|  |  |
| --- | --- |
| Name: | Edward Eisenberger |
| ID# | 1066164 |
| Assignment 8 | |
| Date of Submission | August 2, 2019 |

Table of Contents

[Overview 3](#_Toc15655328)

[Part 1: Convert Multithreaded GA to WCF Service 3](#_Toc15655329)

[Summary 3](#_Toc15655330)

[Results 3](#_Toc15655331)

[Part 2: Distributed System of Web Services 5](#_Toc15655332)

[Summary 5](#_Toc15655333)

[Results 5](#_Toc15655334)

[Part 3: Biogeography Based Optimization Exchange of Population 6](#_Toc15655335)

[Summary 6](#_Toc15655336)

[Result 6](#_Toc15655337)

# Overview

Assignment 08 consisted of converting a Multithreaded Genetic Algorithm implementation of the Travelling Salesman that implements the island exchange model (TSGA\_MT) into a distributed system of web services with periodic exchange.

# Part 1: Convert Multithreaded GA to WCF Service

## Summary

Part 1 consisted of converting a Multithreaded Genetic Algorithm implementation of the Travelling Salesman that implements the island exchange model (TSGA\_MT) into a WCF service. The WCF service provides a callback interface so that the client can periodically be updated with the best result from the service.

A WCF Service, GAWebHost, was created which provides the GAWebHost.IDistributor interface for initiating the Genetic Algorithm and the GAWebHost.ICallback interface so the client may be receive periodic updates. The ICallback interface provides two contracts, MOnUpdate and MOnComplete. The GAWebHost service provides the latest best results to the client via MOnUpdate just before each exchange. When the algorithm completes, the GAWebHost service informs the client and provides the latest best results via the MOnComplete interface.

## Results

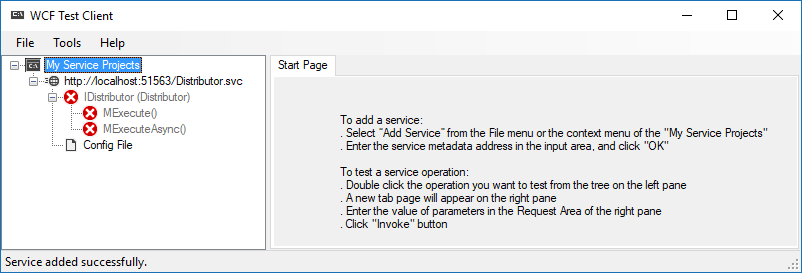


Figure - WCF Service: GAWebHost

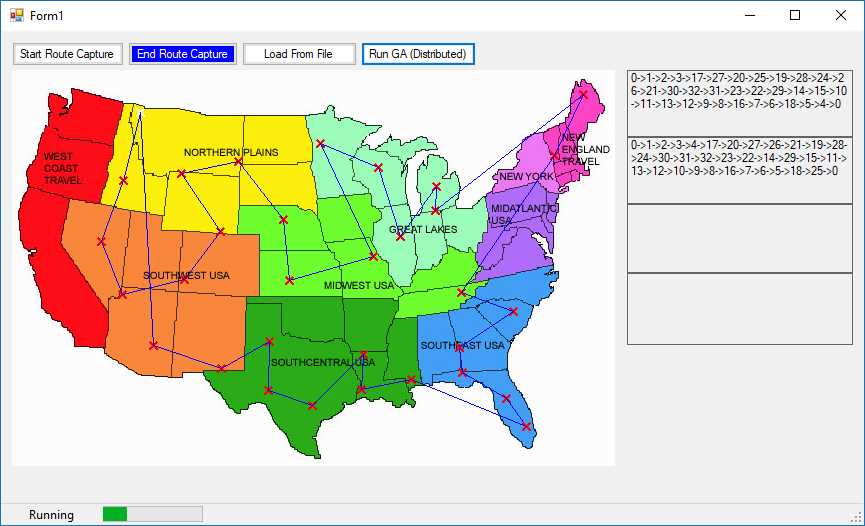


Figure – GAWebClient

Figure 2 illustrates client’s view. Each box on the right contains the result from one of the distributed services described in Part 2: Distributed System of Web Services. The bottom left corner of the window indicates the status of the algorithm: Idle, Running, or Complete. As results are received from the GAWebHost, the status and progress bar are updated.

# Part 2: Distributed System of Web Services

## Summary

Part 2 consisted of implementing a distributed system of web services, where each service implements the multithreaded genetic algorithm for the travelling salesman problem with periodic exchange.

Two WCF Services, GAWebServant and GAWebServant 2, were created. Each servant implements an ICallback interface and an IWorker interface. The GAWebHost.IDistributor uses the IWorker interface to command the servant to execute a specified number of iterations on the Genetic Algorithm. When a servant completes its work, it notifies the GAWebHost.IDistributor service via the ICallback.MOnComplete method. When all servants complete their work, the GAWebHost.IDistributor performs the random exchange between services, then commands them to run again. This cycle repeats until the specified total number of iterations has completed.

## Results

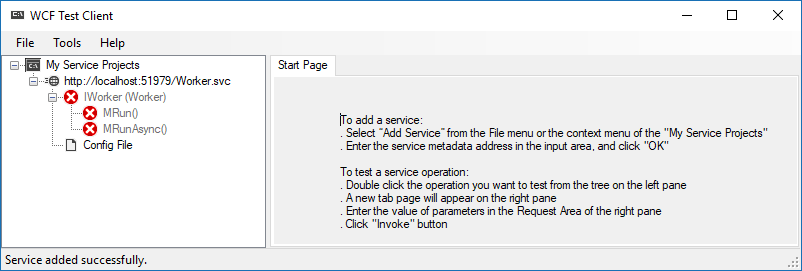


Figure - WCF Service: GAWebServant

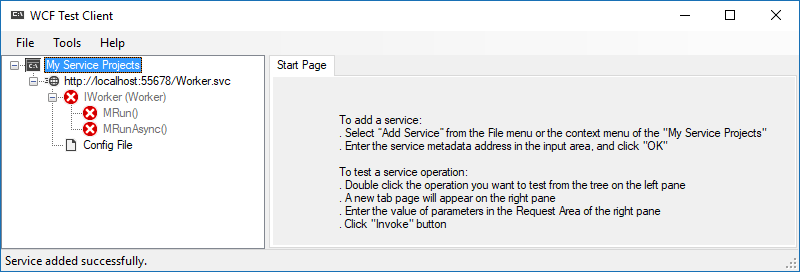


Figure - WCF Service: GAWebServant2

# Part 3: Biogeography Based Optimization Exchange of Population

## Summary

Part 3 consisted of read the “Biogeography based Optimization” paper, authored by Dan Simon, then implementing the exchange of population according to the proposed algorithm.

This was accomplished by modifying the WCF Service, GAWebHost. The GAWebHost Distributor service provides two exchanges methods: mExchangeData and mExchangeBBO. The mExchangeData method implements the same logic as provided by the TSGA\_MT application. The mExchangeBBO method implements the Biogeography based Optimization algorithm. The exchange algorithms can be switched by modifying one line of code, recompiling, and relaunching the service.

The mExchangeBBO method iterates through each island (Servant) and calculates the immigration probability, λ, and emigration probability, µ, based on the best population in each island. The method then iterates through each island again and generates a random number. If the random is greater than the island’s λ, then an inner loop starts which iterates through the remaining islands and generates a random number. If the random number method is greater than the island’s µ, two randomly selected populations are exchanged between each island.

## Result

There are no further screen shots which would distinguish the two implementations. The results of the BBO implementation appear to be as efficient as the original implementation based on the small sample size of tested inputs.