Федеральное государственное автономное образовательное учреждение высшего образования

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

**Факультет информатики, математики и компьютерных наук**

**КУРСОВАЯ РАБОТА**

Реализация муравьиного алгоритма для решения задач комбинаторной оптимизации

по направлению подготовки 09.03.04 "Программная инженерия"

образовательная программа «Программная инженерия»

Выполнил:

Студент группы 19ПИ-1

Балаян Роман Каренович

Руководитель:

Лемайкина Елена Александровна,

старший преподаватель

Нижний Новгород 2019

Содержание

[1.Введение 3](#_Toc42846690)

[2. Постановка задачи 3](#_Toc42846691)

[3. Существующие решения 6](#_Toc42846692)

[4. Муравьиный алгоритм 7](#_Toc42846693)

[5. Программная реализация 9](#_Toc42846694)

[6. Результаты работы программы 17](#_Toc42846695)

[7. Заключение 20](#_Toc42846696)

GitHub: https://github.com/R0maFUN/AntsColonyUI

1.Введение

На сегодняшний день существует множество задач, для которых не существует определенного алгоритма, способного предоставить точное решение за приемлемое время. На помощь приходят приближенные методы решения задач комбинаторной оптимизации, предоставляющие решения, близкие к идеальному, но тратящие на его поиск в разы меньше времени. В данной работе будет рассмотрена проблема маршрутизации на примере перевозки грузов со склада по магазинам. Целью будет являться нахождение наименее затратного пути, посетив при этом все магазины. Существует достаточно много видов алгоритмов для решения задач такого рода, наиболее популярными из них являются: метод эластической сети, генетический алгоритм, муравьиный алгоритм. Именно последний будет рассмотрен в данной работе.

2. Постановка задачи

Задача заключается в проблеме маршрутизации. При большом количестве вершин, которые нужно посетить, обычные алгоритмы становятся неэффективными по времени. Суть задачи: существует склад, грузовики, а также какое-то количество магазинов, работающих в определенный промежуток времени. Нужно найти оптимальный путь, по которому будут двигаться грузовики для доставки нужных товаров во все магазины.

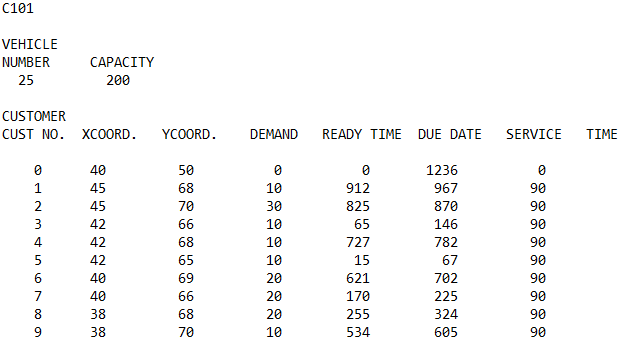
Решение будет находиться при помощи Муравьиного алгоритма. Суть данного подхода заключается в моделировании жизни муравьев, которые находят ближайший путь к источнику питания при помощи феромонов. Более детальное описание данного алгоритма будет представлено в пункте 4

Необходимо разработать программу на языке С++ для построения и отображения оптимального пути грузовиков при перемещении между магазинами и складом. Реализовать муравьиный алгоритм, а также обеспечить удобную работу с программой. Приложение должно иметь следующий функционал:

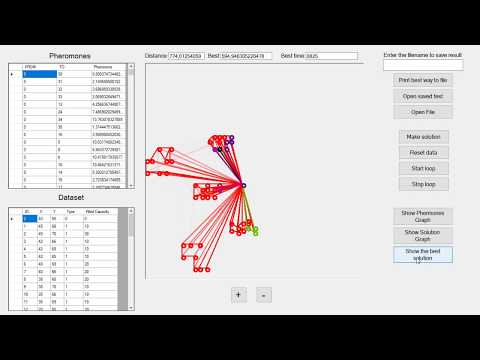
1. Чтение входных данных.
2. Отображение текущих данных в виде таблиц.
3. Построение графов.
4. Вывод наилучших решений в удобном формате.

