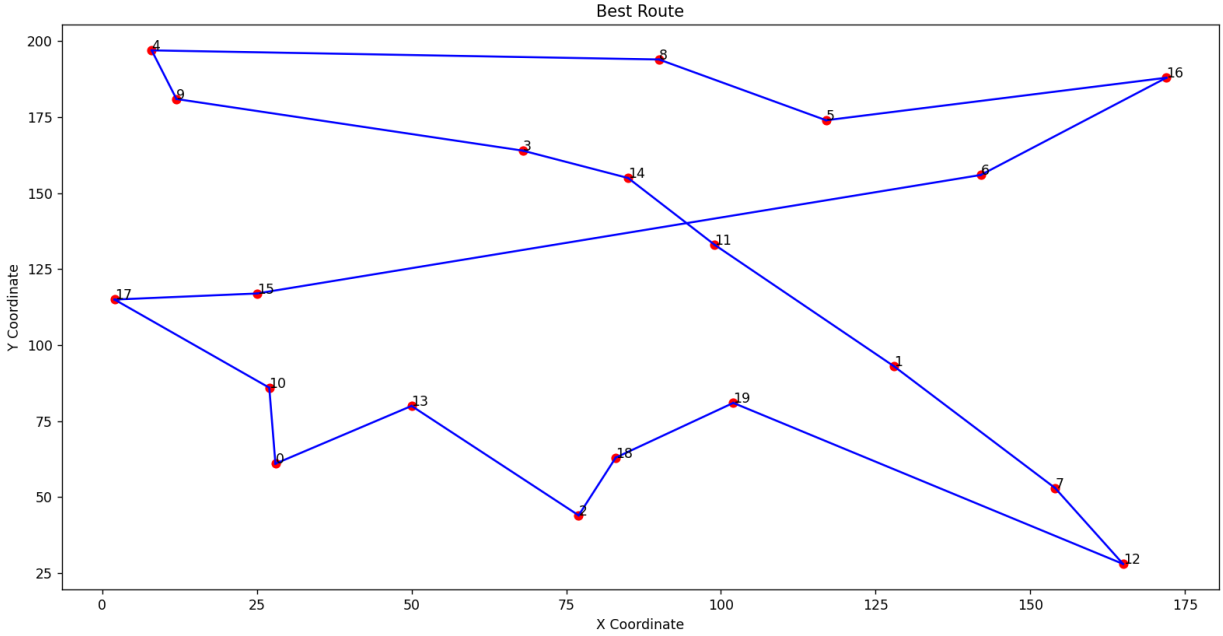




Tournament selection 500

Populácia 100, Generácie – 500, mutácia – 10%, 20 miest

