Ця програма була розроблена для учбових цілей, тому всі спроби використання її у комерційних цілях категорично не схвалюються і засуджуються

Ця програма призначена для розв’язання проблеми комівояжера і проблем, що зводяться до вищезгаданої, засобами і можливостямои методу еластичної сітки.

Короткий опис

Ця програма знаходить один із гарних(не обов’язково найкращого) шляхів між містами

Шлях повинен задовольняти такі вимоги:

* бути неперервним
* проходити через усі точки один і тільки один раз
* бути зв’язним(починатися і закінчуватися в одній точці)

Програма надає можливість міняти(коригувати) практично всі прараметри, які можуть вплинути на ефективність роботи алгоритму

**Перелік умовних термінів:**

Точка кільця – точка нормалізуючого кільця;

Місто – одна з точок, що створюються спочатку(перед початком роботи алгоритму) і встановлення шляху між ними є основним завданням роботи алгоритму;

Нормалізуюче кільце – сукупність усіх точок кільця та послідовних зв’язків між ними; на першій ітерації має вигляд кола, яке згодом розтягується у псевдокриву, що уособлює послідовність зв’язків між містами на останній ітерації роботи алгоритму;

Шлях – один із способів прокладення відстаней між містами; передбачається, що при правильній настройці параметрів довжина шляху буде квазіоптимальною(близькою до отимальної – найкращої довжини шляху); це означає, що шлях являється одним із кращих способів з’єднання міст, проте не гарантовано найкращим; на великих наборах міст визначити, чи є шлях найкращим, не маючи точного рішення задачі методом повного перебору, не являється можливим;

Карта – вікно алгоритму з розташованими на ньому містами;

Абсолютні координати – (реальні координати). Координати міст у такому вигляді, у якому вони були зчитані з файлу або координати/довжини точок кільця, визначені у масштабі абсолютних координат міст;

Нормалізовані координати – координати, за яких найменшою допустимою координатою міста може бути (0;0), а найбільшою – (1;1), і при цьому пропорції відстаней між містами в абсолютних координатах буде збережено.

В графічній інтерпретації точкою (0;0) вважається верхній лівий кут квадрата, а (1;1) – нижній правий; наприклад, для трьох точок з абсолютними координатами А(7;4), В(5;12) і С(7;12) отримаємо нормалізовані координати та відповідно;

**Коротка інструкція**

Nodes per city – кількість точок нормалізуючого кільця, які створюються на кожну з точок-міст

Нормалізуюче кільце складається з (amount of cities \* nodes per city) точок кільця

Max Іterations – максимальна кількість ітерацій, виконуваних алгоритмом

(при досягненні ліміту програма припинить виконання незалежно від того чи було знайдено шлях)

Max Gaussian Radius – максимально можлива кількість точок кільця з кожного боку від обраної у даній ітерації точки кільця(назвемо її цільовою точкою); принцип наступний: обирається місто із множини усіх міст на карті; із множини обирається цільова точка , така що відрізок є найкоротшим із множини відрізків ; обрана цільова точка притягується до міста на відсатнь, що дорівнює певному відсотку довжини відрізка , а ‘Current Gaussian Radius’(якщо цей параметр рівний 1, то це означає, що рухається лише цільова точка без жодного «хвоста») точок кільця зліва і справа від також наближуються до , при цьому задовольняється умова, що кожна наступна точка кільця притягується на відстань, меншу ніж відстань між містом і попередньою точкою; таким чином формується дуга

Експериментальним шляхом встановлено, що оптимальне значення цього параметру зазвичай знаходиться в межах (1~2,5) \* Nodes per city

Min Gaussian Radius – параметр, за своєю дією протилежний до Max Gaussian Radius : мінімально можливе значення параметра ‘Current Gaussian Radius’. Повинен бути додатнім числом, не більшим за Max Radius

Normalizing Ring Radius – не має жодного відношення до Min Gaussian Radius чи Max Gaussian Radius. Визначає початковий радіус нормалізуючого кільця, що буде створене на першій ітерації.

При встановленні цього параметра рівним 0,5 і центру нормалізуючого кільця у цетрі вікна, нормалізуюче кільце вийде вписаним у квадрат, що позначає границі території карти.

Рекомендована величина параметра: 0,55-0,025

Screen Refresh Rate - [разів за 10 секунд];частота оновлення вікна алгоритму; при числах більших за цей параметр фактично озачає, що вікно алгоритму оновлюватиметься зі швидкістю виконання ітерацій і залежатиме лише від потужностей апаратного забезпечення комп’ютера;

Learning Rate Decrease Speed – що більший цей параметр, то повільніше спадатиме Learning Rate ; при високих значеннях цього параметра та параметра Max iterations одночасно, Max Radius може збільшуватися замість зменшуватися;

Max Learning Rate - learning rate параметр спочатку задається цим числом, а далі з ростом кількості ітерацій зменшується;

ComboBox – надає можливість вибрати, звідки починати малювати нормалізуюче кільце: з центру вікна чи з точки, що є координатним центром усіх міст на карті

Write down intermediate data to console – цим параметром контролюється, чи слід виводити поітеративно проміжні дані до консолі

ЗВЕРНІТЬ УВАГУ!!! Немає сенсу вмикати цей параметр, якщо програма була запущена не з .bat, тому що видима консоль відсутня і відслідкувати проміжні дані буде усе одно неможливо, в той час як вони записуватимуться до віртуальної, що може значно сповільнити роботу алгоритму

Write to file – записує результат поточного рішення алгоритму до файлу(див. теку 'output'), нумеруючи їх(файли) починаючи з нуля. Кожен наступний запуск алгоритму з цим параметром генеруватиме новий файл, тож немає ризику втратити уже створені файли

Select file with data – дозволяє вибрати файл, що ляже в основу створення поточної карти.

