**Документация**

**на проект TSP SOLVER**

Изготвена от Тодор Тонев Димитров 81969

1. **Увод**

Гитхъб репо на проекта: <https://github.com/todorakisa/TSP-Project-SDP>

**Описание и идея на проекта**

Проблемът на търговския пътник (TSP) е един от най-известните и най-добре изследваните алгоритми в комбинаторната оптимизация. Той е много лесен за дефиниране но и труден за решаване. Търговски пътник трябва да посети списък от градове. Ако е известно разстоянието между всяка двойка градове, кой е най-късият път, който посещава всеки град. Проблема не може да се реши за разумно време поради неговата природа. Може да се класифицира като такъв със сложност NP. Досега не е измислен алгоритъм който да го решава за линейно време, но има подходящи евристични алгоритми които могат да намерят най- късния път с малко отклонение.

**Цел и задачи на разработката**

Целта на приложението е да се имплементират няколко алгоритъма за решаване на проблема и да се сравни тяхната ефективност при проблеми с различна големина. Трябва да се направи извод за ефективността на всеки алгоритъм.

**Структура на документацията**

В документацията има пет основни глави: 1 Увод 2 Основни дефиниции, концепции и алгоритми , които ще бъдат използвани 3 Резултати и тестване 4 Заключение и 5 Използвани ресурси

1. **Основни дефиниции, концепции и алгоритми, които ще бъдат използвани**

Избрани алгоритми за тестване са BruteForce , Nearest Neighbor, Improved Nearest Neighbor, ACS(Ant Colony System). Ще ги разгледаме поотделно:

* 1. **BruteForce**

Това е един от първите алгоритми за които се сеща човек когато, си помисили за решение на Traveling Salesman Problem. Той е много лесен, но същевременно и много бавен. При него се минава през всяко едно възможно решение на проблема и се избира най-краткото. Проблема е че при графи с над 10 града, времето за решаването започва да расте експоненциално. Използвайки псевдокод може да илюстрираме как работи алгоритъма:

get an initial tour; call it T   
best\_tour ⇦ T  
best\_score ⇦ score(T)   
while there are more permutations of T do the following:  
 generate a new permutation of T   
 if score(T) < best\_score then  
 best\_tour ⇦ T   
 best\_score ⇦ score(T)  
 print best\_tour and best\_score

* 1. **Nearest Neighbor**

Този алгоритъм е с много алчна природа. Той също е един от алгоритмите за които човек веднага. При него се избира една точка за начална и се добавя към решението. След това се избира най-близкия съсед и той също се добавя към решението. След като сме изминали всички точки по този начин, получаваме път, който е с около 20 процента по-дълъг от най-краткия възможен. Въпреки това е добър алгоритъм защото времето за което се изпълнява е в пъти по-добро от BruteForce.

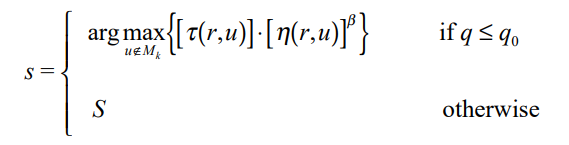
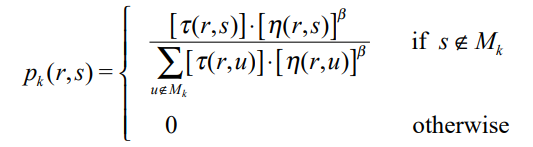
* 1. **Improved Nearest Neighbor**

Този е алгоритъм е почти същия като Nearest Neighbor, но пък разликата е че за начална точка се избира точката, която е най-близко до всички останали. Бързодействието му е сходно с това на NN, но понякога е се получават подобри резултати от NN.

