COMPUTER UNIVERSITY (MANDALAY)

FINAL YEAR PROJECT REPORT

ON

DECISION MAKING SYSTEMS OF DIAGNOSIS FOR DIABETES

Bachelor of Computer Science

(B.C.Sc.)

Presented by Group (No - 20)

2014-2015

Acknowledgements

We would like to express my appreciation and thanks to the following persons whose guidance aided directly or indirectly towards the success of this project.

We would respectfully like to thank **Dr.Win Aye**, Rector of the Computer University (Mandalay) for her guidance and valuable suggestions and for her permission to submit our project.

We would like to express my special thanks to **U Thaung Kyaw**, Associate Professor and Head of the English Department, Computer University (Mandalay for editing this project and giving advice useful us about our project.

We are deeply thankful to teacher **Dr.Saw Thanda Myint**, Head of Application Department, Computer University (Mandalay), for her great editing and guidance.

We wish to express thank to my supervisor **Daw Khin Po**, Lecturer of Information Science Department the Computer University (Mandalay), for her invaluable guidance, and supervision throughout the work of this project.

We thank all the teacher of Computer University (Mandalay) who taught us with their benevolences, compassions and hobbies.

Finally, we would like to express our gratitude all people who helped us to achieve this project.

Group Member List

Sr.No	Name	Roll No.
1	Ma Hsu Wai Myint	4CS-27
2	Ma Win Pa Pa Kyaw	4CS-79
3	Ma Htet Htet Naing	4CS-148
4	Mg Lwin Aung Moe	4CS-158

Supervisor Khi 28.9.15

Name : **Daw Khin Po**

Rank : **Lecturer**

Department : **Software Department Computer University (Mandalay)**

Project Schedule

Project Proposal : : 16.3.2015

First Seminar : : 3 .5.2015

Second Seminar : : 8 .6.2015

Third Seminar : :12.7.2015

Book Submission : :28.9.2015

Time Schedule	March 2015	May 2015	June 2015	July 2015	August 2015
Project Proposal					
First Seminar					
Second Seminar					
Third Seminar					
Book Submission					

Abstract

Diabetes is malfunctioning which causes due to the body's deficiency of insulin. Nowadays it becomes popular globally. Genetic algorithms are a search heuristic that mimics the process of evaluation. If the users want to know about the diabetic, they are use this system for provide for their medical care, more effectively. This system not only simplifies the task of doctor but also helps patients by providing initial medical care for diabetic's symptoms in emergency. Decision making system of diagnosis for diabetes will implement by using the C# Programming language.

List of Figures

List of Tables

List of Equations

CONTENTS

CHAPTER 3 **DESIGN AND IMPLEMENTATION**

CHAPTER 1

INTRODUCTION

1.1 Introduction

Computer based methods are increasingly used to improve the quality of medical services .This system examines the issues in the context of a Genetic Algorithm approach to made decision of diabetes .To evaluate the diabetes diseases, genetic algorithm has been used. Genetic algorithm are a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. Diabetes mellitus refers to a group of diseases that affect how your body uses blood sugar (glucose).Glucose is vital to your health because it's an important source of energy for the cells that make up your muscles and tissues. It's also your brain's main source of fuel.

If you have diabetes, no matter what type, it means you have too much glucose in your blood, although the causes may differ. Too much glucose can lead to serious health problems. Diabetes symptoms vary depending on how much your blood sugar is elevated.

Genetic algorithms are quite popular and applicable to many domains like industrial design, scheduling, network design, routing, time series prediction, database mining, control systems, artificial life systems, as well as in many fields of science. In computer science and AI, GA is used as a process of search through the space of possible solutions. Database mining, control systems, artificial life systems, as well as in many fields of science. In computer science and AI, GAs are used as a process of search through the space of possible solutions. The set of possible solutions defines the search space (also known as state space) for a give problem. Solutions or partial solutions are reviewed as points in the search space. In engineering and mathematics, GA is used as a process of optimization .The problems are first formulated as mathematical models expressed in terms of functions and then

to find a solution, discover the parameters that optimize the model or the function components that provide optimal system performance.

1.2 Objectives of the Project

- To decide diabetes or not diabetes.
- To understand how does Genetic Algorithm work.
- To alleviate cost effectiveness without going to the hospital.
- To assist the medical officers and to get correct diagnosis of the disease, within very short period of time.

1.3 Project Requirements

1.3.1 Hardware Requirements

- **Processors :** Intel ® Core™ i3-3110M CPU @ 2.40GHz
- **RAM** : 2.00 GB
- **Input** : Keyboards & Mouse

1.3.2 Software Requirements

- Microsoft Visual Studios 2012
- Microsoft SQL Server

CHAPTER 2

THEORY BACKGROUND

2.1 Informed Search and Exploration

An informed search strategy that uses problem specific knowledge beyond the definition of the problem itself can find solutions more efficiently.

Informed search a heuristic is used a guide that leads to better overall performance in getting to the goal state.

2.2 Genetic Algorithm

In the field of artificial intelligence, a **genetic algorithm (GA)** is a search heuristic that mimics the process of natural selection. This heuristic (also sometimes called a met heuristic) is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

In computer science, engineering, computational physics, molecular chemistry, statistics and applied probability, genetic algorithms are a class of interacting and nonlinear Monte Carlo methods to sample from complex high-dimensional probability distributions and to estimate their normalizing constants. Genetic particle algorithms approximate the target probability distributions by a large cloud of random samples termed particles or individuals. During the mutation transition, the particles evolve randomly around the space independently and to each particle is associated a fitness weight function. During the selection transitions, such an algorithm duplicates particles with high fitness at the expense of particles with low fitness which dies. These genetic type particle samplers belong to the class of mean field particle methods.

In signal processing and Bayesian inference genetic algorithms allow solving the nonlinear filtering problem. These algorithms are called particle filters or sequential Monte Carlo methods. In computational physics and molecular simulation, genetic algorithms are also used to compute Feynman-Kay path integrals as the number of individuals tends to infinity. These mutation-selection type algorithms are often called Resample or Reconfiguration Monte Carlo. They belong to the class of Quantum Monte Carlo methods and more specifically Diffusion Monte Carlo methodologies.

Genetic algorithms find application in bioinformatics, phylogenetic, computational science, signal and image processing, Bayesian inference, machine learning, risk analysis and rare event sampling, Engineering and robotics, economics, manufacturing, mathematics, mathematical finance, molecular chemistry, computational physics, pharmacokinetic, pharmacometrics , and other fields.

2.2.1 Optimization problems

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a *generation*. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly,

the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

1. A genetic representation of the solution domain,
2. A fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

2.2.2 Initialization

The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the *search space*). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

2.2.3 Evaluation

Evaluation is to associate each individual or chromosome with a fitness value that reflects how good it is based upon its achievement of the objectives. The fitness function is the essence of the problem: it provides the means by which the quality of a solution may be assessed, and the probability that a solution will reproduce. The function needs to be direct measure of the quality of the solution.

