Grade Diffusion and Quantum Inspired Evolutionary Algorithm to Solve Fault Node Recovery Problem in Wireless Sensor Network

Submitted by:

Yeasin Muhammad Nur
ID: 2011-2-60-034

Supervised by:

Dr. Mohammad Rezwanul Huq

Assistant Professor
Department of Computer Science and Engineering
East West University

A project submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering to the Department of Computer Science and Engineering
At the

East West University
Aftabnagar, Dhaka, Bangladesh
January, 2016

Abstract:

Wireless sensor network (WSN) consists of spatially dispersed and dedicated sensors for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location. In most of the case wireless sensor network (WSN) measure environmental conditions such as temperature, sound, pollution levels, humidity, wind speed and direction, pressure, etc. In this paper we applied fault sensor node recovery algorithm to increase the life time of a WSN when few sensors are damaged or shut down due to technical problem or low level of battery power. This algorithm is based on the Grade diffusion algorithm combined with the quantum inspired evolutionary algorithm.

Declaration

We hereby declare that, this project was done under CSE497 and has not been submitted elsewhere for requirement of any degree or diploma or for any purpose except for publication.

Yeasin Muhammad Nur
ID: 2011-2-60-034
Department of Computer Science and Engineering
East West University

Letter of Acceptance

I hereby declare that this thesis is from the student's own work and best effort of mine, and all other source of information used have been acknowledge. This thesis has been submitted with my approval.

______________________________ **Supervisor**

Dr. Mohammad Rezwanul Huq

Assistant professor

Department of Computer Science and Engineering

East West University

____

Dr. Shamim H. Ripon **Chairperson**

Associate Professor and Chairperson

Department of Computer Science and Engineering

East West University

Acknowledgement

Firstly my most heartfelt gratitude goes to my beloved parents for their endless support, continuous inspiration, great contribution and perfect guidance from the beginning to end.

I owe my thankfulness to my supervisor Dr. Mohammad Rezwanul Haq for his skilled, atmost direction, encouragement and care to prepare myself.

My sincere gratefulness for the faculty of Computer Science and Engineering whose friendly attitude and enthusiastic support that has given me for four years.

I am very grateful for the motivation and stimulation from my good friends and seniors.

I also thank the researchers for their works that help me to learn and implement Grade Diffusion and Quantum Inspired Evolutionary Algorithm to Solve Fault Node Recovery Problem in Wireless Sensor network.

TABLE OF CONTENTS

List of Figures

List of Table

List of Algorithms

Abbreviation and Acronyms

QEA	Quantum-inspired Evolutionary Algorithm
GD	Grade Diffusion
WSN	Wireless Sensor Network
DD	Direct Diffusion

Chapter 1

Introduction

1.1 About Wireless sensor network (WSN):

A WSN consists of anywhere from a few hundreds to thousands of sensor nodes. The sensor node equipment includes a radio transceiver along with an antenna, a microcontroller, an interfacing electronic circuit, and an energy source, usually a battery. The size of the sensor nodes can also range from the size of a shoe box to as small as the size of a grain of dust. As such, their prices also vary from a few pennies to hundreds of dollars depending on the functionality parameters of a sensor like energy consumption, computational speed rate, bandwidth, and memory. WSN is using for monitor physical or environmental conditions.

1.2 When fault has happened:

In sensors networks, every sensors node has limited wireless power for processing and transferring the live data to the base station or data collection center [1], [2], [3]. Every sensor networks usually contains lots of sensor nodes. Normally every sensor node has low level of battery of power that cannot be replenished. If the sensor node has low power or damaged ,wireless sensor network leaks will appear and the fault nodes will not able to transmit the data to the other nodes during transmission processing. In that case other nodes will be burdened [4].

1.3 Techniques of solving fault node recovery problem:

There are few techniques to solve FNR problem .The algorithm proposed in this paper which able to solve FNR problem .This algorithm based on GD algorithm combined with the QEA algorithm with the goal of replacing fewer sensor nodes that are damaged or have depleted batteries and of reusing the maximum number of routing paths .These optimizations will ultimately enhance the WSN life time and reduce sensor node replacement cost.

1.4 Objective of the Research:

We create a WSN which is the mimic of real WSN. We frequently create events in different sensor nodes which causes the depletion of the batteries .So after passing messages some nodes were shut down .Our objective is to create a WSN using GD algorithm and replacing fewer sensor nodes that are damaged or have depleted batteries and of reusing the maximum number of routing paths .These optimizations will ultimately enhance the WSN life time and reduce sensor node replacement cost.

1.5 Methodology of Research:

In this paper we are going to propose a algorithm which is the combination of GD algorithm and Quantum-inspired Evolutionary Algorithm .Here QEA also the combination [5] of concept of quantum computing and evolutionary algorithm [6].GD algorithm discussed in chapter 3. QEA uses quantum chromosomes that are actually Q-bit individuals. Binary chromosomes will be produced from these quantum chromosomes. Q-gate is the main variation operator. That means, Q-bit individual will be updated mainly using Q-gate.QEA discussed in chapter 4.

Chapter 2

Related Work

Fault node recovery problem (FNR) it requires a huge amount of a computation because the problem is NP-hard. Previously wireless sensor network proposes lot of routing algorithms and energy algorithms. This project is also continued after observe the previous methodologies are helpful to provide effective communication and efficient energy to a sensor network nodes. Generally wireless sensor contains lot of sensor nodes armed with computing and sensing and communication devices over wireless channels. The main aim of WSN is to collect event data from the source and send it to sink node. In previously used two algorithms are directed diffusion algorithm and grade diffusion algorithm [7]. In this grade diffusion algorithm used finding the grade each and every node also find the nearest neighbor nodes and generates the routing path for each and every sensor node. These developments are will finally improve the life time and drain the sensor node cost of replacement. Next concept of genetic algorithm developed from the M. Gen and R. Cheng, [8]. This genetic algorithm is used to generate an efficient node from the dead nodes

2.1. Directed Diffusion Algorithm

In recent years in order to make WSN efficient, several algorithms have been proposed. C. Intanagonwiwatet. al. presented the Directed Diffusion(DD) algorithm. Directed Diffusion is designed for robustness, scaling and energy efficiency. It is data centric. Directed diffusion consists of several elements: interests, data messages, gradients, and reinforcements [9]. The DD algorithm is a query-driven transmission protocol. The collected data are transmitted only if they fit the query from the sink node, thereby reducing the power consumption from data transmission. First, the sink node provides interested queries in the form of attribute-value pairs to the other sensor nodes by broadcasting the interested query packets to the entire network. Subsequently, the sensor nodes only send the collected data back to the sink node if they fit the interested queries. In DD, all of the sensor nodes are bound to a route when broadcasting the interested queries, even if the route is such that it will never be used. In addition, several circle routes, which are built simultaneously when broadcasting the queries, result in wasted power consumption and storage. In the real world, the number of the sensor nodes in a system is in the hundreds or even thousands. Such a waste of power consumption and storage becomes worse and the circle route problem becomes more serious with larger-sized systems.

2.2. Ladder Diffusion Algorithm

Ladder diffusion (LD) algorithm is used to map out the data relay routes in wireless sensor nodes. The algorithm focuses on balancing the data transmission load, increasing the lifetime of sensor nodes and their transmission efficiency. This study evaluates the performance of this algorithm for random wireless sensor networks with regard to the number of sensor nodes and relay hops required for data collection. The ladder diffusion algorithm is used to identify routes from sensor nodes to the sink node and avoid the generation of circle routes using the ladder diffusion process. The LD algorithm is fast and completely creates the ladder table in each sensor node based on the entire wireless sensor network by issuing the ladder create packet that is created from the sink node[7],[8].

2.3. Grade Diffusion

In 2011, H. C. Shih et al.[12]proposed a Grade Diffusion algorithm that improves upon the LD-ACO algorithm[4] to enhance node lifetime, raise transmission efficiency, and solve the problem of node routing and energy consumption. The GD algorithm broadcast grade completely and quickly creates packages from the sink node to every node in the wireless sensor network by the LD-ACO algorithm. The GD algorithm increases the sensor node's lift time and the sensor node's transmission effect. Moreover, the sensor node can save some backup nodes to reduce the energy for the re-looking routing by GD algorithm in case the sensor node's routing is broken. Finally, the grade diffusion algorithm has the less data package transmission loss and the hop count than the tradition algorithms in our simulate setting.

The Grade Diffusion algorithm overcomes the disadvantages of Direct Diffusion algorithm by broadcasting the neighbors to only first neighbor set. After that nodes are picked up based on hop count or rules and the amount of RREQ exchange is reduced hence amount of power required is less as compared to Direct Diffusion. However problem still persist as the number of routes discovered increases the battery power decreases and node becomes obsolete sooner. Whether the DD or the GD algorithm is applied, the grade creating packages or interested query packets must first be broadcast. Then, the sensor nodes transfer the event data to the sink node, according to the algorithm, when suitable events occur. Grade Diffusion algorithm will discuss in chapter 3.

