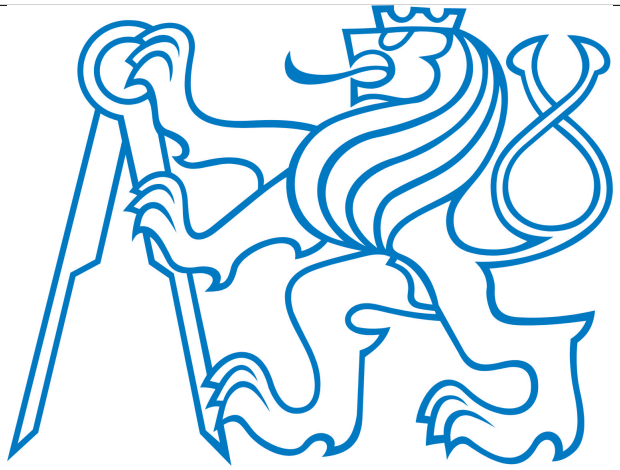
CZECH TECHNICAL UNIVERSITY
IN PRAGUE

FACULTY OF ELECTRICAL ENGINEERING



BACHELOR THESIS

# IMPLEMENTATION OF A DISCRETE FIREFLY ALGORITHM FOR THE QAP PROBLEM WITHIN THE SEAGE FRAMEWORK

Author: Karel Durkota

Advisor: Ing. Richard Málek

May 2011

**Czech Technical University in Prague**
**Faculty of Electrical Engineering**

**Department of Cybernetics**

# BACHELOR PROJECT ASSIGNMENT

**Student:** Karel Durkota

**Study programme:** Software Engineering and Management

**Specialisation:** Intelligent Systems

**Title of Bachelor Project:** Implementation of a Discrete Firefly Algorithm Within the SEAGE Framework

**Guidelines:**

1. Study basic principles of the Firefly algorithm.
2. Design and implement the algorithm within the SEAGE framework.
3. Test the algorithm and implement a solution of the QAP problem using the Firefly algorithm. Validate the algorithm experimentally and evaluate the results for some problem instances from the QAPLIB library.

**Bibliography/Sources:**

Xin-She Yang, Nature-Inspired Metaheuristic Algorithms, Published in 2008 by Luniver Press.

**Bachelor Project Supervisor:** Ing. Richard Málek

**Valid until:** the end of the winter semester of academic year 2011/2012

prof. Ing. Vladimír Mařík, DrSc.
**Head of Department**

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická L.S.

prof. Ing. Boris Šimák, CSc.
**Dean**

Prague, February 3, 2011

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 13. 5. 2011

podpis

ABSTRACT

*Firefly Algorithm* (FA), introduced in 2008 by X. S. Yang, belongs with the nature-inspired algorithms. It is a meta-heuristic, based on the firefly bugs' behaviour, including the light emission, light absorption and the mutual attraction, which was developed to solve the continuous optimization problems. This thesis describes Firefly Algorithm and proposes a possible way it can be adjusted to solve the class of discrete problems Quadratic Assignment Problem (QAP), where the solutions consist of the permutation of the integers. Explained is the Firefly Algorithm's discretization, which consists of constructing a suitable conversion of the continuous functions as are attractiveness, distance and movement, into new discrete functions. New Discrete Firefly Algorithm (DFA) is implemented into the framework SEAGE, where the DFA is examined and experimentally tested on 11 different QAP problems chosen from the public QAPLIB Library. Results of these experiments are analysed and described in this work.

Keywords: Firefly Algorithm, Meta-heuristics, Optimization

---

ABSTRAKT

*Firefly Algoritmus* (FA), který v roce 2008 zveřejnil jeho autor X. S. Yang, patří mezi algoritmy inspirované přírodou. Jedná se o meta-heuristiku založenou na chování svetlušek (vzájemná atrakce za pomoci světelného záření) vyvinutou pro spojité optimalizační problémy. Tato bakalářská práce popisuje Firefly Algoritmus a možný způsob návrhu, jak jej lze přizpůsobit problémům třídy Quadratic Assignment Problem (QAP), kde řešením je permutace celočíselných hodnot. Popisuje diskretizaci Firefly Algoritmu, jenž spočívá v konstrukci vhodného převodu spojitých funkcí jako jsou atraktivita, distance a pohyb na nové diskrétní funkce. Takto nově vzniklý Diskrétní Firefly Algoritmus (DFA) je implementován do projektu SEAGE, kde je jeho funkcionalita experimentálně ověřena na 11 různých QAP problémech z veřejné QAPLIB knihovny. Výsledky jsou analyzované a rovněž uvedené v této práci.

Klíčová Slova: Firefly Algorithm, Metaheuristika, Optimalizace

# ACKNOWLEDGMENTS

I would like to acknowledge Ing. Richard Málek for much advice and consults about this thesis. My big gratitude belongs to Ing. Michaela Urbancová for the long discussions and great ideas of improvements of the algorithm, Eduard Durkota, who offered technical equipment for simulations of the experiments, and Catherine Elliott for reviewing and correcting the grammar of this thesis.

# Contents

# Chapter 1

# Introduction

Over the last 20 years new meta-heuristic algorithm has been introduced almost every year [17]. The nature-inspired ones have become very interesting and distinguished. It is reasonable that people began to explore how nature solves the problems, since nature has evolved for billions of years, and has found almost perfect solution to any problem it has encountered. Erroneous or lesser alternations have already become extinct or were not favoured in a natural selection. So why not be inspired by nature [15]? Such commonly known algorithms are e.g., Evolutionary Algorithms, which are inspired by biological evolution itself and use crossover and mutation for improving solutions or Neural Network [1], based on the biological neural network system in animals' brains. These algorithms are usually proposed for the continuous optimization problems (see section 1.1) that can be solved very effectively, as our world is continuous. In case we want to apply an algorithm to discrete problems, like the Travelling Salesman Problem (TSP) or Quadratic Assignment Problem (QAP), we need to adjust an algorithm for that purpose, which is called *discretization*. For some discretizations it would suffice to round real numbers up to a certain decimal fraction or replace them by the integers, but sometimes solution can be represented as permutation of the integers, as it is in TSP and QAP problems, or even in a more complex way. In these classes of problems, we need to adapt our continuous meta-heuristics to be more sophisticated.

[1] Neural Networks were inspired by nature, but is not a meta-heuristic

## 1.1 Optimization

There are three basic elements of the optimization problems:

- an *objective function* which we want to *minimize* or *maximize*
- a set of *unknowns* or *variable* which affect the value of the objective function
- a set of *constraints* that allow the unknown to take on certain value but exclude others.

The optimization problem is described informally as: Find values of the *variables* that *minimize* or *maximize* the *objective function* while satisfying the *constraints*.

Formal definition is:

$$\begin{array}{llll} \text{given a function} & f : \mathbf{A} \to \mathbb{R} & & \mathbf{A} \subseteq \mathbb{R}^n \\ \text{minimize} & f(\mathbf{x}), & \mathbf{x} = (x_1, x_2, \ldots, x_n)^T & x \in \mathbf{A} \\ \text{subject to} & c_i(\mathbf{x}) = 0, & i = 1, 2, \ldots, m' & i, m' \in \mathbb{N} \\ & c_i(\mathbf{x}) \geq 0, & i = m'+1, \ldots, m & i, m \in \mathbb{N} \end{array}$$

where:

- $f(\mathbf{x})$ is the objective function,
- $\mathbf{x}$ is the column vector of the $n$ independent variables,
- $c_i(\mathbf{x})$ is the set of constraint functions.

Constraint equations of the form $c_i(\mathbf{x}) = 0$ are termned *equality* constraints, and those of the form $c_i(\mathbf{x}) \geq 0$ are *inequality* constraints. Taken together, $f(\mathbf{x})$ and $c_i(\mathbf{x})$ are known as the *problem functions* [8].

## 1.2 Seage

Project SEAGE[2] [9] is a dissertation work created by Ing. Richard Málek at Czech Technical University. This project is an optimization framework written in the Java programming language,

[2] SEAGE is an acronym of SEarch AGEnts

that uses hyper-heuristic approach for solving the optimization problems. Hyper-heuristic is a technique, under which heuristics compete against each other in order to be selected. In other words, hyper-heuristic is “heuristic, which chooses heuristics". Hyper-heuristic does not try to find the best solution for the problem, as it endeavours meta-heuristic, yet it gives us as a solution some heuristic, or combination of several heuristics, that could compute the best solution. Hyper-heuristic's other motivation is robustness, rather than finding an optimal solution [4].

SEAGE contains implementations of some nature-inspired meta-heuristic algorithms and common *NP* complex discrete computational problems. It enables to solve the problem instances by any of the implemented meta-heuristic algorithms, as well as their combinations running simultaneously, which may lead to even better results.

