**A SELF-TUNING FIREFLY ALGORITHM TO SOLVE THE TRAVELLING SALESMAN PROBLEM**

**A thesis by**

**T.K. HETTIARACHCHI**

**CO.SC/13/264**

**Supervised by**

**Prof. A.S. Karunananda**

**Ms. M.K.A. Ariyaratne**

**Submitted in partial fulfillment of the requirements for the award of BSc (Hons) in Computer Science Degree**

**GENERAL SIR JOHN KOTELAWALA DEFENCE UNIVERSITY**

**2016**

Author’s Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma in any University or equivalent institution, and that to the best of my knowledge and belief, contains no material previously submitted or written by any other person , except where due reference is made in the text of this thesis.

I carried out the work described in this dissertation under the supervision of Prof. A.S. Karunananda and Ms. M.K.A. Ariyaratne.

Signature :

Name of the Student : T. K. Hettiarachchi

Registration No : Co.Sc/13/264

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

Signature :

Name of Supervisor : Prof. A.S. Karunananda

Signature :

Name of Supervisor : Ms. M.K.A. Ariyaratne

Abstract

Optimization is a key component in solving various types of problems, in almost every field of research and in industries. Even though some problems can be solved using conventional algorithms, NP-hard problems are pretty much impossible to be solved in that manner. The Traveling Salesman Problem (TSP) is one such problem, where the time complexity increases exponentially with the increase in the number of cities, making it unsolvable using conventional optimization algorithms using the resources available up to date. Nature-Inspired algorithms can be used to solve these kind NP-hard problems. However, these algorithms have algorithm-specific parameters and identifying the optimal value for these variables is normally done using trial and error method, where the values are statistically compared against the performance or the final result. To automate this process, if we use another algorithms it will create an endlessly recurring process because the algorithms applied should be well-tuned. Therefore, to overcome this problem, we should design a multi-objective algorithm where the algorithm itself optimizes its variables. So, one objective of the algorithm will be to optimize the solution while the other objectives will be to optimize the parameters of the algorithm. Such algorithms are called self-tuning (hyper-optimization) algorithms. The aim of this research was to design a self-tuning optimization algorithm using Firefly algorithm to solve the Traveling Salesman Problem. So that, with the solution described by this thesis, the user is able to find the optimized path for the given Travelling Salesman Problem, without worrying about adjusting the parameters of the Firefly Algorithm. It is implemented to solve the Traveling Salesman Problem and is tested using TSPLIB, which is a library of sample instances for the TSP.

Table of Contents

[Author’s Declaration i](#_Toc468357805)

[Abstract ii](#_Toc468357806)

[Table of Contents iii](#_Toc468357807)

[List of Figures v](#_Toc468357808)

[List of Tables vi](#_Toc468357809)

[Chapter 1 Introduction 1](#_Toc468357810)

[1.1 Preamble 1](#_Toc468357811)

[1.2 What are Nature-Inspired Algorithms? 1](#_Toc468357812)

[1.3 Traveling Salesman Problem 2](#_Toc468357813)

[1.4 Problem in brief 3](#_Toc468357814)

[1.5 Aim of the research 3](#_Toc468357815)

[1.6 Objectives of the research 4](#_Toc468357816)

[1.7 Hypothesis 4](#_Toc468357817)

[1.8 Overview of the thesis 4](#_Toc468357818)

[1.9 Summary 4](#_Toc468357819)

[Chapter 2 State of the art 5](#_Toc468357820)

[2.1 Introduction 5](#_Toc468357821)

[2.2 Firefly Algorithm 6](#_Toc468357822)

[2.2.1 Discretization of FA 8](#_Toc468357823)

[2.3 Summary 9](#_Toc468357824)

[Chapter 3 Methodology 10](#_Toc468357825)

[3.1 Introduction 10](#_Toc468357826)

[3.2 Approach 10](#_Toc468357827)

[3.2.1 Input 10](#_Toc468357828)

[3.2.2 Process 11](#_Toc468357829)

[3.2.3 Output 11](#_Toc468357830)

[Chapter 4 Evaluation 13](#_Toc468357831)

[4.1 Introduction 13](#_Toc468357832)

[References 14](#_Toc468357833)

[Appendix A Self-Tuning Algorithm 16](#_Toc468357834)

List of Figures

[Figure 1: Pseudo code of the original firefly algorithm 7](#_Toc468352154)

[Figure 2 The permutation representation of a solution 8](#_Toc468352155)

[Figure 3 Final route in the form of an array 11](#_Toc468352156)

[Figure 4 Final Results 12](#_Toc468352157)

[Figure 5 Optimum route 12](#_Toc468352158)

.

