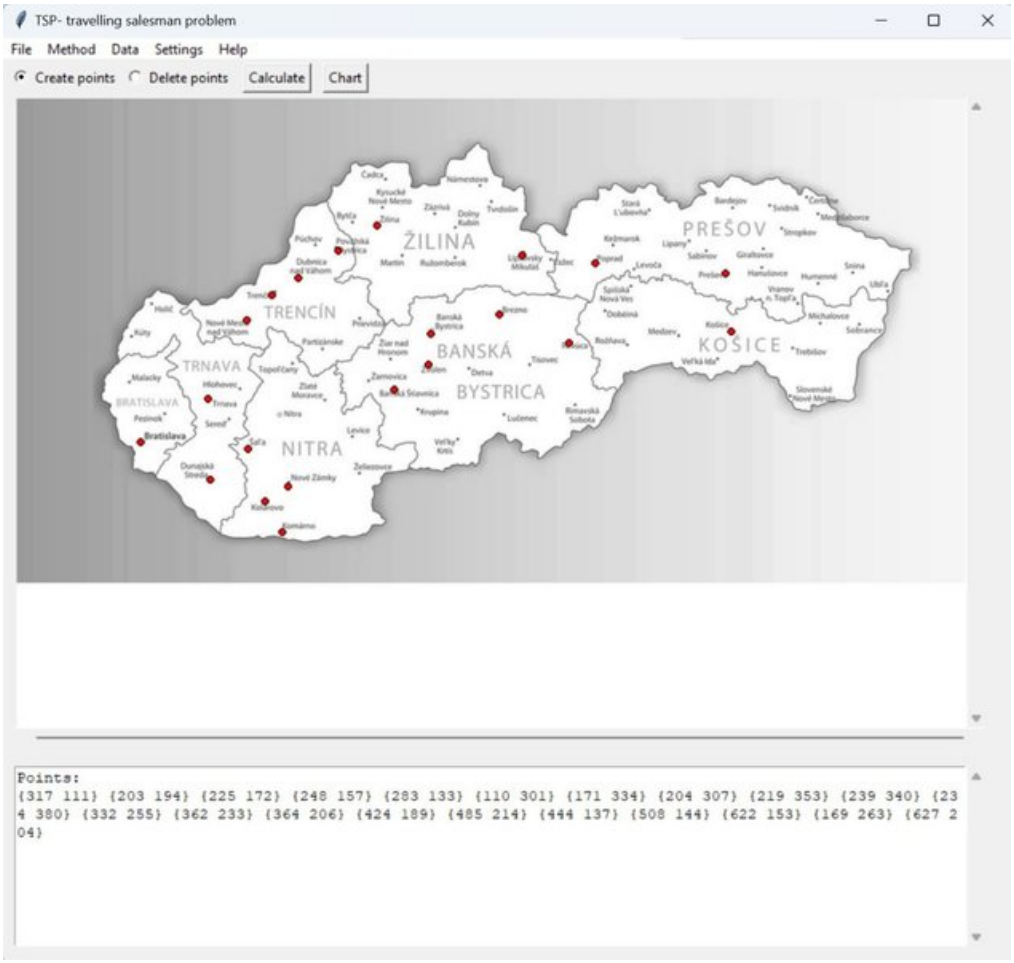
Optimization of the Nearest Neighbor method for the TSP Problem



Autor: Ing. Robert Polák (Robopol), Email: robopol@gmail.com, Slovakia

Dátum/date:19.2.2023

Abstrakt / Abstract:

SK: Táto publikácia sa venuje optimalizácií metódy najbližšieho suseda tak, aby boli dosiahnuté lepšie výsledky celkovej trasy TSP (traveling salesman problem) problému. Táto optimalizácia bola navrhnutá tak, aby sa sa významne nepredĺžil čas tejto efektívnej metódy (metódy najbližšieho suseda) a zároveň sa dosiahlo výrazného zlepšenia. V tejto publikácií je odkaz na algoritmus tohto zlepšenia v programovacom jazyku Python. V publikácií budú predostreté všetky aspekty tohto algoritmu.

Eng: This publication is dedicated to optimizing the nearest neighbor method to achieve better results for the overall TSP (traveling salesman problem) route. This optimization was designed not to significantly extend the time of this efficient method while achieving a significant improvement. The publication includes a reference to the algorithm of this improvement in the Python programming language. All aspects of this algorithm will be presented in the publication.