2.2.4 Selection

During each successive epoch, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected .Certain selection methods are the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.

Most functions are stochastic and designed so that a small proportion of less fit solutions are selected. This helps keep diversity of the population large, preventing premature convergence on poor solutions. Certain selections methods are the fitness of each solution are preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming. Popular and well-suited selection methods include roulette wheel selection, rank selection and tournament selection.

- **Roulette-Wheel Selection:** In the roulette-wheel selection, the probability that a solution will be selected is given by the ratio of its fitness to the fitness of other member of the current population. This method is sometimes called fitness proportionate selection.

- **Tournament Selection:** In this selection, several individuals are chosen at random and the fittest becomes one of parents. The selection process is: select a group of N members (N>=2), then select the fittest member of the group and discard the rest. This operator is excellent suited for applying in parallel genetic algorithm.
- **Rank Selection:** In this selection, the individual in the current population are first sorted by fitness. The probability that solution will be selected is then proportional to its rank in this sorted list, rather than its fitness.

In genetic algorithms, crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. Cross over is a process of taking more than one parent solutions and producing a child solution from them. There are methods for selection of the chromosomes. Those are also given below.

2.2.5 Crossover techniques

Many crossover techniques exist for organisms which use different data structures to store themselves.

2.2.5.1 One point crossover

A single crossover point on both parents' organism strings is selected .All data beyond that point in either organism string is swapped between the two parent organisms. The resulting organisms are the children:

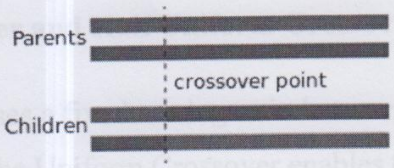

Figure 2.1 One Point Crossovers

2.2.5.2 Two Point crossover

Two-point crossover calls for two points to be selected on the parent organism strings. Everything between the two points is swapped between the parent organisms, rendering two child organisms:

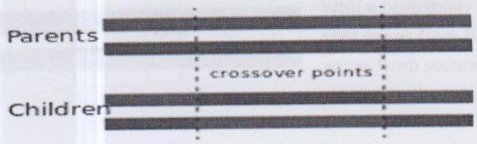

Figure 2.2 Two Point Crossovers

2.2.5.3 Cut and Splice

Another crossover variant, the "cut and splice" approach, results in a change in length of the children strings. The reason for this difference is that each parent string has a separate choice of crossover point.

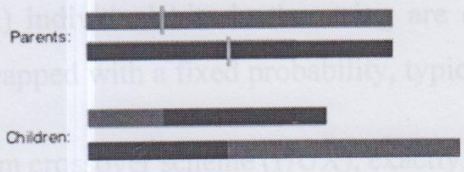

Figure 2.3 Cut and Splice

2.2.5.4 Uniform Crossover and Half Uniform Crossover

The Uniform Crossover uses a fixed mixing ratio between two parents. Unlike one- and two-point crossover, the Uniform Crossover enables the parent chromosomes to contribute the gene level rather than the segment level.

If the mixing ratio is 0.5, the offspring has approximately half of the genes from first parent and the other half from second parent, although cross over points can be randomly chosen as seen below:

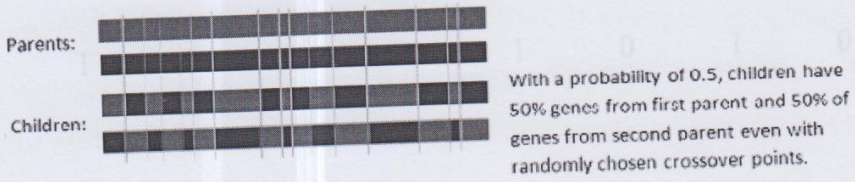

Figure 2.4 Uniform Crossovers and Half Uniform Crossover

The Uniform Crossover evaluates each bit in the parent strings for exchange with a probability of 0.5. Even though the uniform crossover is a poor method[1], empirical evidence suggest that it is a more exploratory approach to crossover than the traditional exploitative approach that maintains longer schemata. This results in a more complete search of the design space with maintaining the exchange of good information. Unfortunately, no satisfactory theory exists to explain the discrepancies between the Uniform Crossover and the traditional approaches. In the uniform crossover scheme (UX) individual bits in the string are compared between two parents. The bits are swapped with a fixed probability, typically 0.5.

In the half uniform crossover scheme (HUX), exactly half of the nonmatching bits are swapped. Thus first the Hamming distance (the number of differing bits) is calculated. This number is divided by two. The resulting number is how many of the bits that do not match between the two parents will be swapped.

2.2.5.5 Three parent crossover

In this technique, the child is derived from three parents. They are randomly chosen. Each bit of first parent is checked with bit of second parent whether they are same. If same then the bit is taken for the offspring otherwise the bit from the third parent is taken for the offspring. For example, the following three parents:

parent1 1 1 0 1 0 0 0 1 0

parent2 0 1 1 0 0 1 0 0 1

parent3 1 1 0 1 1 0 1 0 1

Produces the following offspring:

Offspring 1 1 0 1 0 0 0 0 1

2.2.5.6 Crossover for Ordered Chromosomes

Depending on how the chromosome represents the solution, a direct swap may not be possible. One such case is when the chromosome is an ordered list, such as an ordered list of the cities to be travelled for the traveling salesman problem. There are many crossover methods for ordered chromosomes. The already mentioned N-point crossover can be applied for ordered chromosomes also, but this always need a corresponding repair process, actually, some ordered crossover methods are derived from the idea. However, sometimes a crossover of chromosomes produces recombination's which violate the constraint of ordering and thus need to be repaired. Several examples for crossover operators (also mutation operator) preserving a given order is given in:

1. Partially matched crossover (PMX): In this method, two crossover points are selected at random and PMX proceeds by position wise exchanges. The two crossover points give matching selection. It affects cross by position-by-position exchange operations. In this method parents are mapped to each other, hence we can also call it partially mapped crossover.
2. Cycle crossover (CX): Beginning at any gene i in parent 1, the i-the gene in parent 2 becomes replaced by it. The same is repeated for the displaced gene until the gene which is equal to the first inserted gene becomes replaced (cycle).
3. Order crossover operator (OX1): A portion of one parent is mapped to a portion of the other parent. From the replaced portion on, the rest is filled up by the remaining genes, where already present genes are omitted and the order is preserved.
4. order-based crossover operator (OX2)
5. position-based crossover operator (POS)
6. voting recombination crossover operator (VR)
7. alternating-position crossover operator (AP)
8. sequential constructive crossover operator (SCX)

Other possible methods include the edge recombination operator.

2.2.6 Mutation

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation. Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation

probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search.