List of Tables

**No table of figures entries found.**

# Introduction

## Preamble

Optimization is an essential component in solving various types of problems, in almost every field of research and industry. It can be either reducing time complexity, space complexity or both. Since the industrial revolution, people have been interested in making machines that will reduce the workload of humans. Advancement of this area happened at a rapid rate where we have reduced the physical workload, as well as the mental workload of people with the development of computers, where we implement various optimization algorithms, and by using them; the computer will solve the problem for us. Even though some problems can be solved using conventional optimization algorithms, NP-hard problems are pretty much impossible to be solved in this manner. In computational complexity theory, NP-hard (Non-deterministic Polynomial-time hard) is a class of problems which are at least as difficult as the hardest problems in NP (Non-deterministic Polynomial-time). NP is a class of computational decision problems, for which a given solution can be verified in a polynomial time using a non-deterministic Turing machine.

Traveling Salesman Problem (TSP) is one of the most popular NP-hard problems (Applegate et al., 2011). The problem can be simply described as, the salesman has to visit each city one time and return back to the city where he began the tour, using the most optimized path. This may seem easy but as the number of cities increases the time complexity increases exponentially, making it unsolvable using conventional optimization algorithms utilizing the resources available up to date.

## What are Nature-Inspired Algorithms?

Nature is still a mystery to modern day science. It has inspired number of research in a miraculous way, to make advancements in technology. Inspiring of nature has flooded on to our transportation, industries and in many day-to-day activities. Every machine we use and algorithms we use is bridged with the principles, behaviors, functions and structure in nature. Yet, there are many more functions in nature we are still unable to bridge with. Nature -inspired algorithms is one of the most active research areas in today’s world. It is an attempt to bridge more natural scenarios in the form of evolutionary algorithms, in order to solve complicated problems that cannot be solved using conventional algorithms. As a result, nature-inspired algorithms have developed solutions for a number of complex real-world problems. Among these algorithms, there are some algorithms like swarm optimization, cuckoo search and firefly algorithms which are more popular because of their efficiency in solving problems[1].

## Traveling Salesman Problem

The Traveling Salesman Problem (TSP) which was mathematically formulated in the 1800s by the mathematicians William Rowan Hamilton and Thomas Kirkman[2]. Since then it became one of the most popular NP-hard problems[3] and is commonly used as a benchmark for optimization algorithms. In this problem, there is a set of cities and a salesman has to visit each city and return back to the city that he started. The salesman’s path is known as a Hamiltonian circuit in graph theory where the vertices of the graph represents the cities and every edge represents the path between the two cities connected. Each edge has a non-negative cost and the sum of the whole path is the cost of that tour (i.e. objective function). Minimizing the total distance of the path i.e. finding the optimum path is the challenge of this problem. Even though there are a lot of conventional algorithms that can be used to solve this problem, all those fail as the number of cities increases. The reason behind this is that, with the increase of the number of cities, the time complexity increases exponentially, making it unsolvable using conventional optimization algorithms with the resources available up to date.

The most common approaches to solve this, is by using approximation algorithms and heuristic algorithms such as Nearest neighbour algorithm[4], Interpolation algorithm[5], Probabilistic algorithm, Clark & Wright algorithm, Christofides algorithm, Double spanning tree algorithm and various Hybrid algorithms. Furthermore, TSP has been solved using multi-agent systems and meta-heuristic algorithms which are said to be of a higher level than the heuristic algorithms[1]. Dozens of research have been carried out in order to solve this problem using various Nature-Inspired Meta-heuristic Algorithms including algorithms like Genetic algorithms [6], Ant Colony Optimization Algorithms [7], Firefly algorithm [8] etc.

## Problem in brief

Most of the NP-hard problems have been successfully solved using meta-heuristic algorithms. However, these algorithms have algorithm-specific parameters where the performance of the algorithm is highly dependent on the values of these parameters. Identifying the optimal value for these variables is normally, done using trial and error method, where the values are statistically compared against the performance. It would certainly be better if we could just input the problem and straight away receive the solution. So, in order to achieve this, we should figure out a way to automate the process of optimizing the parameters.