Úvod / Introduction

Metóda najbližšieho suseda sa považuje za jednu z najjednoduchších metód na riešenie problému obchodného cestujúceho. Dnes existuje mnoho iných metód, algoritmov na riešenie TSP problému. Táto publikácia nebude o hodnotení iných metód, ale o zlepšení tejto konkrétnej metódy (metódy najbližšieho suseda). Základný algoritmus sa dá popísať veľmi jednoducho. Z poľa bodov sa vyberie jeden bod (začiatok). Nasledujúci bod, ktorý sa spojí je ten k nemu najbližší. Takto sa pokračuje ďalej a vyberajú sa vždy najbližšie body. Algoritmus prejde všetky body zo zoznamu a každý bod obsahovať iba dve spojnice. Nájdená trasa nie je ideálna, avšak častokrát lepšia ako metóda simulovaného žíhania a iných metód (ak zohľadníme čas výpočtu na nájdenie trasy). Metóda najbližšieho suseda je proste veľmi rýchla a nájde trasu, ktorá sa líši od ideálnej trasy zhruba 20-30% navyše z celkovej dĺžky ideálnej trasy. To všetko záleží aj od toho ako sú rozmiestnené body. Tento odhad sa týka náhodne generovaných bodov, bez nejakej symetrie.

Eng: The nearest neighbor method is considered one of the simplest methods for solving the traveling salesman problem. Today, there are many other methods and algorithms for solving the TSP problem. This publication will not evaluate other methods, but rather focuses on improving this specific method (the nearest neighbor method). The basic algorithm can be described very simply. One point is selected from the list of points (the starting point). The next point to be connected is the one closest to it. This process continues, selecting the closest points each time. The algorithm passes through all the points in the list, with each point having only two connections. The resulting route is not ideal, but often better than simulated annealing and other methods (if we consider the computation time to find the route). The nearest neighbor method is simply very fast and finds a route that differs from the ideal route by approximately 20-30% more than the total length of the ideal route. This all depends on how the points are distributed. This estimate applies to randomly generated points, without any symmetry.

Zdrojový kód v Pythone: /Source code in Python

LINK GITHUB:
https://github.com/robopol/Travelling-salesman-problem/blob/main/traveling_salesman_problem_3.py

SK: Program v Pythone je konzolová aplikácia, kde na vstupe zadávame počet náhodne generovaných bodov na intervale (0,1000) pre každú os x, y. Následne program vypočíta optimalizovanou metódou najbližšieho suseda trasu s tým, že vykreslí graf trasy a vypíše body v postupnosti, vypíše aj celkovú dĺžku trasy. Program ukončíme zadaním 0 a entrom, čím sa program ukončí.

ENG: The Python program is a console application where we enter the number of randomly generated points on the interval (0,1000) for each x,y axis. Subsequently, the program calculates the optimized route using the nearest neighbor method, draws a graph of the route, and prints the sequence of points, as well as the total length of the route. We end the program by entering 0 and pressing enter, which terminates the program.

Optimalizačné kroky algoritmu/ Optimization steps of the algorithm.

SK:
Optimalizačné kroky metódy najbližšieho suseda pozostávajú z týchto krokov:

1. Použitie metódy najbližšieho suseda pre každý počiatočný bod zo zoznamu generovaných bodov. Vybratie najkratšej trasy zo všetkých n- trás.
2. Použitie knižnice intertools pre permutácie, ktoré trasu optimalizujú v nejakom zadanom intervale permutácii, postupne algoritmus prechádza každý bod a v úseku napr. 7 bodov dopredu trasu opraví na kratšie riešenie. Ide teda o optimalizáciu subtúry (časti trasy).
3. V prípade, že existujú kríženia trasy použije sa knižnica shapely, konkrétne: shapely.geometry import LineString. Pomocou funkcii # function intersection a # function Exchange of intersecting lines nájde algoritmus kríženie trasy a následne kríženie odstránením pre usporiadaním bodov trasy. Jedná sa o pomerne jednoduché funkcie.
4. Použitie bodu 2. pre konečnú optimalizáciu.

ENG:
The optimization steps of the nearest neighbor method consist of the following:

1. Applying the nearest neighbor method to each starting point from the list of generated points. Selecting the shortest path from all n paths.