Слід пам’ятати, що навіть не всі файли з розширенням .tsp можуть бути прочитані. Це зумовлено тим, що вони не містять конкретних координат міст у читабельному для програми форматі.

Користувачеві, що бажає створити власний файл генерації карти, слід керуватися зразком data.txt як макетом, оскільки він добре зрозумілий одночасно і для програми, і для людини

Внутрішній формат файлів з розширенням .tsp можна побачити, скопіювавши увесь вміст такого файлу до документу Microsoft Word

CheckBox ‘Generate cities randomly’ – при виборі цього параметру міста будуть створені випадковим чином внутрішніми можливостями програми. У разі вибору цього параметра значення поля ‘Select file with data’ не буде враховано.

Дані карти будуть встановлені відповідно до параметрів Cities Quantity, Higher Bound X, Higher Bound Y. Якщо якесь із цих полів не буде заповнене або формат даних не буде числовим, причому цілочисельним невід’ємним числом, меншим за , алгоритм, швидше за все не запуститься, а в роботі програми станеться збій

Cities Quantity – визначає кількість міст, що будуть згенеровані

Higher Bound X – визначає максимально допустиме значення координати міста по осі ОХ

Higher Bound Y – визначає максимально допустиме значення координати міста по осі ОY

Ці два параметри є параметрами абсолютних координат міст, а не нормалізованих. Найбільше їхній вплив буде помітно при обрахунку абсолютного шляху

ЗВЕРНІТЬ ОСОБЛИВУ УВАГУ!!! Назва файлу міста створюється за шаблоном і складається з ‘generSource’ + кількість міст. Це означає, що при повторному створенні випадкового набору міст з кількістю міст, що вже була згенерована раніше, ПОВНІСТЮ ПЕРЕЗАПИШЕ файл на новий. Для збереження вподобаних створених карт, їх слід перейменувати перед запуском наступного сеансу алгоритму, що використовує випадковий набір міст такої ж кількості

**Принцип роботи алгоритму програми**

Виконання самого алгоритму може бути розбито на три основні блоки:

**I.<*Надання початкового значення параметрів*>;**

**II.<*Обчислення результату ітеративним шляхом*>;**

**III.<*Визначення, чи досягнуто хоча б однієї з умов припинення виконання алгоритму*>;**

Як і більшість подібних програм, даний алгоритм можна розбити на цикли.

Окремо варто виділити перший цикл – він є визначним і має наступну структуру:

1. **<I>**;

2. **<II>**;

3. **<III>**;

Далі виконання алгоритму має таку структуру:

***1.***Якщо **<III>** з попередньої ітерації повертає ‘true’, алгоритм припиняє роботу і виконує кінцеві операції:

-виводить дані на екран у вигляді окремого вікна, створює два додаткових вікна із зображенням нормалізуючого кільця та абсолютного шляху;

- якщо вибрано відповідні пункти у меню з параметрами, записує кінцевий резудьтат у файл;

Інакше:

***2.* <II>**;

***3.* <III>**;

==>> ***1.***

Кожен з цих блоків можна описати у вигляді псевдокоду або алгоритму.

Нижче представлено принциповий алгоритм-схему з елементами псевдокоду для опису роботи кожного з блоків програми:

**I.<*Надання початкового значення параметрів*>;**

1.Визначення найменших та найбільших x та y координат із масиву міст;

2. a) Для створення карти використати найменше значення x як 0 на осі ОX , а найбільше x як 1;

b)встановити найменше значення y як 0 на осі ОY і найбільше як 1;

3. Ініціалізувати змінні Xrange і Yrange, де

Xrange = Xmax – Xmin;

Yrange = Ymax – Ymin;

4.Для кожного міста (з абсолютними (x;y) координатами)встановити відповідні координати в нормалізованих координатах відповідно до формул:

thisCityNormalizedX =

thisCityNormalizedY =

5. Встановити центр, з якого малюватиметься початкове нормалізуюче кільце:

a) якщо вибрано параметр ‘start from cities mass center’ , обчислити за формулами:

XesSum =;

YsSum = ;

Xcenter = , Ycenter = ;

b) інакше встановити: Xcenter = 0.5, Ycenter = 0.5;

6.Створити та визначити кожен елемент масиву для точок нормалізуючого кільця розміром

arraySize = citiesQuantity \* Nodes per Cities;

тобто добутку кількості міст на карті на значення параметру з меню параметрів

7. step = – крок, який визначає відстань між точками кільця при створенні початкового нормалізуючого кільця;

8.radius = Normalizing Ring Radius;