Parameter tuning can be described as a process of optimizing the optimization algorithm. The problem is that, when applying another algorithm (say B) to optimize one algorithm (say A), we will need a third algorithm (say C) to optimize the parameters of the second algorithm (B), not to forget that we are not sure whether C is well-tuned. When this continuously repeats, it creates and endlessly recurring process. This provides us with the idea that there should be an alternate process for this parameter tuning.

## Aim of the research

This thesis proposes a self-tuning firefly algorithm to solve the Travelling Salesman Problem.

## Objectives of the research

As an approach to design and develop a self-tuning firefly algorithm to solve the Travelling Salesman Problem, the following objectives have been formulated.

1. To study the characteristics of Nature-Inspired Algorithms
2. To study the nature of firefly algorithm
3. To learn the behavior of the algorithm-specific parameters
4. To implement firefly algorithm to solve the TSP.
5. Adopt the self-tuning framework to the implemented firefly algorithm.
6. Discuss and compare the performance of the self-tuning firefly algorithm

## Hypothesis

The hypothesis employed in the thesis can be state as the Firefly Algorithm can be self-tuned in order to solve the Travelling Salesman Problem.

## Overview of the thesis

In this chapter, Nature-Inspired Algorithms and Travelling Salesman Problem were briefed. Then, the problem was presented followed by the aim, objectives and the hypothesis of the research.

The second chapter is the literature survey on nature-inspired algorithms, their applications and drawbacks.

## Summary

This chapter describes the background related to the problem i.e. nature-inspired algorithms and the Travelling Salesman Problem. Then the problem addressed in this thesis was stated, followed by the aim, objectives and hypothesis of the research. Finally, the overview of this thesis was presented.

# State of the art

## Introduction

Most of the real-world optimization problems have to deal with NP-hard problems and are, therefore, very challenging to be solved. Optimization tools must be used in order to solve such problems, but there is no guarantee that optimality can be achieved [9]. Because of the inherent solution mechanism of the conventional or classical optimization algorithms, these impose limitations on solving mathematical programming. Furthermore, conventional algorithms are not at all efficient in solving NP-hard problems [10]. But, no matter how complex it becomes, nature finds a way to maintain its equilibrium regardless of the number of external factors disturbing it. So researchers started to explore the nature to gain inspirations, to come up with means to solve these problems. In the current literature, there are dozens of nature-inspired algorithms which are designed with the expectation of overcoming the flaws of the conventional algorithms.

Among those, Genetic Algorithms[11], Artificial Bee Colony[12], Ant Colony Optimization[13], Particle Swarm Optimization[14], Harmony Search[15], Firefly Algorithm[16], Cuckoo Search[17], Bat-Inspired Algorithm[18] etc. have gained popularity over the others [1]. Scientists have always tried to find a universally better algorithm but, the No Free Lunch theorems [19] showed that the average performance of an algorithm cannot be stated because if one algorithm performs better than another for a set of optimization functions, the second algorithm will outperform the first in another set of problems. This showed that the algorithms must be problem-specific [19]. So now, researchers are on exploration to find the best algorithm for a specific set of problems instead of looking for one in general.

The Firefly Algorithm which was originally designed for solving continuous optimization problems [20] [16] is now used by researchers to solve many types of problems including Travelling Salesman Problem (TSP) [21] [22], Scheduling [8] and Clustering [23]. It was developed by Xin-She Yang in 2008 [24], with the inspiration of rhythmic flashes of fireflies where the pattern of flashes is often unique for particular species of fireflies.

Since the original Firefly algorithm has been used to solve continuous optimization problems, it has been adjusted in a way that other classes of problems can be solved. Those variants can be categorized as modifications, where the original firefly algorithm is developed as binary, elitist or based on chaos etc. and hybridizations, where the firefly algorithm is combined with other algorithms or techniques such as genetic algorithm, ant colony, neural network etc. [25]. However, all these algorithms have a unique drawback where the parameters should be tuned in order to optimize the algorithm to obtain better results. So far, those are basically tuned using trial and error method. Researchers are now interested in auto-tuning these parameters. Recently, Xin-She Yang proposed a framework for self-tuning optimization algorithm which is designed to tune parameters in optimization algorithms [26].

## Firefly Algorithm

Firefly algorithm is a nature-inspired metaheuristic algorithm which was developed by Xin-She Yang, originally to solve continuous domain problems. The inspiration from nature that this algorithm is based on is the behavior of fireflies[16]. Fireflies, also called as lighting bugs are well-known for their use of bioluminescence (flashing light). These insects use this light emitted by their extraordinary photogenic organs situated very close to the body surface behind of translucent cuticle to lure mates or prey. This algorithm is built on this flashing or signal mechanism of fireflies. Fireflies use different flashing sequences to send different messages to other fireflies around in its area. Yang formulated this algorithm based on the following assumptions.