2. Using the intertools library for permutations, which optimize the route in a given interval of permutations. The algorithm iterates through each point and, for example, corrects the route for a shorter solution within a segment of 7 points. This is the optimization of a sub-tour (a part of the route).
3. In case there are crossings in the route, the shapely library is used, specifically the LineString function. The algorithm finds the crossing of the route using the intersection and Exchange functions of intersecting lines, and subsequently removes the crossing by rearranging the points of the route. These are relatively simple functions.
4. Using step 2 for final optimization with point 2.

Hlavný program /Main program

SK: V zmysle metódy najbližšieho suseda a optimalizačných krokov hlavná časť programu je volaná nasledovne:
ENG: In accordance with the nearest neighbor method and optimization steps, the main part of the program is called as follows:

```
import math
import sys
import matplotlib.pyplot as plt
import random
from itertools import permutations
from shapely.geometry import LineString
…
# Infinite while loop console. Main program
while True:
    # input numbers of points
    num=get_input()
    # end of program
    if num==0:
        break
    # call function for random points
    points=get_random_points(num)
    print("Random points:")
    print(points)
    # call function for optimal nearest neighbor
    optimal_path=get_optimal_nearest_neighbor(points)
    print("Best path - best nearest neighbor:")
    print(optimal_path[0])
    print(f'Best distance -best nearest neighbor: {optimal_path[1]}')
    # plot best path for optimal nearest neighbor
```

```
    plt.title("Plot of best path : best nearest neighbor")
    plt.grid(True)
    plt.plot([x for (x,y) in optimal_path[0]],[y for (x,y) in
optimal_path[0]],'ko-')
    plt.show()
    # call function for permutation
    best_path_permutation=get_optimize_path(optimal_path[0])
    # delete double points
    best_path_correction=check_double_points(best_path_permutation[0])

best_path_permutation=[best_path_correction,best_path_permutation[1]]
    # call intercection function
    intersection=check_intersection(best_path_permutation[0])
    if intersection[1]!=0 and intersection[2]!=0:
        field_points=exchange_intersecting_lines(intersection[0],
intersection[1], intersection[2])
    else:
        field_points=best_path_permutation
    # cykle for intercection function
    while intersection[1]!=0 and intersection[2]!=0:
        intersection=check_intersection(field_points[0])
        if intersection[1]!=0 and intersection[2]!=0:
            field_points=exchange_intersecting_lines(intersection[0],
intersection[1], intersection[2])
            # delete double points
            best_path_correction=check_double_points(field_points[0])
            field_points=[best_path_correction,field_points[1]]
        else:
            break
    # call function for permutation
    best_path_permutation=get_optimize_path(field_points[0])
    # delete double points
    best_path_correction=check_double_points(field_points[0])
    field_points=[best_path_correction,field_points[1]]
    print("Best path -optimize best NN:")
    print(field_points[0])
    print(f'Best distance -optimize best NN:
{best_path_permutation[1]}')
    # plot best path
    plt.title("Plot of best path : optimize best NN")
    plt.grid(True)
```

```
    plt.plot([x for (x,y) in field_points[0]],[y for (x,y) in
field_points[0]],'ko-')
    plt.show()
```

SK: Potrebné knižnice, ktoré program vyžaduje:
V príkazovom riadku nainštalovať tieto knižnice:

- pip install matplotlib
- pip install shapely

ENG: Required libraries for the program:
Install these libraries in the command line:

- pip install matplotlib
- pip install shapely

Program beží korektne aj pre najnovšiu verziu Pythonu 3.11.1. //The program runs correctly even for the latest version of Python 3.11.1.

Konkrétne použitie optimalizačných funkcii si čitateľ nájde v celom zverejnenom kóde na GITHUB. / Specific use of optimization functions can be found in the entire published code on GITHUB.

Testovanie algoritmu / Testing algorithm

SK: Optimalizačné kroky výrazne zlepšili metódu najbližšieho suseda do zhruba cca 200-300 bodov. Od tej najkratšej trasy sa to líši zhruba do 5%. Pre vyšší počet bodov >200,300 by bolo pre dosiahnutie lepšieho výsledku naprogramovať ešte jednu optimalizačnú funkciu. Táto funkcia bude vysvetlená nižšie (v príklade).