Chapter 3

Grade Diffusion Algorithm

The Grade Diffusion (GD) algorithm [14] is used for improving the ladder diffusion algorithm with the help of ant colony optimization (LD-ACO) for wireless sensor networks [15]. The GD algorithm creates the routing for each sensor node and identifies a set of neighbor nodes for minimizing the transmission loading. Each sensor node can select a sensor node from the set of neighbor nodes when its grade table lacks a node able to perform the relay. The GD algorithm can also record some information regarding the data relay. Then, a sensor node can select a node with a lighter loading or more available energy than the other nodes to perform the extra relay operation. That is, the GD algorithm updates the routing path in real time, and the event data is thus sent to the sink node quickly and correctly. The Grade Diffusion algorithm overcomes the disadvantages of Direct Diffusion algorithm by broadcasting the neighbors to only first neighbor set. After that nodes are picked up based on hop count or rules and the amount of RREQ exchange is reduced hence amount of power required is less as compared to Direct Diffusion. However problem still persist as the number of routes discovered increases the battery power decreases and node becomes obsolete sooner.

3.1 Plotting node:

At the beginning of executing GD algorithm we have to plot sensor nodes. For plotting node we maintain few conditions. We assume 1st nodes will be the sink node. Every information will come here. Then we plot nodes in circular way. When we plotting nodes in every circle we maintain few conditions, like it is keeping Euclidean distance between any nodes in that graph and maximum nodes will be three in every circle. Maintain this condition we plotting nodes randomly.

Experimental Dataset:

Row: 500, Col: 500, Center(Sink node):(100,50), Node:3,minED:17,TotalPlotted Node: 542
.

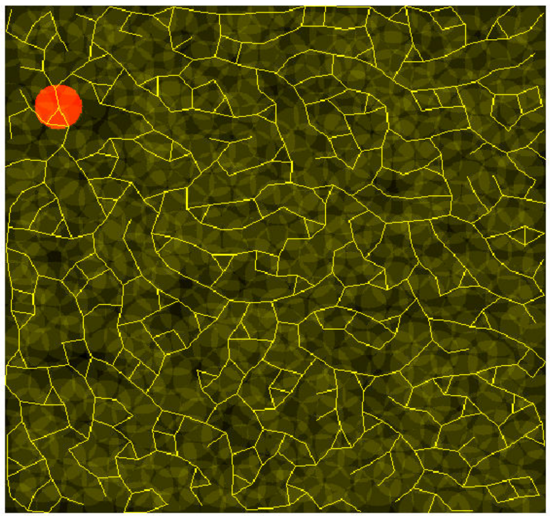

Figure 1: Simulated Area of Plotting Node

3.2 Determine Grade:

After plotting nodes we give grade in every node. When we give the grade for every node we maintained some conditions. Every child nodes grade must greater then parents nodes grade.we are incrementing child nodes grade by one. If parent node grade is one then child grade will be two. After grading all nodes we generate adjacency list.

3.3 All Possible Path:

After plotting and determining grade we calculate all possible path for a node to send information to the sink node. We are taking the record of how many path are exits for each node. And we set initial battery life for every node. Now we are putting all the information of every node in a knowledgebase. Each node containing these information:

- Node number
- Coordinate of the node
- Grade of the node
- How many path exist for the node
- Battery life

Chapter 4

Quantum-inspired Evolutionary Algorithm

4.1 Overview of Quantum-inspired Evolutionary Algorithm

This chapter concerns a new class of meta-heuristics, drawing their inspiration from two fields: evolutionary algorithm and quantum computing.

Quantum computing [13] is a branch of computer science concerning applications of unique quantum mechanical effects to solve computational problems. In theory, by application of unique quantum mechanical phenomena it is possible to solve selected computational problems extremely efficiently [14, 15]. However, there are still several serious difficulties in building a functional and scalable quantum computer. It is possible to simulate quantum computation on a classical computer; yet it is highly inefficient and computationally exhaustive. Nevertheless, quantum computing remains a valuable source of inspiration for contemporary computer science.

On the other hand, Evolutionary algorithms (EAs) are based on the principles of natural biological evolution. It is a generic population-based meta-heuristic optimization algorithm. Compared to traditional optimization methods, such as calculus-based and enumerative strategies, EAs are robust, global and may be applied generally without recourse to domain-specific heuristics, although performance is affected by these heuristics. At each generation of the EA, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and reproducing them using variation operators. This process may lead to the evolution of populations of individuals that are better suited to their environment than the individuals from which they were created, just as in natural adaptation. Genetic algorithms (GAs) [16, 17, 18], evolutionary programming (EP) [19] and evolution strategies (ES) [20] are the three main-stream methods of evolutionary computation.

This paper proposes a novel Evolutionary Algorithm, called a Quantum-inspired Evolutionary Algorithm (QEA), which is based on the concept and principles of quantum computing such as a quantum bit and superposition of states. Like the EAs, QEAis also characterized by the representation of the individual, the evaluation function, and the population dynamics. However, instead of binary, numeric, or symbolic representation, QEAuses a Q-bit as a probabilistic representation, defined as the smallest unit of information. A Q-bit individual is defined by a string of Q-bits. The Q-bit individual has the advantage that it can represent a linear superposition of states (binary solutions) in search space probabilistically. Thus, the Q-bit representation has a better characteristic of population diversity than other representations. A Q-gate is also defined as a variation operator of QEA to drive the individuals toward better solutions and eventually toward a single state. Initially, QEA can represent diverse individuals probabilistically because a Q-bit individual represents the linear superposition of all possible states with the same probability. As the probability of each Q-bit approaches either 1 or 0 by the Q-gate, the Q-bit individual converges to a single state and the diversity property disappears gradually. By this inherent mechanism, QEA can treat the balance between exploration and exploitation. It should be noted that although QEA is based on the concept of quantum computing, QEA is not a quantum algorithm, but a novel evolutionary algorithm for a classical computer [21, 22].

4.2 Basics of Quantum Computing

The smallest unit of information stored in a two-state quantum computer is called a quantum bit or qubit [13]. A qubit may be in the "1" state, in the "0" state, or in any superposition of the two. The state of a qubit can be represented as

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \qquad (1)$$

Where, αandβ are complex numbers that specify the probability amplitudes of the corresponding states.The$|\alpha|^2$and$|\beta|^2$ are probabilities of the quantum bit being in the "0" state and "1" state, respectively, and should satisfy the condition:

$$|\alpha|^2 + |\beta|^2 = 1 \qquad (2)$$

The state of a qubit can be changed by the operation with a quantum gate. A quantum gate is a reversible gate and can be represented as a unitary operator Uacting on the qubit basisstates satisfying$\mathsf{U}^{\boldsymbol{T}}\mathsf{U} = \mathsf{U}\mathsf{U}^{\boldsymbol{T}}$, where $\mathsf{U}^{\boldsymbol{T}}$ is the hermitian adjoint of U. There are several quantum gates, such as the NOT gate, controlled NOT gate, rotation gate, Hadamard gate, etc. [13]. If there is a system of$\boldsymbol{m}$qubits, the system can represent $\boldsymbol{2^m}$states at the same time. However, in the act of observing a quantum state, it collapses to a single state. Inspired by the concept of quantum computing, QEA is designed with a novel Q-bit representation, a Q-gate as a variation operator, and an observation process.

4.3 Representation

A number of different representations can be used to encode the solutions onto individuals in evolutionary computation. The representations can be classified broadly as: binary, numeric, and symbolic [23]. QEA uses a representation, called a Q-bit, for the probabilistic representation that is based on the concept of qubits, and a Q-bit individual as a string of Q-bits, which are defined below.

Definition 1: A *Q-bit* is defined as the smallest unit of information in QEA, which is defined with a pair of numbers $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ as

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Where $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha|^2$ gives the probability that the Q-bit will be found in the "0" state and $|\beta|^2$ gives the probability thatthe Q-bit will be found in the "1" state.A Q-bit may be in the "1" state, in the "0" state, or in a linear superposition of the two.

Definition 2:A *Q-bit individual* as a string of ***m***Q-bits is defined as

$$\left[\begin{array}{c|c|c} \alpha_1 & \alpha_2 & \ldots\ldots \alpha_m \\ \beta_1 & \beta_2 & \ldots\ldots \beta_m \end{array}\right] \qquad (3)$$

Where,$|\alpha_i|^2 + |\beta_i|^2 = 1,\ i = 1, 2\ 3, \ldots, m$.

