APPLICATION OF SBSE TECHNIQUES FOR HIERARCHICAL SOFTWARE CLUSTERING

by

Ibrar Hussain

2009-NUST-MSPhD- CSE(E)-16

MS-09 (SE)

Submitted to the Department of Computer Engineering in fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
SOFTWARE ENGINEERING

Thesis Supervisor

Dr. Aasia Khanum

College of Electrical & Mechanical Engineering

National University of Sciences & Technology

2012

DECLARATION

I hereby declare that I have developed this thesis entirely on the basis of my personal efforts under the guidance of my supervisor Dr. Aasia Khanum. All the sources used in this thesis have been cited and the contents of this thesis have not been plagiarized. No portion of the work presented in this thesis has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

Ibrar Hussain

ACKNOWLEDGEMENTS

First and foremost I want to thank my advisor Dr. Aasia Khanum. It has been an honor to study under his supervision. I appreciate all his contributions of time and ideas to make my MS experience both productive and stimulating.

I would also like to thank the faculty members at the college of E & ME for their constant support during our studies. For this dissertation I would like to thank my committee members Brig. Dr. Muhammad Younus Javed, Dr. Shoab Ahmad Khan, and Dr. Farooque Azam. I would also like to thank Mr. Abdul Qudus Abbasi, Assistant Professor at Quaid-i-Azam University Islamabad for his guidance in the field of software clustering, providing me literature on this topic, and providing me valuable test systems data to compute and evaluate results.

Lastly, I would like to thank my family for all their love and encouragement. For my parents who raised me and supported me in all my pursuits. And most of all for my loving, supportive and encouraging brother and sisters whose faithful support during the final stages of my degree is so appreciated. Thank you it would not have been possible without you.

To my loving parents, brother and sisters…

ABSTRACT

Software systems evolve and change with time due to change in business needs. At some stage the available architectural description may not best represent the current software system. Accurate understanding of software architecture is very important because it helps in estimating where and how much change is required in the software system to fulfill changing business needs. It also helps in making decisions related to reusability of software components. The understanding of software architecture also plays vital role in estimating cost and risk of change in software system. In some cases, especially for legacy systems such a description does not readily exist. For such cases, we can use source code to extract architecture of the software system. Software Clustering is an approach to decompose large software system into smaller manageable sub systems to get system architecture. Software clustering, however, is an NP-hard problem. Search Based Software Engineering (SBSE) provides optimization algorithms which are search based and can be applied to Software Engineering problems. Particle Swarm Optimization (PSO) is a metaheuristic search technique based on biological behaviors and can be used to solve NP-hard problems. This thesis provides a framework for solving software clustering problem with PSO. Experimental results show fast convergence and stable results.

In this thesis, software clustering process is presented in detail. Different Search Based Software Engineering (SBSE) techniques are discussed but focus is on Particle Swarm Optimization (PSO). The thesis focuses on design, implementation and analysis of PSO algorithm applied to software clustering problem. The objective of this paper is to solve software clustering problem using PSO and examine the effectiveness of PSO comparative to Genetic Algorithms (GA). Simulation results show that the PSO approach has stable results and it requires smaller computational effort as compared to GA.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACO	Ant Colony Optimization
CF	Component Factor
ES	Evolutionary Strategies
GA	Genetic Algorithms
GP	Genetic Programming
IP	Integer Programming
LP	Linear Programming
MFC	Microsoft Foundation Classes
MOJO	MOve and JOin
MQ	Modularization Quality
NLP	Non-Linear Programming
PSO	Particle Swarm Optimization
QP	Quadratic Programming
SA	Simulated Annealing
SBSE	Search Based Software Engineering
SCPSO	Software Clustering using Particle Swarm Optimization
TS	Tabu Search
UI	User Interface
pbest	Personal Best
gbest	Global Best
lbest	Local Best

Chapter 1

INTRODUCTION

Architecture of a software system is defined as “the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution” [1]. Software architecture encapsulates higher level design of software, defining its various sub-systems and their relationships. Knowledge of software architecture is needed in various phases of software lifecycle e.g. maintenance, evolution, and reuse [2], [3]. However, for many systems this architectural knowledge is not so readily available and the software managers have to incur extra efforts in recovering the underlying architecture from source code. Manual methods can be considered as last resort measure for architecture recovery, but in the face of large size and complexity of today’s legacy software these measures prove costly and time-consuming. It is now generally recognized that in order for software architecture recovery to be viable, it must be handled by automatic or semi-automatic tools [4], [5].

Software systems become complex due to the complexity of application domain and changing business rules [6]. It also happens that the software developers are not familiar with many concepts of the application domain. Other reasons for the complexity of software systems are development methods, tools, and people involved in the software development process [7]. Over the life time, software applications demand changes to fit in the changed business processes. The timely modification in the software system is very important which sometimes becomes very difficult due to unavailability of the persons who actually developed the system. Changes weaken the architecture of the system if done without enough understanding. Deteriorated software systems are difficult to understand by the software developers and designers [6], [8].

According to Len Bass [9], “The software architecture of a program or computing system is the structure or structures of the system which comprise software elements, externally visible properties of those elements and the relationships among them”. Understand ability of the software system is highly influenced by the architecture of the software system.

1.1 PROBLEM OVERVIEW

Software clustering is an NP-Hard problem therefore it is very difficult to solve it in real time. Search Based Software Engineering (SBSE) provides optimization algorithms which are search based and can be applied to Software Engineering problems. Particle Swarm Optimization (PSO) is a metaheuristic search technique based on biological behaviours and can be used to solve NP-Hard problems. This thesis provides a framework for solving software clustering problem with PSO.

1.2 PROJECT OBJECTIVES

In this thesis, software clustering process is presented in detail. Different Search Based Software Engineering (SBSE) techniques are discussed but focus is on Particle Swarm Optimization (PSO). The thesis focuses on design, implementation and analysis of PSO algorithm applied to software clustering problem. The objective of this thesis is to solve software clustering problem using PSO and examine the effectiveness of PSO comparative to Genetic Algorithms (GA).

1.3 THESIS OUTLINE

The thesis is logically broken down so that each chapter builds on the learning's from the previous chapters. *Chapter 2* provides fundamentals of software clustering, research contributions, and literature review on search based optimization techniques. This includes hill climbing, simulated annealing, tabu search, genetic algorithms, evolutionary strategies, and genetic programming, particle swarm optimization, and ant colony optimization. *Chapter 3* presents architecture of the software clustering system. The architecture discusses in details the system architecture, class diagram, format of relationships file, and formula to compute fitness values. *Chapter 4* analyzes the results of PSO with relation to GA. The results are elaborated with the help of graphs. *Chapter 5* provides conclusion and future work.

Chapter 2

LITERATURE REVIEW

INTRODUCTION

The chapter starts with the explanation of software clustering is and its process. Then different categories of software clustering algorithms are presented. Later in the chapter, Particle Swarm Optimization (PSO) is explained with all its constraints and variations.

2.1 SOFTWARE CLUSTERING

Clustering is the process of decomposing large system into smaller manageable subsystems in such a way that entities within the subsystem are similar to one another and different from those in other subsystems. The similarity and difference is measured based on presence and absence of some features [11] in entities. The terms entities and features are commonly used. Entities include files, classes, and global functions whereas features are the attributes such as number of function calls of one class within another class.

The clustering produced by a clustering technique is also known as partition. Software clustering process is described as [10].

- ***Identification of entities and features*** – Entities are files, classes, and functions that are grouped together. Features may include number function calls by the entity, global variables referred, type of data maintained by a class, etc.

- ***Measuring similarity*** – Different metrics are used to calculate the similarity between entities. Those metrics include association coefficients, correlation measures and distance metrics.
- ***Applying clustering algorithm*** – Optimization techniques are available which lead us to *sub-optimal solution*. Those techniques include classical techniques and metaheuristic search [13].
- ***Evaluation of partition*** – No definite quantitative measures exist to evaluate partitions. Expert decompositions are used which are mostly done by designer of the system. The test clustering is compared with these expert decompositions.

Clustering algorithms are mainly divided into two categories [11]:

- ***Partitional*** – They produce flat decompositions. Clustering process starts with the initial partition with some number of clusters. On each iteration, clustering criteria is optimized that result in the modification in the partition. Number of clusters must be known in advance for the application of partitional algorithms.

- ***Hierarchical*** – These algorithms decompose software system in natural hierarchy which better helps in understanding large software systems. Hierarchical algorithms represent both detailed and high level views of software system. Hierarchical clustering is further divided into *divisive* and *agglomerative* [15].

2.2 SOURCE CODE ELEMENTS

Source code elements are mainly divided into two groups; entities and relationships.

2.2.1 Entities

Entities are further divided into primary entities and secondary entities [15]. Primary entities are part of the clustering process and they become members of clusters. Secondary entities help indirectly in the clustering process i.e., they help to define relationships among primary entities. These secondary entities do not become members of clusters in the final outcome of the clustering process. Classes, structures, and unions are primary entities while files, folders, global data, global functions, and macros are secondary entities.

2.2.2 Relationships

Relationships between entities are meaningful in the context of clustering. Entities are grouped into clusters based on relationships between those entities. Some of the relationships are inheritance depth, inheritance hierarchy, inheritance type, containment as object, containment as pointer, containment as reference, containment at method parameter level, Containment at local method declaration level, both classes exist in the same file or in the same folder.