ENG: The optimization steps significantly improved the nearest neighbor method by approximately 200-300 points. It deviates from the shortest route by approximately 5%. For a higher number of points (>200, 300), it would be necessary to program another optimization function to achieve better results. This function will be explained below (in the example).

Príklad 1: / Example 1:

n= 27
Random points:
[(420, 588), (142, 614), (239, 288), (710, 538), (826, 272), (819, 598), (492, 373), (842, 309), (232, 730), (444, 337), (182, 342), (551, 70), (80, 924), (362, 230), (303, 968), (643, 368), (742, 688), (34, 25), (197, 920), (792, 98), (245, 444), (841, 933), (668, 485), (895, 249), (767, 391), (592, 322), (649, 259)]
Best path - best nearest neighbor:
[(668, 485), (710, 538), (819, 598), (742, 688), (841, 933), (303, 968), (197, 920), (80, 924), (232, 730), (142, 614), (245, 444), (182, 342), (239, 288), (362, 230), (444, 337), (492, 373),

(592, 322), (643, 368), (649, 259), (767, 391), (842, 309), (826, 272), (895, 249), (792, 98), (551, 70), (34, 25), (420, 588), (668, 485)]
Best distance -best nearest neighbor: 5055.386109647314
Best path -optimize best NN:
[(668, 485), (710, 538), (819, 598), (742, 688), (841, 933), (303, 968), (197, 920), (80, 924), (232, 730), (142, 614), (245, 444), (182, 342), (34, 25), (551, 70), (792, 98), (895, 249), (826, 272), (842, 309), (767, 391), (643, 368), (649, 259), (592, 322), (492, 373), (444, 337), (362, 230), (239, 288), (420, 588), (668, 485)]
Best distance -optimize best NN: 4959.811830328142

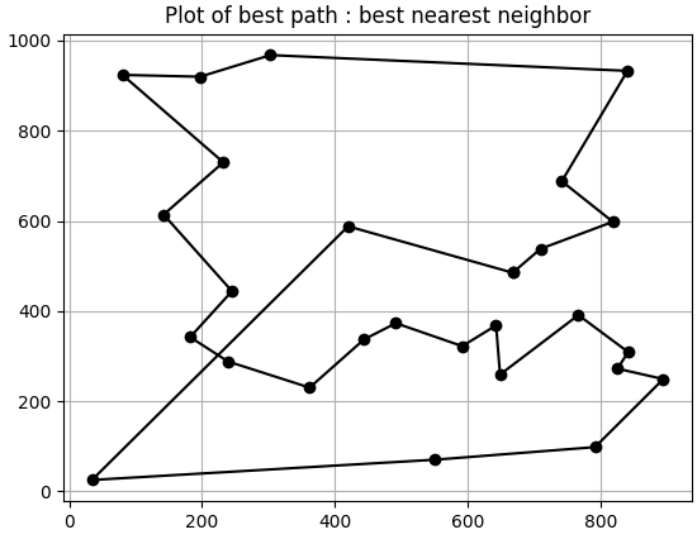


figure No.1 Plot best nearest neighbor

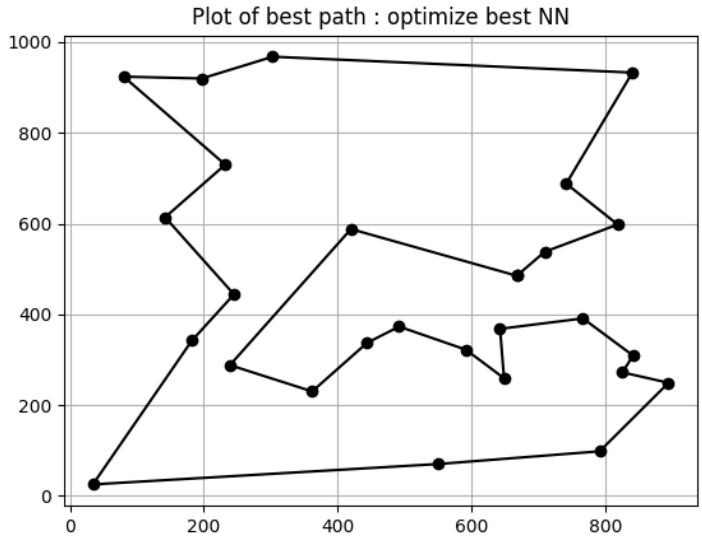


figure No.2 Plot optimize best nearest neighbor

Príklad 2: / Example 2:
n=145
Best distance -best nearest neighbor: 10856.431529849422
Best distance -optimize best NN: 9912.874411452593