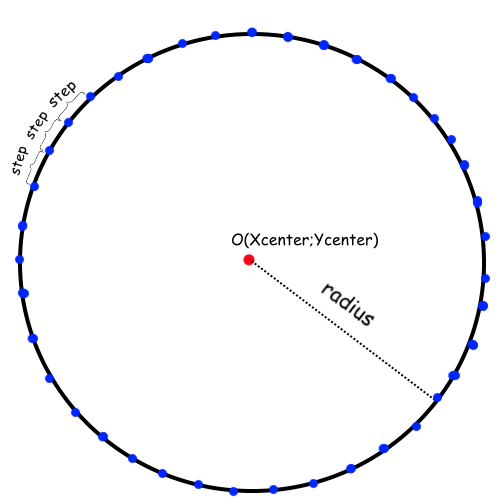
9. for(int i = 0; i < arraySize; i++){

nodesArray[i]:

}

іншими словами, створюється ламана з нодів у вигляді кола радіусу radius з центром у точці (Xcenter ; Ycenter);

схематично результат роботи цієї частини алгоритму зображено на рисунку:



**II.< *Обчислення результату ітеративним шляхом* >**

**якщо** умова finalConditionsAreDone == true

не виконується(інакшими словами, якщо ifSolutionCloseEnough = false), то виконати поточну ітерацію:

{

1.Обрати випадкове місто;

2.Обчислити найближчу до нього точку кільця і записати її як tiedUpNode для даного міста;

3.Змістити tiedUpNode у сторону міста на величини:

difX = (selectedCity.X-tiedUpNode.X)\*currentLearningRate,

difY = (selectedCity.Y-tiedUpNode.Y)\*currentLearningRate;

, де currentLearningRate :

currentLearningRate = (Max Learning Rate)\*

4.Змістити по currentGaussianRadius нодів з кожного боку від tiedUpNode , де:

=

= (selectedCity.X – node[tiedUpNode.Order+i].X)\*currentLearningRate\*;

, де =

, а *і*  дорівює віддаленості точки кільця від tiedUpNode і обчислюється за формулою:

;

, де tiedUpNode.Order та selectedNode.Order – індекси цих точко кільця у масиві точок кільця відповідно;

5. обрахувати нові значення currentLearningRate та currentGaussianRadius за формулами:

currentGaussianRadius =

currentLearningRate = (Max Learning Rate)\*

6. Збільшити значення поточної ітерації currentIteration на 1:

currentIteration = currentIteration+1;

}

**інакше**(якщо finalConditionsAreDone == true) виконати операції на завершенні роботи алгоритму:

{

1.виконати кінцеві операції

2.припинити виконання алгоритму в цьому вікні

3. намалювати остаточне зображення

, де інформація, що виводиться на екран, складається з наступних полів:

1. print out currentIteration;

2. print out currentLearningRate;

3. print out currentNodesRingLength, що обчислюється за формулою нижче:

currentNodesRingLength = ;

4. print out currentGaussianRadius;

5. print out normalizedRingLength та trueRingLength, обчислення яких відбувається згідно формул:

1) відсортувати масиви normalizedCities та trueCities згідно порядкових номерів tiedUpNode – точок кільця, що до них прив’язані

на кожній ітерації для параметр tiedUpNode визначається заново і при цьому він дорівнює індексу тієї точки кільця, відстань від якого до обраного міста є найменшою серед усіх відстаней від обраного міста до кожної з точок кільця:

if (cityA.tiedUpNode.index > cityB.tiedUpNode.index) then

cityB.index = cityA.index, cityA.index = cityB.index;

else do nothing;

2)

normalizedRingLength =

=;

3)

trueRingLength = ;

6. print out executionTime , що дорівнює різниці абсолютного часу, зафіксованого часу на початку алгоритму, і абсолютного часу, зафіксованого після закінчення виведення на екран попереднього параметра endTime – startTime;

}

**III.< *Визначення, чи досягнуто хоча б однієї з умов припинення виконання алгоритму* >**

Алгоритм припиняє свою роботу, якщо:

finalConditionsAreDone == true

значення finalConditionsAreDonе встановлюється true, якщо виконується хоча б одна з наступних трьох умов [finalConditionsAreDone = ((1)||(2)||(3))]

{

currentIteration ≥ maximalIteration,

де currentIteration – поточна ітерація, maximalIteration – ліміт;

maximalIteration = Max Іterations(з форми вводу початкових даних)

currentGaussianRadius ≤ minimumAvailableRadius,

де currentGaussianRadius визначається за формулою

currentGaussianRadius =

оскільки логарифм натуральний не може бути добутий з нуля, при значенні Min Gaussian Radius = 0 параметр Min Gaussian Radius заміняється на одиницю(сам параметр Min Gaussian Radius при цьому залишається незмінним)

, а minimumAvailableRadius = Min Gaussian Radius;

ifSolutionCloseEnough == true;

параметр ifSolutionCloseEnough набуває значення true тоді і лише тоді, коли біля кожного міста розташовано точка, що знаходиться на відстані не більше ніж minimumLearningRate(зазвичай 2) від нього.