## 1.3 Goals of This Work

In 2008, Xin-She Yang has introduced a new meta-heuristics algorithm *Firefly Algorithm*, that is inspired by firefly's social behaviour. In the basic form, this algorithm is designed to solve primarily continuous problems. Goal of this work is to implement a *discretization* of this algorithm, or in other words, to adjust this algorithm to solve discrete problems, particularly the Quadratic Assignment Problems (QAP). Discretization has to be done in such a way, that the new Discrete Firefly Algorithm (DFA) would solve the QAP effectively. Both, the QAP and the DFA will be implemented into the SEAGE framework, and afterwards, DFA will be experimentally tested on 11 QAP instances in the SEAGE's EXPERIMENTATOR environment. Experiments will be set to observe algorithms strengths and weaknesses, as well as to find algorithm's different behaviours dependent on parameters' settings.

# Chapter 2

# Firefly Algorithm

## 2.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO), introduced by Kennedy and Eberhart (1995) [6], is an optimization method based on social behaviours. This method does not calculate a gradient of an *objective function* [1] $f()$, hence the objective function is not required to be differentiable. PSO methods consist of a collection (called a *swarm*) of individual entities (known as *particles*). Each particle represents a candidate solution and abides simple mathematical formulae and rules, observed in behaviour of the bird flock [6]. Every $i^{th}$ particle knows: *a*) its own *position* $\mathbf{x}^i = (x_1^i, \ldots, x_n^i)$, *b*) its own *direction and velocity* [2], usually represented as a vector $\mathbf{v}^i = (v_1^i, \ldots, v_n^i)$, *c*) the *position* of its own best currently reached solution $\mathbf{p}^i = (p_1^i, \ldots, p_n^i)$ and *d*) the *position* of the best currently known solution of the whole swarm $\mathbf{g} = (g_1, \ldots, g_n)$ [3], where $n$ is the space dimension .

At the initialization step, all the particles $\mathbf{p}^1, \ldots, \mathbf{p}^m$, where $m \in \mathbb{N}$ is the number of the particles,

[1] *objective function*, usually $f : \mathbb{R}^n \to \mathbb{R}$ — also called cost function — is a function we want to minimize (or maximize, if desired). Such minimum or maximum is then called an *optimum*

[2] *direction and velocity* can be represented in one vector, where its direction is the vector's direction, and velocity is the vector's length

[3] this is the only shared information among all the particles

are uniformly scattered over the search space and their velocities are set either to zero or to a small random value. Each particle's best current solution $\mathbf{p}^i$ is set to their current position $\mathbf{x}^i$, and $\mathbf{g}$ is set to the position of the best particle. Best particle is such particle $\mathbf{p}^j$, that for all $i \in \{1,\ldots,m\}$ : $f(\mathbf{p}^j) \leq f(\mathbf{p}^i)$ in case of the minimalization [6]. Next in a loop, there is calculated new velocity vector $\mathbf{v}^i$ of each particle in the swarm using the formula [10]:

$$\mathbf{v}^i \leftarrow \omega \mathbf{v}^i + \phi_p r_p(\mathbf{p}^i - \mathbf{x}^i) + \phi_g r_g(\mathbf{g} - \mathbf{x}^i), \tag{2.1}$$

where $r_p$, $r_g \in U[0,1)$ and $\omega \in \mathbb{R}$ are user-defined constants, called *inertia weights* and $\phi_p$,$\phi_g$ are user-defined behavioural constants called *acceleration coefficients*. All the other variables were defined earlier. After calculating every particle's new velocity vector $\mathbf{v}^i$, we need to compute their new positions $\mathbf{x}^i$ — to make their step. New position for each particle is calculated simply by the formula [10]:

$$\mathbf{x}^i \leftarrow \mathbf{x}^i + \mathbf{v}^i. \tag{2.2}$$

Particles' best known positions $\mathbf{p}^i$ and the global best known position ($\mathbf{g}$) are updated in the last step of this loop. If the particle's newly gained solution is better than its current best solution or $f(\mathbf{x}^i) < f(\mathbf{p}^i)$ in a case of minimalization, then $\mathbf{p}^i \leftarrow \mathbf{x}^i$. Find a current best particle (with the maximum of $f(\mathbf{x})$), let say it is $\mathbf{b}$ and then using rule: if $f(\mathbf{g}) < f(\mathbf{b})$ then $\mathbf{g} \leftarrow \mathbf{b}$ we update the global solution. Every particle gets closer towards the combination of its best currently known solution, which is called *cognitive component* and is modelled by vector $\phi_p$, and best known solution, called *social component*, modelled by $\phi_g$ as it is shown in Figure 2.1.

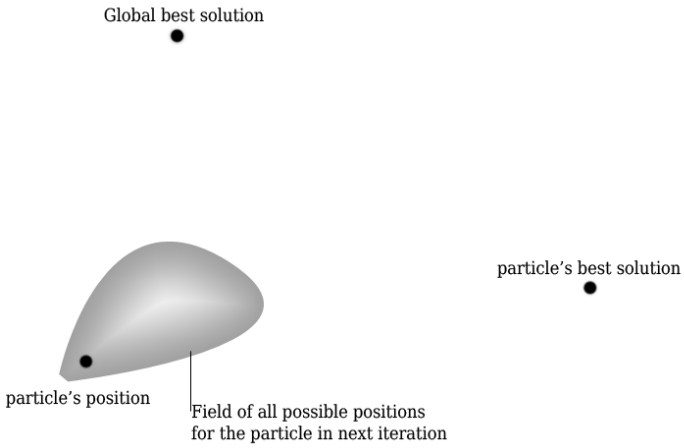



**Figure 2.1** Field of the particle's possible new positions for the next iteration.

## 2.2 Firefly Algorithm

### 2.2.1 Inspiration

Fireflies, belong with family of Lampyridae, are small winged beetles capable of producing a *cold light* [4] flashes in order to attract mates [1]. They are believed to have a capacitor-like mechanism, that slowly charges until the certain threshold is reached, at which they release the energy in the form of light, after which the cycle repeats [12].

Firefly algorithm, developed by Xin-She Yang (2008), is inspired by the light attenuation over the distance and fireflies' mutual attraction, rather than by the phenomenon of the fireflies' light flashing. Algorithm considers what each firefly observes at the point of its position, when trying to move to a greater light-source, than is his own.

[4] *cold light* is a light producing little or no heat

### 2.2.2 Algorithm

The Firefly Algorithm is one of the newest meta-heuristics. Therefore there have been written fairly few articles concerning it. This thesis will mainly draw information from [13, 15, 16]. As it is described in above listed articles, Firefly algorithm idealizes some of the characteristics of the firefly behaviour. They follow three rules: *a*) all the fireflies are unisex, *b*) each firefly is attracted only to the fireflies, that are brighter than itself; strength of the attractiveness is proportional to the firefly's brightness, which attenuates over the distance; the brightest firefly moves randomly and, *c*) brightness of every firefly determines its quality of solution; in most of the cases, it can be proportional to the *objective function*. Using these three rules, pseudo-code of the Firefly Algorithm may look as follows:

**Algorithm 1:** Basic Firefly Algorithm Pseudo-code

**input** : $f(\mathbf{x}), \mathbf{x} = (x_1, x_2, \cdots, x_n)$ ; // objective function
$m, I_0, \gamma, \alpha$ ; // user-defined constants
**output**: $\mathbf{x}^{min}$ ; // position of minimum in objective function

**for** $i \leftarrow 1$ **to** $m$ **do**
  $\mathbf{x}^i \leftarrow$ Initial_Solution();
**end**
**while** *termination requirements are not met* **do**
  $min \leftarrow \arg\min_{i \in \{1,\ldots,m\}} (f(\mathbf{x}^i))$;
  **for** $i \leftarrow 1$ **to** $m$ **do**
    **for** $j \leftarrow 1$ **to** $m$ **do**
      **if** $f(\mathbf{x}^i) < f(\mathbf{x}^j)$ **then** // move $\mathbf{x}^i$ towards $\mathbf{x}^j$
        $d_{i,j} \leftarrow$Distance($\mathbf{x}^i$,$\mathbf{x}^j$);
        $\beta \leftarrow$Attractiveness($I_0, \gamma, d_{i,j}$);
        $\mathbf{x}^i \leftarrow (1-\beta)\mathbf{x}^i + \beta\mathbf{x}^j + \alpha(\text{Random}() - \frac{1}{2})$; // movement
      **end**
    **end**
  **end**
  $\mathbf{x}^{min} \leftarrow \mathbf{x}^{min} + \alpha(\text{Random}() - \frac{1}{2})$ ; // best firefly moves randomly
**end**

In the above algorithm, $m$ is the number of the fireflies, $I_0$ is the light intensity at the source,

$\gamma$ is the absorption coefficient and $\alpha$ is the size of the random step. All these parameters will be explained further in detail.