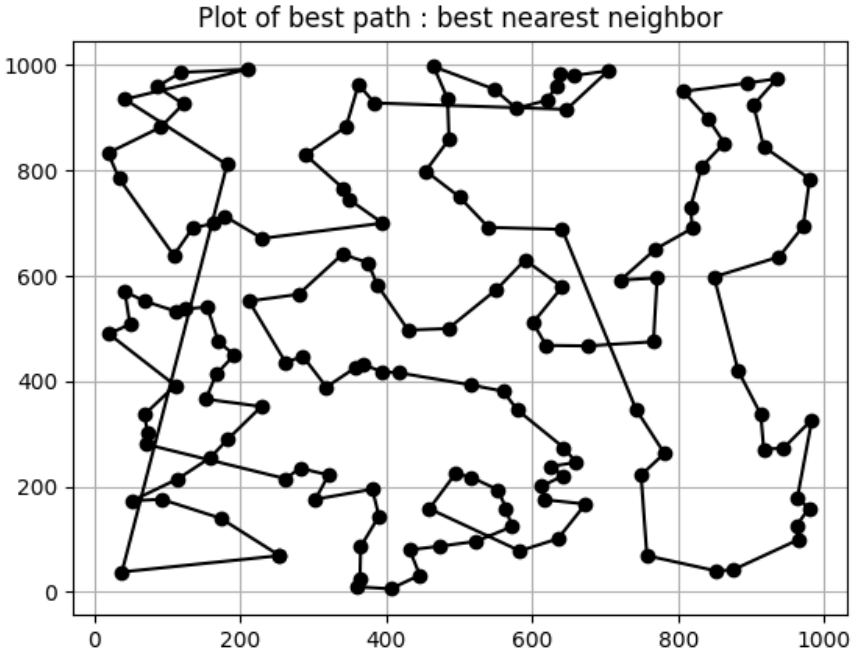


figure. No.3 Plot best nearest neighbor

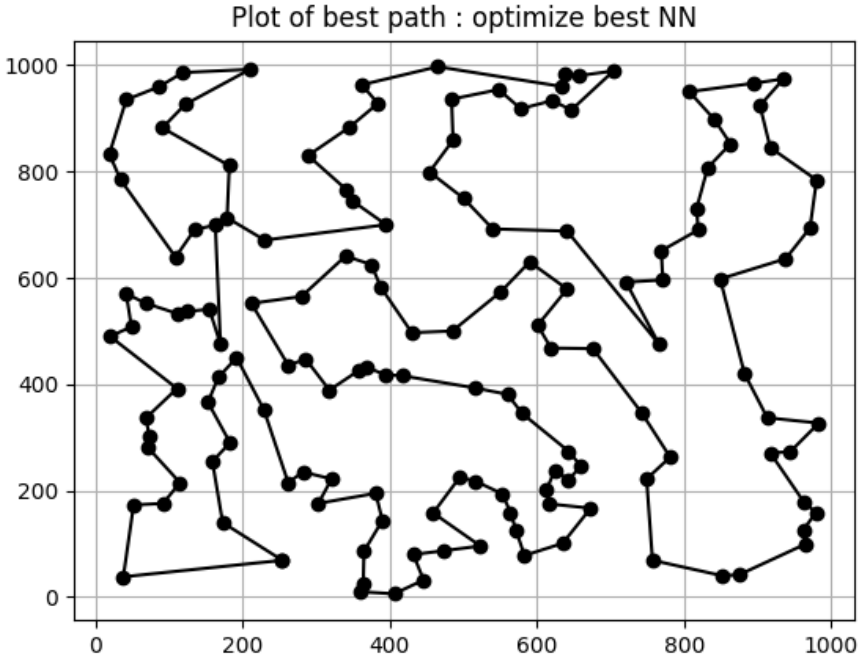


figure. No.4 Plot optimize best nearest neighbor

SK: Optimalizačný krok pre väčší počet bodov spočíva v princípe ako je uvedené na obrázku č.5, opravené červenou farbou. Pri väčšom počte bodov sa stáva, že pri spojnici (nejakej dlhšej) sa v blízkosti nachádza nejaký bod, spojnica ho tesne minie. V takom prípade by bolo žiaduce presunúť tento bod do inej časti cesty, teda to časti kde je ta dlhšia spojnica. Princíp je jasný z obrázku č.5.