2.3 SEARCH BASED SOFTWARE ENGINEERING

Search Based Software Engineering (SBSE) is an approach to software engineering in which search based optimization algorithms are used to identify *acceptable* or *sub-optimal solution* [12].

Most widely used optimization techniques are [13]:

- ***Classical Techniques*** – These techniques include linear programming, integer programming, quadratic programming, non-linear programming, stochastic programming, dynamic programming, combinatorial optimization, infinite-dimensional optimization, constraint satisfaction.

- ***Metaheuristic Search*** – Most commonly search techniques are; hill climbing, simulated annealing, tabu search, genetic algorithms, evolutionary strategies, and genetic programming.

2.3.1 Classical Techniques

2.3.1.1 Linear Programming

Linear programming (LP) can be used as mathematical optimization technique to find out optimum solution. The inputs are n real numbers which are called decision variables. Here the goal is to maximize the value of linear expression in these decision variables with the set constraint [13].

2.3.1.2 Integer Programming

Integer programming (IP) is a type of linear programming in which all variables contain integer values only [28]. IP problems can be classified into *Pure Integer IP Problem*, *Mixed Integer IP Problem*, and *Zero-One IP Problem* [28]. Integer programming allows depicting discontinuous decision variables. It is used to model fixed cost, logical conditions, and discrete level of resources.

2.3.1.3 Quadratic Programming

Quadratic programming (QP) is also a type of mathematical optimization problem in which quadratic function of several variables is optimized (minimized or maximized) where there are linear constraints on these variables. An optimization problem which is linearly constrained and objective function is quadratic, is called a quadratic program [29].

2.3.1.4 Non-Linear Programming

The objective function of some real-world problems may not be linear or some of the constraints may be non-linear. Such problems are called non linear programming (NLP) problems. Applications of non-linear programming include resource allocation, production planning, computer-aided design, modeling human or organizational behavior, or data networks.

2.3.2 Metaheuristic Search Techniques

2.3.2.1 Hill Climbing

The search starts from a randomly chosen point by considering the neighbourhood. Every neighbour is checked for fitness. A move is made if it improves fitness and neighbour is selected if there is increase in fitness [13]. Search terminates if no neighbour qualifies for fitness [14]. Each variable is changed one a time to get better results. The process ends if all the possible combinations of variables are checked and results are worse or same as the current one.

There is a problem with hill climbing approach that is, the hill located may be local maxima which may not meet fitness criteria than the global maxima in search space [13]. A local maximum is a small hill on the surface whose peak is lower than the main peak. If local maximum is found, we're stuck in it because any small move in any direction degrades fitness.

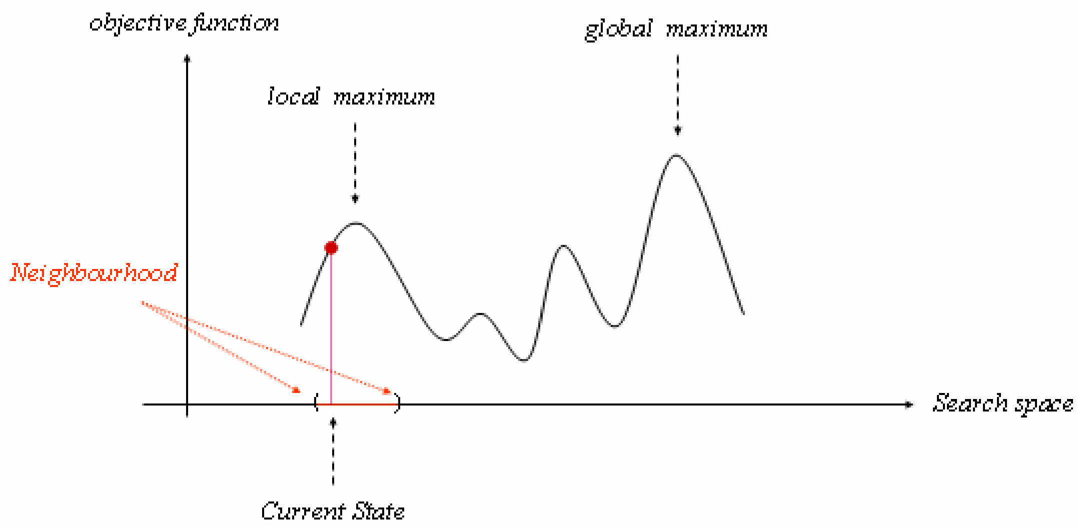


Figure 2.1: Hill Climbing Terminology.

A *move* defines the neighbourhood function, in which new solution is generated by changing one or more attributes of a given solution.

Algorithm:

1. Pick an initial state.
2. Consider all the neighbours of the current state.
3. Choose the neighbour with the best quality and move to that state.
4. Repeat 2 thru 4 until all the neighbouring states are of lower quality.
5. Return the current state as the solution state.

Hill climbing approach can be used in two ways [14]; In *simple hill climbing*, the first optimum neighbour is selected. In *steepest ascent hill climbing* all neighbours are compared and the individual with the best fitness is selected. Both forms fail if search space contains local maxima which are not the solutions.

2.3.2.2 Simulated Annealing (SA)

It uses method of local search for optimization. It considers s' as neighbouring value of s and its cost is evaluated. Minimizing the objective function becomes cost function and maximizing the objective function becomes fitness function [14]. Simulated annealing reduces the probability of making undesirable moves. SA heuristic probabilistically decides whether move the system to the state s' or keep staying in state s. This is repeated until the desired state is reached, or a complete iteration produces no change to current state, or given computational budget has been consumed [14].

SA is based on physical process of annealing a metal to get the best state. If a metal is cooled slowly, this reduces the probability of unfavourable moves of molecules hence it forms into a smooth piece [13]. If a metal is cooled too fast, the metal will form a shape having bumps and jagged edges representing the local minimums and maximums.

New solutions are only accepted if they are better than or equal to current solution. On reducing the cost function of s', the search moves to s' and the process is repeated. However, if the cost function increases, the move to s' is not necessarily rejected; there is a small probability p that the search will move to s' and continue [14].

$$p = e^{-(cost(s')-cost(s))/T}$$
$$p = e^{-\Delta E/T}$$

where ΔE represents energy and T represents temperature [14]. The probability depends on the values of energy and temperature. The negative change in cost function means it is improvement. In this case probability is considered as 1 and move is made. If ΔE is positive, it is considered as unfavorable and the move is considered as accepted with the probability given in the above equation [14].

Algorithm:

1. Start by generating initial solution. Initialize a very high *temperature*.
2. Select a neighbour.
3. Calculate the change in the score due to the move.
4. Depending on the change in fitness, accept or reject the move.
5. Update the temperature value by lowering the temperature.
6. Repeat 2 thru 5 until *freezing point* is reached.
7. Return the current state as the solution state.

2.3.2.3 Tabu Search (TS)

Tabu search is also a method of local search. In TS, all solution space is searched to obtain global optimal solution. Some of the moves are declared as forbidden and some are aspirant [14]. Aspirant moves might lead to global optimal solution unlike forbidden moves. Set of forbidden moves is also called as tabu set [14]. Tabu set also helps in performing more extensive exploration by moving search to the new portions of the search space.

Recording complete solutions requires a lot of storage hence expensive to check whether a potential move is tabu or not. The common practice is to record the last few transformations performed on the current solution in order to prevent reverse transformations.

Algorithm [14]:

1. Generate initial candidate s.
2. Determine neighbourhood set N.
3. Identify tabu set from neighbour.
4. Identify aspirant set from neighbour.
5. Choose the move with best improving solution s' in N.
6. Set $s=s'$.
7. Repeat steps 2 thru 6 until terminating condition is met.

2.3.2.4 Genetic algorithms (GA)

Genetic Algorithms move around the concept of population and recombination. A set of candidate solutions constitute the population whereas recombination is the process of combining and mutating the candidates to generate new solutions [14]. Some fitter function is used for recombination. The population is chosen randomly and iterative process is started. In GA, iterations are named as *generations* and the term *chromosomes*, is used to represent members of population [13]. The optimization process terminates if some pre-set criteria is satisfied or number of iterations are completed. Members of population are recombined on every generation to generate new population by using the fitness function. The candidates with best fitness values are likely to be selected for recombination [13].

Algorithm [23]:

1. Represent the problem variable domain as a chromosome of fixed length; choose the size of the chromosome population N, the crossover probability P_c and the mutation probability P_m.
2. Define a fitness function to measure the performance of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction.
3. Randomly generate an initial population of size N: $x_1, x_2, \ldots, x_N$.
4. Calculate the fitness of each individual chromosome: $f(x_1), f(x_2), \ldots, f(x_N)$.
5. Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness. High fit chromosomes have a higher probability of being selected for mating than less fit chromosomes.
6. Create a pair of offspring chromosomes by applying the genetic operators.
7. Place the created offspring chromosomes in the new population.

8. Repeat Step 5 until the size of the new population equals that of initial population N.
9. Replace the initial (parent) chromosome population with the new (offspring) population.
10. Go to Step 4, and repeat the process until the termination criterion is satisfied.

A cost function is applied on input variables (a chromosome) to generate an output. The cost function may be some mathematical function [16]. The objective is to modify the output in some desirable fashion by finding the appropriate values for the input variables. The term fitness is extensively used to designate the output of the objective function [16].