1. All fireflies are unisex therefore, one firefly will be attracted to any other firefly regardless of their sex.
2. Attractiveness is directly proportional to their brightness, hence for any two glowing fireflies, the less bright one has to move towards the brighter one. The attractiveness and the brightness, both decrease as the distance between the fireflies increase. If there is no brighter firefly than a particular firefly, it will move randomly.
3. The brightness of a firefly is affected or determined by the landscape of the objective function. For a maximization problem, the brightness can be directly proportional to the value of the objective function. Other forms of brightness can be defined in a similar way to the fitness function in genetic algorithms.

Based on these three assumptions, the basic steps of the firefly algorithm (FA) can be summarized as the pseudo code shown in Figure 1.

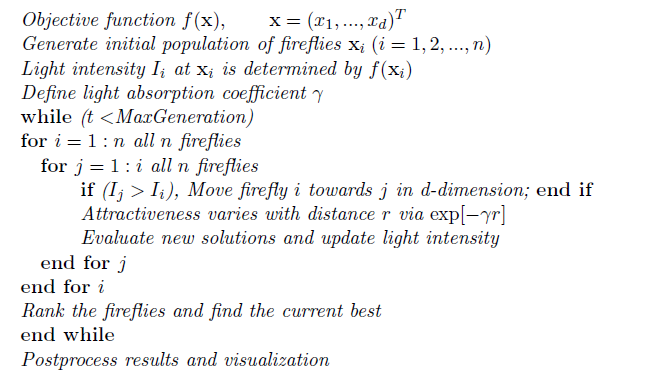


Figure 1: Pseudo code of the original firefly algorithm

Discretization of FA

As mentioned in the previous section, the FA is a population algorithm proposed in 2008 by Yang. The basic FA is based on the flashing behavior of fireflies, and its first version was proposed for solving continuous optimization problems. Since its first implementation, the FA has been applied in a broad range of areas. Some of these areas are the multi-modal optimization, continuous optimization, in which some further works have been published apart from to the original one, combinatorial optimization, and multi-objective optimization. Although the first version FA was intended for continuous problems, it has been remodeled many times in the literature with the aim of addressing discrete optimization problems. In [23], for instance, we can find a discrete FA changed to solve the class of discrete problems named Quadratic Assignment Problem. Another discrete FA was formed by Sayady et al. in 2010 to solve, minimizing the makespan for the permutation flow shop scheduling problem which is named as a NP-Hard problem. Furthermore, another discrete FA is formulated by Marichelvam et al. in 2014 for the multi-objective flexible job shop scheduling problem. On the other hand, a new evolutionary discrete FA applied to the symmetric TSP is presented in [24].

Representation of a Firefly

A solution representation for the TSP is a permutation representation as illustrated by Figure 2. Here, a firefly represents one solution. It is just like a chromosome that represents an individual in genetic algorithm. In this representation, an element of array represents a city (node) and the index represents the order of a tour. Although the original FA is to cater continuous problems, TSP is a discrete problem. Therefore, representation of a firefly should be changed accordingly. A solution of the TSP is expressed as a permutation representation as illustrated in Figure 1. Here, a firefly represents one solution. It is just like a chromosome that represents an individual in genetic algorithm. In this representation, a cell of the array represents a city (node) and the index represents the order of the tour.

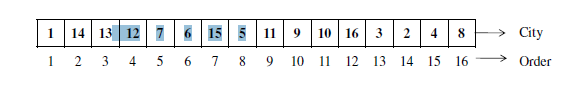


Figure 2 The permutation representation of a solution

Movement of a Firefly

The movement of a firefly *i* attracted to another brighter (more attractive) firefly *j* is determined by

*xi* = *random*(2, r*ij*),

where is the distance between firefly *i* and *j*. The length of movement of a firefly will be randomly selected from 2 to *rij*. When a firefly moves, existing solutions in the firefly is changed. Since the representation of firefly is a permutation representation, then we use Inversion Mutation to represent the movement. With inversion mutation, the path that has been formed can be maintained so the good path formed previously is not damaged.