ENG: The optimization step for a larger number of points is based on the principle as shown in the Figure 5, highlighted in red. With a larger number of points, it may happen that a point is located near a longer connection and the connection passes very close to it. In such a case, it would be desirable to move this point to another part of the path, namely the part where the longer connection is. The principle is clear from Figure 5.

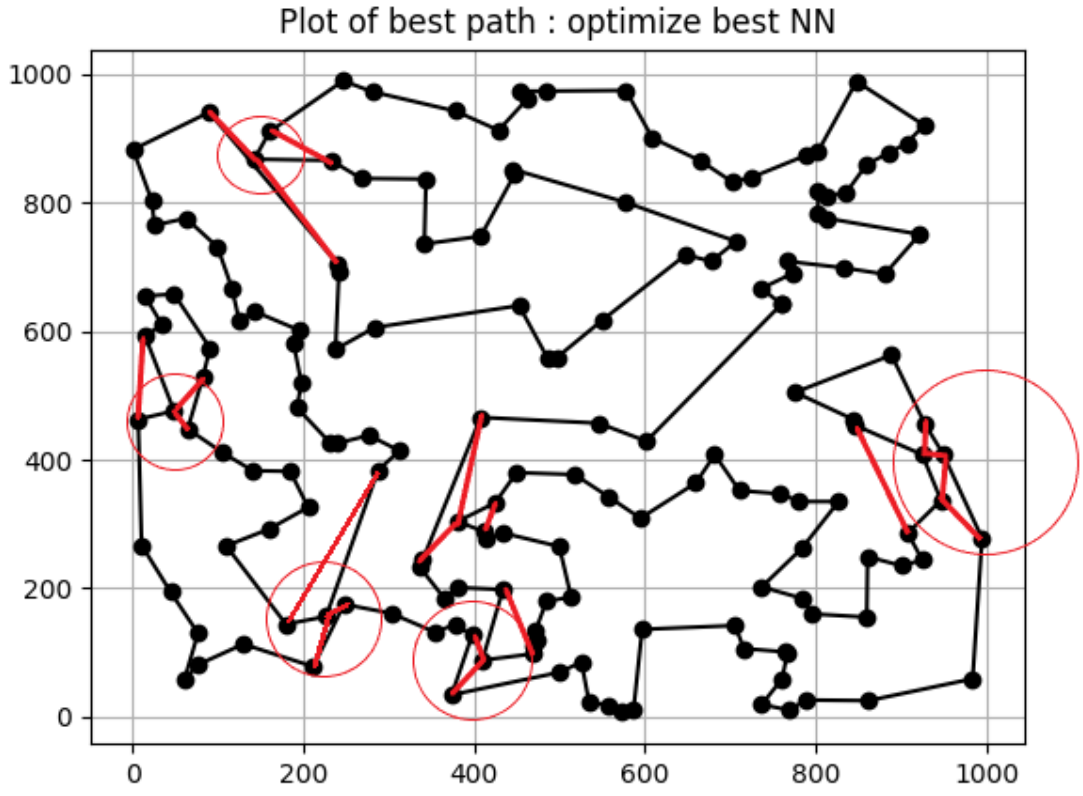


figure. No.5 Plot optimize best nearest neighbor

Záver / Conclusion

SK: Na záver sa dá povedať, že táto metóda má svoje výhody, ale aj nevýhody. Výhoda je, že poskytuje výsledky blízke optimalnej trasy za pomerne krátky čas. Medzi nevýhody patrí hlavne to, že potrebuje ďalšie optimalizačné kroky pri väčšom počte bodov a výsledok nie je nikdy dokonalé riešenie.

ENG: In conclusion, it can be said that this method has its advantages and disadvantages. The advantage is that it provides results close to the optimal route in a relatively short time. The disadvantages include the need for additional optimization steps with a larger number of points, and the result is never a perfect solution.