2.3.2.5 Evolutionary Strategies (ES)

This is an alternative form of GA and not widely applied on SBSE [13]. Iteration is named as *generation* [17]. Best individuals are used to create new population during each generation. [17].

The objective function describes the fitness value of each member in the population. The individual or solution having higher fitness value is considered as best solution. ES uses selection operator, mutation operator and recombination operator to evolve solutions [17]. In ES, individuals with best fitness values are involved in reproduction. New generation is produced by selecting individuals with best fitness values from the previous generation [18]. Mutation prevents falling GA into local maxima. If the change is beneficial to the general population then that individual will tend to survive and participate in the future generation processes. If the change causes a weakness then it is likely the individual will be discarded [18].

Best individuals are recombined to produce new offspring which shares many of the characteristics of their parents [18]. Again new parents are selected for each new child, and this process continues until a desired fitness value is achieved.

Algorithm [18]:

1. Collect an initial population of N individuals randomly.
2. Generate K offspring, where each offspring is generated as:
   - Select P parents from N
   - Recombine the P parents to form a new individual I.
   - Apply mutation operator to the strategy parameter to adapt it.
   - Apply the mutation operator to the individual I using the updated strategy parameter.
3. Select new parent population consisting of N best individuals from the pool of N and K.
4. Go to step 2 until termination condition occurs.

2.3.2.6 Genetic Programming (GP)

This is a variation of GA. In genetic programming, chromosome is like a tree instead of list [12], [13]. Genetic programming is used in SBSE to formulate predictive models of software projects [13]. The idea is to develop a program to solve the particular problem.

The main difference between genetic algorithms and genetic programming is how the solution is represented. Genetic algorithms create a list of numbers that represent the solution [13]. Genetic programming creates computer programs as the solution. The individuals in genetic programming are the programs developed in LISP or in some other artificial intelligence language [16], [26].

Algorithm [26]:

1. Generate an initial population of random created computer programs.
2. Execute each program in the population and assign it a fitness value according to how well it solves the problem.
3. Create a new population of computer programs.
    - Copy the best existing programs.
    - Create new computer programs by mutation or crossover.
4. The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming.

2.3.2.7 Ant Colony Optimization (ACO)

The Ant Colony Optimization (ACO) metaheuristic proposed by M. Dorigo is based on the cooperating behaviour of real ants to solve optimization problems [33]. Some of the applications of ACO are combinatorial optimization, scheduling, networking and communication, and assignment [22], [33]. An ant colony is able to find the shortest path to the food sources by using a very simple communication method. The ant colony has access to the food source though different paths from the colony's nest. During the trips, a chemical trail (pheromone) is left on the ground. The role of this trail is to guide the other ants toward the target point [33]. The larger the amount of pheromone on a particular path, the larger is the probability that the ants will select the path [33]. This chemical substance has a decreasing action over time. This decrease over time can is called as evaporation process [33].

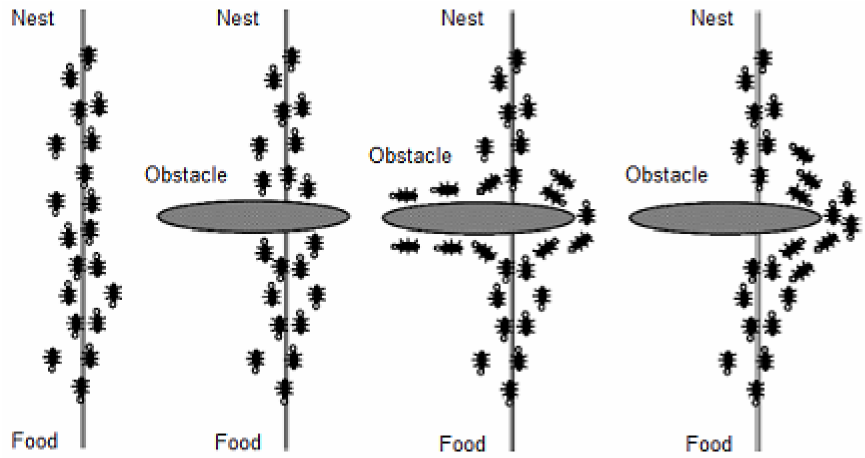


Figure 2.2 [33]: Ant colony searching an optimal path between the food and the nest.

Algorithm [33]:

```
Initialize the pheromone trails ;
Repeat
   For each ant Do
      Solution construction using the pheromone trail ;
      Update the pheromone trails:
         Evaporation ;
         Reinforcement ;
Until Stopping criteria
Output: Best solution found or a set of solutions.
```

2.3.2.8 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a nature inspired optimization algorithm introduced by Eberhart and Dr. Kennedy in 1995 [30]. It is a probabilistic metheuristic search technique based on social behavior of bird flocking and fish schooling [21], [33]. In PSO, particles are called potential solutions. These particles fly through the problem space by following the best positions found by neighbour particles and by themselves. Every particle keeps the record of its best position achieved so far. This is particle's personal best value and called as *pbest* [21]. There is another value *gbest* or *lbest*. *gbest* (global best) is the best position obtained so far by any particle in the swarm [21] whereas lbest (local best) is the position for a given subset of the swarm [33]. Swarm is similar to population as in Genetic Algorithms and particle is analogous to an individual [31]. PSO is considered as collective and iterative method due to its emphasis on cooperation [20].

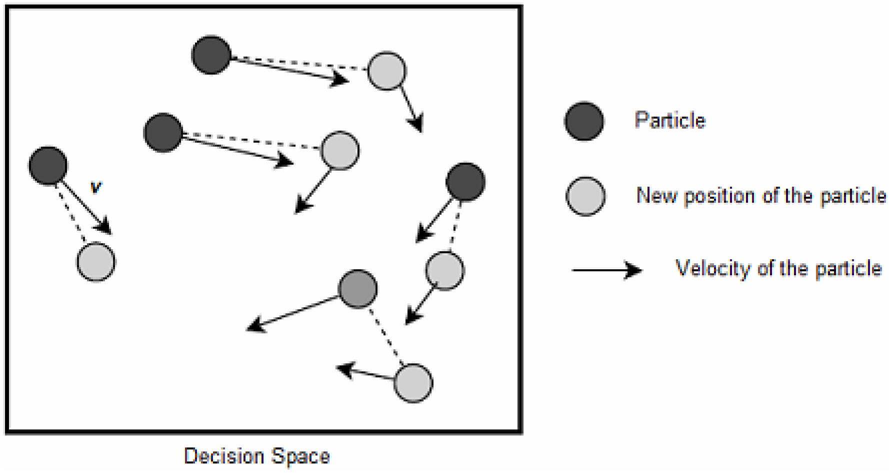


Figure 2.3 [33]: Particle swarm with their positions and velocities.

PSO can be mapped to continuous as well as discrete search space values. In PSO, each particle has position and velocity. Initially positions and velocities of all the particles are randomly initialized [23]. On each iteration, first velocity of the particle is updated and then its position. The PSO algorithm use *pbest* and *gbest* (or *lbest*) for adjusting the velocity of the particle. This process continues until desired fitness level is achieved. In other words, PSO algorithm is mainly composed of three steps; velocity update, position update, and fitness calculation until desired convergence level is achieved.

Traditionally there are two methods to define particles neighbourhood [21], [33]. In ***gbest*** method, the neighbourhood is defined as the whole swarm of particles. On the other hand, in the ***lbest*** method, the neighbourhood of a particle is defined by several fixed particles. In other words we can say there are multiple *lbest* in the swarm. The *gbest* offers faster rate of convergence but it is not robust [19]. The *gbest* particle attracts all the particles towards itself. Using only the *gbest* in velocity update process, may lead to premature convergence of swarm. The *lbest* prevents premature convergence because many *lbest* positions are kept. In other words there are many attractors [19].

According to the neighbourhood, *a leader* (lbest or gbest) represents the particle that is used to guide the search of a particle toward better regions of the search space [33].

Equations 1 and 2 are used to update velocity and position of the particle [22], [23], [24] as described in Figure 1.

$$v(t + 1) = vt + (c1r1(pbest - xt)) + (c2r2(gbest - xt)) \tag{1}$$

$$x(t + 1) = xt + v(t + 1) \tag{2}$$

Where c1 and c2 are self confidence factor and swarm confidence factor respectively [25]. c1 and c2 are also called acceleration coefficients [19]. The parameter c1 is the cognitive learning factor that represents the attraction that a particle has toward it own process [34]. The parameter c2 is the social learning factor that represents the attraction that a particle has toward the success of its neighbours [34]. r1 and r2 are uniform random numbers in the range [0,1].

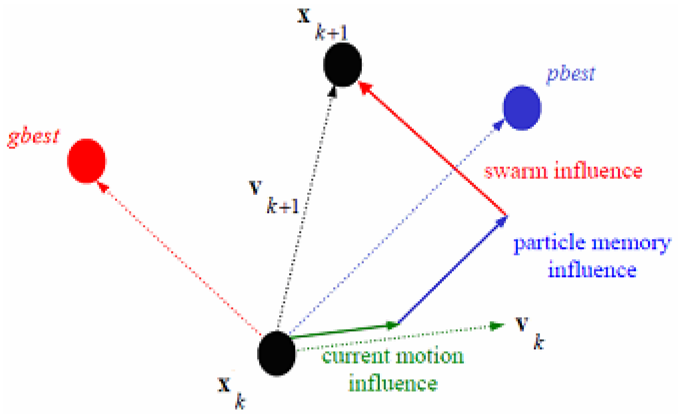


Figure 2.4 [25]: Depiction of the velocity and position updates.