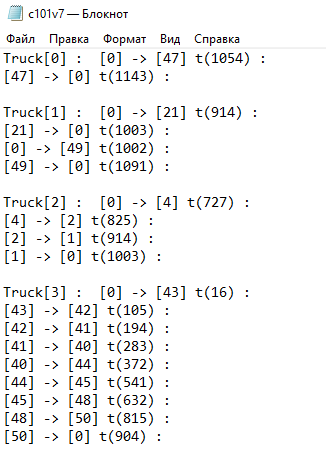
Входные данные представлены в виде таблицы, содержащей информацию о грузовиках (количество и вместимость) и магазинах/складах (координаты, время работы, количество единиц необходимого товара). Выходными данными является иллюстрация найденного пути (последовательное отображение откуда, куда и какой грузовик поехал, а также время прибытия).

Пример входных данных:



Пример выходных данных:

[](https://www.youtube.com/embed/IrzAO0ybiyw?feature=oembed)



3. Существующие решения

Существуют следующие классификации техник поиска решений для задач данного типа:

1. Exact Approaches(точные подходы)

Наиболее успешным решением здесь является “Branch and bound”, которое использует стратегию “divide and conquer” для деления задачи на подпроблемы, а затем оптимизируя каждую подпроблему.

2. Heuristic (эвристический)

Эвристические методы как правило исследуют небольшой объем решений, но при этом находят довольно хорошие за небольшое количество времени.

3. Constructive methods

При использовании данного метода решение создается постепенно, следя за затратами, но не содержат в себе этап улучшения.

Наиболее популярным здесь является “Savings Algorithm” , сначала он вычисляет все возможные пути для всех машин, а затем находит и соединяет несколько наиболее подходящих дорог в одну.

4. 2-Phase Algorithm

Задача разбивается на 2 компонента: (1) сортировка вершин по возможным маршрутам (на некие районы) и (2) само построение маршрутов с возможностью пересортировать вершины.

5. Metaheuristics

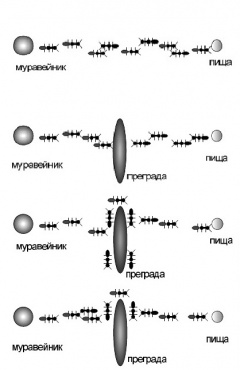
- Genetic Algorithm. На данный момент является наиболее популярным среди мета-эвристических алгоритмов. Он симулирует природную эволюцию, используя селекции, мутации, естественный отбор и т.п. На входе у него может быть случайное решение, но через какое-то время это решение эволюционирует, “приспособляется” и становится достаточно близким к идеальному. Отличительным фактором является то, что генетический алгоритм работает не с одним решением, а сразу с несколькими(“популяцией”).

- Simulated annealing. Имитация отжига, симуляция физического процесса , когда при некоторых температурах атомы уже выстроились в кристаллическую решетку, но все еще могут менять своё место с какой-то вероятностью

-Ants Algorithm.

4. Муравьиный алгоритм

Это мета-эвристический алгоритм, предназначенный для решения задач, связанных с маршрутизацией, поиска пути на графах и т.п. Суть заключается в симуляции жизни колонии муравьёв. Муравьи обладают некоторым феромоном, который остается на поверхности, где побывал муравей. Чем чаще по такой тропинке ходят муравьи, тем больше феромона на ней остается. Стоит заметить, что чем длиннее дорожка, тем быстрее феромон на ней исчезает. В совокупности всех факторов, на более выигрышных путях остается больше феромона.



Дойдя до преграды, муравьи с равной вероятностью будут обходить её справа и слева. То же самое будет происходить и на обратной стороне преграды. Однако, те муравьи, которые случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащён феромоном. Поскольку движение муравьёв определяется концентрацией феромона, то следующие будут предпочитать именно этот путь, продолжая обогащать его феромоном до тех пор, пока этот путь по какой-либо причине не станет недоступен.