The classic example of a mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence. This random variable tells whether or not a particular bit will be modified. This mutation procedure, based on the biological point mutation, is called single point mutation. Other types are inversion and floating point mutation. When the gene encoding is restrictive as in permutation problems, mutations are swaps, inversions, and scrambles.

The purpose of mutation in Gas is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. This reasoning also explains the fact that most GA systems avoid only taking the fittest of the population in generating the next but rather a random (or semi-random) selection with a weighting toward those that are fitter.[1]

For different genome types, different mutation types are suitable:

2.2.6.1 Bit string mutation

The mutation of bit strings ensue through bit flips at random positions.

Example:

1 0 1 0 0 1 0

↓

1 0 1 0 1 1 0

The probability of a mutation of a bit is $\frac{1}{l}$, where l the length of the binary vector is. Thus, a mutation rate of 1per mutation and individual selected for mutation is reached.

2.2.6.2 Flip Bit

This mutation operator takes the chosen genome and inverts the bits (i.e. if the genome bit is 1, it is changed to 0 and vice versa).

2.2.6.3 Boundary

This mutation operator replaces the genome with either lower or upper bound randomly. This can be used for integer and float genes.

2.2.6.4 Non-Uniform

The probability that amount of mutation will go to 0 with the next generation is increased by using non-uniform mutation operator. It keeps the population from stagnating in the early stages of the evolution. It tunes solution in later stages of evolution. This mutation operator can only be used for integer and float genes.

2.2.6.5 Uniform

This operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes.

2.2.6.6 Gaussian

This operator adds a unit Gaussian distributed random value to the chosen gene. If it falls outside of the user-specified lower or upper bounds for that gene, the

new gene value is clipped. This mutation operator can only be used for integer and float genes.

2.3 The General Scheme of Simple Genetic Algorithm

```
Function GENETIC_ALGORITHM (population, FITNESS_FN) returns an individual
    inputs : population , a set of individuals
             FITNESS_FN, a function that measure the fitness of an individual
    repeat
        new _ population ← empty set
        loop for i from 1 to SIZE (population) do
            x ← RANDOM_SELECTION (population , FITNESS_FN)
            y ← RANDOM_SELECTION (population , FITNESS_FN)
            child ← REPRODUCE(x , y)
            if(small random probability) then child ← MUTATE(child)
            add child to new_ population
        population ← new_ population
    until some individual is fit enough , or enough time has elapsed
    return the best individual in population , according to FITNESS –FN
Function REPRODUCE(x , y ) returns an individual
    inputs : x , y , parent individuals
    n ← LENGTH(x)
```

c ← random number from 1 to n

return APPEND(SUBSTRING (x , 1 , c). . SUBSTRING (y , c+1 , n))