### 2.2.3 Attractiveness, Distance and Movement

#### Attractiveness

Suppose it is a night with absolute darkness, where the only visible light is the light produced by fireflies. The light intensity of each firefly is proportional to the quality of the solution, it is currently located at. In order to improve own solution, the firefly needs to advance towards the fireflies that have brighter light emission than is his own. Each firefly observes decreased light intensity, than the one fireflies actually emit, due to the air absorption over the distance. Light intensity reduction abides the law:

$$I(I_0, d) \leftarrow \frac{I_0}{d^2}, \tag{2.3}$$

where $I_0$ is the light intensity at zero distances, and $d$ is the observer's distance from the source. Attenuation through the air absorption coefficient follows this rule:

$$A(I_0, \gamma, d) \leftarrow I_0 e^{-\gamma d}, \tag{2.4}$$

where $\gamma$ is the absorption coefficient. By combining both equations, resulted *attractiveness function* can be well approximated by the formula[5]:

$$\text{Attractiveness}(I_0, \gamma, d) \leftarrow I_0 e^{-\gamma d^2}, \tag{2.5}$$

For the faster computation, or in cases intensity function needs to be decreased in a slightly slower rate, it can be well emulated by function:

$$\text{Attractiveness}(I_0, \gamma, d) \leftarrow \frac{I_0}{1 + \gamma d^2}. \tag{2.6}$$

[5]this formula leaves us out of the concerns for $d = 0$, for which the domain of definition of the Formula 2.3 is not defined

### Distance

The distance function $Distance: \mathbb{X} \times \mathbb{X} \to \mathbb{R}$, where $\mathbb{X}$ is the set of all solutions, supplies information of the diversity between two given solutions. In case solution space $\mathbb{X} = \mathbb{R}^n$, distance function can be following:

$$d_{i,j} = \text{Distance}(\mathbf{x}^i, \mathbf{x}^j) = \sqrt{\sum_{k=1}^{n} (x_k^i - x_k^j)^2}. \tag{2.7}$$

For the strings, distance function may be represented as Hamming's distance of given strings; for the more complex structures, the distance function should provide information about the amount of different entities in the structures.

### Movement

The movement itself consists of two elements: *approaching the better local solutions* and the *random step*[6]. Formula for an attracting firefly $\mathbf{x}^i$ towards a firefly $\mathbf{x}^j$ is the following:

$$\mathbf{x}^i \leftarrow \mathbf{x}^i + \text{Attractiveness}(I_0, \gamma, \text{Distance}(\mathbf{x}^i, \mathbf{x}^j)) \cdot (\mathbf{x}^j - \mathbf{x}^i) + \alpha \cdot (Random() - \frac{1}{2}) \tag{2.8a}$$

$$\leftarrow \mathbf{x}^i + \text{Attractiveness}(I_0, \gamma, d_{i,j}) \cdot (\mathbf{x}^j - \mathbf{x}^i) + \alpha \cdot (Random() - \frac{1}{2}) \tag{2.8b}$$

$$\leftarrow \mathbf{x}^i + I_0 e^{-\gamma d_{i,j}^2} \cdot (\mathbf{x}^j - \mathbf{x}^i) + \alpha \cdot (Random() - \frac{1}{2}) \tag{2.8c}$$

$$\leftarrow \mathbf{x}^i + \beta \cdot (\mathbf{x}^j - \mathbf{x}^i) + \alpha \cdot (Random() - \frac{1}{2}) \tag{2.8d}$$

$$\leftarrow (1 - \beta) \cdot \mathbf{x}^i + \beta \cdot \mathbf{x}^j + \alpha \cdot (Random() - \frac{1}{2}), \tag{2.8e}$$

where $\gamma \in [0, \infty)$ is the user-defined constant *absorption coefficient* and $I_0 \in [0, \infty)$ is the light intensity at the source; $I_0$ may be the value of the objective function ($I_0 = f(\mathbf{x}^i)$), then we have to

[6] approaching the better local solution is *exploitation* and random step is *exploration* of the searching space. We want the *exploration-exploitation* to keep in a balance for the most efficient search

make sure that $\forall \mathbf{x} \in \mathbb{X} : f(\mathbf{x}) \geq 0$, or usually in practice is set by user to a constant, e.g. $I_0 = 1$. The $\alpha$ is the user-defined value that affects the maximal random step. The function $Random()$ — in these equations — generates a uniformly distributed vector of $[0,1)^n$.

By examining Eq. (2.8) one can observe, that parameters $\alpha$ and $\beta$ influence the step of the firefly in following way: $\beta$ determines the step-size towards the better solution and $\alpha$ determines the maximum radius of the random step.

For example, assume we want to compute the probabilistic distribution of the firefly's position for the next iteration, where the firefly has three rivals with a better solution than is its own. Progress of a such computation illustrates Fig. 2.2. In Fig. 2.2(a) is shown a prior distribution of the fireflies, where the black dot is the computed firefly, and the red dots are the fireflies with the better objective values than the objective value of the computed firefly. In Fig. 2.2(b) are done steps $\beta_1$ and $\alpha_1$ towards the left firefly, where the grey circle marks possible distribution of the firefly's position after these two steps. Since the Random() function is uniformly distributed, our circle[7] is uniformly distributed. In (c) the firefly's next step $\beta_2$ is shown — the whole probability distribution shifts according to this $\beta_2$. After executing another $\alpha_2$ step, which is again a uniform distribution, resulting distribution becomes a triangular distribution [14], having peaked at the centre, as it is depicted in (d), and fading linearly away with growing distance from the centre. Triangular distribution's radius is $\alpha_1 + \alpha_2 = \alpha + \alpha = 2\alpha$. Using the same steps and rules we obtain (e) and (f). The more $\alpha$s we add together, the more resulting distribution will resemble the Gaussian distribution [14]. All the $\alpha$ steps are alike for the length parts, whereas lengths of $\beta$ steps' are proportional to the distance between given fireflies.

[7] the colour intensity indicates the probability at that point

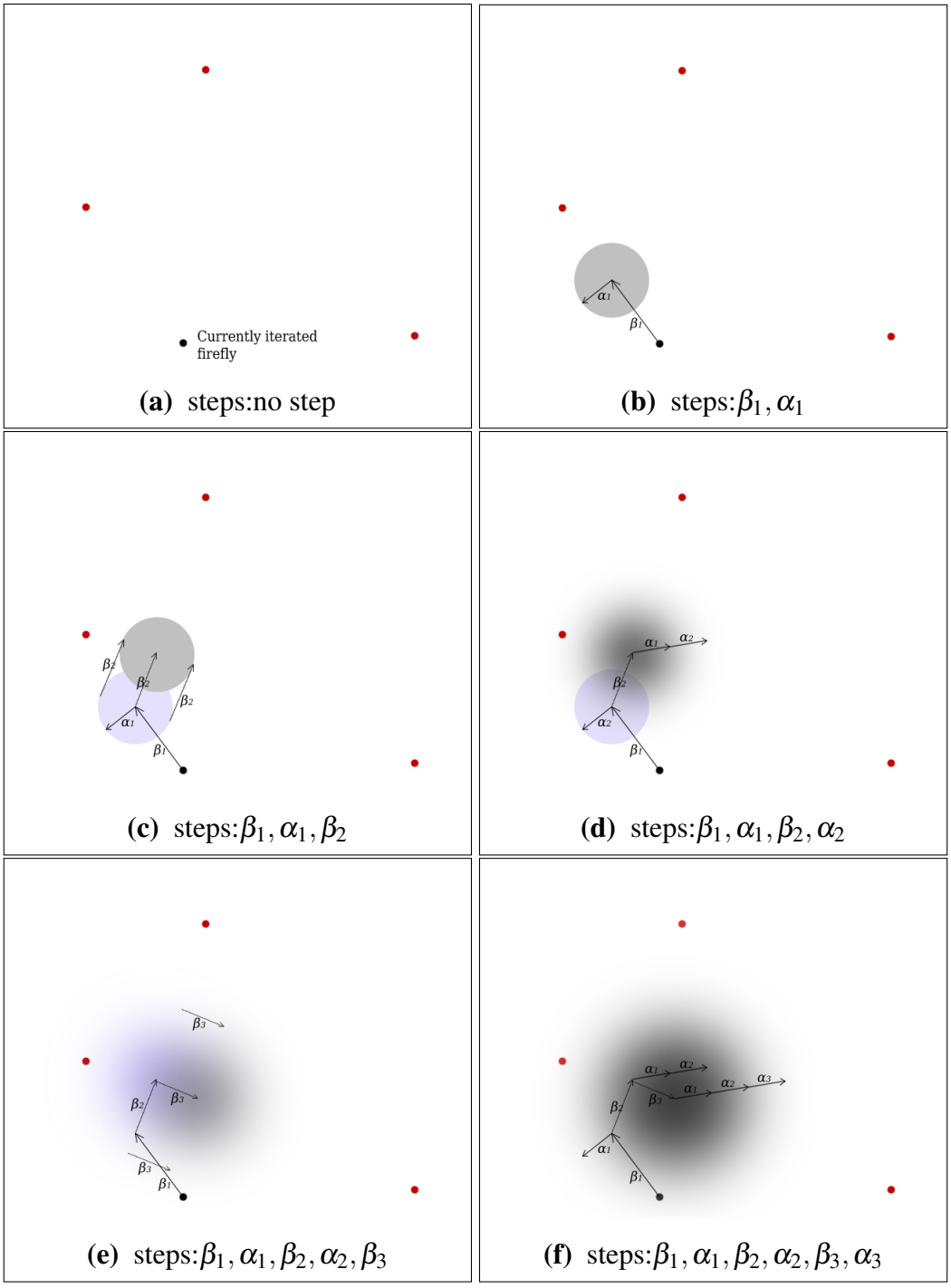



**Figure 2.2** Computation process of the probabilistic distribution of the firefly's position for the next iteration

### 2.2.4 Firefly Algorithm's Special Extreme Cases

There are two special cases in the Firefly Algorithm. The two exceptions are in a relation to the absorption coefficient $\gamma = 0$ and $\gamma = \infty$ [15].