Таким образом мы формируем граф, на каждом ребре которого будет находиться какое-то число феромона, а выбор ребра, по которому будет проезжать грузовик будет зависеть от количества феромона на нём. Количество феромона напрямую отражается на вероятности прохода по ребру.

Применяя данный алгоритм к поставленной задаче, будем называть муравьями – грузовики, муравейником – склад, а источниками питания – магазины.

Краткое описание реализации главного алгоритма:

1. Берём первый грузовик и обнуляем текущее время

2. Отбираем магазины, которые будут готовы принять грузовик ко времени, когда он приедет.

3. Для текущего грузовика запускаем функцию выбора следующего пункта назначения. (Выбирается исходя из вероятностей(чем больше феромона на ребре и меньше длина ребра, тем больше вероятность перехода по нему)).

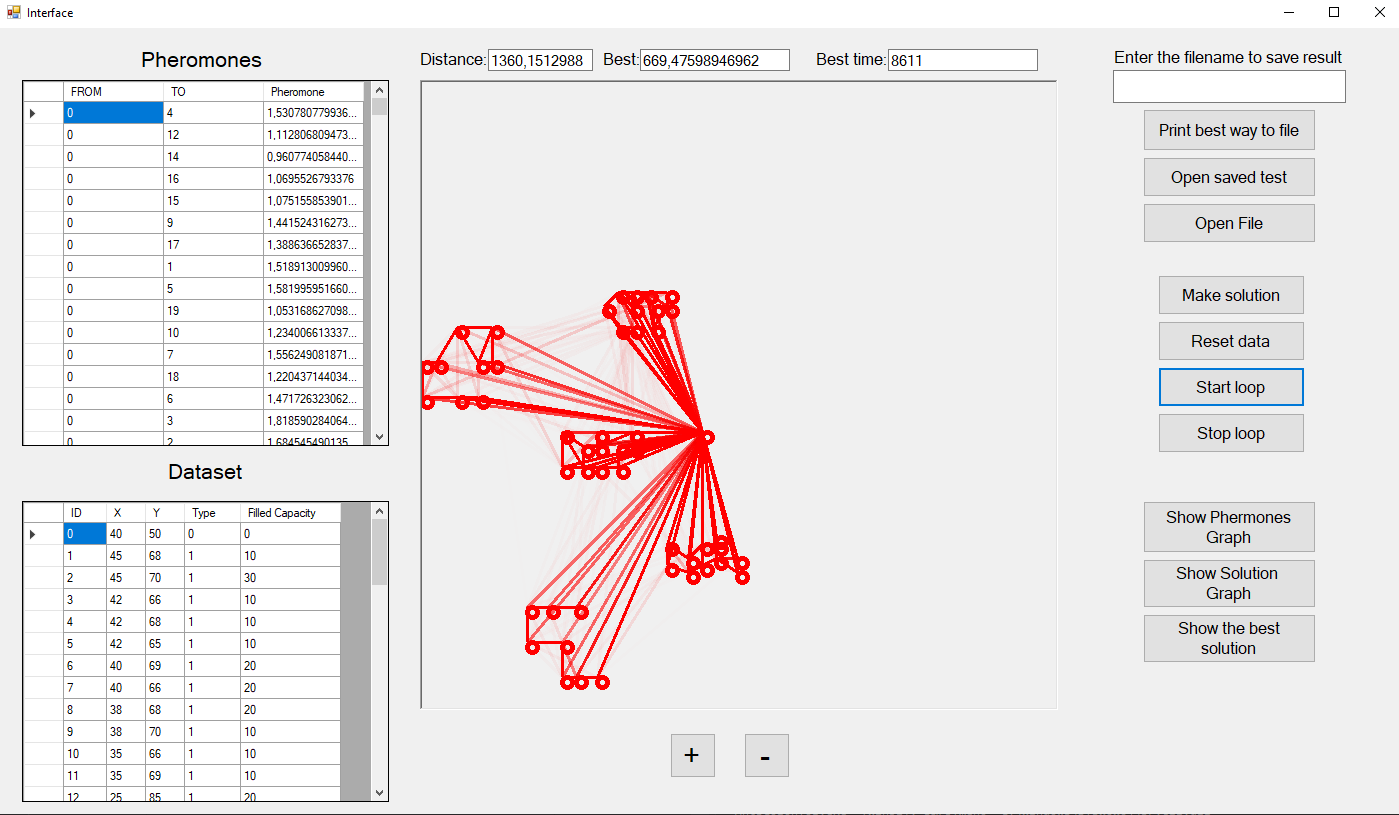
4. Обновляется время, на выбранном ребре увеличивается количество феромона.