Equation 2 is used to update position in continuous PSO. This version of PSO is also called as real-valued PSO [24] because velocities and positions are represented using real values. The other version is the discrete version also proposed by Kennedy and Eberhart [24], [32] in which velocity is used to make Boolean decision. This is called Binary PSO and used to solve binary problems. In binary version of PSO, new position of the particle is decided using sigmoid function [24].

$$x(t+1) = \begin{matrix} 1 \quad if \quad r < sig(v(t+1)) \\ 0 \quad otherwise \end{matrix} \tag{3}$$

Where r is uniform random number in the range [0,1].

$$sig(v(t+1)) = 1 / (1+exp(-v(t+1))) \tag{4}$$

Algorithm [23]:

1. Randomly initialize velocities and positions of all particles.
2. On each iteration, update velocities of all the particles according to equation 1.
3. Update positions of all the particles using equations 2 and 3 for Continuous PSO and Binary PSO respectively.
4. Update *pbest* and *gbest* when condition is met.

$$pbest = x(t+1) \qquad if \;\; x(t+1) > pbest$$

$$gbest = x(t+1) \qquad if \;\; x(t+1) > gbest$$

5. Repeat steps 2 to 4 until certain termination conditions are met, such as a pre-defined number of iterations, desired fitness value is achieved, or failure to make progress for a certain number of iterations.

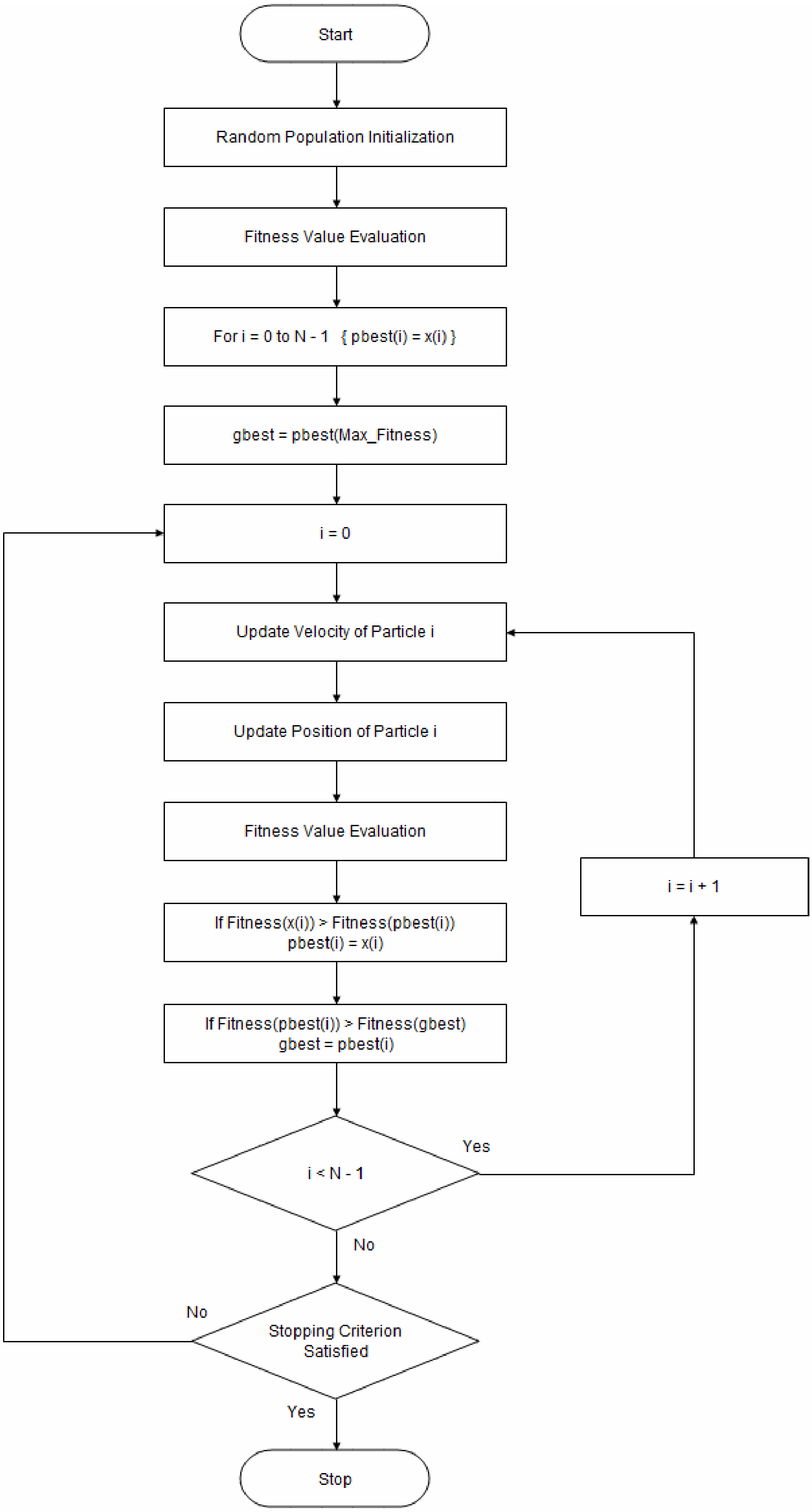


Figure 2.5: PSO Flowchart.

2.4 COMPARISON BETWEEN GA AND PSO

PSO is analogous to GA in many aspects. Both starts with the randomly generated population and there is some fitness function to evaluation the population [23]. In contrary to GA, PSO does not support crossover and mutation. On iteration in PSO, velocity of the particle is updated and thus particle occupy some memory to hold best position [22]. Information sharing mechanism is also different in PSO. In PSO, only *gbest* and *lbest* share information with others whereas in GA, chromosomes share information with each other [23]. The advantages of PSO include easy to implement and small number of parameters to adjust [23].

2.5 SUMMARY

In this chapter a background study on software clustering was presented. The presented concepts form the foundations of the project. Details regarding PSO have been discussed in detail. The “Application of SBSE Techniques for Hierarchical Software Clustering” project has been studied for its dedications towards providing solution to software clustering problem. Algorithms, techniques and various directions that have been discussed form the foundation of the research work in the project.

Chapter 3

DESIGN & IMPLEMENTATION

INTRODUCTION

This chapter presents the architectural design and implementation details of the PSO on software clustering problem. This system has been designed to meet the highest levels of usability. All the required information such as fitness values, the iteration numbers, and the time span are shown on the screen. The solution is presented in the form of MOJO files.

3.1 CLASS DIAGRAM

The following figure represents the class diagram of proposed system.

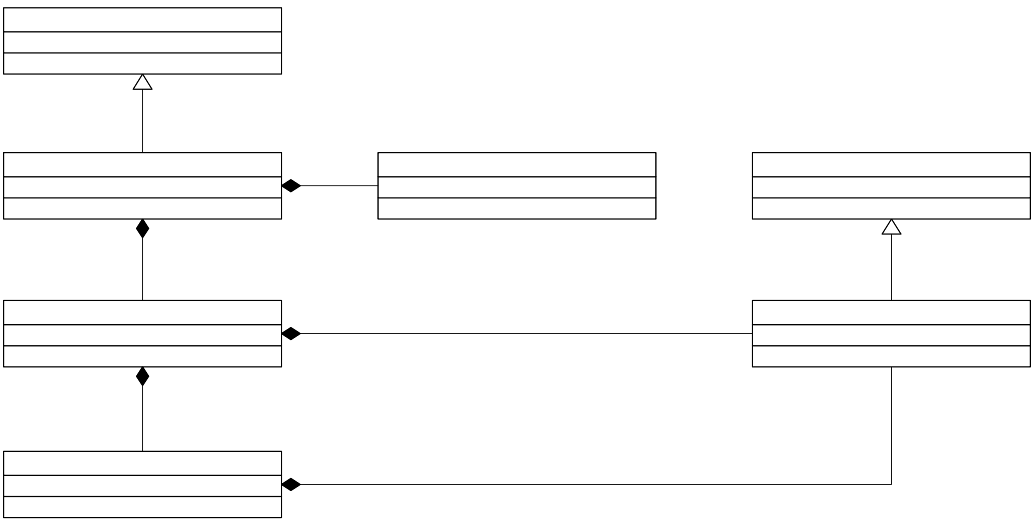


Figure 3.1: Class Diagram.

3.1.1 CDialog

The MFC class used as base class to display dialog boxes on screen.

3.1.2 CSCPSODlg

This class is derived from CDialog class and is used to provide user interface functionality. Operations of this class include facts file selection, accept cluster size, start software clustering process, and display fitness value with iteration no. and time.

3.1.3 CPersistence

This class handles all the file related operations. File operation includes reading facts file and writing solution file.

3.1.4 CParticleSwarm

This acts as manager class. It creates population of particles and updates their velocities and positions on each iteration. It also keeps global best solution obtained during position update process.