- if $\gamma = 0$, then $\beta = I_0 e^{-0 \cdot d^2} = I_0$; in our story of the fireflies it means that air among fireflies is absolutely clear with no light dispersion, hence each firefly can clearly see another fireflies as bright as they emit the light at any distance. Here $\beta$ is always the largest it could possibly be, therefore, fireflies attempt to advance towards other fireflies with the largest possible steps. The *exploration-exploitation*, in this case, is out of the balance, because exploitation is maximal and exploration normal.

- if $\gamma = \infty$, then $\beta = I_0 e^{-\infty \cdot d^2} = 0$; in the story this would mean, that fireflies are surrounded with very thick fog and could not see any of the other fireflies. The only movement fireflies would be doing is the $\alpha$-steps, which are random steps. Here the *exploration-exploitation* is out of the balance as well, as fireflies would do only the exploration, with no exploitation the search space.

The most efficient searching is when the *exploitation-exploration* is in balance, or first to emphasize the exploration, and then the exploitation. Therefore the best when $\gamma$ is in a range $[0, \infty)$.

# Chapter 3

# Quadratic Assignment Problem

## 3.1 Problem Statement

The Quadratic Assignment Problem, introduced by Koopmans and Beckmann in 1957 as a mathematical model for the location of indivisible economic activities [3], is one of the basic computational problems in computer science with complexity strong *NP*-hard [11]. For the better comprehension of the problem, it is often simplified as follows: for the given $n$ facilities and $n$ locations, with the given flows between all the pair of the facilities and the given distances between all the pairs of the locations, find an assignment of each facility to the unique location, such that the total cost, computed as the sum of distances times corresponding flows, is minimum. Mathematical formulation of the problem is as follows: we are given two $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$, where usually $a_{ij}, b_{ij} > 0$, and the task is to find:

$$\min_{\pi \in \mathbb{S}_n} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)}, \tag{3.1}$$

where $\mathbb{S}_n$ is set of all possible permutations of $(1,2,\cdots,n)$. Sometimes there is an accessory $n \times n$ matrix $C = (c_{ij})$, which is added to the latter equation. Problem formula then becomes:

$$\min_{\pi \in \mathbb{S}_n} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi(i)\pi(j)} + \sum_{i=1}^{n} c_{i\pi(j)}. \tag{3.2}$$

There are many different interpretations for the matrices, one of which is:

- $a_{ij}$ represents the flow from the facility $i$ to the facility $j$,
- $b_{ij}$ represents the distance from the location $i$ to the location $j$,
- $c_{ij}$ represents the cost of the placing facility $i$ to the location $j$ [3].

Applications of the QAP in real life can and have been used to solve problems as are: backboard wiring, scheduling, process communication, design typewriter keyboard, other ergonomic designs, scheduling parallel production lines, turbine runner problem, etc. [3, 7]

## 3.2 Methods of Solving the Problem

There were three methods developed for solving QAP problem exactly, that finds an optimal solution. These methods are: *dynamic programming*, *cutting plane techniques*, and *branch and bound procedure*. The most efficient appears to be the *branch and bound procedure*, although for the problems $n > 30$ these methods remain intractable.

Another approach to solve QAP is using heuristic methods. A heuristic approach does not guarantee finding an optimal solution, but in a relatively short time can find solutions very close to the optimum. The heuristic methods that are mainly used are: Local-Search, Simulated Annealing, Tabu Search, Genetic Algorithm, Greedy Randomized Adaptive Search Procedure (GRASP). Authors in [7] assert, that "Drezner [5] designed a new GA with a problem-specific crossover rule and a tabu search .... This new genetic algorithm is currently one of the best heuristics to solve QAPs.".

# Chapter 4

# Firefly Algorithm Discretization for QAP

## 4.1 What Firefly Algorithm Needs?

Alg. 1 shows us that in order to execute Firefly Algorithm we need to implement functions Initial_Solution() and Distance($x_i$,$x_j$) in a way how it is represented in QAP. We also need to redefine the step movements of the fireflies, and the Attract function, as our search space $\mathbb{X}$ is not $\mathbb{R}^n$, as it was assumed in the pseudo-code, but $\mathbb{S}_n$, the all possible permutations of $(1,2,\cdots,n)$. Thus the line

$$x_i \leftarrow (1-\beta)x_i + \beta x_j + \alpha(\text{Random}() - \frac{1}{2}); \tag{4.1}$$

will be replaced by

$$x_i \leftarrow \text{Attract}(x_i,x_j,\alpha,\beta);. \tag{4.2}$$

In order to define these function for the QAP, we need to represent solutions as the permutations and operate with them accordingly. Any operation with permutations must result in another permutation. Following sections describe the way each function is implemented in the project SEAGE.

### 4.1.1 Initial Fireflies

In the basic form of the Firefly Algorithm, as X. S. Yang describes it, the initial solutions/fireflies are scattered over the search space in a uniform distribution. As our search space is $\mathbb{S}_n$, the set of permutations, we need to produce $m$ random permutations of $(1, 2, \cdots, n)$ as the initial fireflies.

It is worth considering whether using a greedy method for solution initialization would improve the performance of the algorithm. By putting one relatively good solution among the random solutions, would cause the random solutions immediately to advance towards the better solution. This would probably contribute to find the local optima faster, but not for better exploration of the search space. Though this could be the future investigation, I have not used the greedy initialization in my implementation.

## 4.2 Distance Function

Basically, here are two possible ways how to measure the distance between two permutations: (a) Hamming's distance and (b) the number of the required swaps of the first solution in order to get the second one.

Let's examine this example. Consider permutations $\pi_1, \pi_2, \pi_3 \in \mathbb{S}_n$:

$$\pi_1 = [1\ 2\ 3\ 4\ 5\ 6], \tag{4.3}$$

$$\pi_2 = [1\ 2\ 4\ 3\ 6\ 5], \tag{4.4}$$

$$\pi_3 = [1\ 2\ 4\ 5\ 6\ 3]. \tag{4.5}$$

The Hamming's distance between two permutations is the number of non-corresponding elements in the sequence. Therefore, HammingDistance($\pi_1$,$\pi_2$) is 4 (only the first two positions have the same elements). The Swap distance is the number of minimal required swaps of one permutation in order to obtain the other one. SwapDistance($\pi_1$,$\pi_2$) is thereby, 2 (in the $\pi_2$ we swap elements:

4 with 3 and then 5 with 6). HammingDistance($\pi_1$,$\pi_3$) is again 4, but SwapDistance($\pi_1$,$\pi_3$) is now 4. We need such measurement of the distances, so that the closer permutations/solutions are to one another, the smaller difference of their objective functions is produced. In our case, the objective function is computed by the formula Eq. 3.1, respectively Eq. 3.2, thus, it is obvious, that the difference in the results of the objective function of two permutations will decrease with smaller Hamming's distance, not the Swap distance. Following reason can arise: if we compare results of the objective function for $\pi_1$ with $\pi_2$ and then $\pi_1$ with $\pi_3$, we will observe, that they have in common nothing but first two elements, so objective function of all of them will contain the product of 1st and 2nd row and column of the matrices, according to Eq. 3.1 resp. Eq. 3.2. The rest of the rows and the columns in $\pi_1, \pi_2$ and $\pi_1, \pi_3$ will be always different, so $\pi_2$ and $\pi_3$ should be equally distant from the $\pi_1$, which is manifested by Hamming's distance.