Here is a three-Q-bit system with three pairs of amplitudes,

$$\left[\begin{array}{c|c|c} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & \frac{\sqrt{3}}{2} \end{array}\right] \qquad (4)$$

Then the states of the system can be represented as

$$\frac{1}{4}|000\rangle + \frac{\sqrt{3}}{4}|001\rangle - \frac{1}{4}|010\rangle - \frac{\sqrt{3}}{4}|011\rangle + \frac{1}{4}|100\rangle + \frac{\sqrt{3}}{4}|101\rangle - \frac{1}{4}|110\rangle - \frac{\sqrt{3}}{4}|111\rangle \qquad (5)$$

The above result means that the probabilities to represent the states $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$, $|100\rangle$, $|101\rangle$, $|110\rangle$ *and* $|111\rangle$are 1/16, 3/16, 1/16, 3/16, 1/16,3/16, 1/16, and 3/16 respectively. By consequence, the three-Q-bit system of (4) contains the information of eight states.

Evolutionary computing with Q-bit representation has a better characteristic of population diversity than other representations, since it can represent linear superposition of states probabilistically. Only one Q-bit individual such as (4) is enough to represent eight states, but in binary representation at least eight strings, (000), (001), (010), (011), (100), (101), (110), and (111) are needed.

4.4 The structure of QEA

The pseudo-code of QEA [21, 22] is given below.

Algorithm 1: Basic structure of Quantum-inspired Evolutionary Algorithm.

```
Procedure QEA
01.     begin
02.     t←0
03.     initialize Q(t)
04.     make P(t) by observing the states of Q(t)
05.     evaluate P(t)
06.     store the best solutions among P(t) into B(t)
07.     while(not termination-condition) do
08.     begin
09.     t ← t + 1
10.     make P(t) by observing the states of Q(t - 1)
11.     evaluate P(t)
12.     update Q(t) using Q-gates
13.     store the best solutions among B(t - 1) and P(t) into B(t)
14.     store the best solution b among B(t)
15.     if(migration-condition)
16.     then migrate b or $b_j^t$ to B(t) globally or locally, respectively
17.     end
18.     end
```

In **line 03** Q-bits of all individualsin the quantum population **Q(0)**are initialized with$\frac{1}{\sqrt{2}}$. It means that one Q-bit individual, represents the linear superposition of all possible states withthe same probability.

In **line 04**, this step makes binary solutions in **P(0)** by observing thestates of **Q(0)**, where **P(0)** is population of binary strings at **t=0**. Binary string is formed by selecting either 0 or1 for each bit using the probability, either $|\alpha^0|^2$or$|\beta^0|^2$. In a quantum computer, in the act of observing a quantum state, it collapses to a single state. However, collapsing into a single state does not occur in QEA, since it is working on a classical computer.

In **line 05**, each binary solution is evaluated to give a level of its fitness.

The initial best solutions are then selected among the binary solutions in **line 06**.

In the **while** loop, binary solutions in **P(t)** are formed by observing the states of **Q(t-1)(line 10)**and each binary solution is evaluated for the fitness value**(line 11)**.

In **line 12**, Q-bit individuals in Q(t) are updated by applying Q-gates defined below.

Definition 3:A *Q-gate* is defined as a variation operator of QEA, by which operation the updated Q-bit should satisfy the normalization condition, $|\alpha'|^2 + |\beta'|^2 = 1$, where $\alpha'\ and\ \beta'$are the values of the updated Q-bit.

The following rotation gate is a basic Q-gate,

$$U(\Delta\theta_i) = \begin{bmatrix} cos(\Delta\theta_i) & -sin(\Delta\theta_i) \\ sin(\Delta\theta_i) & cos(\Delta\theta_i) \end{bmatrix} \qquad (6)$$

Where,$\Delta\theta_i, i = 1, 2, \dots, m,$ is a rotation angle of each Q-bit toward either 0 or 1 state depending on its sign.$\Delta\theta_i$shouldbe designed in compliance with the application problem. It should be noted that NOT gate, controlledNOT gate, or Hadamard gate can be used asa Q-gate. NOT gate changes the probability of the1 (or 0) state to that of the 0 (or 1) state. It can beused to escape a local optimum. In controlled NOTgate, one of the two bits should be a control bit. Ifthe control bit is 1, the NOT operation is applied tothe other bit. It can be used for the problems whichhave a large dependency of two bits. Hadamardgateis suitable for the algorithms which use the phase informationof Q-bit, as well as the amplitude information,and H_ϵ gate which is a novel Q-gate as a variation operator is designed in section 3.5.

In **lines 13 and 14,** The best solutions among **B(t – 1)** and **P(t)** are selected and stored into **B(t)**, and if the best solution stored in **B(t)** is fitter than the stored best solution **b**, the stored solution **b** is replaced by the new one.

If a migration condition is satisfied**(line 15)**, the best solution **b** is migrated to **B(t)**, or the best one among some of the solutions in **B(t)** is migrated to them. The migration condition is a design parameter, and the migration process defined below can induce a variation of the probabilities of a Q-bit individual.

Definition 4:A migration in QEA is defined as the process of copying $\mathbf{b}_j^t$in **B(t)** or **b** to **B(t)**. A global migration is implemented by replacing all the solutions in **B(t)**by **b**, and a localmigration is implemented by replacing some of the solutions in**B(t)**by the best one of them.

The binary solutions in **P(t)** are discarded at the end of the loop because **P(t + 1)** will be produced by observing the updated **Q(t)** in step **7**. Until the termination condition is satisfied, QEA is running in the **while** loop.

4.5 H_ϵ Gate

The rotation gate used as a Q-gate induces the convergence of each Q-bit to either 0 or 1. However, a Q-bit converged to either 0 or 1 cannot escape the state by itself, although it can be changed passively by a global or local migration. If the value of $|\beta|^2$is 0 (or 1), the observing state of the Q-bit is always 0 (or 1). To prevent premature convergence of Q-bit, H_ϵ gate can be defined as a Q-gate.

Definition 3*:* AnH_ϵ*gate* is defined as a Q-gate extended from the rotation gate

$$[\alpha_i'\beta_i']^T = H_\epsilon(\alpha_i, \beta_i, \Delta\theta_i) \qquad (7)$$

Where for $[\alpha_i''\beta_i'']^T = U(\Delta\theta_i)[\alpha_i'\beta_i']^T$:

i) if$|\alpha_i''|^2 \leq \epsilon \ and\ |\beta_i''|^2 \geq 1-\epsilon$

$$[\alpha_i'\beta_i'] = [\sqrt{\epsilon}\sqrt{1-\epsilon}]^T;$$

ii) if$|\alpha_i''|^2 \geq 1-\epsilon \ and\ |\beta_i''|^2 \leq \epsilon$

$$[\alpha_i'\beta_i'] = [\sqrt{1-\epsilon}\sqrt{\epsilon}]^T;$$

iii) otherwise

$$[\alpha_i'\beta_i'] = [\alpha_i''\beta_i''];$$

Where, $0 < \epsilon \ll 1$, $U(\Delta\theta_i)$is the rotation gate, and $\Delta\theta_i, i = 1, 2, \ldots, m,$ is the rotation angle of each Q-bit toward either 0 or 1 state, depending on its sign. **Figure 3** shows the $\boldsymbol{H_\epsilon}$gate, where $\boldsymbol{lim_{\epsilon\to 0} H_\epsilon}(.)$is the same as the rotation gate. While the rotation gate makes the probabilityof $|\alpha|^2\ or\ |\beta|^2$ converge to either 0 or 1,$\boldsymbol{H_\epsilon}$gate makes it converge toϵor $(1 - \epsilon)$. It should be noted that if $\epsilon$is too big, the convergence tendency of a Q-bit individual may disappear.

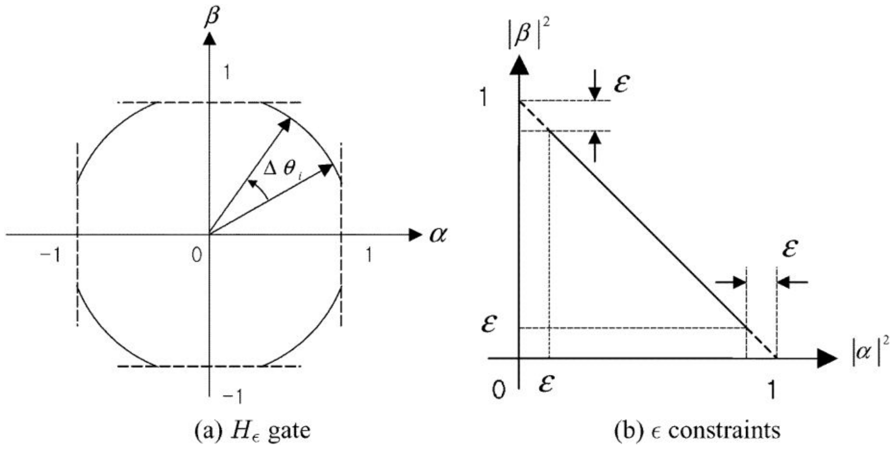

Figure 3:$\boldsymbol{H_\epsilon}$ gate based on rotation gate.

4.6 Application of Quantum-inspired Evolutionary Algorithm

This section will provide an overview of the numerous applications of QEAs. Traditional NP-Optimization problems are one of the most typical application fields of QEA.