Figure 2.5 The General Scheme of Simple Genetic Algorithm

CHAPTER 3

DESIGN AND IMPLEMENTATION

3.1 System Flow Diagram

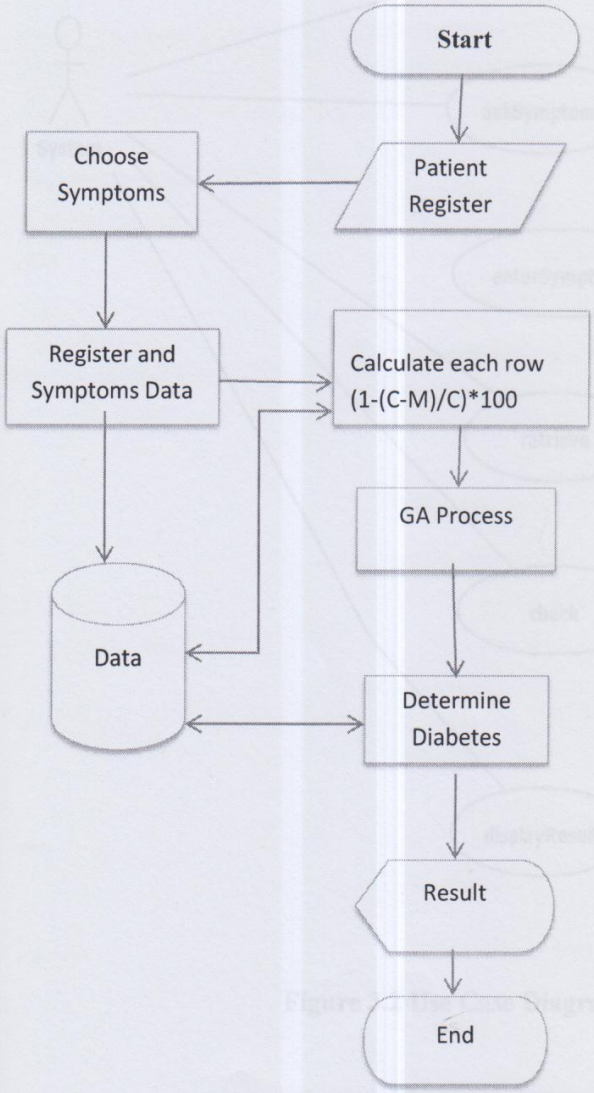

Figure 3.1 Process Flow of the System

3.2 Use Case Diagram

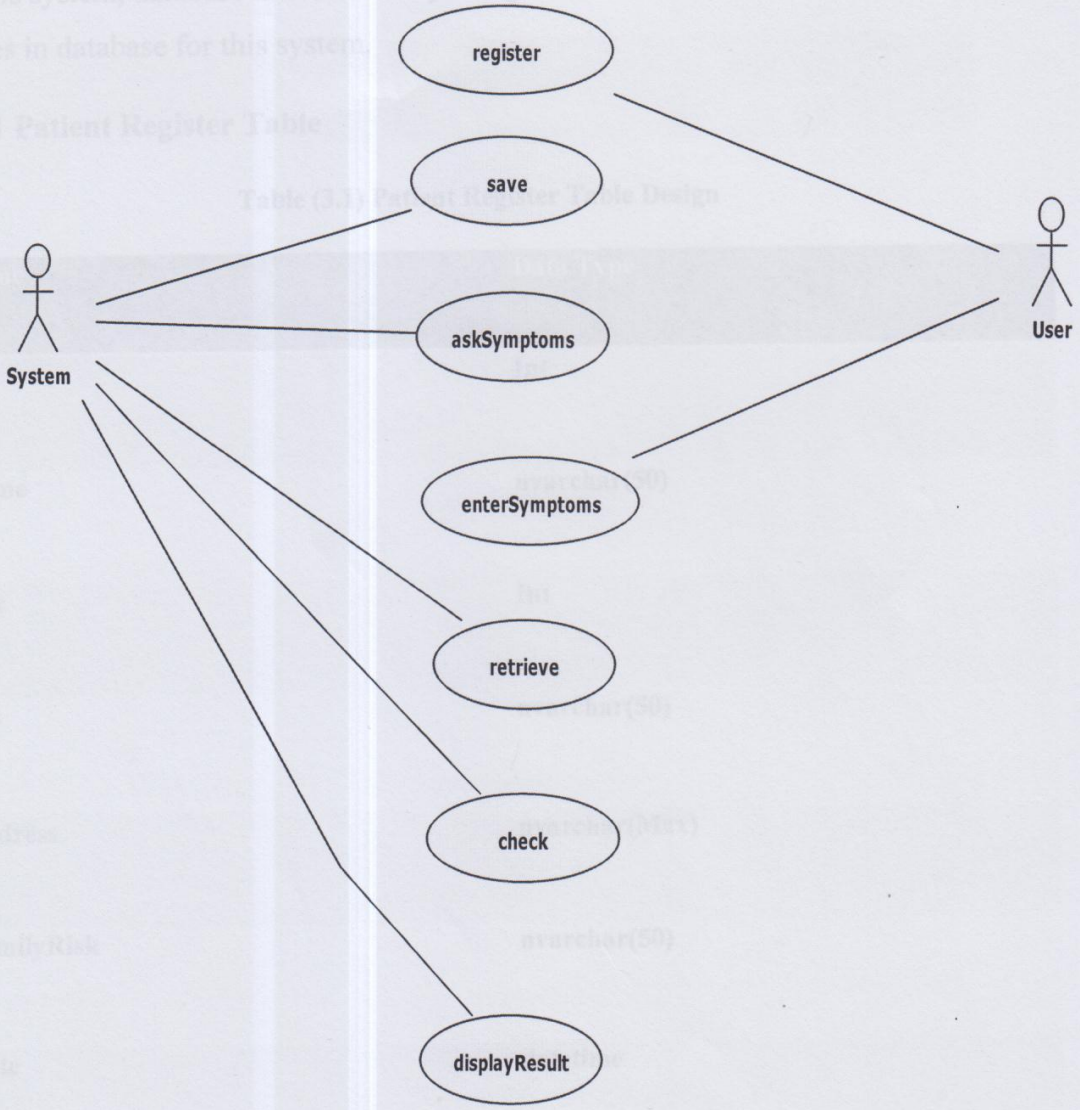

Figure 3.2 Use Case Diagram of System

3.3 Database Design

In this system, database is described by using Microsoft SQL Sever. There are three tables in database for this system.

3.3.1 Patient Register Table

Table (3.1) Patient Register Table Design

Name	Data Type
Pid	**Int**
Name	**nvarchar(50)**
Age	**Int**
Sex	**nvarchar(50)**
Address	**nvarchar(Max)**
FamilyRisk	**nvarchar(50)**
Date	**datetime**
Condition	**nvarchar(50)**

3.3.2 Symptoms Description Table

Table (3.2) Symptoms Description Table Design

Name	Type
SID	nchar(10)
s1	nvarchar(50)
s2	nvarchar(50)
s3	nvarchar(50)
s4	nvarchar(50)
s5	nvarchar(50)
s6	nvarchar(50)
s7	nvarchar(50)
s10	nvarchar(50)
s11	nvarchar(50)
s12	nvarchar(50)

3.3.3 Decision Table

Table (3.3) Decision Table Design

Name	Type
SD	nchar(10)
S1	nvarchar(50)
S2	nvarchar(50)
S3	nvarchar(50)
S4	nvarchar(50)
S5	nvarchar(50)
S6	nvarchar(50)
S7	nvarchar(50)
S10	nvarchar(50)
S11	nvarchar(50)
S12	nvarchar(50)
Conditions	nvarchar(50)

3.4 Data Set Tables

3.4.1 Patient Register Table

Table (3.4) Patient Register Table

PId	Name	Age	Sex	Adress	FamilyRisk	Date	Condition
1	Ba Ba	34	Male	Mandalay	No	7/21/2015 2...	Diabetes
2	Mg Mg	23	Male	Mandalay\...	No	7/22/2015 5...	No Diabetes
3	Khin Khin	12	Female	Yangon	No	7/22/2015 6...	Diabetes
4	Mya Mya	34	Female	Mandalay	Yes	7/22/2015 9...	Diabetes
5	Bo Bo	30	Male	Pyi Oo Lwin	No	7/22/2015 1...	No Diabetes
6	Kyaw Kyaw	20	Male	Mdy	No	7/22/2015 1...	No Diabetes
7	U Mya	44	Male	Mdy	No	7/22/2015 1...	Diabetes
8	Mya Moe	23	Female	Mdy	No	7/22/2015 1...	Diabetes
9	Yo To	23	Male	Mya Mya	No	7/22/2015 1...	No Diabetes
10	jaw moe	12	Male	mdy	Yes	7/22/2015 1...	Diabetes
11	Mg Mg	12	Male	hfhjkahjha	Yes	8/4/2015 10...	No Diabetes
12	Lwin Aung ...	21	Male	Mandalay	Yes	8/5/2015 2:...	Diabetes
13	Htet Htet ...	21	Female	Pyin Oo Lwin	No	8/5/2015 2:...	No Diabetes
14	Mg Kyaw	23	Male	Mandalay	Yes	8/7/2015 1:...	Diabetes
15	Ko Mya	23	Male	Pyi Oo Lwin	Yes	8/7/2015 11...	Diabetes
16	U Mya	44	Male	Mdy	No	8/8/2015 1:...	No Diabetes
17	mg mg	18	Male	mandalay	No	8/11/2015 1...	No Diabetes
18	Ko Zaw	23	Male	Mandalay	Yes	8/12/2015 1...	No Diabetes
19	mg mg	18	Male	mandalay	Yes	8/12/2015 2...	No Diabetes
20	bo bo	30	Male	mandalay	Yes	8/12/2015 2...	No Diabetes
21	Ma Ma	12	Female	khkajkjkhs...	Yes	8/14/2015 7...	No Diabetes
22	Moe Moe	23	Female	58 30 Yang...	No	8/18/2015 7...	No Diabetes