3.1.5 CParticle

This class represents a particle. Collection of this class is used to represent all particles in the population. Operations of this class include particle initialization, update velocity and position, Binarize position, and fitness value calculation on new position.

3.1.6 CBSVector

CBSVector is a template class of sequence containers that arrange elements of a given type in a linear arrangement and allow fast random access to any element.

3.1.7 CBSMatrix

A collection class derived from CBSVector to store facts file, velocities and positions of the particles. This class is also used to store local and global best position for each particle.

3.2 USER INTERFACE

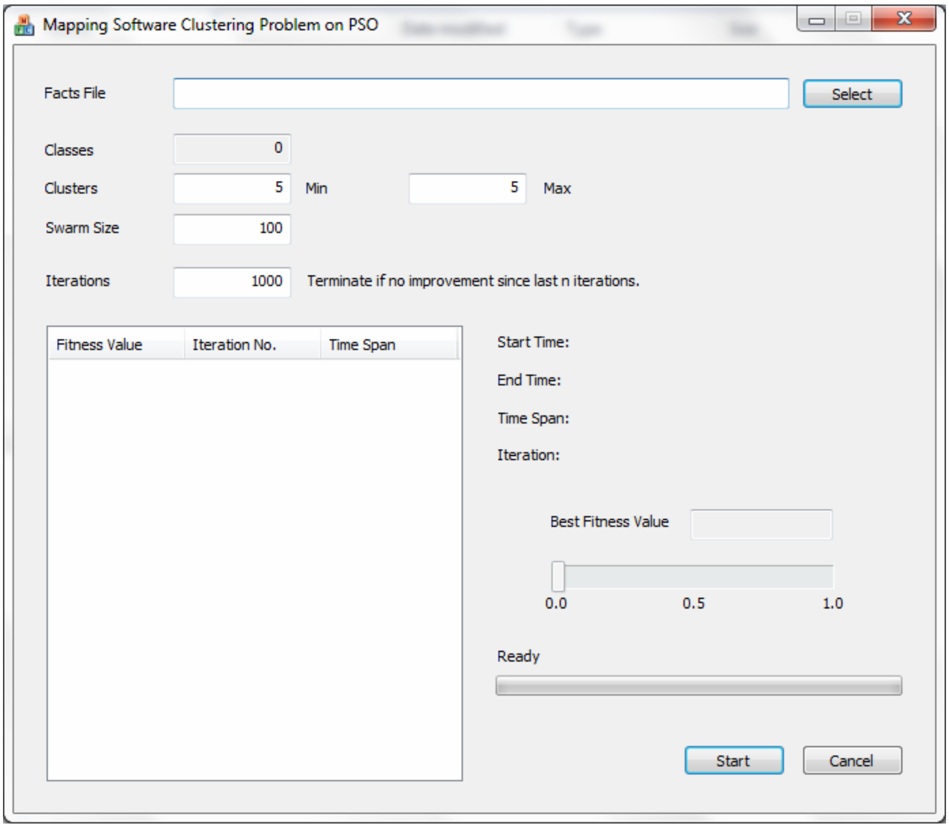


Figure 3.2: User Interface.

3.2.1 Description of User Interface Items

Facts File	Input is matrix of size n x n. Where n is the number of classes. This vector contains relationship strengths between classes and the file is stored in text format.
Classes	No. of classes in the Test System
Clusters	No. of clusters to be created or no. of sub-systems to be formed.
Swarm Size	No. of particles.
Iterations	This is termination criteria. If no improvement since last n iterations, the process is terminated. Where n is the no. of iterations to be performed.
List (Fitness Value, Iteration No., Time Span)	During optimization, It shows best fitness value along with the iteration no. and time.
Start Time	Start time of the optimization process.
End Time	End time of the optimization process.
Time Span	Difference between End Time and Start Time.

Iterations	Total no. of iterations performed.
Best Fitness Value	Best fitness values on process completion.
Start button	Used to start software clustering process.
Cancel button	Used to cancel software clustering process.

Table 3.1: Description of UI Items.

3.3 INPUT FILE DESCRIPTION

Input is the Facts file which is a matrix of size n x n. Where n is the number of classes. Facts file contains relationship strengths between classes.

		Classes						
		0	1	2	3	4	5	6
	0	0	3	2	1	7	1	0
	1	0	0	4	5	0	2	1
	2	0	0	0	3	0	0	1
Classes	3	0	0	0	0	2	1	0
	4	0	0	0	0	0	0	2
	5	0	0	0	0	0	0	1
	6	0	0	0	0	0	0	0

Figure 3.3: Tabular Representation of Facts File.

Suppose there are seven classes in the test system. Names are A, B, C, D, E, F, and G. They are represented with numeric ids in the range 0 ~ 6.

	0	1	2	3	4	5	6
Classes	A	B	C	D	E	F	G

Figure 3.4: Class Ids.

3.4 OUTPUT FILE DESCRIPTION

Output is the Solution file. The Solution file is a text file with "mjo" file extension. The file contains the list of clusters or sub-systems and the classes contained in them.

The structure of the Solution file is,

```
contain ss1 cls01
contain ss1 cls02
contain ss1 cls04
contain ss2 cls03
contain ss2 cls05
contain ss3 cls06
contain ss3 cls07
```

ss1 represents sub-system 1 and cls01 represents class 1.

The classes which are more related to each other are placed in the same sub-system. Graphically we can represent solution as:

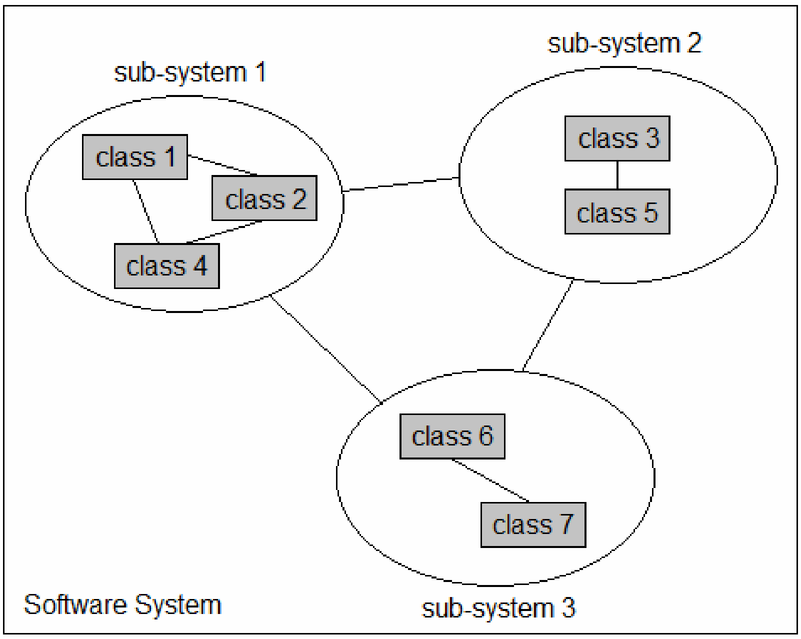
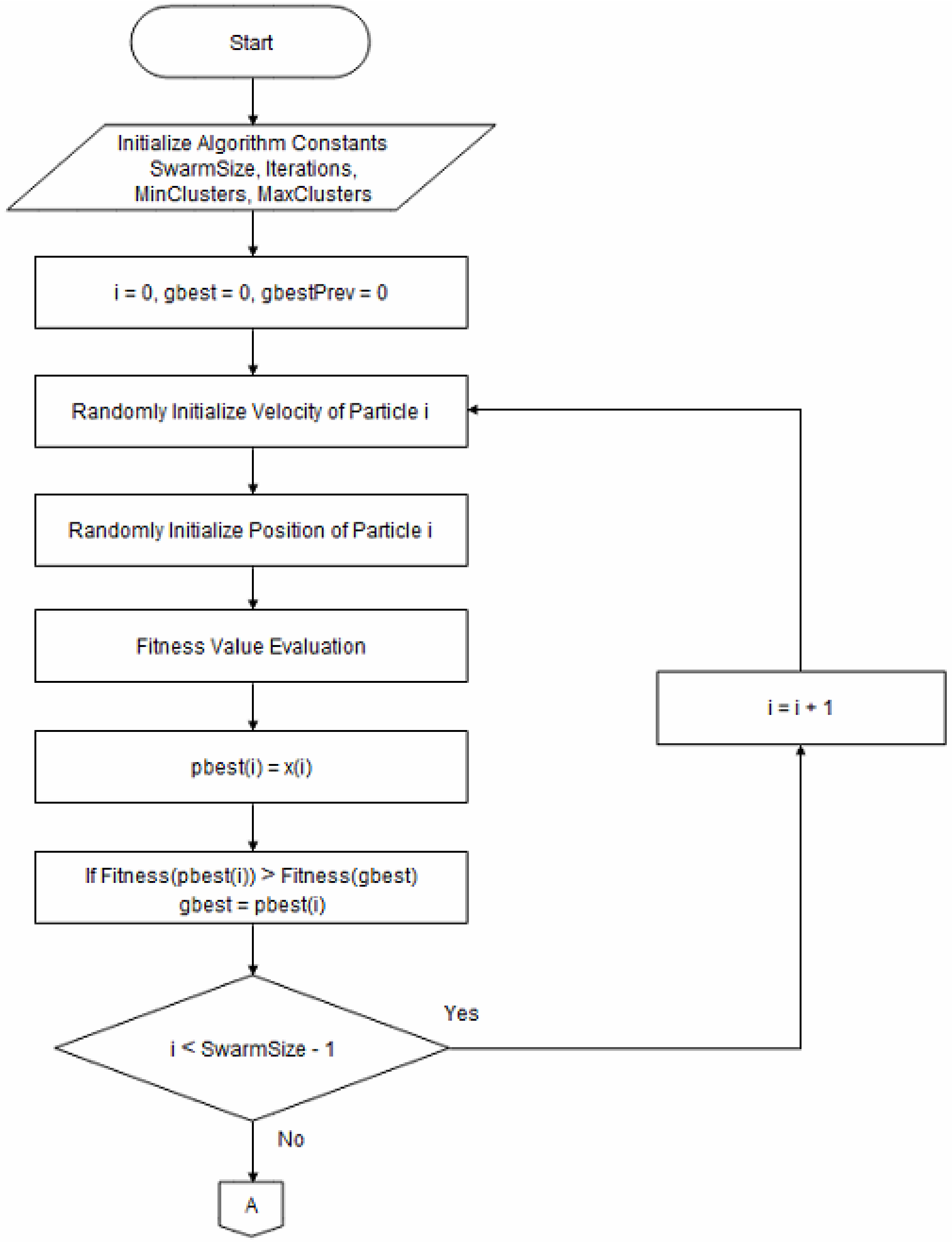