Recent years have witness successful applications of QEA in variety of different areas, including image processing [11, 12, 13, 14], network design problems[15, 16, 17], flow shop scheduling problems [18]thermal unit commitment, power system optimization ,discovering structures in time series etc.

QEA with real numbers representation have been also successfully applied in various fields: engineering optimization problems option pricing modeling [19,20], power system optimization financial data analysis[21], training fuzzy neural networks and ceramic grinding optimization.Consequently, it has been demonstrated that Quantum-inspired Evolutionary Algorithms are capable to outperform classical metaheuristics for a wide range of problems.

4.7 Proposed QEA for Fault node recovery problem

Here, we are going to start our discussion with the proposed representation and We present the detailed algorithm of QEA for the node recovery problem.

Algorithm 2: Proposed QEA for Fault node recovery problem

Procedure QEA for the Fault node:

```
01: begin
02:     t ← 0
03:     initialize Q(t)
04:     make P(t) by observing the states of Q(t)
05:     repair P(t)
06:     improve P(t)
07:     evaluate P(t)
08:     store the best solutions among P(t) into B(t)
09:    while (t < MAX GEN) do
10:     begin
11:         t ← t + 1
12:         make P(t) by observing the states of Q(t - 1)
13:         repair P(t)
14:         improve P(t)
15:         evaluate P(t)
16:         update Q(t)
17:         store the best solutions among B(t - 1) and P(t) into B(t).
18:         store the best solution b among B(t).
19:         if (migration-period)
20:        then migrate b or $b_j^t$ to B(t) globally or locally,
           respectively
```

```
21:    end
22: end
```

Procedure Make (x)

```
01: begin
02:    i← 0
03:    while (i< m) do
04:    begin
05:        i← i + 1
06:        if (random[0, 1) <$|\beta_i|^2$)
07:            then $x_i$ ← 1
08:            else $x_i$ ← 0
09:    end
10: end
```

Procedure Repair (x)

```
01: begin
02:    for (i=0 to m)do
03:    begin
04:      if ($x_i$ is 1) then
05:      begin
06:        for (j=0 to m) do
07:        begin
08:          if ($x_j$ is 1)then
09:          begin
```

```
10:             if ($i$is not equal $j$and $x_i$ is not adjacent to $x_j$) then
11:             begin
12: randomly choose $i$ or $j$.
13:               if ($i$ is choosen) then
14:                 $x_i \leftarrow 0$
15:               else
16:                 $x_j \leftarrow 0$
17:             end
18:         end
19:        end
20:      end
21:    end
22: end
```

Here, H_ϵ gate is used as a Q-gate to update a Q-bit individual $\boldsymbol{q}$as a main variation operator. Q-bit of ***i***th position (α_i, β_i) is updated as follows:

$$[\alpha_i'' \beta_i'']^T = U(\Delta\theta_i)[\alpha_i \beta_i]^T \ :$$

Where, $U(\Delta\theta_i)$ is a simple rotation gate,

$$U(\Delta\theta_i) = \begin{bmatrix} cos(\Delta\theta_i) & -sin(\Delta\theta_i) \\ sin(\Delta\theta_i) & cos(\Delta\theta_i) \end{bmatrix}$$

if$|\alpha_i''|^2 \leq \epsilon \ and \ |\beta_i''|^2 \geq 1 - \epsilon$

then $[\alpha_i' \beta_i'] = [\sqrt{\epsilon}\sqrt{1-\epsilon}]^T$;

if$|\alpha_i''|^2 \geq 1 - \epsilon \ and \ |\beta_i''|^2 \leq \epsilon$

then $[\alpha_i' \beta_i'] = [\sqrt{1-\epsilon}\sqrt{\epsilon}]^T$;

Otherwise, $[\alpha_i' \beta_i'] = [\alpha_i'' \beta_i'']$;

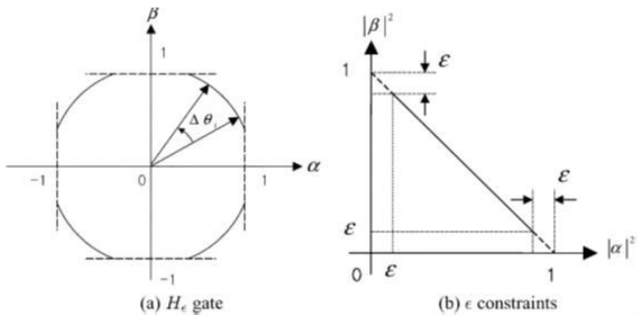

Figure 2: H_ϵ gate based on rotation gate.

In this QEA for MCP, the angle parameters used for the rotation gate are shown in table 1. Let us define an angle vector$\Theta = [\theta_1\theta_2 \dots \theta_8]^T$, where $\theta_1, \theta_2, \dots, \theta_8$ can be found from table 1. We have used, $\theta_3 = 0.01\pi, \theta_5 = -0.01\pi,$ and 0 for the rest. The values from 0.001π to 0.05π are recommended for the magnitude of $\Delta\theta_i$. Otherwise, it may converse prematurely. The sign of $\Delta\theta_i$determines the direction of convergence. We have chosen $\epsilon = 0.01$. In table 1, $f(.)$ is the fitness, and $x_i$and b_i are the ith bits of the best solution x and the binary solution b, respectively. Here, $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 0.01\pi$, $\theta_4 = 0, \theta_5 = -0.01\pi$, $\theta_6 = 0$, $\theta_7 = 0$, $\theta_8 = 0$ are used.

Table 1: lookup table of $\Delta\theta_i$

x_i	b_i	$f(x) \geq f(b)$	$\Delta\theta_i$
0	0	$false$	θ_1
0	0	$true$	θ_2
0	1	$false$	θ_3
0	1	$true$	θ_4
1	0	$false$	θ_5
1	0	$true$	θ_6
1	1	$false$	θ_7
1	1	$true$	θ_8

Chapter 5

Experimental Results and Analysis

5.1 Environment Setup:

A simulation of the proposed algorithm as described in chapter 3 and chapter 4 was performed to verify the method. The experiment was designed based on 2-D space, using 500*500 units, and the scale of the coordinate axis for each dimension was set at 0 to 500. The radio ranges (transmission range) of the nodes were set to 22 units. In each of these simulations, the sensor nodes were distributed uniformly over the space. There are three sensor nodes randomly distributed in 500*500 space, and the Euclidean distance is 17 units between any two sensor nodes. Therefore, there are 542 sensor nodes in the 2-D wireless sensor network and the center node is the sink node. The data packages were exchanged between random source/destination pairs with 1000 event data packages.

For simulation:

IDE	: Code::Blocks 13.12 and Mathlab (2012b)
Language	: C++

5.2 Experimental Data sets Results

Dataset for environment-

Row: 500

Column: 500

Center (Sink Node): 100, 50

Radius: 22

Maximum node: 3

Minimum Distance: 17

Result:

Total node: 542

Dataset for simulation:

Maximum iteration: 2000

Event: 1000

Optimum value: 30

Message per deduction: 2w

Result:

Success: 912 (250 QEA call)

In figure -3 we can see that the number of message reached for each QEA call. Before QEA call maximum messages send in the sink node. After every calling the rest of the messages were sending gradually.