## Summary

In this chapter, the

# Methodology

## Introduction

There are various types of nature-inspired algorithms which have different kinds of input parameters, evolutionary mechanisms and are applied in various areas [27]. All these algorithms have algorithm-specific parameters and tuning those parameters is the key in finding a good solution for a problem. In the original FA there are two parameters namely, light absorption coefficient (gamma) and randomization parameter (alpha). These parameters must be tuned in order to gain the optimum solution for a problem. Tuning these parameters are generally done using trial and error method while the results are statistically compared against the parameter values. For this, Xin-She Yang proposed a self-tuning framework. Like the original FA, this too was proposed to solve continuous domain problems.

This research proposes a self-tuning FA to solve a problem in the discrete domain. The TSP is used as a sample problem to present the new algorithm.

## Approach

As previously mentioned, a discrete FA is used to solve the TSP. The discretized FA uses one parameter i.e. the light absorption coefficient (gamma). In order to tune this parameter, the ability to solve multi-objective problems is used with the help of the self-tuning framework introduced by Yang to solve continuous domain problems. In this research, along with the discrete FA is used to solve the TSP, the self-tuning framework is adopted to optimize both a discrete problem (TSP) and a continuous problem (tuning of parameters) at the same time.

Input

To solve the TSP is a set of cities along with their coordinates must be given to the algorithm by the user as a tsp file. The user can also input the population size as well as the number of iterations the algorithm will follow.

Process

The system will read the tsp file and generate the distance matrix between the cities. Then, the initial population of fireflies will be generated. These fireflies will carry a random permutation of the cities along with a randomly generated gamma value. In each iteration, a new set of fireflies moving towards the best firefly will be generated. But if a firefly cannot find a better firefly in its vicinity, it will move randomly. Then all the fireflies are sorted according to the objective function. In the case of TSP, the objective function is the reciprocal of the total distance of the route carried by the firefly. After sorting the fireflies, the best firefly (best local solution) is compared with the global best solution and replace the global best solution with the local best solution if a better solution is found. A new set of fireflies are also chosen from the sorted list of fireflies.

Output

At the end of the last iteration in the form of an array of cities, the best route found (Figure 3), the total distance of the route (Figure 4), the difference between the optimum solution(Figure 4) and the best gamma value (self-tuned parameter) (Figure 4), is presented. The final route is shown in a graph where the vertices represent the cities and the path formed by the edges represent the optimum route found. A sample graph is shown in Figure 5.

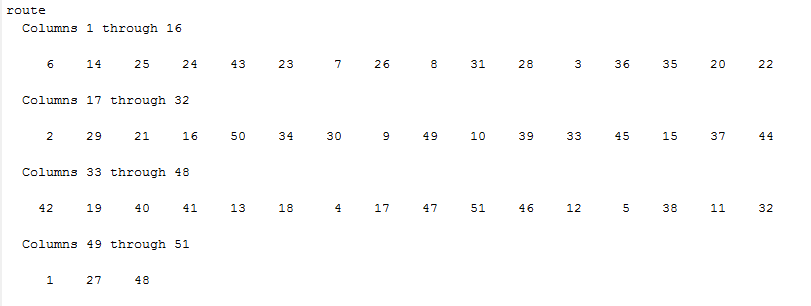
****

Figure 3 Final route in the form of an array

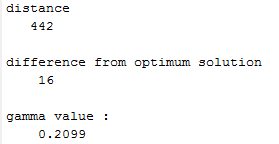


Figure 4 Final Results

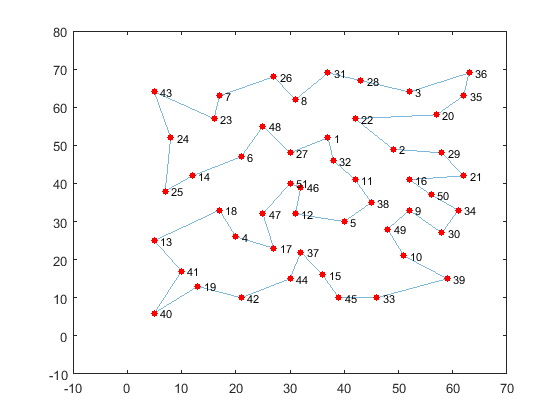


Figure 5 Optimum route

# Evaluation

## Introduction

This chapter will help us to identify how the system will fulfil the requirement of this project. Here we will discuss the overall evaluation process of the Self-Tuning Firefly Algorithm to solve the Travelling Salesman Problem system. This also tries to express the kind of evaluation process that is going to be carried out in order to assess the success of the Self-Tuning Firefly Algorithm (Attached in the Appendix ).

# References

[1] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*, 2nd ed. Luniver Press, 2010.