## 4.3 Attraction

Attraction has to be implemented and interpreted for the QAP in the same way, as it is intended for the continuous Firefly Algorithm. To calculate next position of a certain firefly in the continuous Firefly Algorithm, we can break up an attraction step into to sub-steps: $\beta$-step and $\alpha$-step. We can take the liberty of doing this, since we know that the result will not change. So the

$$x_i \leftarrow (1-\beta)x_i + \beta x_j + \alpha(\text{Random}() - \frac{1}{2}) \tag{4.6}$$

now becomes

$$x_i \leftarrow (1-\beta)x_i + \beta x_j, \tag{4.7}$$

$$x_i \leftarrow x_i + \alpha(\text{Random}() - \frac{1}{2}). \tag{4.8}$$

Note that the order of the steps $\alpha$ and $\beta$ is not interchangeable. The $\beta$-step must always be computed first and afterwards the $\alpha$-step. If it was the other way round, the $\alpha$-step could move

firefly closer to or farther from the compared firefly, which would result in different $\beta$ parameters in the $\beta$-step equation.

### 4.3.1 $\beta$-step

In the Fig. 2.2 can be seen that the $\beta$-step brings the iterated firefly always closer to another firefly. In other words, after applying a $\beta$-step on a firefly towards the other firefly, their distance is always decreased, and the decrement is proportional to their former distance. Since we use Hamming's distance as our distance function, it means that in order for the one permutation to get closer to the another permutation, the amount of their common elements have to increase. In the $\beta$-step process, first of all we have to extract whatever it is that both permutations have in common, or that is definitely in a resulting permutation. For example, we want the $\pi_1 = [4\ 9\ 3\ 7\ 6\ 8\ 2\ 1\ 5]$ attract to $\pi_2 = [4\ 1\ 3\ 2\ 6\ 5\ 9\ 7\ 8]$, the resulting $\pi_{1\to 2}$ after the first step would look following:

$$\pi_1 = [4\ 9\ 3\ 7\ 6\ 8\ 2\ 1\ 5], \tag{4.9}$$
$$\pi_2 = [4\ 1\ 3\ 2\ 6\ 5\ 9\ 7\ 8], \tag{4.10}$$
$$\pi_{1\to 2} = [4\ \_\ 3\ \_\ 6\ \_\ \_\ \_\ \_]. \tag{4.11}$$

Secondly, we fill in the gaps in $\pi_{1\to 2}$ regarding to permutations former distance. It can be achieved as following: with the probability $\beta = \frac{1}{1+\gamma \cdot d_{\pi_1,\pi_2}}$, where $d_{\pi_1,\pi_2} = HammingDistance(\pi_1,\pi_2)$ we insert into $\pi_{1\to 2}$ an element from $\pi_2$, otherwise retain an element from $\pi_1$. While doing this, we need to watch not to create duplicate elements in the resulting permutation. Therefore, if the element, that is about to be inserted from $\pi_2$ or $\pi_1$ have been used previously, we skip filling in this gap and proceed to the filling of the next one. If after going through all the gaps in this manner, there are still left empty gaps in the result, we fill them randomly with unused elements.

In our example, in order to continue, we need compute the $\beta$ probability. Let us say $\gamma = 0.1$, then $\beta = \frac{1}{1+\gamma d_{1,2}} = \frac{1}{1+0.1\cdot 6^2} = \frac{1}{4.6} \approx 0.217391304$. It is also important to fill in the gaps in random order, otherwise the beginnings would be always filled, and in the ends there would be many conflicts

(meaning that both elements have already been used), hence a retention of empty gaps. The filling process could be as follows:

$$\pi_1 = [4\ 9\ 3\ 7\ 6\ 8\ 2\ 1\ 5], \tag{4.12}$$
$$\pi_2 = [4\ 1\ 3\ 2\ 6\ 5\ 9\ 7\ 8], \tag{4.13}$$
$$\text{starting: } \pi_{1\to 2} = [4\ \_\ 3\ \_\ 6\ \_\ \_\ \_\ \_], \tag{4.14}$$
$$\text{Filling } 6^{th} \text{ position: } \pi_{1\to 2} = [4\ \_\ 3\ \_\ 6\ 8\ \_\ \_\ \_], \tag{4.15}$$
$$\text{Filling } 2^{nd} \text{ position: } \pi_{1\to 2} = [4\ 1\ 3\ \_\ 6\ 8\ \_\ \_\ \_], \tag{4.16}$$
$$\text{Filling } 9^{th} \text{ position: } \pi_{1\to 2} = [4\ 1\ 3\ \_\ 6\ 8\ \_\ \_\ 5], \tag{4.17}$$
$$\text{Filling } 4^{th} \text{ position: } \pi_{1\to 2} = [4\ 1\ 3\ 7\ 6\ 8\ \_\ \_\ 5], \tag{4.18}$$
$$\text{Filling } 8^{th} \text{ position: } \pi_{1\to 2} = [4\ 1\ 3\ 7\ 6\ 8\ \_\ \_\ 5] \text{(stays empty, since 1 and 7 are used)}, \tag{4.19}$$
$$\text{Filling } 7^{th} \text{ position: } \pi_{1\to 2} = [4\ 1\ 3\ 7\ 6\ 8\ 2\ \_\ 5]. \tag{4.20}$$

The last step is to fill the rest with unused elements. In our example only the number 9 has not been used, so the $\pi_{1\to 2}$ = [4 1 3 7 6 8 2 9 5]. So the $HammingDistance(\pi_1, \pi_2) = 6$, and $HammingDistance(\pi_{1\to 2}, \pi_2) = 5$, so $\pi_{1\to 2}$ has approached towards $\pi_2$ of one distance unit closer.

### 4.3.2 $\alpha$-step

The $\alpha$-step is much simpler than the $\beta$-step. It should allow us to shift the permutation into one of the neighbouring permutations. The smallest distance of two neighbouring permutations involves one swapping of two elements, therefore all the neighbours are distant from iterated firefly always by 2 distance units. There are two ways how to apply the $\alpha$-step: either to make $\alpha$·Random() many swaps of randomly chosen two elements, or to choose $\alpha \cdot Random()$ many elements, and shuffle their positions. The first option is easier to implement, but results are not as abundant as the second one. Reason for this is following. If we make two consequent swaps, the distance from the original permutation can increase up to 4, for three swaps it is up to 6, etc. Whereas the second option allows us to move odd distance units as well, therefore I have picked the second option. Since $\alpha$ represents a maximal allowed step for the permutation, that consists of $n$ elements, to

make, we need $\alpha$ to be from the set $\{1,\ldots,n\}$. Then $\alpha = 1$ means no step is made [1] and $\alpha = n$ means to shuffle all the elements in the permutation. To clarify, the Hamming Distance $\pi$ from the $\pi_{new}$, that was created from $\pi$ by $\alpha$-step, is always $\leq \alpha$.

[1] choosing 1 element's position to be shuffled in a permutation does not change permutation at all

# Chapter 5

# Experiments

## 5.1 Arrangement

There were made tens of different experiments, several of which were chosen into this paper to illustrate efficiencies and abilities of the Discrete Firefly Algorithm designed for the QAP. Configuration of the DFA simulation consists in setting: of the population size $m$, of the number of the iterations $iter$, of the absorption coefficient $\gamma$ and of the time step $timeStep$. Since experiments are not to be compared among one another in the time domain, type of the processor, the memory size and other technical components are irrelevant. All the experiments were made on 11 QAP problems: chr12a, tai12a, esc16h, nug18, had20, chr20a, chr25a, tai25b, bur26a, tho30 and esc32g, chosen from the QAPLIB at [2], and were executed using built-in EXPERIMENTER in the framework SEAGE. EXPERIMENTER environment enables user to execute comfortably experiments by choosing what algorithms are to be ran upon what problems. User also has to set ranges in which algorithm's parameters are to be chosen, number of iterations and number of simulations. EXPERIMENTER will run everything automatically and results will be stored in the generated XML files.

## 5.2 Experiments' Description

### 5.2.1 Experiment I

In this experiment, there were launched 100 Firefly Algorithm simulations per problem, and each simulation consisted of the population size $m = 100$ and $iter = 100$ iterations. The parameters $\gamma$ and $timeStep$ were chosen randomly in the ranges $\gamma \in [0.0; 1.0]$ and $timeStep \in [0.1, 2.0]$. This gives us the survey of what configuration ($\gamma$ and $timeStep$) has best results for each QAP problem.

### 5.2.2 Experiment II

This experiment consisted of: 10 Firefly Algorithm simulations per each problem with the $iter = 500$ iterations and the population size $m = 500$. The parameters $\gamma$ and $timeStep$ were chosen randomly in the range $\gamma \in [0.0; 1.0]$ and $timeStep \in [0.1, 2.0]$. This experiment is to be compared with the results of the Experiment I, and should provide us an information, whether and how much, the decrease of the amount of the simulations together with the increasing the population size and iterations, would make an improvement.