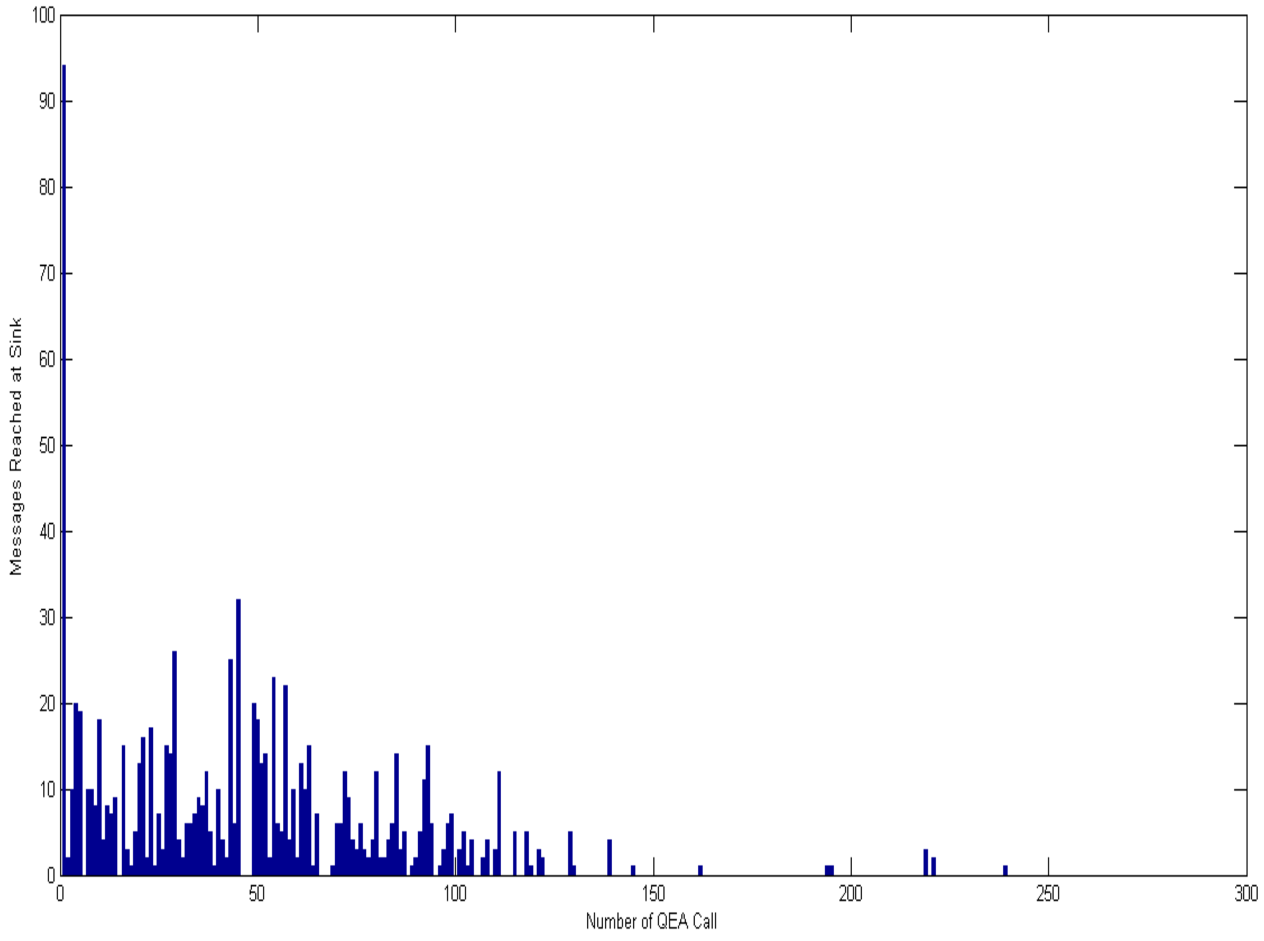

Figure 3: Total messages reaching the sink node for each QEA call

But we also see that after 147 QEA call maximum messages ware sent. Some messages are hanged up in different nodes. After some QEA call that messages also sent into the sink node.

In figure 4 we can see that the recovered sensor nodes after each QEA call. In this graph we able to figure it out that after 1st QEA call the total number of recovered nodes are 22.Thus this graph shows up the amount of recovered node after each QEA call.

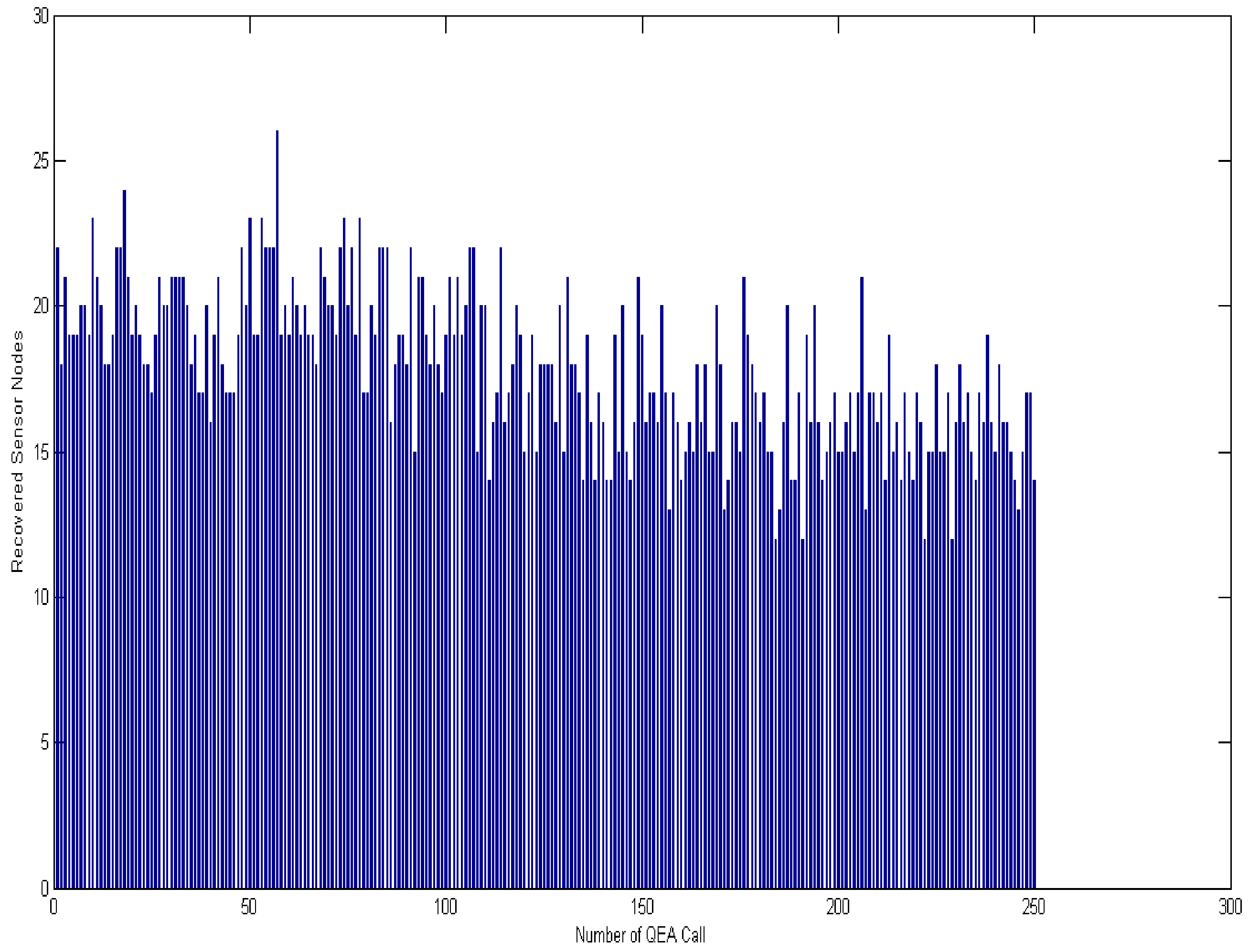

Figure 4: Number of recovery sensor nodes in each QEA call

In Fig. 5, the average energy consumption for each grade is calculated after 1000 events. Using the GD algorithm, the sensor nodes consume their energy rapidly because they try to transfer event data to the sink node using neighbor nodes if the grade 1 sensor nodes are energy-depleted or their routing table is empty. The proposed algorithm has ample energy for each grade sensor node because the algorithm can replace the sensor nodes, but it reuses more routing paths compared to using the traditional algorithm.