3.4.2 Decision Table

Table (3.5) Decision Table

SID	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	Decision
1	1	1	1	1	1	1	1	1	1	1	0	0	Diabetes
2	1	0	0	0	0	0	0	0	0	0	0	1	No Diabetes
3	1	1	0	0	0	0	0	0	0	1	1	1	Diabetes
4	1	1	0	0	0	1	1	0	0	1	1	0	No Diabetes
5	0	0	1	1	1	0	0	0	0	0	0	1	Diabetes
6	1	0	0	0	1	0	0	1	0	0	0	0	No Diabetes
7	1	0	0	1	0	0	0	0	0	0	1	0	No Diabetes
8	1	0	0	0	0	1	0	0	0	0	0	0	No Diabetes
9	1	1	1	1	0	0	0	0	0	1	0	0	Diabetes
10	1	0	1	1	0	0	1	0	0	0	1	0	Diabetes
11	0	1	0	1	0	1	0	1	0	1	1	1	No Diabetes
12	1	0	0	0	0	0	0	0	0	1	1	1	No Diabetes
13	0	1	1	0	1	0	1	0	1	0	1	0	Diabetes
14	1	1	0	1	0	1	0	1	1	0	1	0	Diabetes
15	1	1	0	0	0	0	0	1	1	1	1	1	No Diabetes
16	0	1	0	0	0	1	0	1	0	0	1	1	No Diabetes
17	0	0	0	1	0	0	0	1	1	0	1	0	Diabetes
18	1	0	1	1	1	0	1	0	1	0	1	0	Diabetes
19	1	0	1	0	1	0	1	1	1	1	1	1	Diabetes
20	0	0	1	1	0	0	0	0	0	0	0	0	Diabetes
21	0	0	0	0	0	1	1	0	0	0	0	1	No Diabetes
22	1	1	1	1	1	1	1	0	0	0	0	0	Diabetes
23	1	0	1	1	1	1	1	0	1	0	1	1	Diabetes
24	1	0	1	0	1	0	1	1	1	1	1	1	Diabetes
25	0	0	0	1	0	1	0	1	0	0	0	1	No Diabetes

3.4.3 Symptoms Description Table

Table (3.6) Symptoms Description Table

SD	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12
SD1	Polyuria	Increased t...	Septic	Unexplain...	Blurred visi...	Fatigue	Extreme hu...	Unexplaine...	Persistent,...	Nausea an...	Cough	Sneezing

3.5 Implementation of the Project

This system is implemented by two main parts. These are step by step implementation of the system and iteration of the system. The main from of the system is shown in Figure 3.6

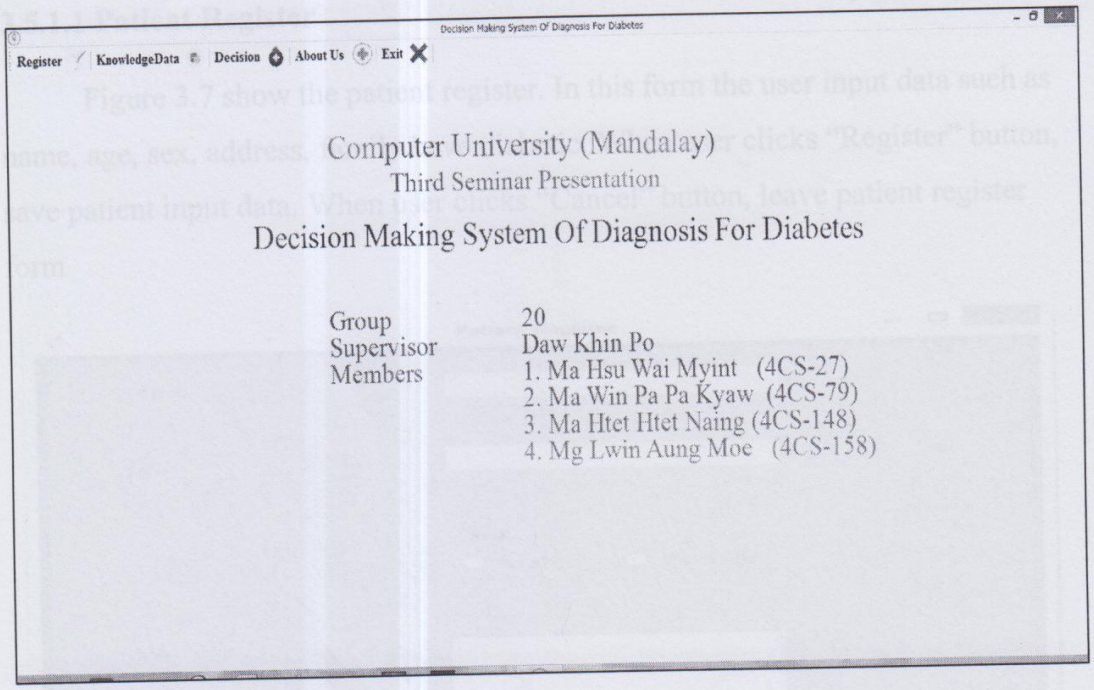

Figure 3.6 Main form of the system

3.5.1 Implementation of the system

In this part, the system is implemented by nine steps. There are

1. Patient Register
2. Check Symptoms
3. Show Weight Processes
4. Show Fitness Value
5. Show Binary Population

6. Show Symptoms Results
7. Show Patient Information
8. Show Dataset
9. Show about this System

3.5.1.1 Patient Register

Figure 3.7 show the patient register. In this form the user input data such as name, age, sex, address, family have diabetic. When user clicks "Register" button, save patient input data. When user clicks "Cancel" button, leave patient register form

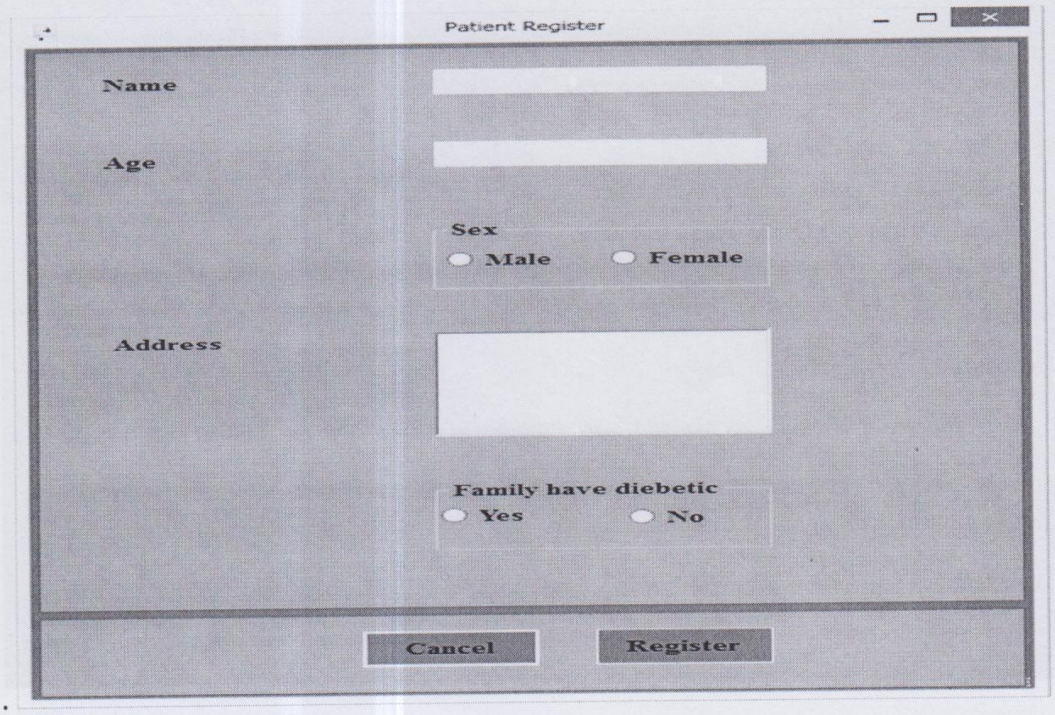

Figure 3.7 Patient Register

3.5.1.2 Check Symptoms

Figure 3.8 Show the symptom. In this form, the user check symptom .When user click "Check " button ,calculate the decision for the patient using Weight process and GA process. When user click "Restart" button , return can be choose the symptom data of patient. When user click "Cancel" button, leave out the symptom check form.