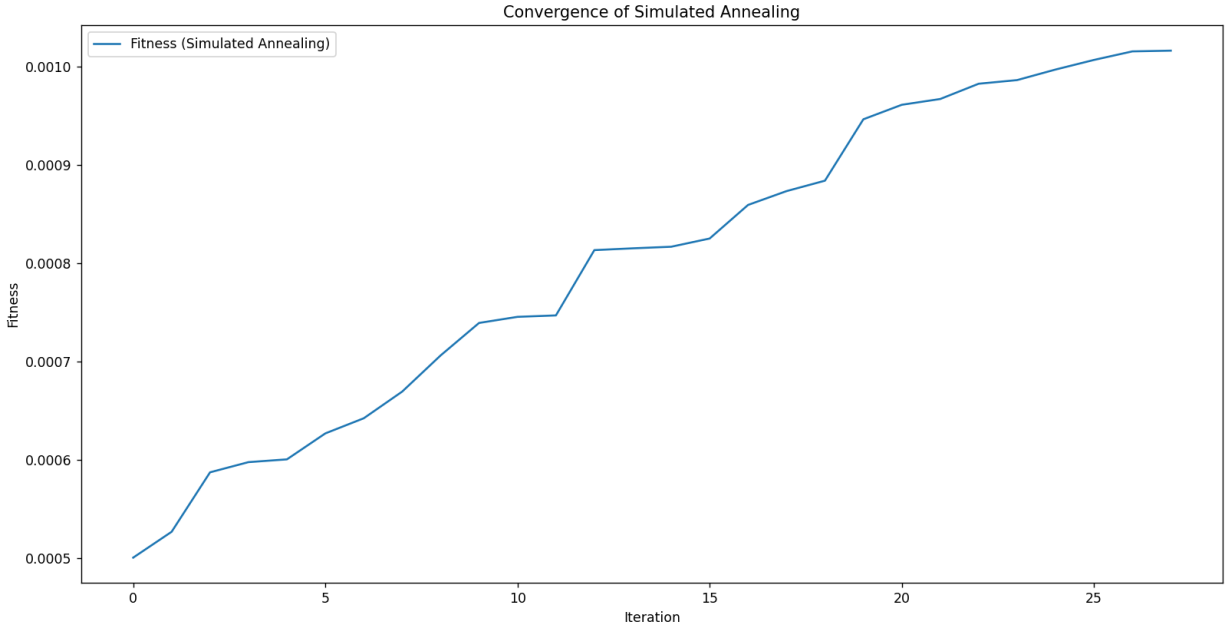


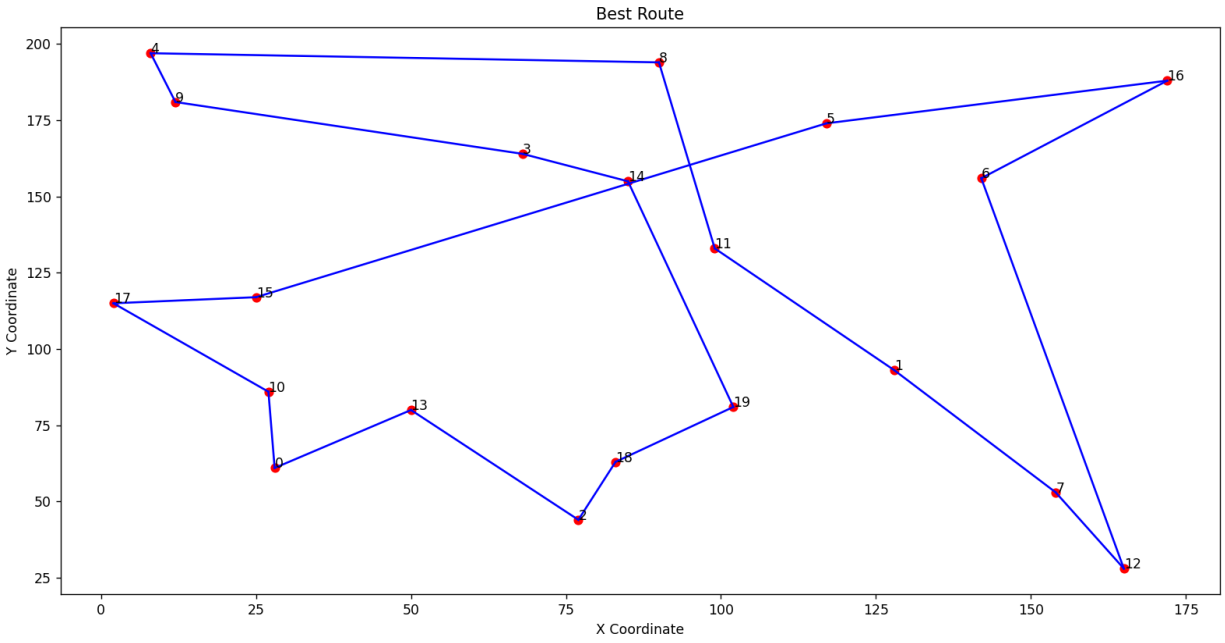


Testovanie simulovaného žíhania