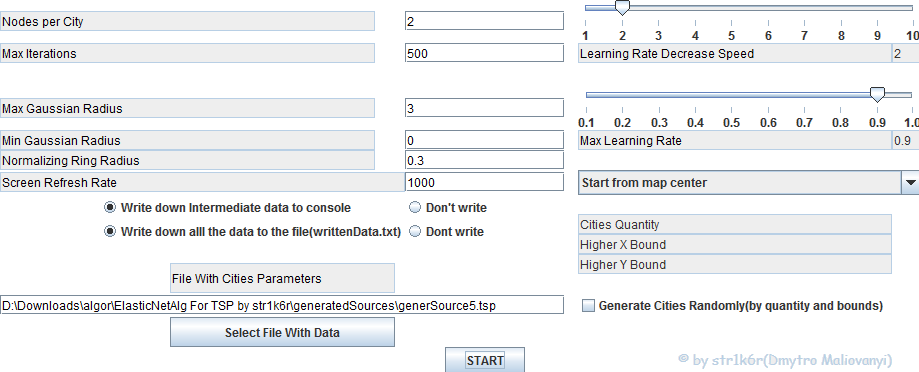
Інакше ifSolutionCloseEnough надається значення false.

}

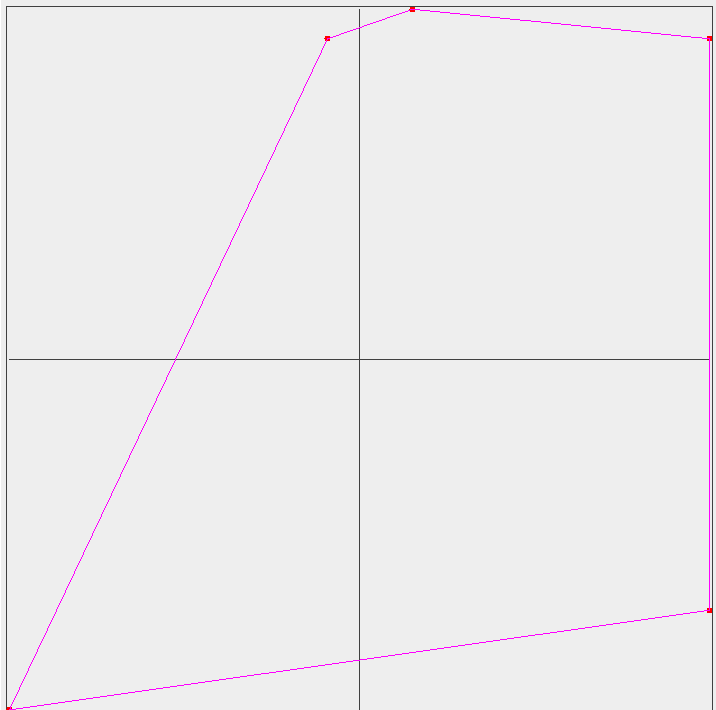
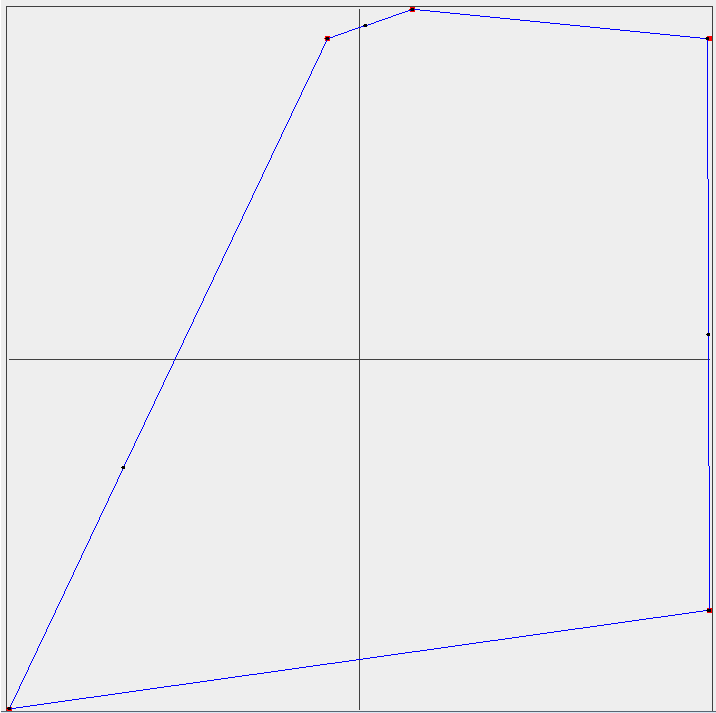
**Приклад для п’яти міст**