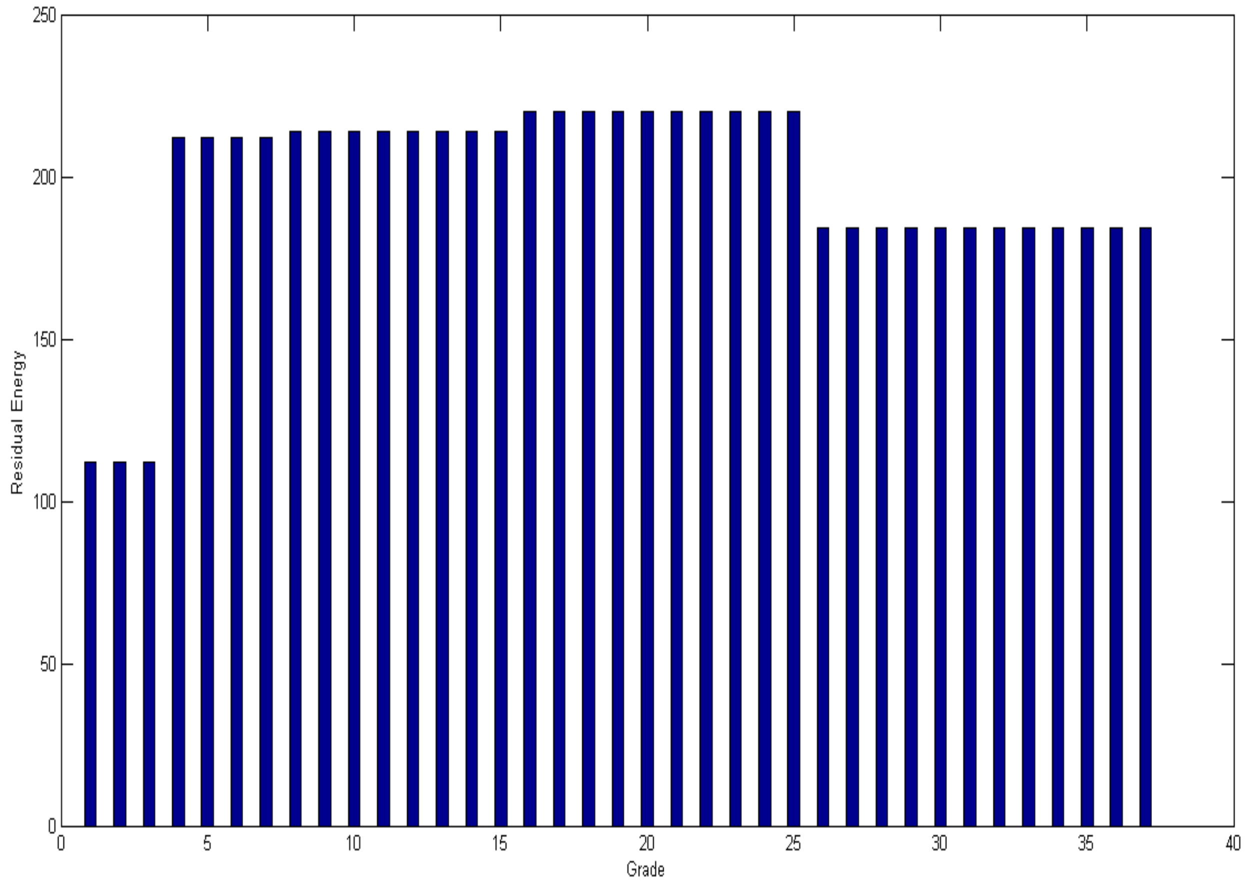

Figure 5: Average residual energy

Chapter 6

Conclusion and Future Work:

Wireless sensor network (WSN) consists of spatially dispersed and dedicated sensors for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location. In most of the case wireless sensor network (WSN) measure environmental conditions such as temperature, sound, pollution levels, humidity, wind speed and direction, pressure, etc. In this paper we applied fault sensor node recovery algorithm to increase the life time of a WSN when few sensors are damaged or shut down due to technical problem or low level of battery power. Here we proposed a algorithm is based on the Grade diffusion algorithm combined with the quantum inspired evolutionary algorithm for solution of fault node recovery problem.

In the future work, we will try to improve the execution time of the algorithm. We also have plans to test on bigger data sets and compare different approach like DD with this approach. In this paper we use battery as only constrain. But we are also interested to implement this approach with different constrain like distance, node overhead etc.

References

[1] F. C. Chang and H. C. Huang, "A refactoring method for cache-efficient swarm intelligence algorithms," Inf. Sci., vol. 192, no. 1, pp. 39–49,Jun. 2012.

[2] S. Corson and J. Macker, Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. New York,NY, USA: ACM, 1999.

[3] M. Gen and R. Cheng, Genetic Algorithms and Engineering Design. New York, NY, USA: Wiley, 1997.

[4] Z. He, B. S. Lee, and X. S. Wang, "Aggregation in sensor networks with a user-provided quality of service goal," Inf. Sci., vol. 178, no. 9, pp. 2128–2149, 2008.

[5] T. P. Hong and C. H. Wu, "An improved weighted clustering algorithm for determination of application nodes in heterogeneous sensor networks," J. Inf. Hiding Multimedia Signal Process., vol. 2, no. 2,pp. 173–184, 2011.no. 4, pp. 387–401, 2008.

[6] T. H. Liu, S. C. Yi, and X. W. Wang, "A fault management protocol for low-energy and efficient wireless sensor networks," J. Inf. Hiding Multimedia Signal Process., vol. 4, no. 1, pp. 34–45, 2013.

[7] E. M. Royer and C. K. Toh, "A review of current routing protocols for ad-hoc mobile networks," IEEE Personal Commun., vol. 6, no. 2,
pp. 46–55, Apr. 1999.

[8] J. H. Ho, H. C. Shih, B. Y. Liao, and S. C. Chu, "A ladder diffusion algorithm using ant colony optimization for wireless sensor networks," Inf. Sci., vol. 192, pp. 204–212, Jun. 2012.

[9] T. Hey, "Quantum computing: An introduction," in Computing & Control Engineering Journal. Piscataway, NJ: IEEE Press, June 1999, vol.10, no. 3, pp. 105–112.

[10] Yu, Xinjie, Gen, Mitsuo: Introduction to Evolutionary Algorithms, 2010, XVII, 418p. 168 illus, ISBN 978-1-84996-129-5.

[11] Fu, X., Ding, M., Zhou, C., Sun, Y.: Multi-threshold image segmentation with improved quantum-inspired genetic algorithm, in Proceedings of SPIE, volume 7495, 2009

[12] Jang, J., Han, K., Kim, J.: Quantum-inspired evolutionary algorithm-based face verification, Lecture Notes in Computer Science (2003)

[13] Talbi, H., Batouche, M., Draa, A.: A Quantum-Inspired Evolutionary Algorithm for Multiobjective Image Segmentation, International Journal of Mathematical, Physical and Engineering Sciences, Vol 1, pages 109-114, 2007.

[14] Talbi, H., Batouche, M., Draa, A.: A quantum-inspired genetic algorithm for multi-source affine image registration, Lecture Notes in Computer Science, 147-154, 2004

[15] Lin, D. Waller, S.: A quantum-inspired genetic algorithm for dynamic continuous network design problem, Transportation Letters: The International Journal of Transportation Research, Vol. 1, 2009

[16] Xing, H., Ji, Y., Bai, L., Liu, X., Qu, Z., Wang, X.: An adaptive evolution-based quantum-inspired evolutionary algorithm for QoS multicasting in IP/DWDM net-works, Computer Communications, Vol. 32, 2009

[17] Xing, H., Liu, X., Jin, X., Bai, L., Ji, Y.: A multi-granularity evolution based Quantum Genetic Algorithm for QoS multicast routing problem in WDM networks, Computer Communications, Vol. 32, 2009

[18] Fan, K., O'Sullivan, C., Brabazon, A., O'Neill, M., McGarraghy, S.: Calibration of the VGSSD option pricing model using a quantum-inspired evolutionary algorithm, Quantum Inspired Intelligent Systems, Springer Verlag, 2008.

[19] Al-Othman, A., Al-Fares, F., EL-Nagger, K.: Power System Security Constrained Economic Dispatch Using Real Coded Quantum Inspired Evolution Algorithm, International Journal of Electrical, Computer, and Systems Engineering Vol. 1, 2007

[20] de Pinho, A., Vellasco, M., da Cruz, A., A new model for credit approval problems: A quantum-inspired neuro-evolutionary algorithm with binary-real representation, Nature Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on, pages 445-450, 2009

[21] Fan, K., Brabazon, A., O'Sullivan, C., O'Neill,M.: Quantum-Inspired Evolutionary Algorithms for Financial Data Analysis, in EvoWorkshops, pages 133-143, 2008

Appendix

CREATE SQUARE AREA

```
int min(int x, int radius)
{
    if(x-radius<0)  return 0;
    else return x-radius;
}

int max(int x, int radius, int y)
{
    if(x+radius>y) return y;
    else return x+radius;
}
```

CHECK CIRCLE

```
int checkCircle(int x,int y, int m, int n, int radius)
{
    int a,b;

    a = ((m-x)*(m-x)) + ((n-y)*(n-y));
    b = radius*radius;

    if(a>=b) return 0;                          // outside circle
    else if(a==0) return 0;                     // center itself
    else return 1;                              // inside circle
}
```

CHECK EUCLIDEAN DISTANCE

```
int checkEuclideanDist(int x2, int y2,int minDist)
{
    int dist,x1,y1,f;
    for(int i=0; i<plotedNode.size(); i++)                          //Check with every ploted node
    {
            x1 = plotedNode[i].x;
            y1 = plotedNode[i].y;
            dist = sqrt(((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1)));

            if(dist<=minDist) {f=0; break;}
            else f=1;
    }
    return f;
}
```

CHECK EXISTED NODE

```
int checkExistedNode(int x,int y,int rowMin,int rowMax,int colMin,int colMax,int radius)
{
    int count=0,f;
    for(int i=rowMin; i<=rowMax; i++)
    {
        for(int j=colMin; j<=colMax; j++)
        {
            if(net[i][j]>0)
            {
                f = checkCircle(x,y,i,j,radius);
                if(f==1)                                    //node found circle
                {
                    count++;
                }
            }
        }
    }
    return count;
}
```

CHECK EVERY POINT

```
int checkEveryPoint(int m, int n,int radius, int nodes,int row,int col)
{
    int x,y,rowMin,rowMax,colMin,colMax,count,d=1;
    net[m][n]=99999999;

    for(int i=0; i<plotedNode.size(); i++)
    {
         x=plotedNode[i].x;
         y=plotedNode[i].y;

         rowMin = min(x,radius);
         rowMax = max(x,radius,row);
         colMin = min(y,radius);
         colMax = max(y,radius,col);

        //cout<<"rowMin: "<<rowMin<<" rowMax: "<<rowMax<<" colMin: "<<colMin<<" colMax: 
"<<colMax<<endl;
        count = checkExistedNode(x,y,rowMin,rowMax,colMin,colMax,radius);

        if(count>nodes)
        {
            d=0;
            break;
        }
    }
    net[m][n]=0;

    return d;
}
```