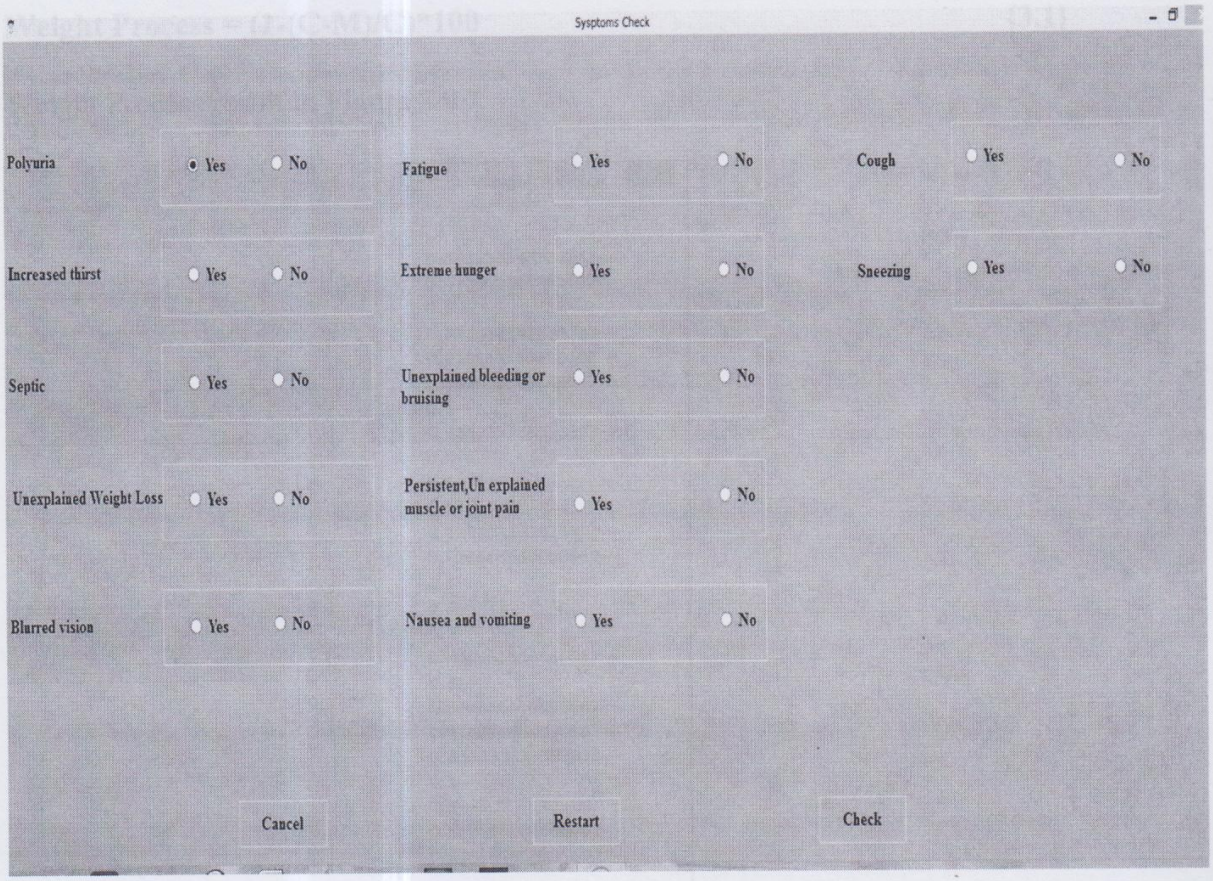

Figure 3.8 Symptom Check Form

3.5.1.3 Show Weight Processes

After the "Check "button click. Weight process is start. Weight process calculate by this formulated as

C=Number of Columns

M=Number of Match

Weight Process = (1-(C-M)/C)*100 {3.1}

Weight Process result in Figure 3.9

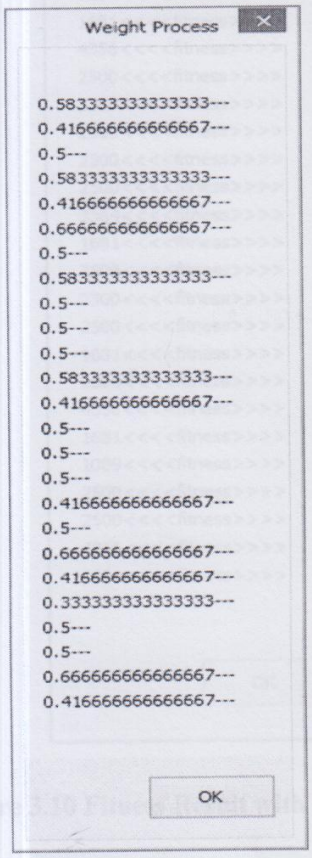

Figure 3.9 Weight Process with Message Box

3.5.1.4 Show Fitness Value

After weight process calculate fitness function. A **fitness function** should return higher values for better states. To apply our genetic algorithm to a particular optimization problem step by step. Consider the problem of **maximizing the function f(x) = x^2.** Fitness result show in Figure 3.10

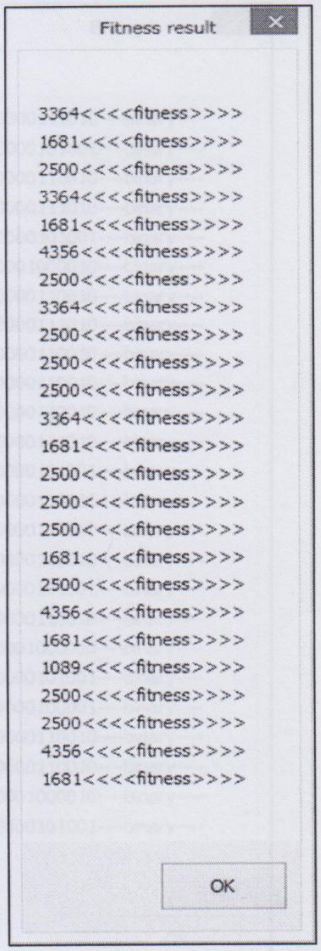

Figure 3.10 Fitness Result with Message Box

3.5.1.5 Show Binary Population

After fitness function calculate binary. Binary process use weight process result. Binary process result use in **GA** process. GA processes include **two point crossover** and **bit string mutation**. GA process return max value. Binary process result show in Figure 3.11