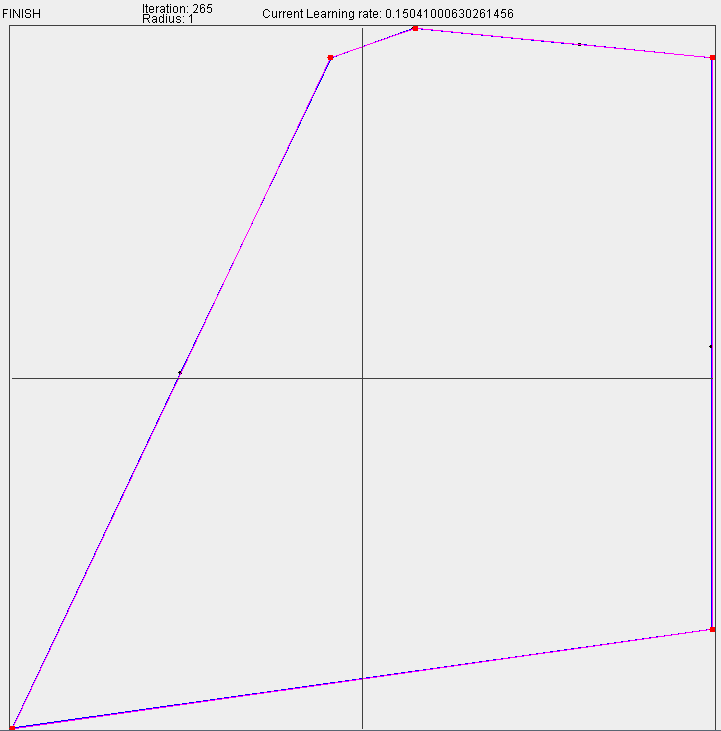
З використанням функції рандомної генерації міст і порівнянням отриманих результатів із результатами самої програми

Стартові параметри:

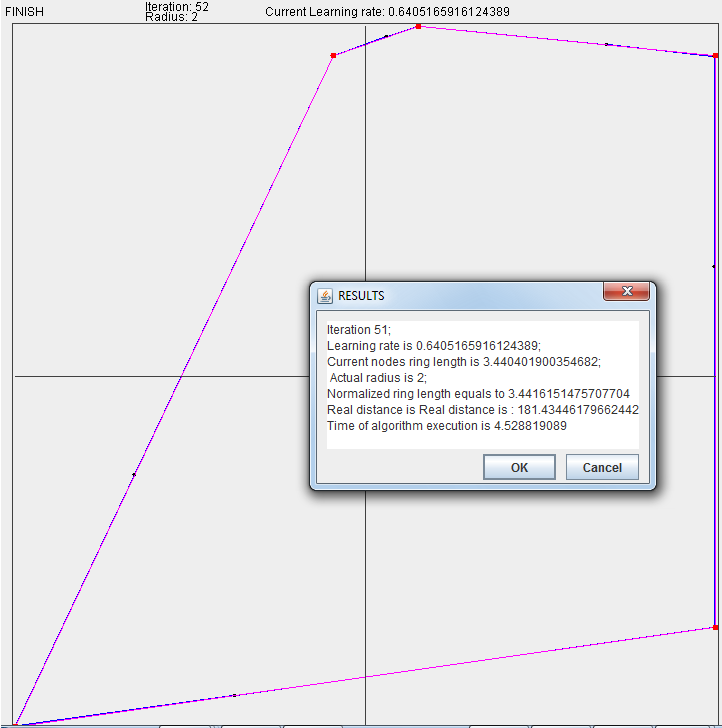
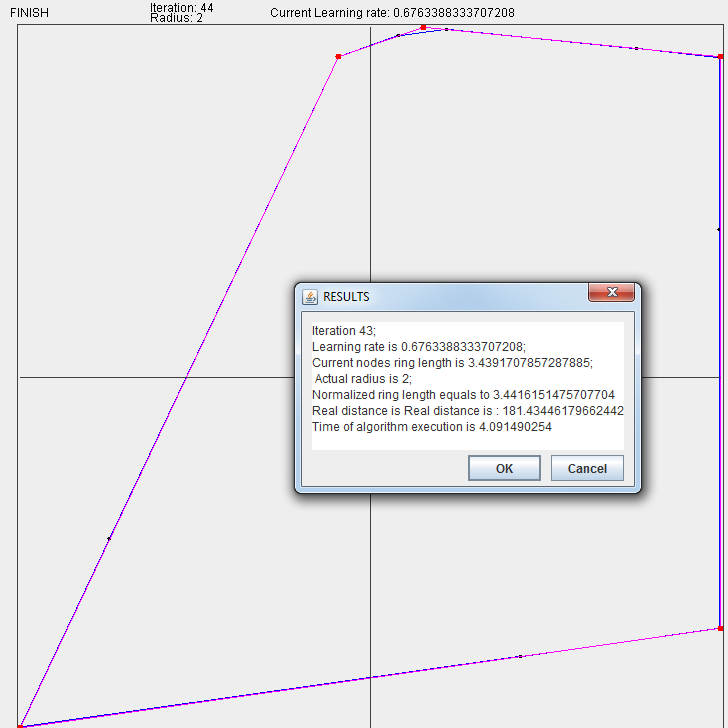


Запустимо алгоритм:





Для наочного представлення було вибрано 3 з найкращих результатів. Як бачимо, значення оптиуму було досягнуто на кожному з них, проте було витрачено різну кількість часу(перший результат має разючу різницю в часі між другим і третім), хоча зоден з параметрів і критеріїв запуску не було зміненою Це спричинено випадковістю вибору міста, проте відсутність цього параметру і заміна його на циклічне виконання ітерації для кожної з точок гарантовано зводить усі результати в єдиний, найчастіше один із не дуже хороших розв’язків(під хорошим мається на увазі розв’язок, що відрізняється від оптимального не більш ніз на 15%, під дуже хорошим – на ~5%).