### 5.2.3 Experiment III

Experiment setting was: 100 Firefly Algorithm simulations per problem with $iter = 500$ iterations and population size $m = 500$. Parameters $\gamma$ and $timeStep$ were chosen randomly in the range $\gamma \in [0.0; 1.0]$ and $timeStep \in [0.1, 2.0]$.

**Table 5.1** Experiment I: 100 Firefly Algorithm simulations, consisted of $iter = 100$ iterations and $m = 100$ of the population size, for each problem

| problem | optimum | obj. value | found | $\gamma$ | $timeStep$ |
|---|---|---|---|---|---|
| chr12a | 9552 | 9552 | opt. | 0.8509 | 0.8341 |
| tai12a | 224416 | 224416 | opt. | 0.2209 | 0.2712 |
| esc16h | 996 | 996 | opt. | 0.2209 | 0.2712 |
| nug18 | 1930 | 1938 | - | 0.7856 | 1.1389 |
| had20 - | 6922 | 6922 | opt. | 0.9482 | 0.1957 |
| chr20a | 2192 | 4206 | - | 0.1201 | 0.5693 |
| chr25a | 3796 | 9196 | - | 0.1251 | 1.5856 |
| tai25b | 34435646 | 386867346 | - | 0.1719 | 1.4515 |
| bur26a | 5426670 | 5465662 | - | 0.8879 | 0.1099 |
| tho30 | 149936 | 174590 | - | 0.8299 | 1.1934 |
| esc32g | 6 | 6 | opt. | 0.2209 | 0.2712 |

## 5.3 Results

### 5.3.1 Results of Experiment I

From 100 simulations for each problem, there were chosen simulations with the best results and put into the Table 5.1. Algorithm found optimum for 5 out of 11 problems and in the rest the found results are relatively close to the optimum, except the problems chr20a and chr25a, which seems to be difficult for the DFA to solve.

Table 5.2 shows $\mu$, the *means*, and $\sigma$, the *standard deviations*, of the *objective values*, $\gamma$s and $timeStep$s, calculated from the 10 best results out of 100 of each problem. Problems tai12a,

**Table 5.2** Experiment I: Means ($\mu$) and standard deviations ($\sigma$) of the *objective values*, $\gamma$s and *timeStep*s, calculated from 10 best results from Experiment I

| problem | $\mu(obj.\ value)$ | $\sigma(obj.value)$ | $\mu(\gamma)$ | $\sigma(\gamma)$ | $\mu(timeStep)$ | $\sigma(timeStep)$ |
|---|---|---|---|---|---|---|
| chr12a | 9590.4 | 114.4797 | 0.47648 | 0.31114 | 1.0946 | 0.56714 |
| tai12a | 224416 | 0 | 0.34626 | 0.22607 | 1.0837 | 0.46997 |
| esc16h | 996 | 0 | 0.58051 | 0.2758 | 0.96023 | 0.68788 |
| nug18 | 1965 | 14.1814 | 0.67373 | 0.28836 | 1.0259 | 0.69435 |
| had20 | 6943.8 | 15.4186 | 0.59939 | 0.3195 | 0.96493 | 0.68728 |
| chr20a | 4472.6 | 112.5425 | 0.46691 | 0.3158 | 1.0825 | 0.57128 |
| chr25a | 9663.4 | 174.1786 | 0.59425 | 0.34911 | 0.85152 | 0.57947 |
| tai25b | 394876343.5 | 4858277.7263 | 0.601 | 0.32635 | 1.139 | 0.62356 |
| bur26a | 5474216.1 | 3356.0987 | 0.6557 | 0.29502 | 0.96376 | 0.66804 |
| tho30 | 177192.6 | 1171.9947 | 0.49934 | 0.36375 | 1.0484 | 0.57131 |
| esc32g | 6 | 0 | 0.64008 | 0.3396 | 0.79045 | 0.59141 |

esc16h and esc32g have $\sigma(obj.\ value) = 0$, which means all the 10 most successful simulations have found exactly the same value, which is in these cases was the optimum solution. Problems nug18 and had20 had small dispersion of the results, since their $\sigma(obj.\ value)$ is small. On the contrary, problem tai25b had the biggest dispersion. The $\gamma$s are around the value 0.5, which is the mean of the uniform distribution $U[0,1)$, from which the values were chosen. Same for the *timeStep*s that were chosen from $U[0,2)$ and are around the value 1.0, so no general conclusion can be made out.

**Table 5.3** Experiment II: 10 Firefly Algorithm simulations on each problem of $iter = 500$ iterations and $m = 500$ of the population size

| problem | optimum | obj. value | found | $\gamma$ | $timeStep$ |
|---|---|---|---|---|---|
| chr12a | 9552 | 9552 | opt. | 0.8600 | 1.2797 |
| tai12a | 224416 | 224416 | opt. | 0.4220 | 1.7274 |
| esc16h | 996 | 996 | opt. | 0.4220 | 1.7274 |
| nug18 | 1930 | 1958 | - | 0.4688 | 1.8760 |
| had20 | 6922 | 6922 | opt. | 0.4688 | 1.8760 |
| chr20a | 2192 | 4188 | - | 0.4688 | 1.8760 |
| chr25a | 3796 | 9120 | - | 0.8393 | 0.6063 |
| tai25b | 34435646 | 379405014 | - | 0.9836 | 1.4223 |
| bur26a | 5426670 | 5432767 | - | 0.7896 | 1.7993 |
| tho30 | 149936 | 175874 | - | 0.3061 | 1.0531 |
| esc32g | 6 | 6 | opt. | 0.4220 | 1.7274 |

### 5.3.2 Results of Experiment II

Results of the Experiment II are in the Table 5.3. In spite of running only 10 simulations per problem, but raising iterations and population size, results are better (meaning closer to the optima) in all the problems but one, the chr20a instance, which got slightly worse.

### 5.3.3 Results of Experiment III

This experiment took computation time of 12 days on processor AMD Athlon II X2 260 Dual Core 3.2GHz. Results are illustrated in the Table 5.4, and are again, as it was expected, better than the results from the Experiments I and II. The means and standard deviations are in the Table 5.5. In this

**Table 5.4** Experiment of 100 Firefly Algorithm simulations on each problem of 500 iterations and 500 of the population size

| problem | optimum | FA | found | $\gamma$ | $timeStep$ |
|---|---|---|---|---|---|
| chr12a | 9552 | 9552 | opt. | 0.1823 | 0.6544 |
| tai12a | 224416 | 224416 | opt. | 0.5955 | 1.5959 |
| esc16h | 996 | 996 | opt. | 0.2336 | 1.9660 |
| nug18 | 1930 | 1946 | - | 0.5959 | 1.7005 |
| had20 | 6922 | 6922 | opt. | 0.2336 | 1.9560 |
| chr20a | 2192 | 3824 | - | 0.4339 | 1.6649 |
| chr25a | 3796 | 8716 | - | 0.0177 | 1.3468 |
| tai25b | 34435646 | 372821202 | - | 0.1946 | 1.9393 |
| bur26a | 5426670 | 5432634 | - | 0.1823 | 0.6544 |
| tho30 | 149936 | 171546 | - | 0.9816 | 0.6227 |
| esc32g | 6 | 6 | opt. | 0.5959 | 1.7005 |

table we can observe, that all the $\sigma(obj.\ value)$ has lessened, which means that this configuration, of longer run ($iter = 500$) and bigger population ($m = 500$), results in having lower dispersion, hence is more stable, than it was in the Experiment I ($iter = 100$ and $m = 100$). Another noticeable change is that the standard deviations of $\gamma$s, the $\sigma(\gamma)$, of the problems chr12a, esc16h, had20 and esc32g have decreased. It means that these best 10 runs had their $\gamma$s set close to the $\mu(\gamma)$. This way we can determine best configuration of DFA for these problems. For example to obtain best solution of the problem esc16h, the best setting is to set $\gamma \approx 0.47$. Of course to gain more accurate values, we need to run much more experiments.