Figure 3.5: Graphical representation of the solution.

Class 1, 2, and 4 are placed in one sub-system because they are strongly related to each other. Similarly class 3 and 5 are placed on one sub-system. Class 2 may have some relationships with class 5 but those relationships are not as much strong as it relationships with class 1 and class 4.

3.5 PSO ALGORITHM

The Standard Binary version of PSO is implemented using *gbest* neighbourhood method.



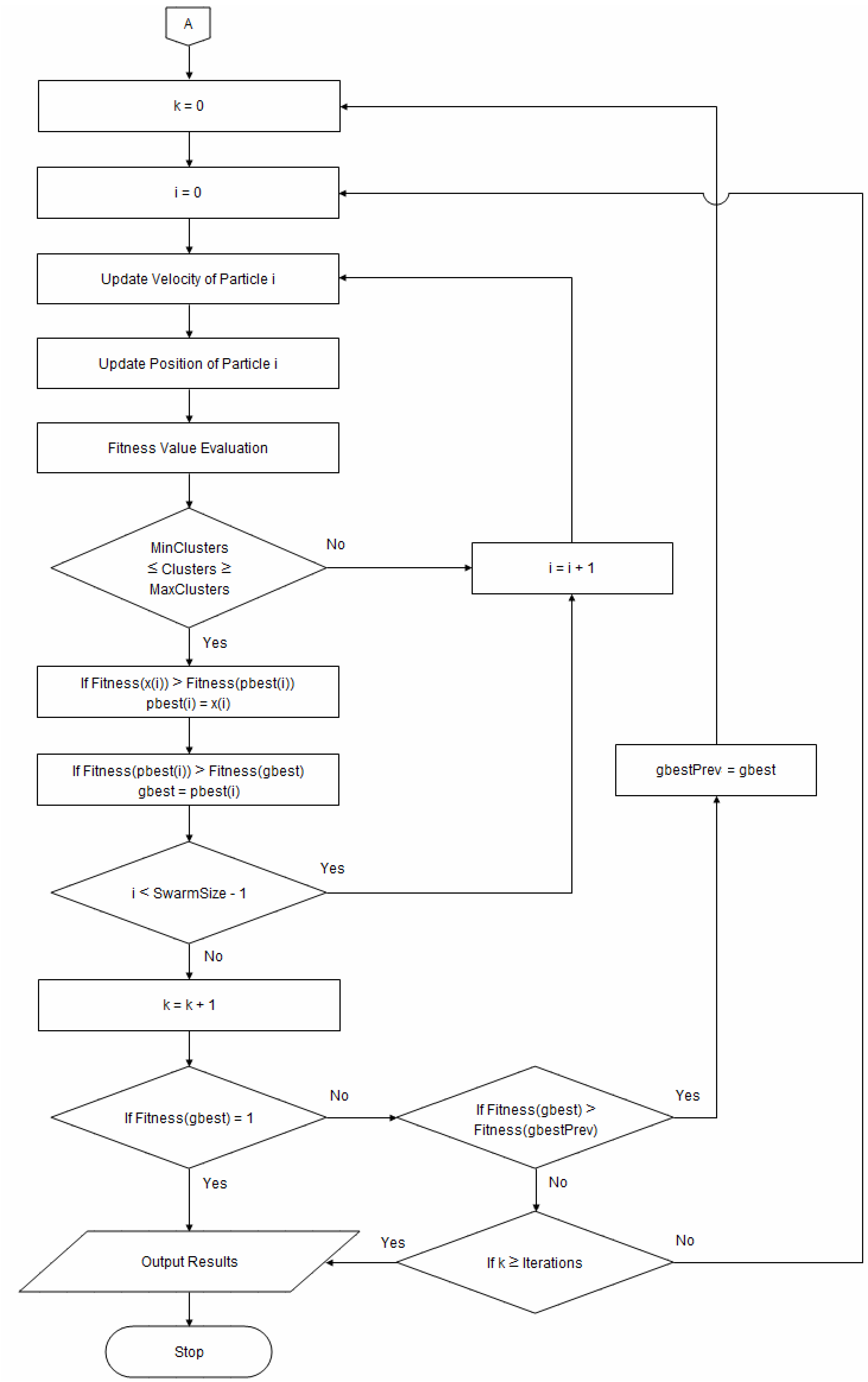


Figure 3.6: Implemented PSO Algorithm.

3.6 MAPPING SOFTWARE CLUSTERING PROBLEM ON PSO

3.6.1 Problem Formulation

A particle in a swarm is represented by n x m matrix where n is the number of clusters to be formed and m is the number of classes in test system. Each particle is a candidate solution to the clustering problem. The collection of particle is called a swarm which is analogous to population in genetic algorithms. Following swarm size is used.

Swarm Size = Number of classes x 10

3.6.2 Fitness Calculation

Every new position of the particle indicates a possible solution. The quality of the solution is evaluated using Modularization Quality (MQ) [8]. The MQ is designed based on the assumption that sub-systems in a good software system are highly cohesive [8]. The modularization quality differentiates between good and bad decompositions.

$$MQ = \sum_{i=1}^{k} CF_i \tag{5}$$

Where CF_i is Component Factor for i^{th} cluster; it is computed from cohesion and coupling among classes in the clusters [15].

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \dfrac{\mu_i}{\mu_i + \frac{1}{2} \sum\limits_{\substack{j=1 \\ j \neq i}}^{k} (\varepsilon_{i,j} + \varepsilon_{j,i})} & otherwise \end{cases} \tag{6}$$

μ represents intra-clusters relationships. In other words, μ_i is the cohesion of i^{th} cluster. ε represents inter-cluster relationships. $\varepsilon_{i,j}$ and $\varepsilon_{j,i}$ denote coupling between i^{th} cluster and any other cluster j.

TurboMQ is the normalized form of MQ [15]. It is achieved by dividing MQ by total number of clusters to keep values in the range 0 and 1.

$$TurboMQ = MQ / k \tag{7}$$

Where k is the total number of clusters.

Let us consider the following example to calculate sum of intra-edges (μ) and sum of inter-edges (ε).

	0	**1**	**2**	**3**	**4**	**5**	**6**
Sub-systems	**S0**	**S0**	**S1**	**S0**	**S1**	**S2**	**S2**

Here S0, S1, and S2 are the 3 clusters or we can call them sub-systems. Cluster S0 contains classes 0, 1, and 3. Cluster S1 contain classes 2 and 4. Class 5 and 6 belongs to cluster S2.

To compute values of μ and ε, we have to read relationships strengths from Facts file.