***Розглянемо перший зображений результат*:**

Через 265 ітерацій отримали відповідь. У цьому випадку варто відмітити дві особливості графіка: шлях нодів зійшовся з абсолютним, алгоритм завершився раніше максимально допустимої кількості ітерацій, що означає, що алгоритм гарантовано знайшов розв’язок, а також те, що в даному випадку шлях був оптимальним.

Оптимальний шлях можливо отримати при правильних настройках коефіцієнтів, проте зі збільшенням кількості міст вірогідність отримання оптимального розв’язку стрімко падає.

Крім того, навіть при малих кількостях міст і однакових параметрах при різних запусках алгоритм може повертати абсолютно різні результати.

Підсумовуючи сказане, оптимальний шлях реально гарантовано отримати на невеликих наборах міст(до 20-30) при правильному підборі коефіцієнтів(зачасту встановлюються експериментально).

Крім того, навіть отримавши оптимальний розв’язок для доволі великої кількості міст(100-200), людина здатна виявити погані і дуже погані розв’язки(лінії шляхів переплітаються між собою, утворюють петлі, колізії), проте не може відрізнити оптимальний і найближчі до нього розв’язки від дуже хороших. Таке можливо лише для наборів міст до 8-10, оскільки в більшості таких ситуацій найкращий шлях є інтуїтивно зрозумілим.

Саме тому для оптимізації швидкості обчислень застосовуються оптимізаційні алгоритми, котрі, жертвуючи абсолютною точністю, дають величезний виграш у швидкодії.

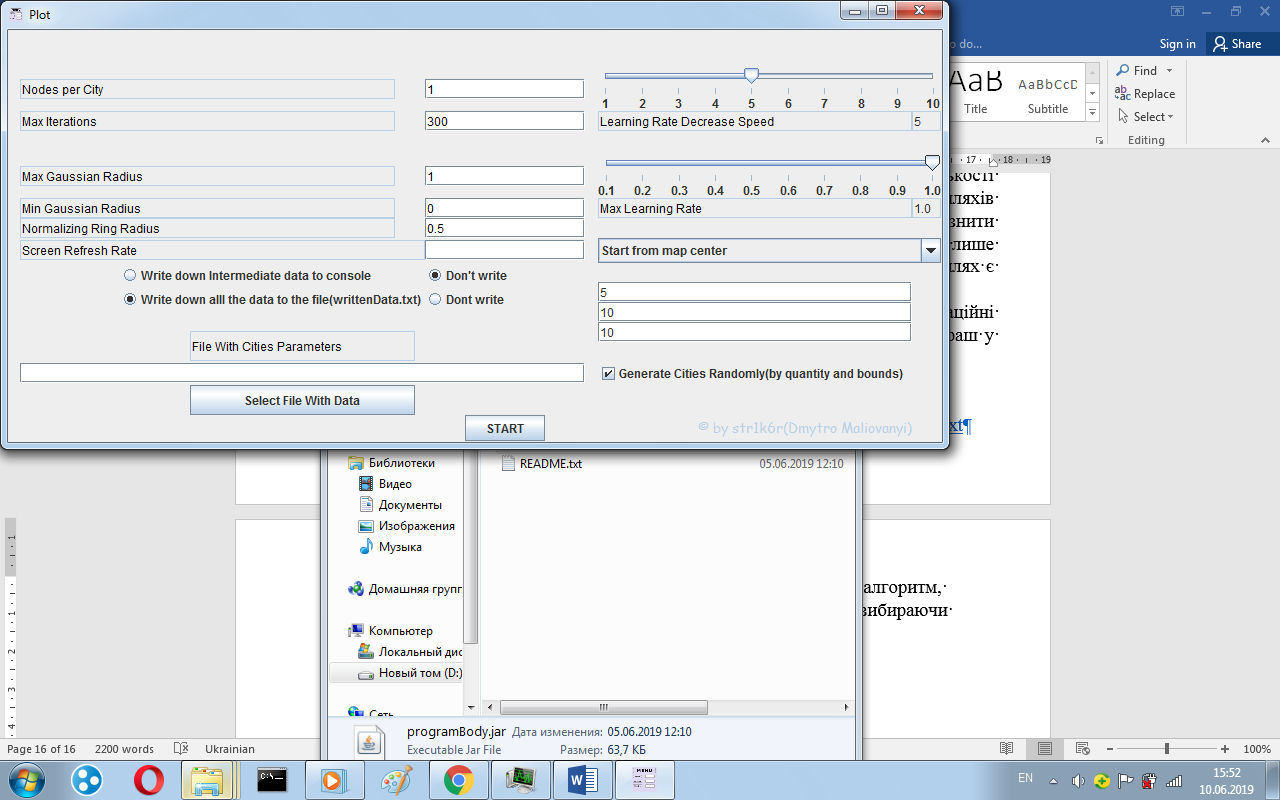
Джерело карти міст: [generatedSources\sampleResult.tsp](generatedSources/sampleResult.tsp)

Ресурс із сесією запусків конкретного алгоритму: [output\sampleResults.txt](output/sampleResults.txt)

**Дійсний наочний розрахунок відповідно до кроків алгоритму**

Проведемо розрахунок для наступної карти міст, використовуючи алгоритм, вказаний вище, і замінивши випадковий генератор індексу міста , вибираючи найбільш вигідне місто для скорочення кількості обчислень:

Стартові параметри:



Отримали наступний набір міст:

1 9 9; 2 7 4; 3 5 2; 4 5 9; 5 7 5;

З цих даних створюється масив розміром 5 елементів з точками: ; ; ; .