Graphs in Fig. 5.1 and Fig. 5.2 show the progress of the search for the optima of all the 11

**Table 5.5** Best 10 result from experiment III

| problem | $\mu(obj.\ value)$ | $\sigma(obj.\ value)$ | $\mu(\gamma)$ | $\sigma(\gamma)$ | $\mu(timeStep)$ | $\sigma(timeStep)$ |
|---|---|---|---|---|---|---|
| chr12a | 9552 | 0 | 0.36800 | 0.1987 | 0.94430 | 0.46860 |
| tai12a | 224416 | 0 | 0.52923 | 0.32234 | 1.18590 | 0.64542 |
| esc16h | 996 | 0 | 0.47245 | 0.12869 | 1.27930 | 0.47849 |
| nug18 | 1995 | 46.7166 | 0.58521 | 0.24987 | 1.33140 | 0.54945 |
| had20 | 6922 | 0 | 0.42085 | 0.18225 | 1.29800 | 0.55875 |
| chr20a | 3982 | 84.5327 | 0.53728 | 0.26636 | 0.93169 | 0.61692 |
| chr25a | 9077.8 | 174.1786 | 0.59425 | 0.34911 | 0.85152 | 0.57947 |
| tai25b | 375488560 | 1208102.9 | 0.50898 | 0.30296 | 0.92502 | 0.60225 |
| bur26a | 5433182.4 | 567.1596 | 0.55538 | 0.25451 | 0.97128 | 0.55667 |
| tho30 | 174076.6 | 1470.5156 | 0.42932 | 0.36307 | 1.16220 | 0.42789 |
| esc32g | 6 | 0 | 0.47245 | 0.12869 | 1.27930 | 0.47849 |

problems:

- problem chr12a — in the Fig. 5.1(a) the DFA found optimum 23 times out of 100 runs. After 100 iterations the improvement is very weak, which means that solutions stuck in a local optima, so for this problem it is better to run more simulations with lower iteration number than the other way around.

- problem tai12a — the Fig. 5.1(b) is a similar case as it is in the Fig. 5.1(a). Improvement progress here usually stops even earlier, after about 50 iterations. Optimum found about 18 times out of 100 runs.

- problem esc16h — is very easy for the DFA, since the optimum was found in all 100 cases and under 200 iteration as it is shown in Fig. 5.1(c).

- problem nug18 — seem to be on the other hand quite difficult to solve for the DFA, since it did not find an optimum at all. Fig. 5.1(d) shows that after 100 iterations results are scattered (big dispersion).

- problem had20 — belongs among the simple problems for DFA since optimum was found 34 times out of 100 runs, as it is depicted in Fig. 5.1(e).

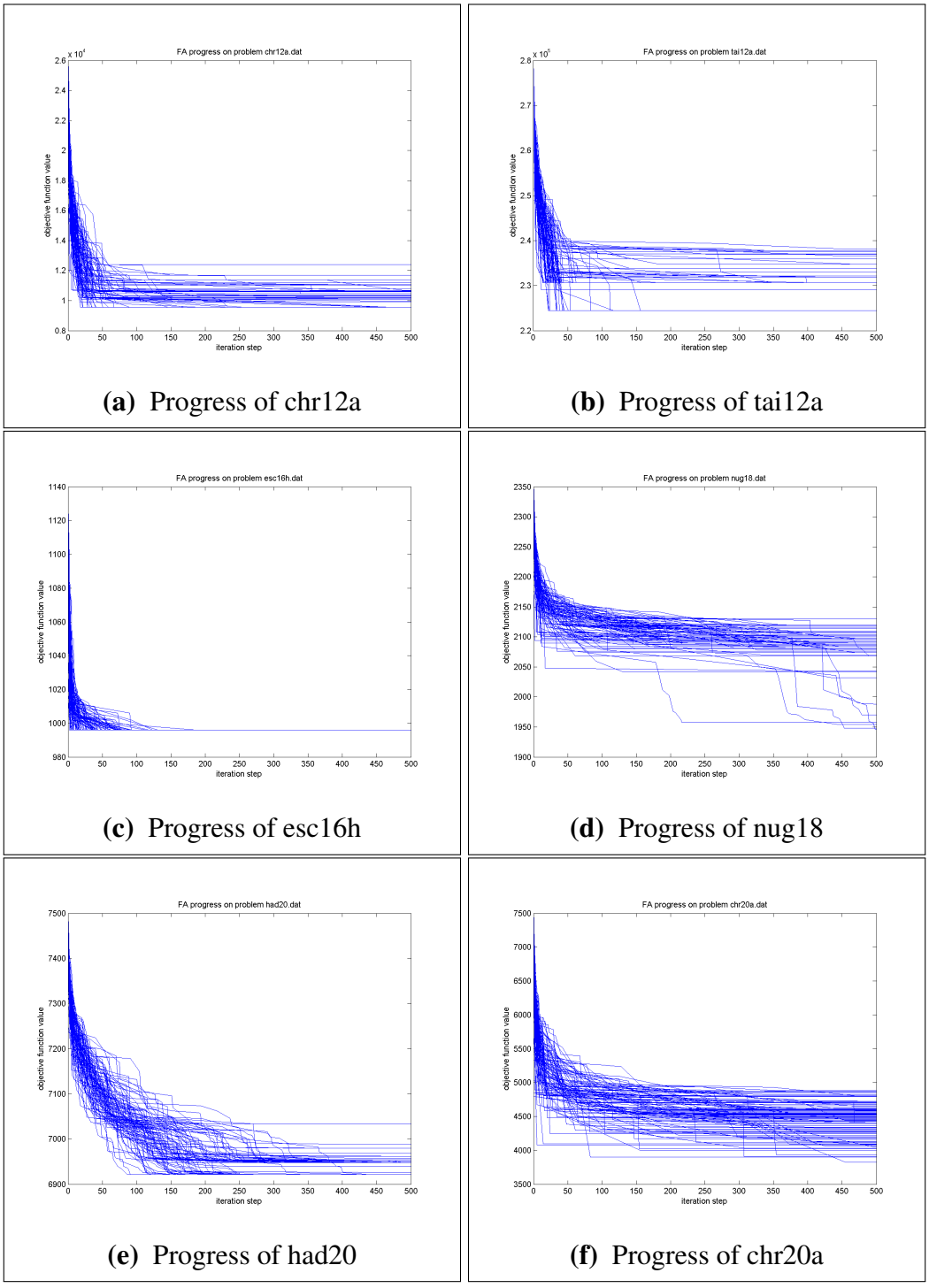
- problems chr20a, chr25a, tai25b, bur26a and tho30 — optimum for these problems were not found in 500 runs at all. Progress is depicted in Fig. 5.1(f) and Fig. 5.2(a)-(d). There were still ongoing improvements, but in only few simulations. These problems have very large search space[1], and require more than 500 iterations to find their optima. Point of heuristics is to find a relatively good solution in short time, not the optimum. And as it can be seen, improvement of these problems from the random starting solution is big

[1] problem chr20a has got $20! = 2.4 \times 10^{18}$ permutations; chr25a and tai25b have got $25! = 1.5 \times 10^{25}$ permutations; bur26a has got $26! = 4 \times 10^{26}$ permutations and tho30 $30! = 2.6 \times 10^{32}$ permutations.

- problem esc32g — though its space has got $32! = 2.6 \times 10^{35}$ different permutations, optimum was always found in fewer than 100 iterations. The reason is because in this problem, there are many optima scattered around the search space. All the 100 runs found unique permutation that is a optimum for this problem.

## 5.4 Discussion

Before launching Experiments, I assumed that the larger the space search is, the lower absorption coefficient $\gamma$ should be set, in order to find better results. My reasoning was, that if the fireflies for the large search space are set into a thick fog (absorption coefficient represents the fog), they would constantly roam randomly since they would not be able to see any of the other fireflies, whereas in a thin fog, they would be able to see others advance toward them, hence to the balance *exploration* and *exploitation* in the searching. On the other hand, in a small search space, the fog should be thicker to prevent fireflies rapidly advance towards the best solution and stuck in it's suboptimal valley. Experiments have shown it is not true, and that the absorption coefficient depends more on a problem itself rather than the search space size. Fortunately it was possible to deduce best $\gamma$ for the couple of problems.