5. Уменьшается количество феромона на всех рёбрах.

6. Повторяем пока есть подходящие вершины и грузовик не пуст. Затем переходим к следующему грузовику.

5. Программная реализация

Интерфейс:



“Show Pheromones Graph” рисует граф по текущим феромонам на ребрах, чем насыщеннее цвет – тем больше феромона на ребре.

“Show Solution Graph” последовательно рисует последнее решение, каждая машина рисуется своим цветом.

“Show the best solution” последовательно рисует лучшее решение.

“Print best way to file” – печатает лог лучшего решения в файл, название которого задано в TextBox.

“Open saved test” – открывает стандартный тест “C101.txt”

“Open file” – открывает диалоговое окно, в котором можно выбрать файл со входными данными, после выбора файла, все данные автоматически инициализируются.

“Make solution” – запускает функцию основного алгоритма, которая построит решение.

“Reset data” – установит все феромоны на 1

“Start loop” – построит 50 решений, при этом феромоны на каждом новом решении обнуляться не будут.

“+” и “-” увеличивают и уменьшают изображение графа.

“Pheromones” table – содержит информацию о феромонах на всех ребрах

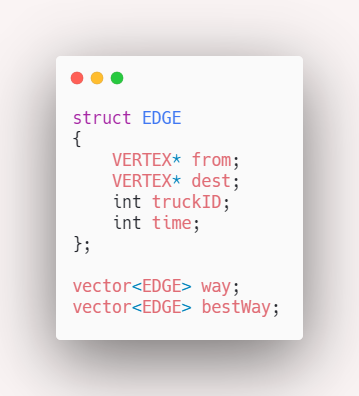
“Dataset” table – содержит данные о вершинах

Создан класс вершины:

Вершины хранятся в: vector<VERTEX\*> vertexes;