		Classes						
		0	**1**	**2**	**3**	**4**	**5**	**6**
	0	0	3	2	1	7	1	0
	1	0	0	4	5	0	2	1
	2	0	0	0	3	0	0	1
Classes	**3**	0	0	0	0	2	1	0
	4	0	0	0	0	0	0	2
	5	0	0	0	0	0	0	1
	6	0	0	0	0	0	0	0

Graphically we can represent classes and the relationship strengths between them as:

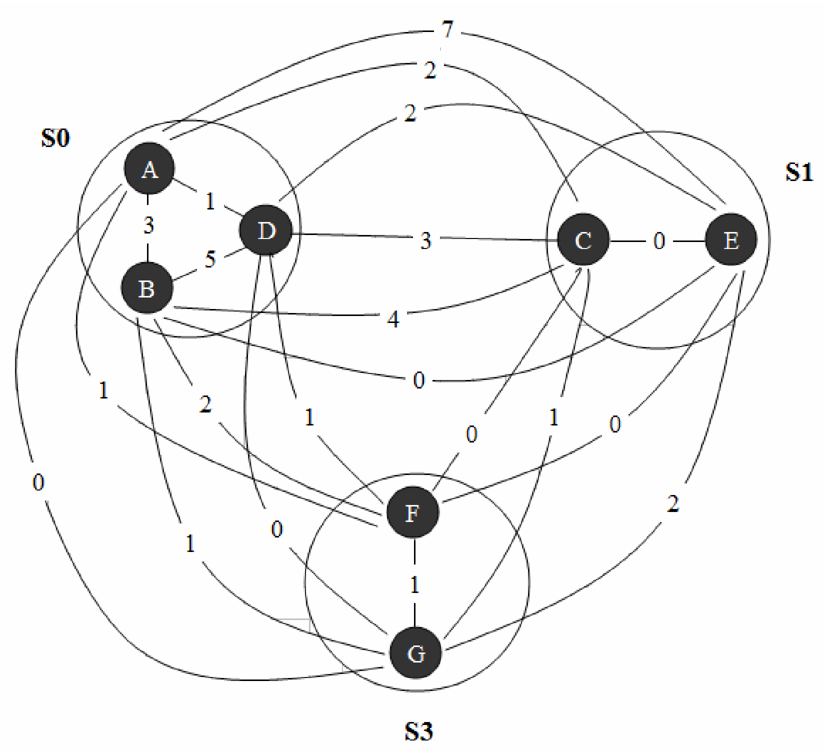


Figure 3.7: Intra Edges and Inter Edges.

$$
\begin{aligned}
\Sigma \text{ Intra Edges} &= IAG = S0 + S1 + S2 \\
&= (3+1+5) + 0 + 1 \\
&= 10
\end{aligned}
$$

$$
\begin{aligned}
\Sigma \text{ Inter Edges} &= IEG = S0S1 + S1S2 + S0S2 \\
&= (7+2+2+3+4+0) + (0+1+0+2) + (0+1+1+2+0+1) \\
&= 18 + 3 + 5 \\
&= 26
\end{aligned}
$$

3.6.3 Termination Criteria

The optimization process terminates if there is no improvement in the fitness value since last 1000 (one thousand) iterations OR the fitness value reached to 1.

3.6.4 Test Environment

Tests are performed on the following environment.

- Windows 7 Professional 32-bit
- 2.26GHz Intel Core i3
- 3 GB RAM

3.7 SUMMARY

In this chapter the architecture and implementation of the newly proposed system have been described in detail. Complete Parallel Binary PSO algorithm is presented. Mapping software clustering on PSO, fitness value calculation, and termination criteria are also described.

Chapter 4

TESTING & EVALUATION

INTRODUCTION

Parallel Binary PSO is tested using three test systems. Firstly, since the solution is proposed for efficient computation of results therefore it has to be evaluated for computational time. Secondly, the improvement in fitness value is checked. The proposed system cannot be tested and evaluated in total isolation. It has to be tested and evaluated in comparison with GA.

4.1 DESCRIPTION OF TEST SYSTEMS

4.1.1 Power Economic Dispatch System (PEDS)

This system is related to electrical power systems. It solves economic power dispatch problem using conventional and evolutionary computing techniques [15]. It uses MFC document viewer architecture and implements conventional and genetic algorithms.

S. No.	**Entity Related Information**	**Count**
1	Lines of source code	16360
2	Header files	31
3	Implementation files	27
4	Classes	41

Figure 4.1: Entities in PEDS Test System.

S. No.	Relationships	Count
Inheritance:		
1	Inheritance Depth	13
2	Same Inheritance Hierarchy	70
3	Virtual Method Overridden Count	6
Containment:		
4	Containment as Object	12
5	Containment as Pointer	9
6	Containment as Reference	0
7	Containment at Method Parameter Level	22
8	Containment at Method Local Declaration Level	29
9	Two Classes using Same Class at Class Level	12
10	Two Classes using Same Class at Method Level	76
Member Access:		
11	Data Member Access Count in Inheritance	43
12	Data Member Access Count in Containment	20
13	Method Access Count in Inheritance	35
14	Method Access Count in Containment	17
15	Both Classes Access Same Global Data	0
16	Both Classes Access Same Global Function	0
Files & Folders		
17	Both Contained in Same Source File	36
18	Both Source Files are in the Same Folder	0

Figure 4.2: Relationships in PEDS Test System.

4.1.2 Statistical Analysis Visualization Tool (SAVT)

This application helps statistical analysts in analysis and visualization of statistical data. It provides complete support of user interface for input and visualize along with the saving and loading data files.

S. No.	Entity Related Information	Count
1	Lines of source code	27311
2	Header files	70
3	Implementation files	76
4	Classes	97

Figure 4.3: Entities in SAVT Test System.

S. No.	Relationships	Count
Inheritance:		
1	Inheritance Depth	26
2	Same Inheritance Hierarchy	986
3	Virtual Method Overridden Count	21
Containment:		
4	Containment as Object	41
5	Containment as Pointer	41
6	Containment as Reference	0
7	Containment at Method Parameter Level	77
8	Containment at Method Local Declaration Level	153
9	Two Classes using Same Class at Class Level	1032
10	Two Classes using Same Class at Method Level	1900
Member Access:		
11	Data Member Access Count in Inheritance	100
12	Data Member Access Count in Containment	49
13	Method Access Count in Inheritance	83
14	Method Access Count in Containment	77
15	Both Classes Access Same Global Data	0
16	Both Classes Access Same Global Function	0
Files & Folders		
17	Both Contained in Same Source File	264
18	Both Source Files are in the Same Folder	0

Figure 4.4: Relationships in SAVT System.

4.1.3 Print Language Converter (PLC)

This application is a sub-system of a large software system. It provides conversion support from intermediate data structures to a well known printer language.

S. No.	Entity Related Information	Count
1	Lines of source code	51768
2	Header files	27
3	Implementation files	27
4	Classes	69

Figure 4.5: Entities in PLC Test System.

S. No.	Relationships	Count
Inheritance:		
1	Inheritance Depth	99
2	Same Inheritance Hierarchy	26
3	Virtual Method Overridden Count	26
Containment:		
4	Containment as Object	24
5	Containment as Pointer	12
6	Containment as Reference	0
7	Containment at Method Parameter Level	25
8	Containment at Method Local Declaration Level	69
9	Two Classes using Same Class at Class Level	58
10	Two Classes using Same Class at Method Level	162
Member Access:		
11	Data Member Access Count in Inheritance	87
12	Data Member Access Count in Containment	41
13	Method Access Count in Inheritance	92
14	Method Access Count in Containment	34
15	Both Classes Access Same Global Data	0
16	Both Classes Access Same Global Function	0
Files & Folders		
17	Both Contained in Same Source File	1812
18	Both Source Files are in the Same Folder	0

Figure 4.6: Relationships in PLC Test System.

4.2 EXPERIMENTAL SETUP

For each test system, we applied PSO and GA and performed comparison between decompositions of each algorithm. The comparisons are based on fitness values, rate of convergence, and computation time. Fitness values are in the range 0 ~ 1. Zero specifies the worst decomposition and while 1 indicates best decomposition [15]. Classes and clusters to be formed are listed in the following table:

4.2.1 Swarm Size

In this implementation of PSO, swarm size is taken as

Swarm Size = Number of classes x 10

Same is the population size in GA.

4.2.2 Number of Clusters

Same numbers of clusters as used in GA are used in PSO for comparison. The value for number of clusters is the mean of expert decompositions.

Test System	Alias	Classes	Clusters
1	PEDS	41	4
2	SAVT	97	8
3	PLC	69	4

Table 4.7: No. of Clusters.

4.2.3 Experimental Results

For each of three test systems, total ten readings are taken. Five readings using PSO and five are using GA. On the basis of these readings, comparative analysis is performed. Time is shown in “HH:mm:ss” format.

4.2.3.1 PEDS

#	PSO		GA	
	Fitness Value	Time	Fitness Value	Time
1	0.805339	0:02:15	0.773470	0:09:55
2	0.805339	0:02:12	0.770880	0:09:50
3	0.824819	0:03:24	0.858480	0:09:45
4	0.824819	0:03:00	0.773470	0:09:50
5	0.805339	0:02:11	0.858480	0:09:39

Table 4.8: Experimental Results of PEDS.

4.2.3.2 SAVT

#	PSO		GA	
	Fitness Value	Time	Fitness Value	Time
1	0.505032	1:04:17	0.474990	2:05:43
2	0.505032	1:03:11	0.434460	2:05:40
3	0.505032	1:03:14	0.461820	2:04:46
4	0.506467	1:04:56	0.435390	2:05:29
5	0.505032	1:04:51	0.492440	2:04:37

Table 4.9: Experimental Results of SAVT.

4.2.3.3 PLC

#	PSO		GA	
	Fitness Value	Time	Fitness Value	Time
1	0.688900	0:10:43	0.640430	0:42:21
2	0.688900	0:10:01	0.652140	0:43:24
3	0.688900	0:09:28	0.640430	0:43:22
4	0.688900	0:09:58	0.652140	0:44:42
5	0.688900	0:09:47	0.652140	0:45:39

Table 4.10: Experimental Results of PLC.

4.3 EVALUATION

4.3.1 Fitness Values

4.3.1.1 PEDS

There is 0.76% improvement in fitness value.

Reading No.	Fitness Values	
	PSO	GA
1	0.805339	0.773470
2	0.805339	0.770880
3	0.824819	0.858480
4	0.824819	0.773470
5	0.805339	0.858480
Average	**0.813131**	**0.806956**
Ratio	**0.992406**	
Percent	**0.765221**	

Table 4.11: Fitness Values of PEDS.