**(a)** Progress of chr12a

**(b)** Progress of tai12a

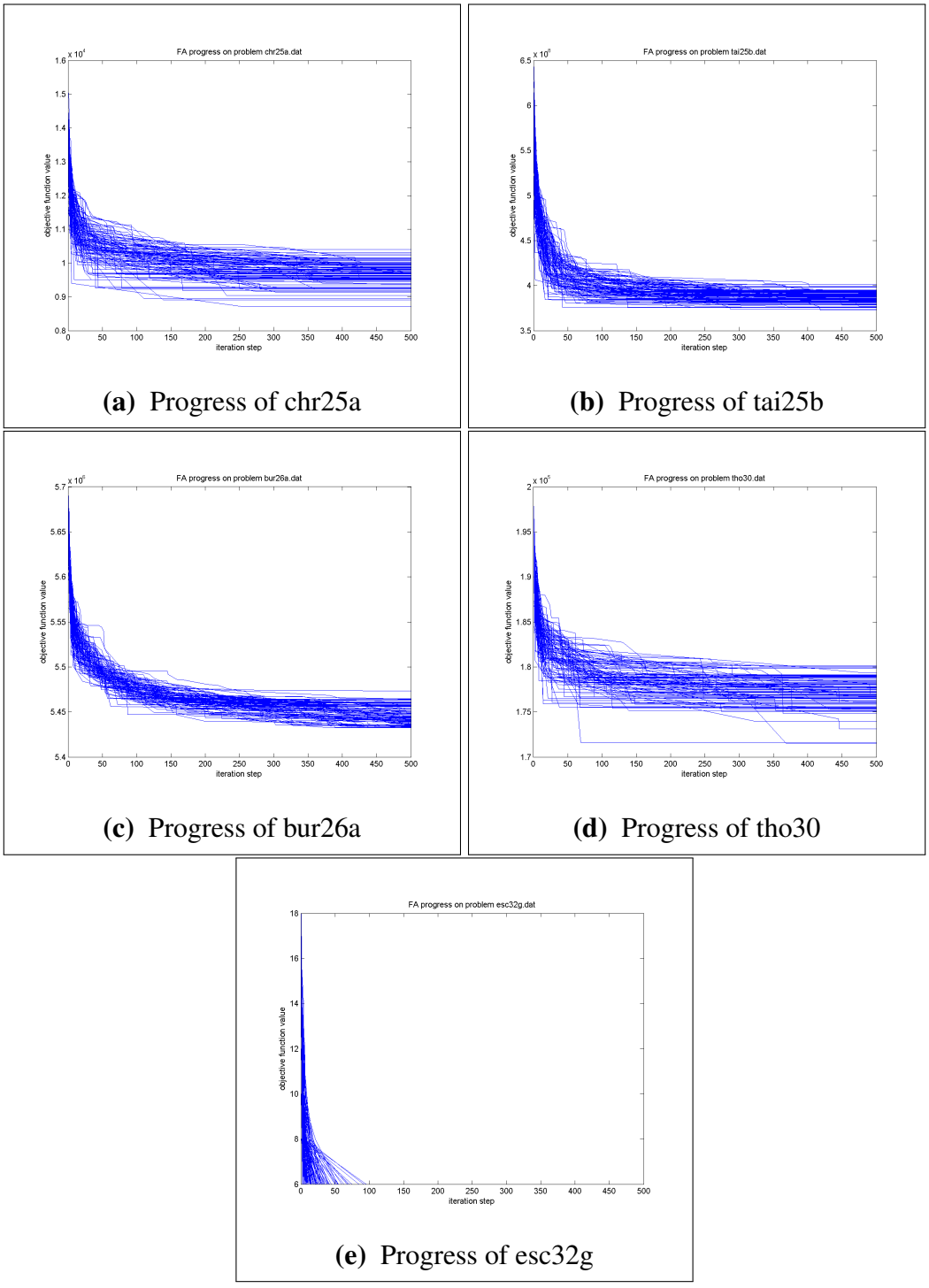
**(c)** Progress of esc16h

**(d)** Progress of nug18

**(e)** Progress of had20

**(f)** Progress of chr20a

**Figure 5.1** Progress' of finding the lowest objective function value of different problems. Data are from the results of Experiment III




**(a)** Progress of chr25a

**(b)** Progress of tai25b

**(c)** Progress of bur26a

**(d)** Progress of tho30

**(e)** Progress of esc32g

**Figure 5.2** Progress' of finding the lowest objective function value of different problems. Data are from the results of Experiment III

# Chapter 6

# Conclusion

The goal of this thesis, to convert continuous Firefly Algorithm into Discrete Firefly Algorithm to solve the Quadratic Assignment Problems, was successfully achieved. There were proposed discrete versions of the continuous functions *Distance*, *Attractiveness* and *Movement*. DFA was implemented into the SEAGE framework in abstract level, giving the user liberty in defining the functions *Distance*, *Attractiveness* and *Movement* anyway it is desired. This gives the algorithm ability to solve other than QAP kinds of problems, since it relies only on the functions listed above. If someone wants to use DFA to solve e.g., Graph Colouring Problem (GCP), all he is required to do is to define how to measure distances between two GCP solutions (*Distance* function), how to measure *Attractiveness* between them, and to define a *Movement* of one solution towards the another. The Quadratic Assignment Problem was implemented into the SEAGE framework as well, which of 11 instances were created. DFA functionality was experimentally tested on 11 QAP instances and showed to be able to find successfully good solutions of these problems. Optima of the simple problems (instances chr12a, tai12a, esc16h, had20 and esc32g) were often reached in fewer than 150 iterations, though for the hard problems it did not find an optimum even in 500 iterations, but was seen a big improvement from the starting random solution.

# Appendix A

# List of used symbols

$\mathbb{X}$ – is the set of all the possible solutions within the search space

$n \in \mathbb{Z}$ – is the search space's dimension

$\vec{x}, x \in \mathbb{X}$ – is a particular solution in the search space $\mathbb{X}$; if $\mathbb{X} = \mathbb{R}^n$, then is used $\vec{x} = (x_1, x_2, \cdots, x_n)$, otherwise $x$ is used

$\vec{p}, p \in \mathbb{X}$ – is the best solution that individual/particle/solution $\vec{x}$ have reached during it's lifetime

$\vec{g}, g \in \mathbb{X}$ – is a globally best so far found solution; $\vec{g} = \text{argmin}_{p_i}(f(p_1), f(p_2), \cdots, f(p_m))$[1]

$\vec{v} \in \mathbb{R}^n$ – is a velocity vector; in some optimizations next position of $x_i^t$ is $x_i^{t+1} = x_i^t + v_i$

$m \in \mathbb{Z}$ – is a size of a population/swarm; number of the solutions/individuals/particles that concurrently try for improvement

$d_{i,j} \in [0, \infty)$ – is the distance — or amount of differentness — of the $i$-th solution from the $j$-th solution

[1] in case of the minimalization; for the maximalization instead of min there is max

$\gamma \in [0,\infty)$ – is the *absorption coefficient* that denotes the rate of attenuation of the light intensity over the distance

$f : \mathbb{X} \to \mathbb{R}$ – is *objective function* — called also cost function — that returns evaluation of given solution. To find the best solution means to find $x = \text{argmin}_{x \in \mathbb{X}}(f(x))$ (or *argmax*). Such minimum or maximum is then called an *Optimum* or an *Optimal solution* of $\mathbb{X}$

$U(a,b)$ – is a uniform distribution on the interval $(a,b)$ $(a,b \in \mathbb{R}; a < b)$

$\mathbb{S}_n$ – is set of all permutations $(1,2,\cdots,n)$

***objective function*** $f : \mathbb{X} \to \mathbb{R}$ – also called cost function, is a function we want to minimize (or maximize, if desired)

***optimum*** $x_{opt} \in \mathbb{X}$ – is the minimum (resp. the maximum) of the *objective function*. Formally, $x_{opt}$ is *optimum* if and only if: $\forall x \in \mathbb{X} : \ f(x) \leq f(x_{opt})$ (resp. $f(x) \geq f(x_{opt})$)

***objective value*** **of** $x \in \mathbb{X}$ – is the value produced by *objective function* $f(x)$

# Bibliography


[1] Encyclopadia Britannica. Firefly. http://www.britannica.com/EBchecked/topic/207935/firefly, May 2011.

[2] Rainer Burkard, Eranda Çela, Stefan Karisch, and Franz Rendl. QAPLIB - a quadratic assignment problem library. http://www.opt.math.tu-graz.ac.at/qaplib/.

[3] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problem*. Society for Industrial and Applied Mathematics, Philadelphia, 2009.

[4] Konstantin Chakhlevitch and Peter Cowling. Hyperheuristics: Recent developments. *Adaptive and Multilevel Metaheuristics*, 2008.

[5] Zvi Drezner. A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 2003.

[6] Marco Dorigo et al. Particle swarm optimization. http://www.scholarpedia.org/article/Particle_swarm_optimization, November 2008.

[7] Ping Ji, Yongzhong Wu, and Haozhao Liu. A solution method for the quadratic assignment problem (qap). *Operations Reasearch and Its Application*, Sixth International Symposium, 2006.

[8] Richard Málek. Global optimization through meta-heuristic collaboration in a multi-agent system, August 2009.

[9] Richard Málek. Seage: Search agents for optimization. http://www.seage.org, July 2009.

[10] Magnus Erik Hvass Pedersen and Andrew John Chipperfield. Simplifying particle swarm optimization. *Applied Soft Computing Journal*, 2010.

[11] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the Association for Computing Machinery*, 1976.

[12] Michal Smith. *Simple Firefly Synchronization*. PhD thesis, University of Daleware, 2008.

[13] Szymon Łukasik and Sławomir Žak. Firefly algorithm for continuous constrained optimization tasks, 2009. Lecture Notes in Computer Science.

[14] Eric Weisstein. Uniform sum distribution. http://mathworld.wolfram.com/UniformSumDistribution.html, 2011.

[15] Xin-She Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[16] Xin-She Yang. Firefly algorithm for multimodal optimization. *Stochastic Algorithms: Foundation and Application*, 5th, 2009.

[17] Xin-She Yang. Metaheuristic optimization. http://www.scholarpedia.org/article/Metaheuristic_Optimization, December 2010.