ADJACENCY LIST

```
int adjListCreate(int radius)
{
    int i, j;
    for(i=0; i<plotedNode.size(); i++)
    {
        for(j=i+1; j<plotedNode.size(); j++)
        {
            if(i==j) continue;
            if(checkCircle(plotedNode[i].x, plotedNode[i].y, plotedNode[j].x, plotedNode[j].y,
radius))
            {
                adjList[i].push_back(j);
                //adjList[j].push_back(i);
            }
        }
    }
    return 0;
}
```

GRAPH PLOT

```
void graphPlot(int row,int col,int x,int y,int radius,int nodes,int minDist)
{
    int m,n,circle,dist,nodeLevel=1,g;
    int rowMin=0,rowMax=0,colMin=0,colMax=0;
    node pointP, pointC;

    rowQ.push(x);
    colQ.push(y);

    net[x][y]=nodeLevel;
    pointP.insert_xy(x, y);
    plotedNode.push_back(pointP);

    int faild=0;
    while(!rowQ.empty() && !colQ.empty())
    {

        x = rowQ.front();
        y = colQ.front();
        //cout<<"                                          popX="<<x<<" popY="<<y<<endl;
        rowQ.pop();
        colQ.pop();

        rowMin = min(x,radius);
        rowMax = max(x,radius,row);
        colMin = min(y,radius);
        colMax = max(y,radius,col);

        //cout<<"rowMin: "<<rowMin<<" rowMax: "<<rowMax<<" colMin: "<<colMin<<" colMax: "<<colMax<<endl;

        int count = checkExistedNode(x,y,rowMin,rowMax,colMin,colMax,radius); // count existed nodes on that area

        cout<<"No Of nodes already exist: "<<count<<endl;

        if(count>nodes)
        {
            printf("%a");
            system("pause");

        }

        int nodePlot=nodes-count;
        if(nodePlot<0) nodePlot=0;
        if(nodePlot>0) nodeLevel++;

        int trial=0;
       // cout<<"                    nodePlot = "<<nodePlot<<endl;

        while(nodePlot)
        {
            if(trial==500)
            {
                faild++;
                cout<<"Faild: "<<faild<<endl;
                break;
            }

            m = rand() % (rowMax-rowMin) + rowMin;
            n = rand() % (colMax-colMin) + colMin;

            circle = checkCircle(x,y,m,n,radius);                 //Check new nodes on circle or
    not

            if(circle==1) dist = checkEuclideanDist(m,n,minDist); // Check Euqlidean Distance

            int countPoint=0;
            if(circle==1 && dist==1) countPoint = checkEveryPoint(m,n,radius,nodes,row,col); //
    count nodes in every nodes circuler area after ploting new node

           // cout<<"m="<<m<<"n="<<n<<endl;
            if(net[m][n]==0 && countPoint==1)
            {
                //cout<<"m="<<m<<" n="<<n<<endl;
                net[m][n] = nodeLevel;

                //cout<<"                    pushX="<<m<<" pushY="<<n<<endl;
                rowQ.push(m);
                colQ.push(n);

                pointC.insert_xy(m, n);
                plotedNode.push_back(pointC);

                nodePlot--;
            }
            trial++;
        }
    }

    adjListCreate(radius);  // Calling adjacency list creator function
}

//********************** MAIN FUNCTION ***********************//

int main()
{
    int x,y,radius,nodes,row,col,minDist;

    cout<<"row: "; cin>>row;
    cout<<"col: "; cin>>col;
    cout<<"x: "; cin>>x;
    cout<<"y: "; cin>>y;
    cout<<"radius: "; cin>>radius;
    cout<<"nodes: "; cin>>nodes;
    cout<<"minDist: "; cin>>minDist;

    graphPlot(row,col,x,y,radius,nodes,minDist);

    cout<<"###################### Ploted Node #############################\n";
    for(int i = 0; i<plotedNode.size(); i++)
    {
        cout<<plotedNode[i].x<<" "<<plotedNode[i].y<<endl;
    }

```

```
        PLOTED NODE FILE
            FILE * pNf;
            pNf=fopen("PlotNodeFinal.txt","w");
            fprintf(pNf,"%d\n%d\n%d\n%d\n%d\n%d\n",row,col,x,y,radius,nodes);
            for(int i=0; i<plotedNode.size(); i++)
            {
                fprintf(pNf,"%d %d\n",plotedNode[i].x, plotedNode[i].y);
            }
            fclose(pNf);

            cout<<"#################### Adjacency List Print #######################\n";
            int edge=0;
            for(int i=0; i<adjList.size(); i++)
            {
                for(int j=0; j<adjList[i].size(); j++)
                {
                    cout<<i<<" "<<adjList[i].at(j)<<endl;
                    edge++;
                }
            }
        //################################## Adjacency List FILE
#####################################
            FILE * adj;
            adj=fopen("AdjacencyList.txt","w");

            fprintf(adj,"%d %d\n", plotedNode.size(),edge);
            for(int i=0; i<adjList.size(); i++)
            {
                for(int j=0; j<adjList[i].size(); j++)
                {
                    fprintf(adj,"%d %d\n",i,adjList[i].at(j));
                }
            }
            fclose(adj);
            cout<<"Finish\n";

            return 0;
        }
```

Determine Grade

```
#include<stdio.h>
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<queue>
#include<vector>
using namespace std;
int net[1005][1005];
int visited[1005][1005];
queue<int>rowQ;
queue<int>colQ;


struct node
{
    int x, y, grade;

    int insert_xy(int a, int b, int c)
    {
        x=a;
        y=b;
        grade=c;
    }
} point;
vector< node >plotedNode;
vector< vector< int > > adjList(100000);


//*********************** PRINT *******************************//
void print(int row,int col)
{
    for(int i=0; i<row; i++)
    {
        for(int j=0; j<col; j++)
        {
            cout<<net[i][j]<<" ";
        }
        cout<<endl;
    }
}

//****************** CREATE SQUARE AREA *******************//
int min(int x, int radius)
{
    if(x-radius<0)  return 0;
    else return x-radius;
}

int max(int x, int radius, int y)
{
    if(x+radius>y) return y;
    else return x+radius;
}

//**************************** CHECK CIRCLE *****************************//

int checkCircle(int x,int y, int m, int n, int radius)
{
    int a,b;

    a = ((m-x)*(m-x)) + ((n-y)*(n-y));
    b = radius*radius;

    //cout<<"rad: "<<a<<endl;
    if(a>=b) return 0;          // outside circle
    else if(a==0) return 0;     // center itself
    else return 1;             // inside circle
}
```

```

//********************** CHECK EXISTED NODE ***************************//

int checkExistedNode(int x,int y,int rowMin,int rowMax,int colMin,int colMax,int radius)
{
    int count=0,f;
    for(int i=rowMin; i<=rowMax; i++)
    {
        for(int j=colMin; j<=colMax; j++)
        {
            if(net[i][j]>0)
            {
                f = checkCircle(x,y,i,j,radius);
                if(f==1)  //node found circle
                {

                    count++;
                }
            }
        }
    }
    return count;
}

//********************** ADJACENCY LIST **************************//
int adjListCreate(int radius)
{
    int i, j;
    for(i=0; i<plotedNode.size(); i++)
    {
        for(j=i+1; j<plotedNode.size(); j++)
                {
                    if(i==j) continue;
                    if(checkCircle(plotedNode[i].x, plotedNode[i].y, plotedNode[j].x,
plotedNode[j].y, radius))
                    {
```

```
                    adjList[i].push_back(j);
                }
            }
        }
        return 0;
    }

    //*********************** GRADE *******************************//

    void grade(int row,int col,int x,int y,int radius,int node,int totalNode)
    {
        int rowMin=0,rowMax=0,colMin=0,colMax=0;
        int f=0,count=0;
        net[x][y]=1; //root node grade = 1


        point.insert_xy(x, y, 1);
        plotedNode.push_back(point);


        rowQ.push(x);
        colQ.push(y);
        int h=1;
        int c=1;
        while(!rowQ.empty() && !colQ.empty())
        {

             x = rowQ.front();
             y = colQ.front();

             rowQ.pop();
             colQ.pop();

             rowMin = min(x,radius);
             rowMax = max(x,radius,row);
             colMin = min(y,radius);
```

```
                 colMax = max(y,radius,col);

                for(int i=rowMin; i<=rowMax; i++)
                {
                    for(int j=colMin; j<=colMax; j++)
                    {
                        if(net[i][j]>0) // it is a node
                        {
                            f = checkCircle(x,y,i,j,radius);
                            if(f==1)                                    // this node found in circle
                            {

                                c++;
                                if(net[x][y]<net[i][j] && visited[i][j]==0)  // if adjacent node grade > node grade
                                {
                                    visited[i][j]=1;
                                    net[i][j]=net[x][y]+1; // adjacent node grade = node grade + 1

                                    h++;
                                    rowQ.push(i);
                                    colQ.push(j);

                                    point.insert_xy(i, j, net[i][j]);
                                    plotedNode.push_back(point);
                                }


                            }
                        }
                    }
                }
            }
            adjListCreate(radius);
```

```
          }

        }

        int main()
        {
            int x,y,radius,node,row,col,r1,c1,minDist,totalNode;

            cout<<"row: "; cin>>row;
            cout<<"col: "; cin>>col;
            cout<<"x: "; cin>>x;
            cout<<"y: "; cin>>y;
            cout<<"radius: "; cin>>radius;
            cout<<"node: "; cin>>node;
            cout<<"Total node: "; cin>>totalNode;
            cout<<"Take Input All Ploted Node: "<<endl;

            for(int i=0; i<totalNode; i++)
            {
                cin>>r1>>c1;
                net[r1][c1]=9999;
            }
            grade(row,col,x,y,radius,node,totalNode);

        //################################################ FILE ####################################

            FILE *fp;
            fp=fopen("GradeNode .txt","w");

            fprintf(fp,"row: %d\ncol: %d\nx: %d\ny: %d\nTotal Node: %d\n",row,col,x,y,totalNode);
            fprintf(fp,"N X Y G\n");
            for(int i=0; i<totalNode; i++)
            {
```

```
            fprintf(fp,"%d %d %d %d\n",i,plotedNode[i].x,plotedNode[i].y,plotedNode[i].grade);
        }
        fprintf(fp,"\n#######################################################################\n");
        fclose(fp);
        cout<<"Finish\n";

        cout<<"################### Adjacency List Print #######################\n";
        int edge=0;
        for(int i=0; i<adjList.size(); i++)
        {
            for(int j=0; j<adjList[i].size(); j++)
            {
                cout<<i<<" "<<adjList[i].at(j)<<endl;
                edge++;
            }
        }
    //################################## Adjacency List FILE #####################################
        FILE * adj;
        adj=fopen("AdjacencyListNew.txt","w");

        fprintf(adj,"%d %d\n",totalNode,edge);
        for(int i=0; i<adjList.size(); i++)
        {
            for(int j=0; j<adjList[i].size(); j++)
            {
                fprintf(adj,"%d %d\n",i,adjList[i].at(j));
            }
        }
        fprintf(adj,"######################################################################################");
        fclose(adj);

    //################################## CONNECTION LIST #####################################
        FILE * con;
        con=fopen("ConnectionList.txt","w");

        for(int i=0; i<adjList.size(); i++)
        {
            for(int j=0; j<adjList[i].size(); j++)
            {
                //if(i==j) continue;
                fprintf(adj,"%d %d %d %d\n",plotedNode[i].x,plotedNode[i].y,plotedNode[adjList[i][j]].x,plotedNode[adjList[i][j]].y);
            }
        }
        fclose(con);

    return 0;
    }
```