[2] N. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory, 1736-1936*. New York, NY, USA: Clarendon Press, 1986.

[3] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2011.

[4] A. A. Khan, “A COMPARITIVE STUDY OF NEAREST NEIGHBOUR ALGORITHM AND GENETIC ALGORITHM IN SOLVING TRAVELLING SALESMAN PROBLEM,” 2016.

[5] Yuan-bin Mo, “The Advantage of Intelligent Algorithms for TSP,” in *Traveling Salesman Problem, Theory and Applications, Prof. Donald Davendra (Ed.)*, vol. 1, InTech, 2010, pp. 26–40.

[6] Z. H. Ahmed, “Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator,” *Int. J. Biom. Bioinforma. IJBB*, vol. 3, no. 6, p. 96, 2010.

[7] M. Dorigo and L. M. Gambardella, “Ant colony system: a cooperative learning approach to the traveling salesman problem,” *Evol. Comput. IEEE Trans. On*, vol. 1, no. 1, pp. 53–66, 1997.

[8] M. K. Marichelvam, T. Prabaharan, and X. S. Yang, “A Discrete Firefly Algorithm for the Multi-Objective Hybrid Flowshop Scheduling Problems,” *IEEE Trans. Evol. Comput.*, vol. 18, no. 2, pp. 301–305, Apr. 2014.

[9] S. Bhattacharyya, *Handbook of Research on Swarm Intelligence in Engineering*. IGI Global, 2015.

[10] A. Baykasoglu, L. Ozbakir, and P. Tapkan, “Artificial bee colony algorithm and its application to generalized assignment problem,” *Swarm Intell. Focus Ant Part. Swarm Optim.*, pp. 113–144, 2007.

[11] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[12] D. Karaboga, “An idea based on honey bee swarm for numerical optimization,” Oct. 2005.

[13] M. Dorigo and C. Blum, “Ant colony optimization theory: A survey,” *Theor. Comput. Sci.*, vol. 344, no. 2–3, pp. 243–278, Nov. 2005.

[14] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. New York: Oxford University Press, 1999.

[15] X.-S. Yang, “Harmony search as a metaheuristic algorithm,” in *Music-inspired harmony search algorithm*, Springer, 2009, pp. 1–14.

[16] X.-S. Yang, “Multiobjective Firefly Algorithm for Continuous Optimization,” *Eng. Comput.*, vol. 29, no. 2, pp. 175–184, 2013.

[17] X.-S. Yang and S. Deb, “Cuckoo search via Lévy flights,” in *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, 2009, pp. 210–214.

[18] X.-S. Yang, “A new metaheuristic bat-inspired algorithm,” in *Nature inspired cooperative strategies for optimization (NICSO 2010)*, Springer, 2010, pp. 65–74.

[19] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *Evol. Comput. IEEE Trans. On*, vol. 1, no. 1, pp. 67–82, 1997.

[20] S. Lukasik and S. Żak, “Firefly algorithm for continuous constrained optimization tasks,” in *Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems*, Springer, 2009, pp. 97–106.

[21] G. Jati and Suyanto, “Evolutionary Discrete Firefly Algorithm for Travelling Salesman Problem,” in *Adaptive and Intelligent Systems*, vol. 6943, A. Bouchachia, Ed. Springer Berlin Heidelberg, 2011, pp. 393–403.

[22] László Kota and Károly Jármai, “DISCRETIZATION OF THE FIREFLY ALGORITHM FOR THE TRAVELLING SALESMAN PROBLEM,” 2008.

[23] J. Senthilnath, S. N. Omkar, and V. Mani, “Clustering using firefly algorithm: Performance study,” *Swarm Evol. Comput.*, vol. 1, no. 3, pp. 164–171, 2011.

[24] X.-S. Yang, *Nature-inspired metaheuristic algorithms*. Frome: Luniver Press, 2008.

[25] Iztok Fister, Janez Bresta, Xin-She Yang, and Iztok Fister Jr., “A comprehensive review of ﬁreﬂy algorithms.pdf,” *Swarm Evol. Comput.*, vol. 13, pp. 34–46, 2013.

[26] X.-S. Yang, Suash Deb, Martin Loomes, and Mehmet Karamanoglu, “A Framework for Self-Tuning Optimization Algorithm,” *Neural Comput. Appl.*, vol. 23, no. 7\*8, pp. 2051–2057, 2013.

[27] P. Agarwal and S. Mehta, “Nature-Inspired Algorithms: State-of-Art, Problems and Prospects,” *Nature*, vol. 100, no. 14, 2014.