Класс грузовика

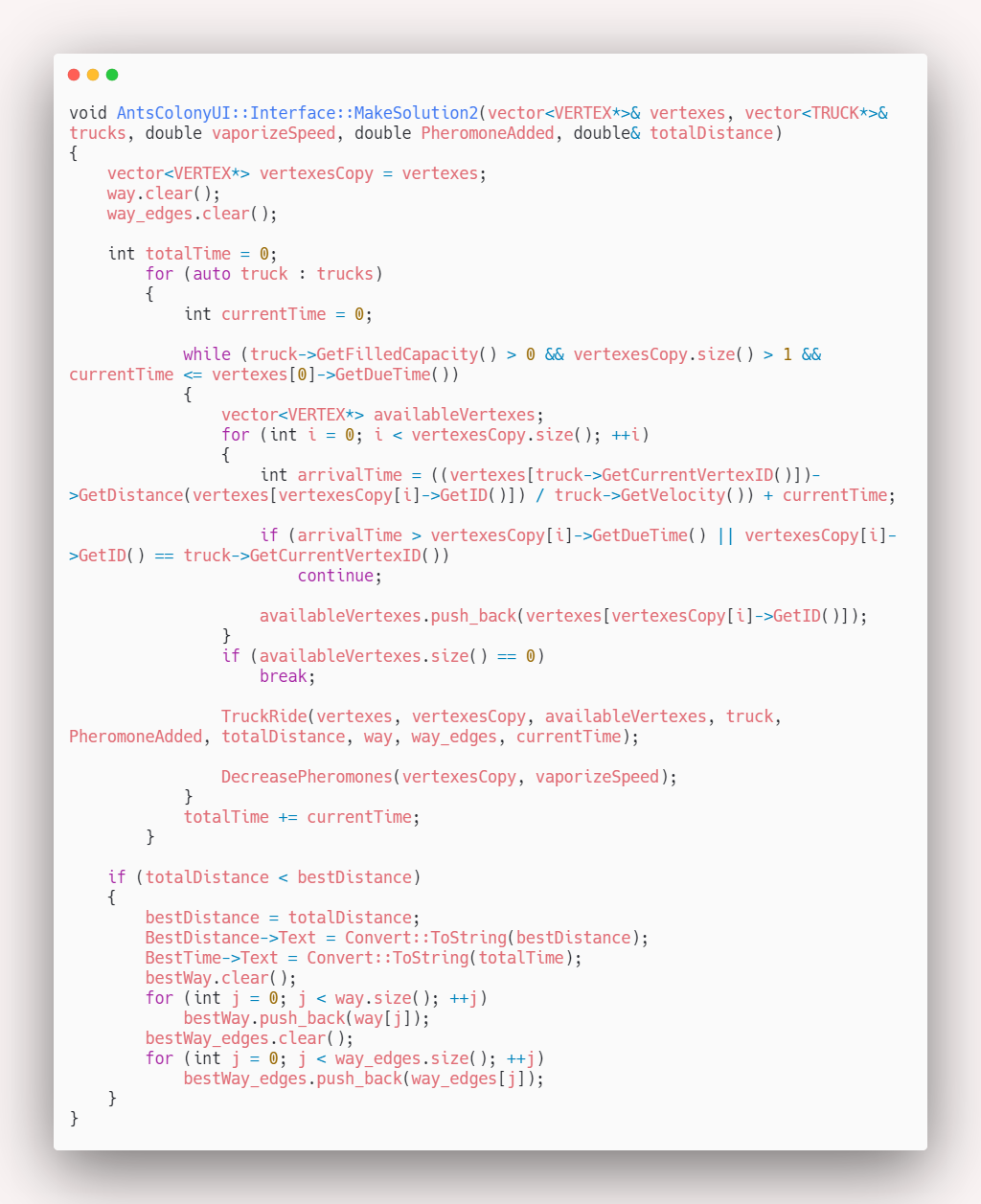
Текущий и лучший пути сохраняются в следующих переменных



Функция отрисовки графа феромонов



Функция создания решения



Функция перемещения в новую вершину

Функция выбора следующей вершины



Если мы выполнили запрос магазина(заполнили его максимально), тогда удаляем его из копии вектора, хранящего вершины. После того, как решение сформировано, феромоны в оригинальном векторе обновятся(так как в копии мы храним указатели на те же вершины), а элементы удалены из него не будут. Экспериментально выяснено, что данный подход позволяет существенно экономить время и память, т.к. не приходится создавать много новых данных и класть их в копию.

В этих трёх функциях происходит всё самое главное.

(TruckRide) Поиск следующей вершины с учетом вероятностей(GetNextVertexID), обновление текущего времени для данного грузовика, занесение ребра в вектор(для сохранения решения), добавление феромона на ребро, по которому мы только что прошли, перемещение данного грузовика в следующую вершину.

Далее мы возвращаемся в MakeSolution2 и уменьшаем феромоны на всех ребрах.

Стоить заметить, что мною были подобраны наиболее подходящие формулы и коэффициенты для более эффективного решения.