Нормалізуємо їх:

Xmin = 5; Ymin = 2; Xmax = 9; Ymax = 9;

Xrange = 9 – 5 = 4; Yrange = 9 – 2 = 7; використаємо формули

thisCityNormalizedX =

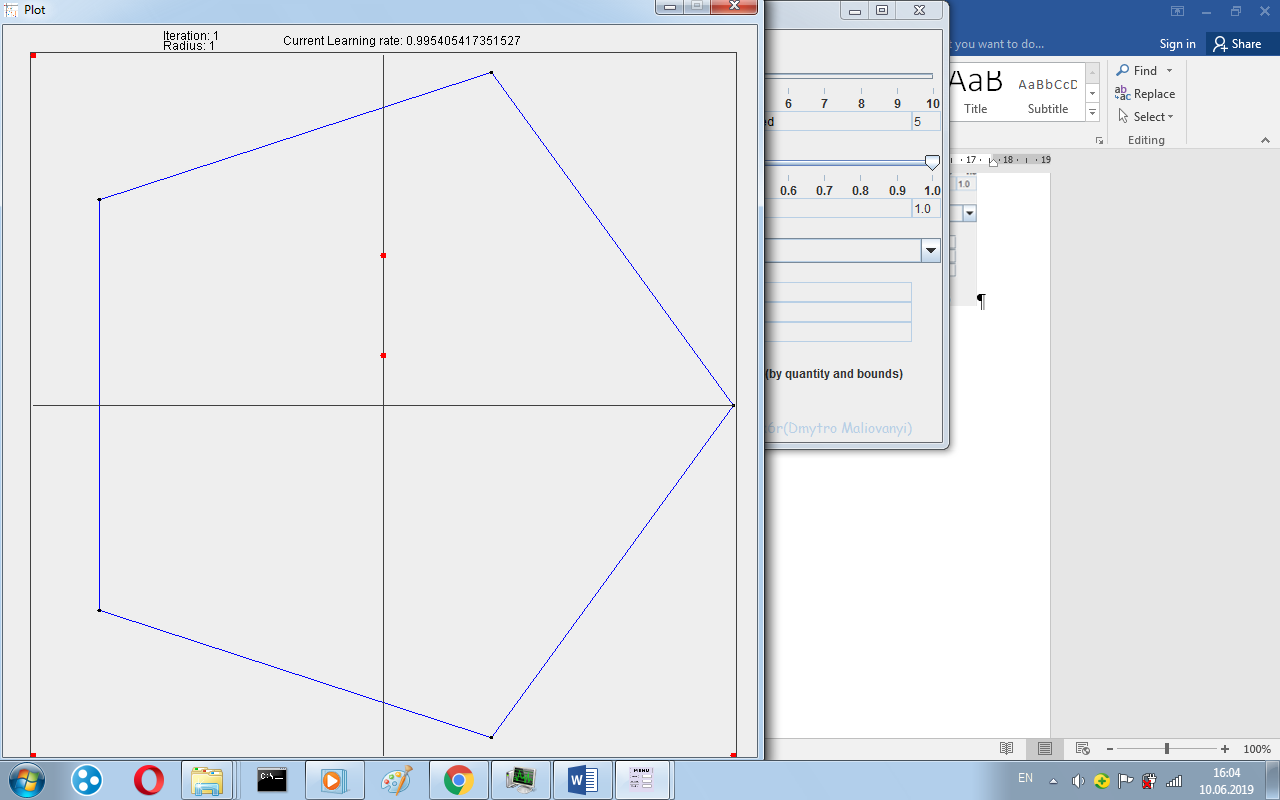
thisCityNormalizedY =

step = = 1,256 rad;

далі за формулами :

обчислимо положення точок кільця у відносних координатах;

Отримаємо наступну карту(точка О(0;0) – верхній лівий кут):



Початок виконання основного тіла алгоритму:

1.Оберемо місто

2.Знайдемо найближчу точку кільця:

tiedUpNode = 3;

3. difX = (0-0,0949)\*1 = -0,0949, difY = (0-0,2069)\*1 = -0,2069;

4.Оскільки currentGaussianRadius = 1, то жодні точки-сусіди не зазнають змін;

5. currentGaussianRadius =

6. currentLearningRate = 1\*

7. currentIteration = currentIteration+1 = 1+1 = 2;

9.Перевіряємо умови завершення алгоритму: очевидно, що жодна з трьох умов не виконується.