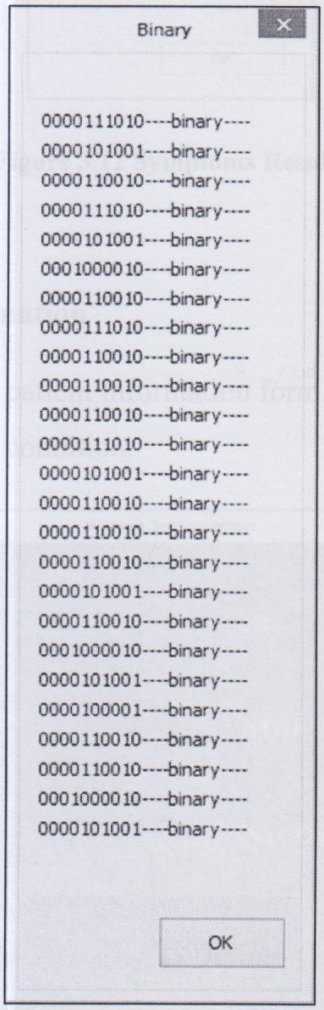

Figure 3.11 Binary Results with Message Box

3.5.1.6 Show Symptoms Results

After binary process calculate GA process.GA process decision diabetes or no diabetes. After decision show result in Figure 3.12

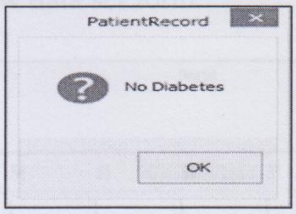

Figure 3.12 Symptoms Results

3.5.1.7 Show Patient Information

After symptoms result show patient information form. Patient information forms include patient id, name and condition.

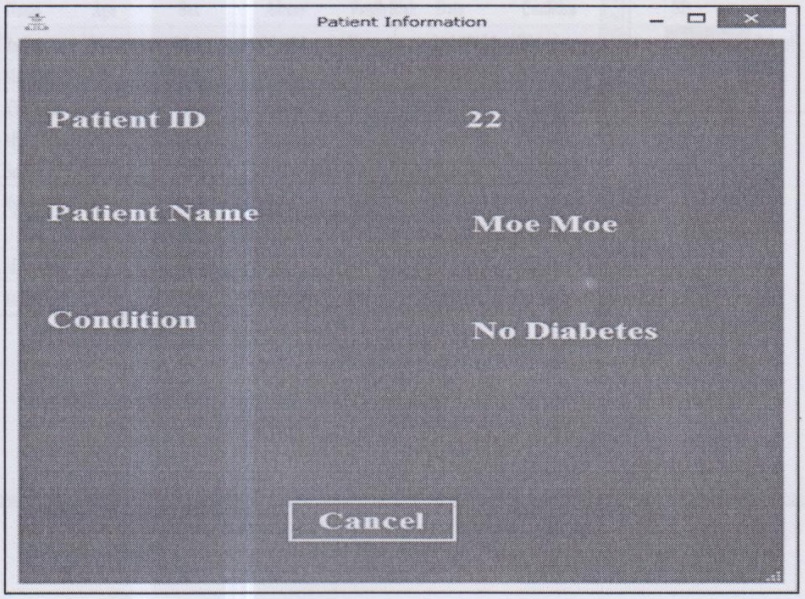

Figure 3.13 Patient Information Form

3.5.1.8 Show Dataset

Figure 3.14 shows the data set of this system. In this form the user look up decision table, patient register table and symptom description table. When you click "Knowledge Data" button show data set form.

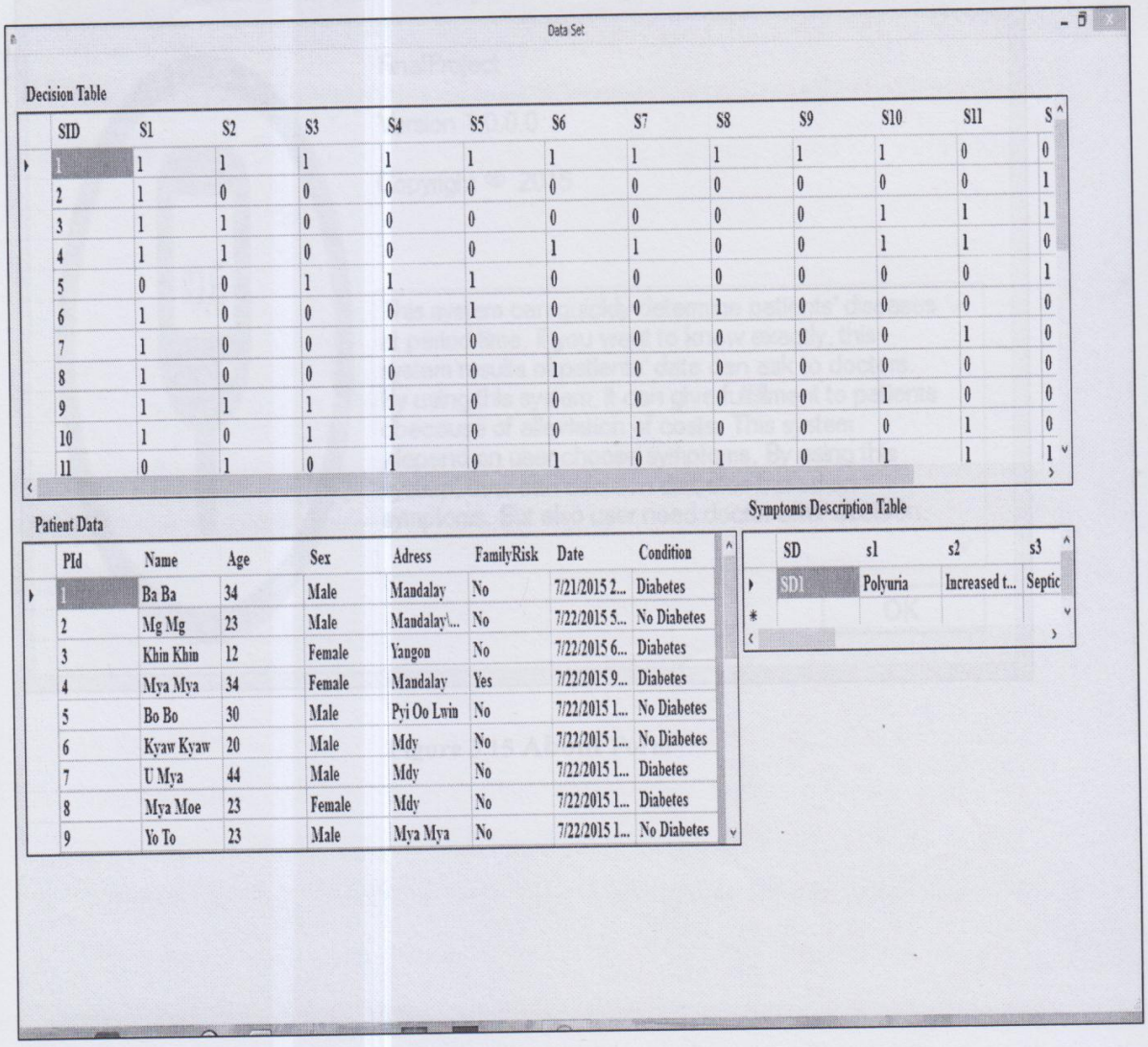

Figure 3.14 Data Set Form

3.5.1.9 Show about this System

Figure 3.15 shows about of this system. In this form the user look up our system description. When you click "About Us " button show data set form.