# Appendix A Self-Tuning Algorithm

function fa\_tsp()

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*inputs\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

clearvars -global;

global nFF;

global N;

global best;

global alpha;

global delta;

global gamma;

nFF = 50; %number of fireflies

movements = 30; %number of times a firefly moves

%gamma = 10; %light absorption coeffient

alpha = 0;

delta = 0;

iterations = 300; %number of times the FFs will evolve

file = 'eil51.tsp'; %file name

minDist = 426;

%\*\*\*\*\*\*\*\*\*\* Read tsp file \*\*\*\*\*\*\*\*\*\*\*\*\*\*

fileID = fopen(file);

%read file and store data in cell format

datacell = textscan(fileID, ' %f %f %f','CollectOutput',1);

fclose(fileID);

%convert data format from cell to matrix

datamat = cell2mat(datacell(1));

N = length(datamat); %No. of cities = No. of rows

cities = datamat(:,1); %city numbers

xValues = datamat(:,2); %x coordinates

yValues = datamat(:,3); %y coordinates

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Start \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

distMat = zeros(N,N);

solutions = zeros(iterations, 1);

distMat = disMat(distMat, xValues, yValues);

initFF = init(nFF, N);

newFF = calcObjFunc(initFF, N, distMat);

newFF = sort(newFF);

%\*\*\*\*\*\*\*\*\*\* evolve \*\*\*\*\*\*\*\*\*\*\*\*\*\*

for iteration=1:iterations

newPop = newSols(newFF, movements, best);

newPop = calcObjFunc(newPop, N, distMat);

newFF = selectFFs(newPop, nFF);

disp(best.')

solutions(iteration) = best(1,N+1);

gammaSol(iteration) = best(1,N+2);

end

[nodes1, nodes2] = getGraphNodes(best);

disp('route');

disp(best(1:N));

disp('distance');

disp(best(1,N+1));

disp('difference from optimum solution');

disp(best(1,N+1) - minDist)

disp('gamma value :');

disp(gamma);

figure

plot (solutions);

figure

plot (gammaSol);

figure

G = graph(nodes1,nodes2);

p = plot(G);

p.NodeColor = 'r';

p.XData = xValues;

p.YData = yValues;

plot(G,'XData',xValues,'YData',xValues,'EdgeLabel');

function setGlobalBest(val)

global best;

best = val;

function setGlobalGamma(val)

global gamma;

gamma = val;

% N

% xValues

% yValues

% distMat

% initFF

%opt = [1 22 8 26 31 28 3 36 35 20 2 29 21 16 50 34 30 9 49 10 39 33 45 15 44 42 40 19 41 13 25 14 24 43 7 23 48 6 27 51 46 12 47 18 4 17 37 5 38 11 32];

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% FUNCTIONS

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%\*\*\*\*\*\*\*\*\*\* Create Distance Matrix \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function distMat = disMat(distMat, xValues, yValues)

global N;

for i = 1:N

for j = i:N

%enter distance between i and j into distMat

x1 = xValues(i); %x coodinate of c1

y1 = yValues(i); %y coodinate of c1

x2 = xValues(j); %x coodinate of c2

y2 = yValues(j); %y coodinate of c2

%calculate the distance between c1 and c2

%dist = sqrt(((x1-x2)^2) + ((y1-y2)^2));

X = [x1,y1;x2,y2];

dist = pdist(X,'euclidean');

distMat(i,j)= round(dist); %round off distance

distMat(j,i) = distMat(i,j);

end

end

%\*\*\*\*\*\*\*\*\*\* Create Initial Population \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function FFs = init(FF, nCity) %nFF = No. of FF, N = No. of cities

FFs = zeros(FF,(nCity+2));

for i= 1:FF

FFs(i,1:nCity) = (randperm(nCity));

FFs(i,(nCity+2)) = (rand());

end

%\*\*\*\*\*\*\*\*\*\* Calculate Distance \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function FFs = calcObjFunc(FFs, N, distMat)

num = size(FFs, 1);

for i = 1:num

sum = 0;

for j = 1:(N-1)

x = (FFs(i,j));

y = (FFs(i,(j+1)));

d = distMat(x,y);

sum = sum + d;

end

sum = sum + distMat(FFs(i, N),FFs(i, 1));

FFs(i,(N+1)) = sum;

end

% disp(sum);

%\*\*\*\*\*\*\*\*\*\* Sort \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function SortedFF = sort(FFs)

global N;

global best;