**Добавление феромона:** new+=

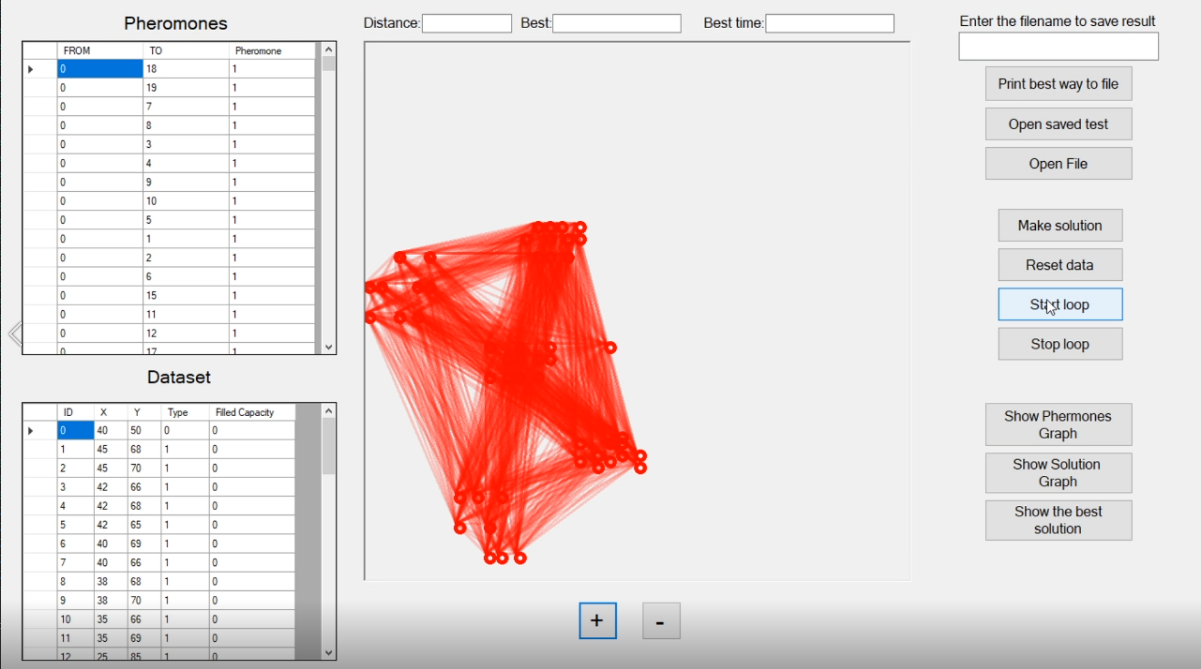
**Испарение феромона:** new = new\*(1 - vaporizeSpeed); где vaporizeSpeed = 0.0008