All Possible Paths

```
#include <stdio.h>
#include <vector>
#include <algorithm>
#include <vector>
#include <queue>
#include <iostream>
#include "sstream"
#include "fstream"
using namespace std;
vector<vector<int> >GRAPH(100000);


struct node
```

```
{
    int node, x, y, grade, pathCount, battry;
}point;
//typedef pair<int,int>node;
vector< node >NodeInfo;
stringstream ss;

//################################### PATH PRINT #######################
inline void print_path(vector<int>path)
{
    //cout<<"[ ";
    ss<<path.size()<<endl;
    for(int i=0;i<path.size();++i)
    {
        //cout<<path[i]<<" ";
        ss<<path[i]<<" ";
    }
    ss<<endl;
    //cout<<"]"<<endl;
}

//###################### ADJACENCY NODE EXIST ON PATH OR NOT #####################
bool isadjacency_node_not_present_in_current_path(int node,vector<int>path)
{
    for(int i=0;i<path.size();++i)
    {
        if(path[i]==node)
        return false;
    }
    return true;
}

/////////////////////////////////////////////////////////////////////////////////////
void routingTable(vector<int>new_path)
{
    FILE *fp;
    fp = fopen("routingTable.txt","a");

    for(int i=0; i<new_path.size(); i++)
    {
        fprintf(fp,"%d ",new_path[i]);
    }
    fprintf(fp,"\n");

    fclose(fp);

}

//################################### FIND PATH ###################################
int findpaths(int source ,int target ,int totalnode,int totaledge )
{
    vector<int>path; // PATH VECTOR INIITIALISATION
    path.push_back(source);
    queue<vector<int> >q; // QUEUE OF PATH VECTOR

    q.push(path);
    int countPath = 0; // PATH COUNT INITIALISATION

    while(!q.empty())
    {
        path=q.front();
        q.pop(); // POP PATH FORM QQUEUE

        int last_nodeof_path=path[path.size()-1]; // LAST NODE OF PATH VECTOR

        if(last_nodeof_path==target) // COMPLETE PATH BETWEEN SOURCE AND DESTINATION
        {
            //cout<<"The Required path is:: ";
            print_path(path);
            routingTable(path);
            countPath++; // PATH COUNT
        }
        else
        {
           // print_path(path);
        }

        for(int i=0;i<GRAPH[last_nodeof_path].size();++i) // GRAPH[last_nodeof_path].size() = SIZE OF ADJACENCY NODE
        {
           // IF GRADE OF LAST NODE > GRADE OF LAST NODES ADJACENCY NODE
            if(NodeInfo[last_nodeof_path].grade>NodeInfo[GRAPH[last_nodeof_path][i]].grade)
            {
                // send LAST UPDATED PATH and (GRAPH[last_nodeof_path][i] = NODES OF ADJACENCY NODE
                // IF NEW ADJACENCY NODE NOT EXIST ON UPADATED PATH THEN ADD NEW NODE AND UPDATE THE PATH

                if(isadjacency_node_not_present_in_current_path(GRAPH[last_nodeof_path][i],path))
                         {
                             vector<int>new_path(path.begin(),path.end());
                             new_path.push_back(GRAPH[last_nodeof_path][i]); // ADD NEW NODE TO PATH
                             q.push(new_path);                             // PUSH UPDATED PATH INTO QUEUE
                         }
                     }
                 }
             }

             return countPath;
         }
         int main()
         {
             int totalNode,totalEdge,u,v,source,target;

             printf("Enter Total Nodes & Total Edges\n");
             scanf("%d%d",&totalNode,&totalEdge);

         //################################ INPUT ADJACENCY LIST ########################

             printf("Enter Adjacency Lists\n");
             for(int i=1;i<=totalEdge;++i)
             {
                 scanf("%d%d",&u,&v);
                 GRAPH[u].push_back(v);
                 GRAPH[v].push_back(u);
             }

         //######################### INPUT ALL NODE INFORMATION ###########################

             int n,x,y,g;
             cout<<"Enter Node Information\nNode X Y Grade"<<endl;
             for(int i=0; i<totalNode; i++)
             {
                 scanf("%d%d%d%d",&n,&x,&y,&g);
                 point.node=n;
                 point.x=x;
                 point.y=y;
                 point.grade=g;
                 point.battry=100;
                 NodeInfo.push_back(point);
             }

             printf("Find and Count All Possible Path from All Node to Source\n");
             int pathCount[100000];
             ofstream fout;
             fout.open("routingTable.txt");
             string str;
             for(int i=0; i<totalNode; i++)
             {
                 source = i;
                 target = 0;
                 int c = findpaths(source,target,totalNode,totalEdge);
                 cout<<endl<<"Node:"<<i<<":: Path Count:: "<<c<<endl;
                 fout<<i<<" "<<c<<endl;
                 str = ss.str();
                 cout<<str;
                 fout<<str;
                 pathCount[i]=c;
                 ss.str("");
             }
             fout.close();


         //################################ CHECKING #############################################
             printf("Node X Y Grade pathCount Battry \n");
             for(int i=0; i<totalNode; i++)
             {
                 printf("%d %d %d %d %d %d\n",NodeInfo[i].node,NodeInfo[i].x,NodeInfo[i].y,NodeInfo[i].grade,pathCount[i],NodeInfo[i].battry);
             }



         //###################################### FILE ###########################################

             FILE *fp;
             fp=fopen("NodeInfo.txt","w");

             fprintf(fp,"Node X Y Grade pathCount Battry \n");
             for(int i=0; i<totalNode; i++)
             {
                 fprintf(fp,"%d %d %d %d %d %d\n",NodeInfo[i].node,NodeInfo[i].x,NodeInfo[i].y,NodeInfo[i].grade,pathCount[i],NodeInfo[i].battry);
             }
             fprintf(fp,"\n#######################################################################\n");
             fclose(fp);

             cout<<"Finish\n";

             return 0;
         }
```