* 1. **ACS(Ant Colony System)**

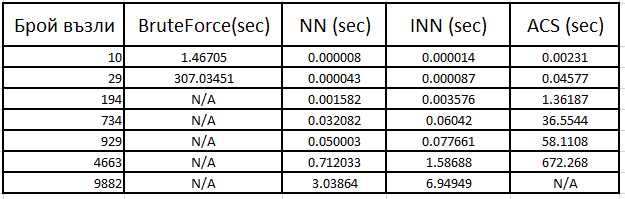
Това е доста по сложен алгоритъм от останалите. Той се клалифицира като генетичен алгоритъм или като reinforcement machine learnig. На практика взима идея от природата, а тя е че мравките винаги намират най-късния път до техния мравуняк, използвайки феромони. С помощта на тази идея се генерира матрица на съседтство в графа която съдържа всички бедра и тяхните феромони и дължини. Избира се колко генерации да се възпроизведат и по колко мравки да бъде всяка генерация. Всички мравки от дадена генерация започват да обикалят графа от случайна точка. Вяко бедро си има феромони и когато дадена мравка го избере и мине през него се променят феромоните. За да може алгоритъма да работи трябва да има глобално и локално правило по което се променят феромоните. Когато приключи дадена генерация най-добрата мравка(тази с най-кратък път) усилва феромоните по всяко бедро от пътя, по който е минала. За да не избират мравките едно и също бедро се използва локалното правило. То намалява феромоните когато една мравка е минала през него. Трябва да има и правило по което да се избира кое бедро ще е следващото за дадена мравка. За реализирането на алгоритъма са използвани ресурс 1 и 2 в глава „Използвани Ресурси“. Правилата може да ги видите на следващите снимки:

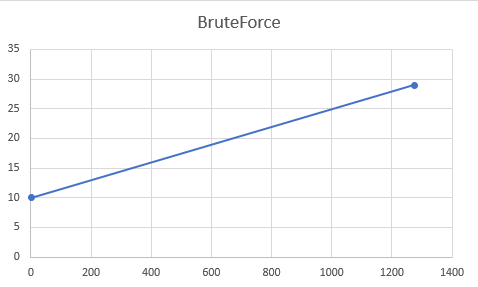
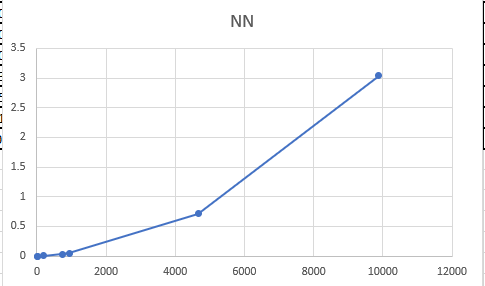
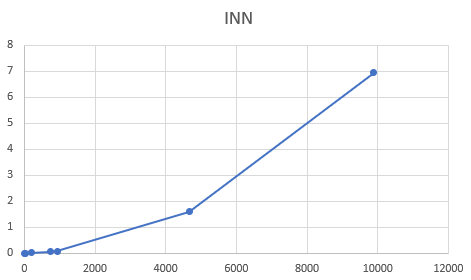
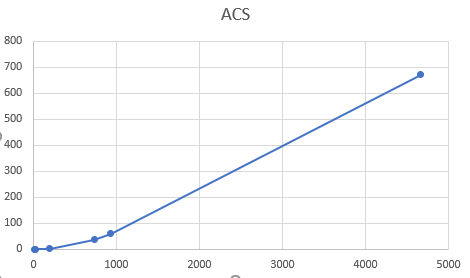
Локално правило: 

Глобално правило:  
   
  
  
 Правило за избор на следващ град:  

1. **Резултати и тестване**

От резултатите се забелязват няколко основни неща. BruteForce алгоритъма не е практичен за повече от 10 върха. Относно ACS алгоритъма намира доста добри пътища, но поради голямата му сложност е много бавен със сравнение с NN и INN. Въпреки че, тези два алгоритъма не намират един от най-близките пътища, са сравнително бързи за големи проблеми. За тестването е използван ресурс 3 от глава „Използвани Ресурси“.



1. **Заключение**
   1. **Обобщение на изпълнението на началните цели**

Имплементирани са 4 алгоритъма за решаване на проблема и са тествани с различни по големина проблеми. Изработен е и малък интерфейс за потребителя.

* 1. **Насоки за бъдещо развитие и усъвършенстване**

Като бъдещо развитие може да се счита оптимизацията на ACS алгоритъма за решаване на TSP. Извикването на много нишки ще подобри многократно бързодействието на алгоритъма за големи проблеми. Другата част която може да се оптимизира е началния брой на генерирани мравки. Той трябва да звиси по някакъв начин с големината на проблема, защото при малко мравки не намира къс път, а при прекалено много прави излишни изчисления.

1. **Използвани ресурси**
   1. <http://staff.washington.edu/paymana/swarm/dorigo97-itec.pdf>
   2. <http://people.idsia.ch/~luca/acs-bio97.pdf>
   3. <http://www.math.uwaterloo.ca/tsp/world/countries.html>