**Вероятность перехода:** (количество феромона) +

6. Результаты работы программы

C101.txt

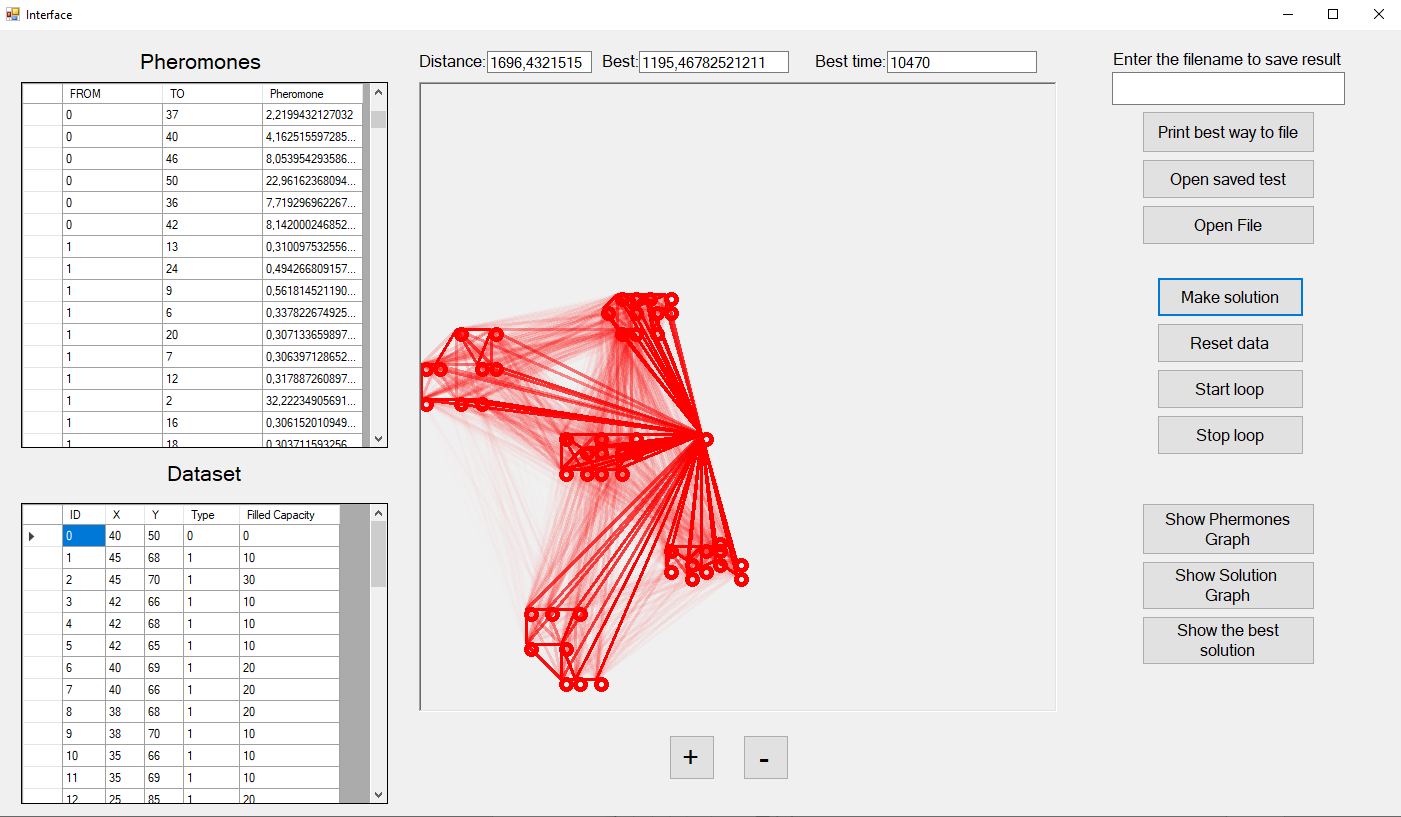
Начало



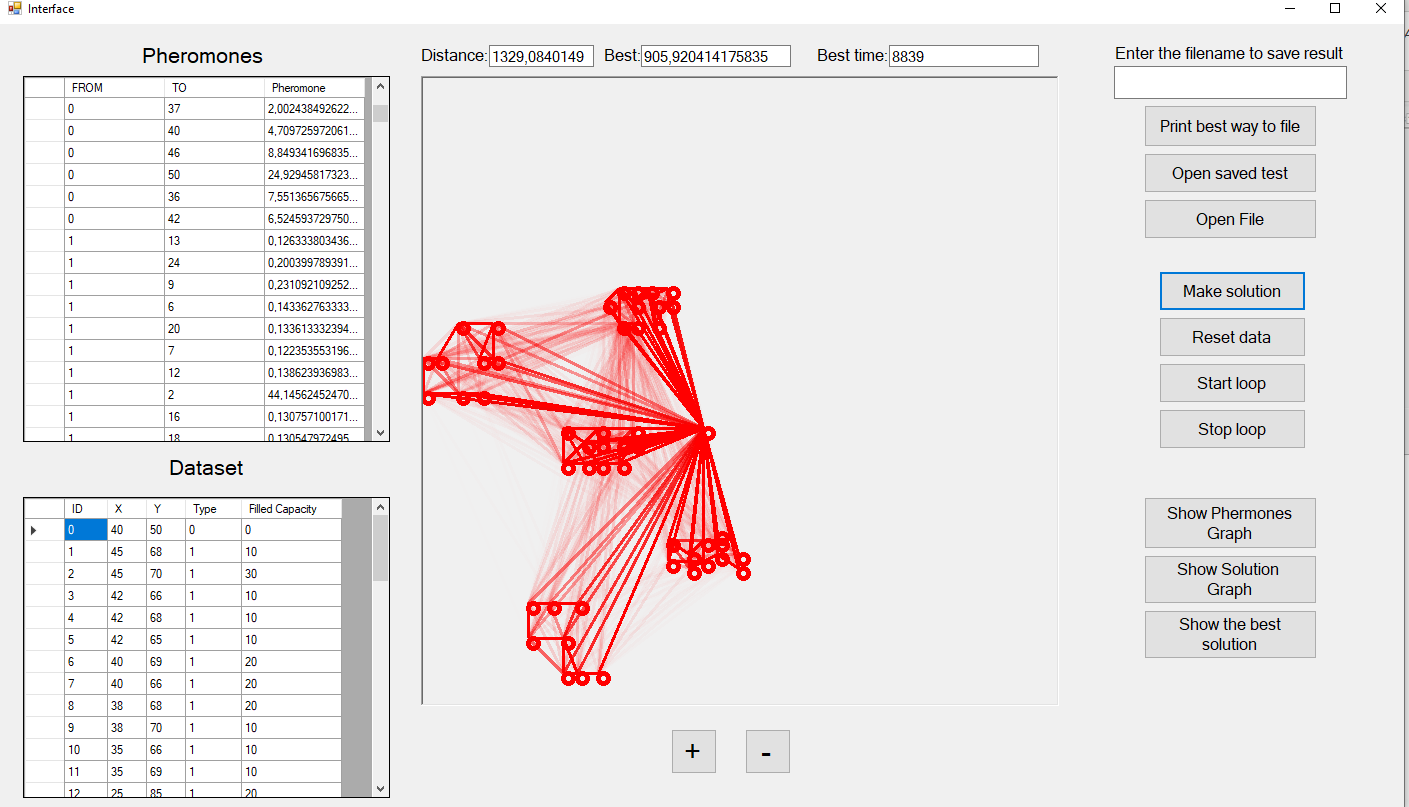
Первые 50 итераций



100 итераций

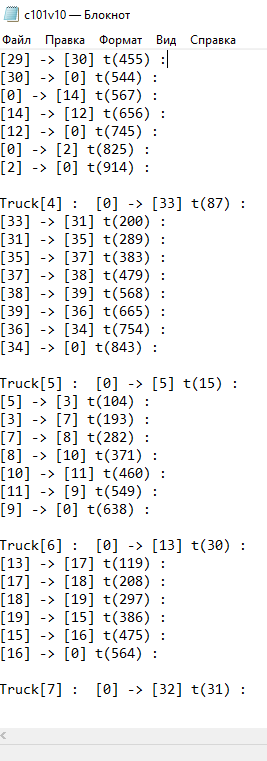
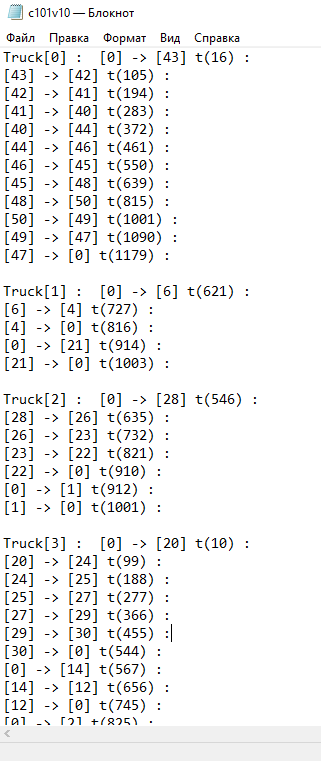


200 итераций



450 итераций





В результате обхода магазинов, отсутствуют какие-либо прыжки из района в район, обход происходит достаточно оптимально. Не могу сказать точно, но на первый взгляд получившиеся результаты даже лучше результатов сайта, с которого были взяты данные тесты.

7. Заключение

В дальнейшем хотелось бы для начала проверить, действительно ли мои результаты лучше результатов того сайта. Но и без этого считаю полученный результат хорошим. Благодаря данной работе научился работать с windows forms/ c++ cli. А также пополнил свой список самостоятельно написанных мета-эвристических алгоритмов.

Еще хотелось бы добавить в данную программу что-то из 2-Phase Algorithm, так как в приведенных тестах магазины делятся на небольшие кучки(группы), вероятно можно было бы их объединять и строить решения для отдельных участков. Также я уверен, что можно улучшить алгоритм подбора коэффициентов в формулах для еще более точных результатов.

GitHub: https://github.com/R0maFUN/AntsColonyUI