CR 0.003

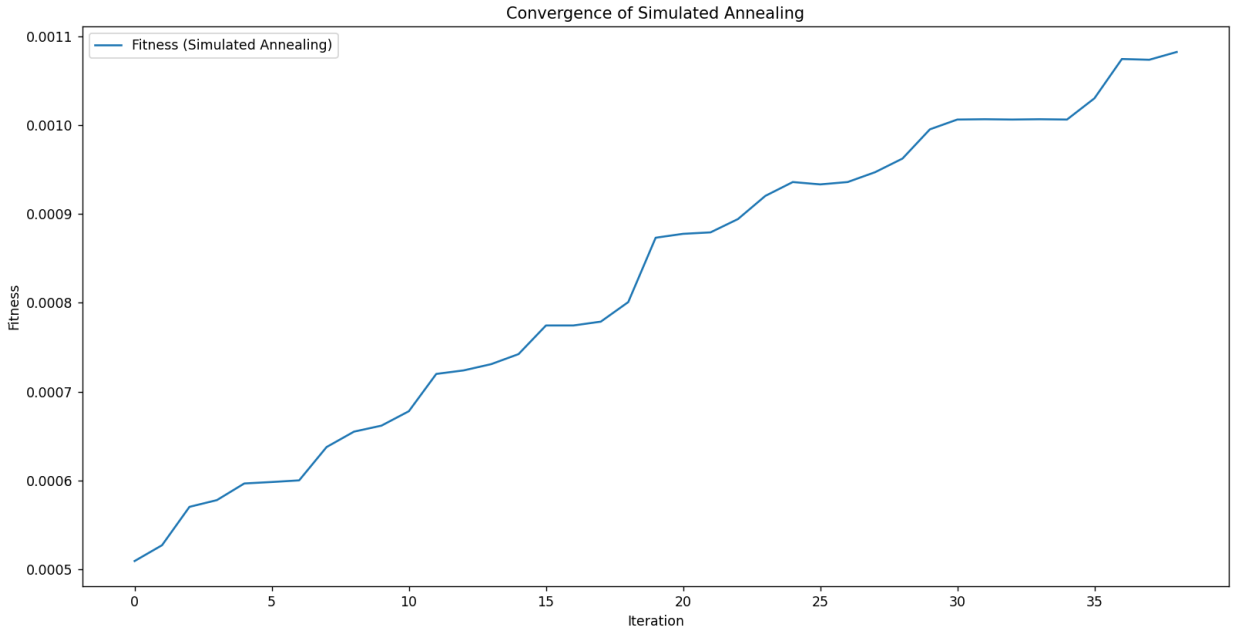
Starting temperature - 1, cooling rate – 0.003