SortedFF = sortrows(FFs,(N+1));

localBest = SortedFF(1,:);

gamma = SortedFF(1,N+2);

if isempty(best) || (localBest(1,N+1) < best(1,N+1))

setGlobalBest(localBest);

setGlobalGamma(gamma);

end;

%\*\*\*\*\*\*\*\*\*\* Calculate Distance between 2 Fireflies (solutions) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function arcs = calDistSol(FF1, FF2)

global N;

arcs = 0;

for i = 1:(N-1)

temp1 = FF1(i);

temp2 = FF1(i+1);

j = 1;

while FF2(j) ~= temp1

j = j+1;

end

if j<N

if FF2(j+1) ~= temp2

arcs = arcs + 1;

end

end

end

%arcs

%\*\*\*\*\*\*\*\*\*\* Calculate Brightness \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function bri = brightness(FF)

global N;

d = FF(1,N+1);

bri = 1/d;

%\*\*\*\*\*\*\*\*\*\* Calculate Objective function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function ob = obj(FF)

global N;

d = FF(1,N+1);

ob = 1/d;

%\*\*\*\*\*\*\*\*\*\* Calculate Attractiveness \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function attr = calAttr(FF1, FF2)

global N;

attr0 = brightness(FF2);

r = calDistSol(FF1, FF2);

gamma = FF1(1, N+2);

pow = -(gamma \* r \*r);

attr = attr0 \* exp(pow);

%\*\*\*\*\*\*\*\*\*\* Find More Attractive Firefly \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function attrFF = getAttrFF (FF, FFset)

% ob = brightness(FF2);

nFF = size(FFset, 1);

attrFF = -1;

FFs = (randperm(nFF));

for i= 1:nFF

temp = FFset(FFs(i),:);

% disp(brightness(FF));

% disp(calAttr(FF, temp));

if brightness(FF) < calAttr(FF, temp)

attrFF = temp;

break;

end

end

%\*\*\*\*\*\*\*\*\*\* Inverse Mutation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function FF = invMutation(FF, length)

global N;

startAt = randi(N, 1);

temp = zeros(1,length);

for i = 1:length

if mod((startAt + (i-1)), N) ~= 0

to = mod((startAt + (i-1)), N);

else

to = N;

end

temp(i) = FF(to);

end

for i = 1:length

if mod((startAt + (i-1)), N) ~= 0

to = mod((startAt + (i-1)), N);

else

to = N;

end

FF(to) = temp(length -(i-1));

end

%\*\*\*\*\*\*\*\*\*\* Select new Solutions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function selectedFFs = selectFFs(FFs, nFF)

FFs = sort(FFs);

selectedFFs = FFs(1:nFF,:);

%\*\*\*\*\*\*\*\*\*\* New Solutions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function newFFs = newSols(FFset, movements, best) %TODO

global N;

nFF = size(FFset, 1);

newFFs = zeros((nFF \* movements)+1, N+2);

for i = 1:nFF

FF = FFset(i, :);

attrFF = getAttrFF (FF, FFset);

if attrFF == -1

for j = 1:movements

move = randi([2,N],1);

newFFs(movements\*(i-1) + j,:) = invMutation(FF, move);

newFFs(movements\*(i-1) + j, N+2) = newGamma(FF, attrFF);

end

else

dist = calDistSol(FF, attrFF);

if dist <2

dist = 2;

end

for j = 1:movements

move = randi([2,dist], 1);

newFFs(movements\*(i-1) + j,:) = invMutation(FF, move);

newFFs(movements\*(i-1) + j, N+2) = newGamma(FF, attrFF);

end

end

end

newFFs((nFF \* movements)+1, :) = best;

function [nodes1, nodes2] = getGraphNodes(best)

global N;

nodes2 = zeros(N,1);

for i = 1:(N-1)

nodes2(i) = best(i+1);

end

nodes2(N) = best(1);

nodes1 = best(1:N);

function g = newGamma(FF1, FF2)

global gamma

global N;

attr0 = 1;

if FF2 == -1

g = FF1(1,N+2) + (rand()\*0.1 - 0.05);

else

gamma1 = FF1(1,N+2);

gamma2 = FF2(1, N+2);

r = gamma1 - gamma2;

pow = -(gamma \* r \*r);

if gamma1 < gamma2

g = abs(gamma1 + attr0 \* exp(pow)\*0.1);

else

g = abs(gamma1 - attr0 \* exp(pow)\*0.1);

end;

end;