10. Оберемо місто

11.Знайдемо найближчу точку кільця:

tiedUpNode = 2;

12. difX = (0-0,0959)\*0,9939 = -0,0953, difY = (1-0,7944)\*0,9939 = 0,2044;

13. currentGaussianRadius = 1\*

14. currentLearningRate = 1\*0,9878;

currentIteration = currentIteration+1 = 2+1 = 3;

15. Жодна з умов не виконується, тому продовжуємо:

16. Оберемо місто

17. Знайдемо найближчу точку кільця:

tiedUpNode = 1;

18. difX = (1-0,6548)\*0,9878 = 0,341 , difY = (1-0,)\*0,9878 = 0,0243;

19. currentGaussianRadius = 1\*

20. currentLearningRate = 1\*0,9818;

currentIteration = currentIteration+1 = 3+1 = 4;

21. Жодна з умов не виконується, тому продовжуємо:

22. Оберемо місто

23. Знайдемо найближчу точку кільця: )

tiedUpNode = 4;

24. difX = (0,5-)\* 0,9818 = -0,151 , difY = (0,286-0,0241)\* 0,9818 = 0,2572;

25. currentGaussianRadius = 1\*

26. currentLearningRate = 1\*0,9758;

currentIteration = currentIteration+1 = 4+1 = 5;

27. Жодна з умов не виконується, тому продовжуємо:

28. Оберемо місто

29. Знайдемо найближчу точку кільця : найближчою з-поміж усіх є точка), проте вона вже зазначена як tiedUpNode для міста , тому обрано точку ;

tiedUpNode = 5;

30. difX = (0,5-0,9999)\*0,9758 = -0,4878 , difY = (0,429 -0,)\*0,9758 = 0,0243;

31. currentGaussianRadius = 1\*

32. currentLearningRate = 1\*0,9698;

currentIteration = currentIteration+1 = 5+1 = 6;

33. Таким чином, було виконано по одній ітерації для кожного міста. Як видно з результатів, на даний момент лише одне місто має точку, що знаходиться у межах minLearningRate. Отже, виконується наступний цикл ітерацій:

34. Оберемо місто

35. На даний момент до нього прив’язана точка кільця . Провівши повторний обрахунок, отримаємо: . Отже, найближча точка залишилася прив’язаною.

tiedUpNode = 5;

36. difX = (0,5-0,5121)\*0,9698 = -0,01173 , difY = (0,429-0,4307)\*0,9698 =

= -0,00164;

37. currentGaussianRadius = 1\*

38. currentLearningRate = 1\*0,96383;

currentIteration = currentIteration+1 = 6+1 = 7;

39. Два міста відповідають третій умові. Продовжуємо виконання алгоритму:

40. Оберемо місто

41. На даний момент до нього прив’язана точка кільця . Провівши повторний обрахунок, отримаємо: . Отже, найближча точка залишилася прив’язаною.

tiedUpNode = 4;

42. difX = (0,5-0,5023)\*0,96383 = -0,00222 , difY = (0,286-0,2813)\*0,96383 =

= 0,00453;

43. currentGaussianRadius = 1\*

44. currentLearningRate = 1\*0,95793;

currentIteration = currentIteration+1 = 7+1 = 8;

45. Три міста відповідають третій умові. Продовжуємо виконання алгоритму:

46. Оберемо місто

47. На даний момент до нього прив’язана точка кільця . Провівши повторний обрахунок, отримаємо: . Отже, найближча точка залишилася прив’язаною.

tiedUpNode = 4;

48. difX = (1-0,9958)\*0,95793 = 0,00403 , difY = (1-0,9997)\*0,95793= 0,00029

49. currentGaussianRadius = 1\*

50. currentLearningRate = 1\*0,95207;

currentIteration = currentIteration+1 = 8+1 = 9;

51. Чотири міста відповідають третій умові. Продовжуємо виконання алгоритму:

52. Оберемо місто

53. На даний момент до нього прив’язана точка кільця . Провівши повторний обрахунок, отримаємо: Отже, найближча точка залишилася прив’язаною.

tiedUpNode = 4;

54. difX = (0-0,0006)\*0,95207 = -0,00057 , difY = (1-0,9988)\*0,95207= 0,00114

55. currentGaussianRadius = 1\*

56. currentLearningRate = 1\*0,94624;

currentIteration = currentIteration+1 = 9+1 = 10;

57. Усі 5 міст задовольняють третю умову завершення виконання алгоритму:

ifSolutionCloseEnough == true;

параметр ifSolutionCloseEnough набуває значення true тоді і лише тоді, коли біля кожного міста розташовано точка, що знаходиться на відстані не більше ніж minimumLearningRate(зазвичай 2) від нього;

Інакше ifSolutionCloseEnough надається значення false.

Наразі маємо ifSolutionCloseEnough = true;

Відповідно finalConditionsAreDone == true;

Це означає, що основне тіло алгоритму закінчило роботу.

Далі виконується блок

На цьому алгоритм припиняє виконуватися.

Вікна з даним алгоритмом після цього можна закрити(або зробити скріншоти), і якщо результат був незадовільним(або за бажанням користувача чи у цілях експерименту) змінити значення параметрів на розсуд користувача та повторити запуск.

Варто звернути увагу, що якщо користувач хоче повторно запустити карту, отриману як результат автоматичного створення випадкової карти, слід забрати вибір з чекбоксу ‘Generate Cities Randomly’ та вибрати відповідний файл з папки ‘generatedSources’. Назва потрібого файла міститиме кількість міст, що були згенеровані.

Також слід відмітити, що файл зі згенерованими містами оновлюється по завершенню виконання алгоритму, отож не слід поспішати, бо такий поспіх може привести до отримання неочікуваних результатів.

Візуалізація завершення роботи алгоритму та виконання блоку