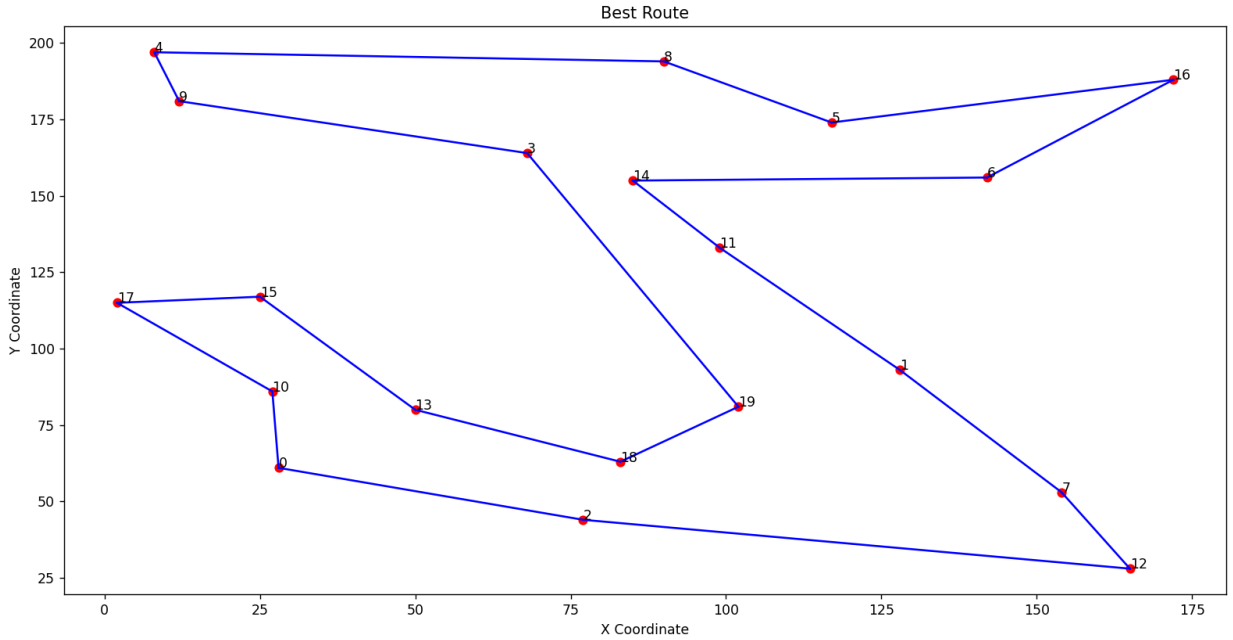




CR 0.0003

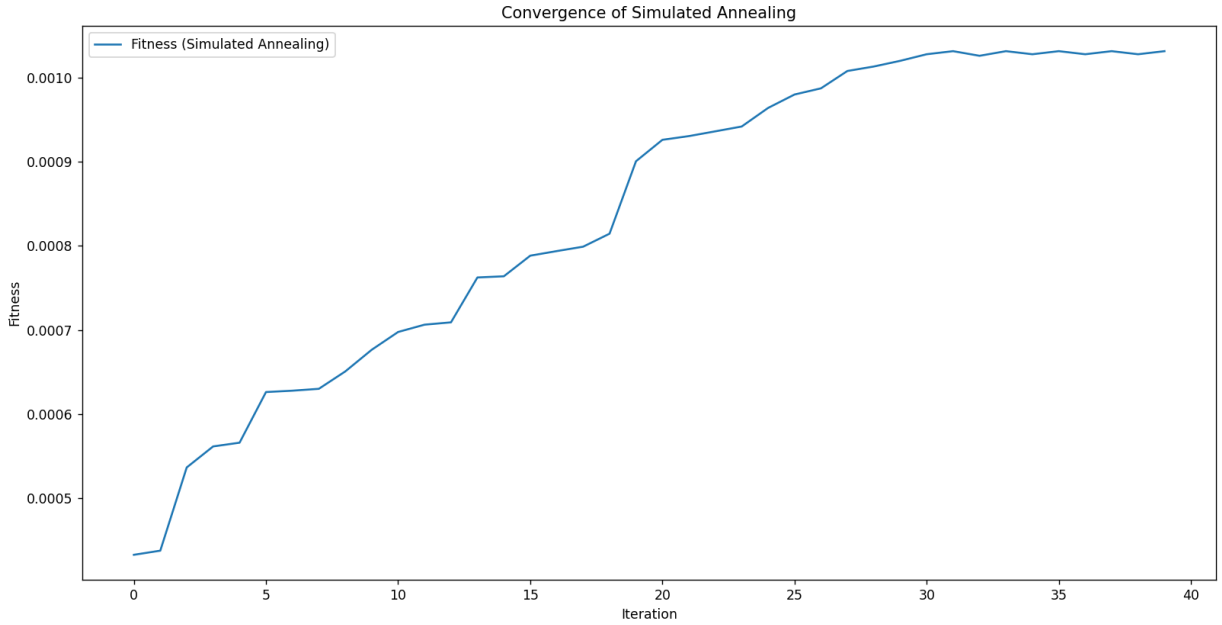
Starting temperature – 1, cooling rate 0.0003