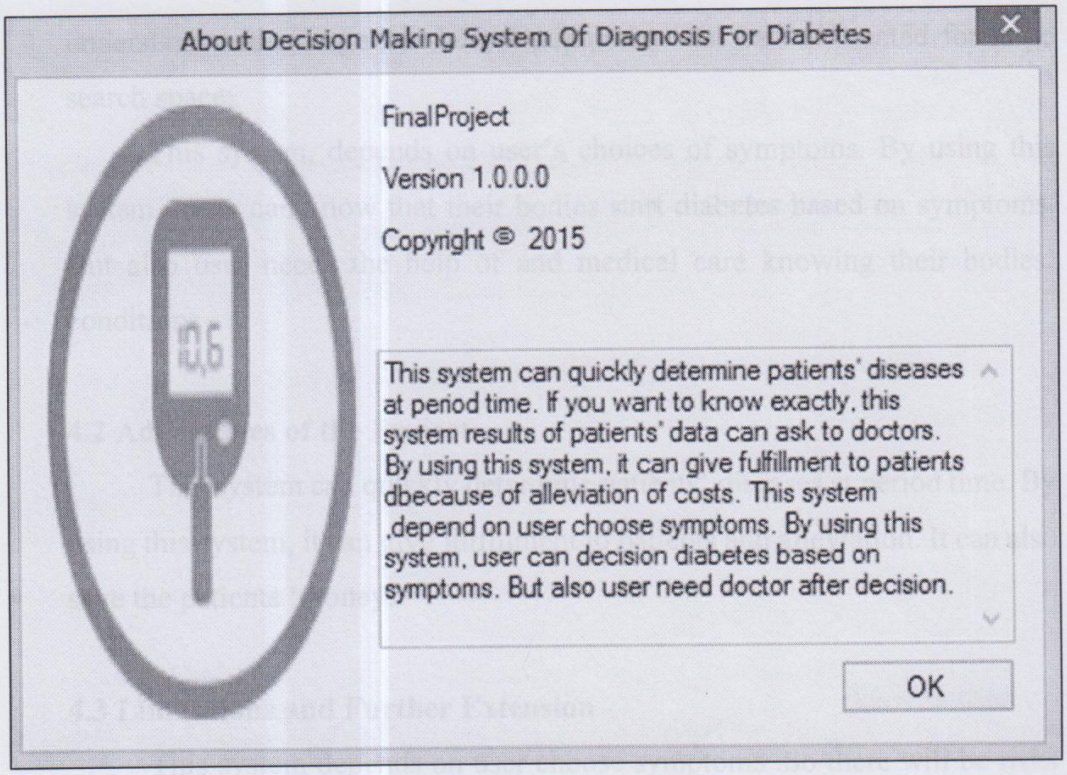

Figure 3.15 About Form

CHAPTER 4

CONCLUSION

4.1 Conclusion

This system, binary representation for and individual are easy to understand and a large number of population can be represented for large search space.

This system, depends on user's choices of symptoms. By using this system, users can know that their bodies start diabetes based on symptoms. But also user needs the help of and medical care knowing their bodies' conditions

4.2 Advantages of the Project

This system can quickly determine patients' diseases at period time. By using this system, it can give fulfillment to patients and alleviation. It can also save the patients 'money.

4.3 Limitations and Further Extension

This system depends on user choose symptoms .So there will be risks in the systems. This system can only be solved to decision but not including medical guide line feature for diabetic. So user needs doctor for medical care.

This system will be extended to medical data set such as cancer, different diabetes condition (e.g. Type I and Type II) and other diseases. Nowadays diabetes test uses A1C test. The A1C test is the primary test used for diabetes management and diabetes research.

References

1. Ruggiero, Murray A (2009-08-01) Fifteen years and counting. Futuresmag.com. Retrieved on 2013-08-07.
2. Herman, Kahn; Harris, Theodore, E. (1951). "Estimation of particle transmission by random sampling" (PDF). *Natl. Bur. Stand. Appl. Math. Ser.* 12: 27–30.
3. Fraser, Alex (1957). "Simulation of genetic systems by automatic digital computers. I. Introduction". *Aust. J. Biol. Sci.* 10: 484–491.
4. Fraser, Alex; Burnell, Donald (1970). *Computer Models in Genetics*. New York: McGraw-Hill. ISBN 0-07-021904-4.
5. Crosby, Jack L. (1973). *Computer Simulation in Genetics*. London: John Wiley & Sons. ISBN 0-471-18880-8.
6. Goldberg, David E. (1991). "The theory of virtual alphabets". *Parallel Problem Solving from Nature, Lecture Notes in Computer Science* 496: 13–22. doi:10.1007/BFb0029726. Retrieved 2 July 2013.
7. Eiben, A. E. et al (1994). "Genetic algorithms with multi-parent recombination". PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: 78–87. ISBN 3-540-58484-6.
8. Del Moral, Pierre (1996). "Non Linear Filtering: Interacting Particle Solution." (PDF). *Markov Processes and Related Fields* 2 (4): 555–580.
9. Bäck, Thomas (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford Univ. Press.
10. Falkenauer, Emanuel (1997). *Genetic Algorithms and Grouping Problems*. Chichester, England: John Wiley & Sons Ltd. ISBN 978-0-471-97150-4.
11. Del Moral, Pierre; Miclo, Laurent (2000). "A Moran particle system approximation of Feynman-Kac formulae.". *Stochastic Processes and their Applications* 86 (2): 193–216. doi:10.1016/S0304-4149(99)00094-0.

12. Ting, Chuan-Kang (2005). "On the Mean Convergence Time of Multi-parent Genetic Algorithms Without Selection". Advances in Artificial Life: 403–412. ISBN 978-3-540-28848-0.

13. *DAVID E.GOLDBERG*, "Genetic Algorithm" ,ISBN 0-201-157675-5,Print in 27th Printing June 2005 page15-16-17.

14. Website (www.wikipedia.com)

15. Website (www.projectcode.com)

16. Website (www.mayoclinic.org)