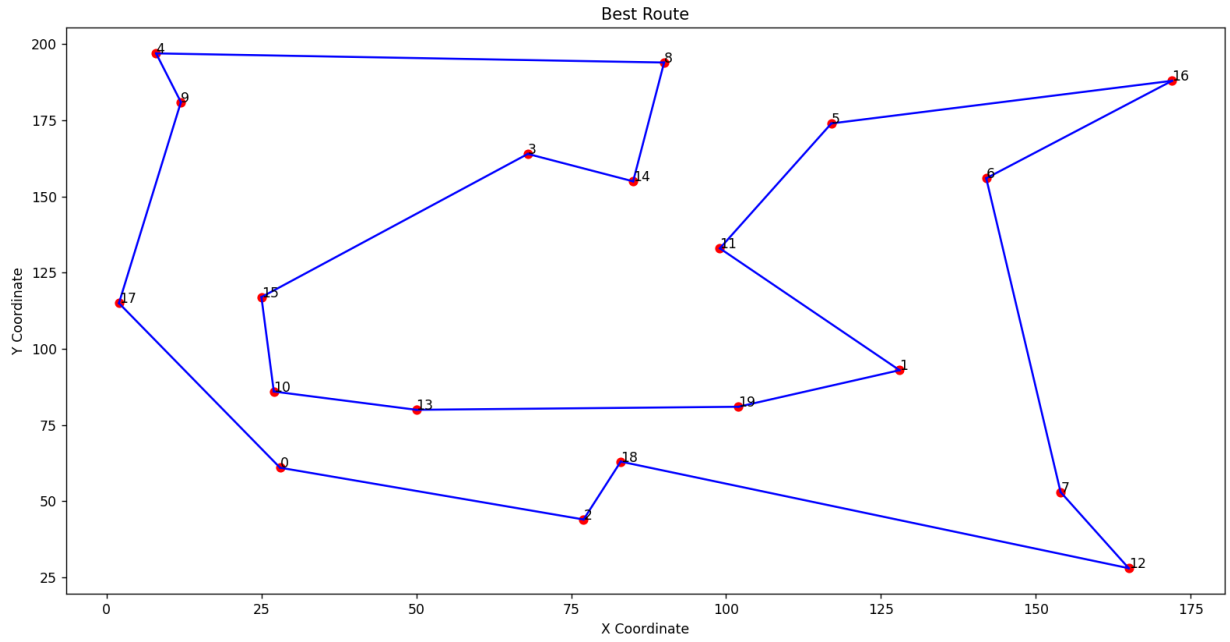




CR 0.00003

Starting temperature – 1, cooling rate – 0.00003





Implementácia

Implementácia bola prevedená v programovacom jazyku Python. Vstupné hodnoty sú manuálne upravovateľné v programovacom kóde. Pre lepšiu kontrolu výsledkov bola použitá knižnica matplotlib.pyplot pre vykresľovanie výsledkov.

Používateľská príručka

Je potrebné mať nainštalovaný python, ako aj knižnicu matplotlib. Ďalej stačí program iba spustiť a vybrať, či je potrebné použiť tournament metódu selekcie, alebo roulette wheel.

```
Choose a selection method:
1 - roulette wheel selection
2 - tournament selection
```

Záver

Pri riešení problému obchodného cestujúceho (TSP) boli použité dve odlišné výpočtové prístupy: Genetický algoritmus (GA) a Simulované žíhanie (SA). Každá metóda má svoje jedinečné silné stránky a charakteristiky, ktoré ponúkajú cenné pohľady pri riešení kombinatorických a optimalizačných problémov.

Porovnanie

- Prístup GA založený na populácii je všeobecne robustnejší v pokrývaní priestoru hľadania a môže byť efektívnejší pri nájdení takmer optimálnych riešení za kratší čas. Avšak schopnosť SA unikať lokálnym optimám môže byť výhodná v určitých typoch problémoch.
- GA je zložitejší na implementáciu kvôli jeho mnohým operátorom (výber, kríženie, mutácia), zatiaľ čo SA je relatívne jednoduchý s jedným vyvíjajúcim sa riešením.
- GA má tendenciu lepšie škálovať s rastom veľkosti problému, pretože môže udržiavať diverzitu v širšom priestore hľadania. Naopak, jednoriešenkový prístup SA môže vyžadovať viac iterácií na adekvátne pokrytie väčších priestorov hľadania.
- Oba algoritmy môžu byť prispôsobené špecifickým charakteristikám daného problému. Ladanie parametrov, ako je miera mutácie v GA alebo chladiaci rozvrh v SA, môže výrazne ovplyvniť ich efektívnosť.

Genetický algoritmus a Simulované žíhanie ponúkajú cenné techniky na riešenie problému obchodného cestujúceho, pričom každá má svoje výhody a obmedzenia. Výber medzi nimi by mal byť riadený špecifickými požiadavkami problému, dostupnými výpočtovými zdrojmi a požadovanou rovnováhou medzi preskúmavaním a využívaním. Kým GA ponúka robustný a adaptívny prístup, SA exceluje v navigácii komplexnými krajinami s mnohými lokálnymi optimami. Kombinovaním postrehov z oboch možno dosiahnuť komplexnejšie pochopenie a efektívnejšie riešenie zložitých optimalizačných problémov.

Zdroje

https://en.wikipedia.org/wiki/Simulated_annealing

https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

http://www2.fiit.stuba.sk/~kapustik/obchodny_cestujuci.html