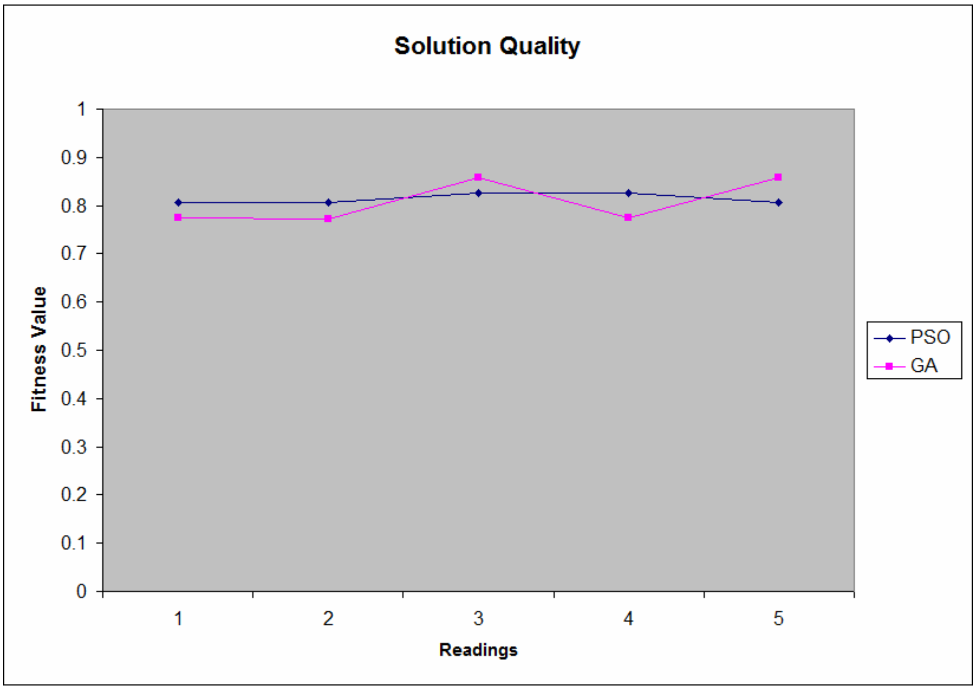


Figure 4.1: Graphical Representation of Solution Quality of PEDS.

4.3.1.2 SAVT

There is 9.89% improvement in fitness value.

Reading No.	Fitness Values	
	PSO	GA
1	0.505032	0.474990
2	0.505032	0.434460
3	0.505032	0.461820
4	0.506467	0.435390
5	0.505032	0.492440
Average	**0.505319**	**0.459820**
Ratio	**0.909960**	
Percent	**9.894959**	

Table 4.12: Fitness Values of SAVT.

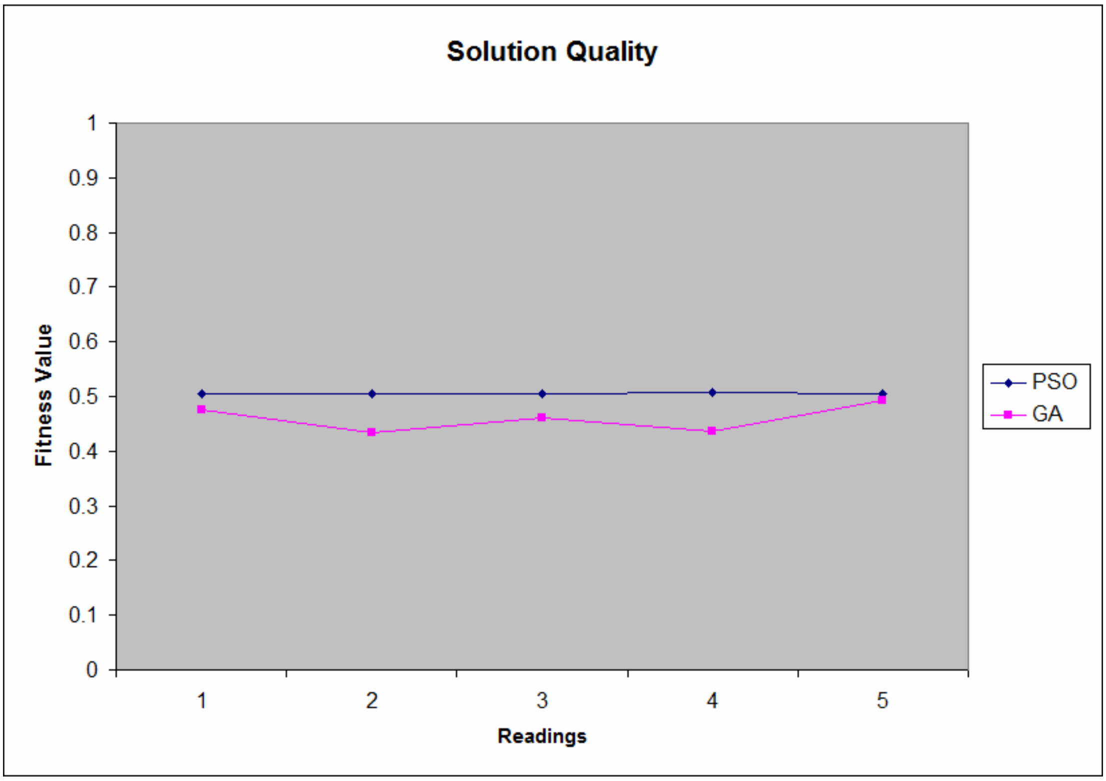


Figure 4.2: Graphical Representation of Solution Quality of SAVT.

4.3.1.3 PLC

There is 6.4% improvement in fitness value.

Reading No.	Fitness Values	
	PSO	GA
1	0.688900	0.640430
2	0.688900	0.652140
3	0.688900	0.640430
4	0.688900	0.652140
5	0.688900	0.652140
Average	**0.688900**	**0.647456**
Ratio	**0.939840**	
Percent	**6.401053**	

Table 4.13: Fitness Values of PLC.

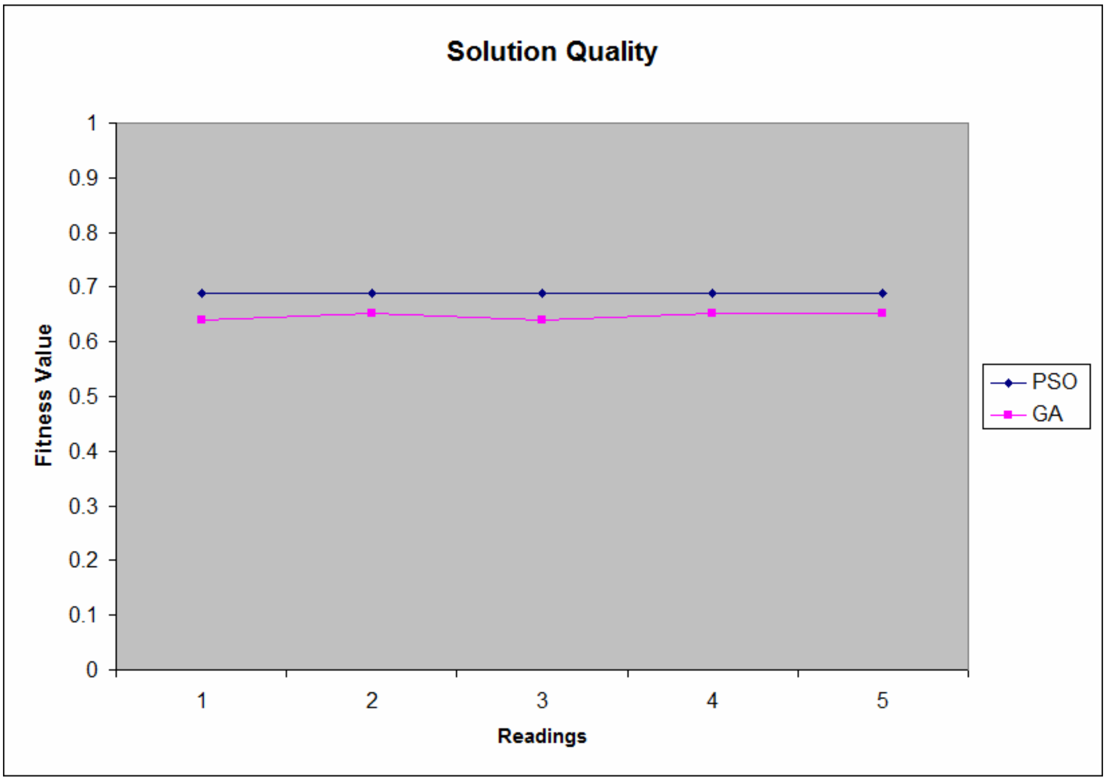


Figure 4.3: Graphical Representation of Solution Quality of PLC.

PSO quickly converged to maximum fitness value within first 500 iterations. Analyzing fitness values revealed that results of PSO are more stable and consistent because of the minimum variations in the fitness values. The reason behind is, in GA only best individuals are selected for reproduction whereas in PSO all the particles in swarm participate in velocity process and position update process. Reproduction can eliminate good solutions in GA while good solutions always survive into the next generation in PSO.

4.3.2 Computational Time

4.3.2.1 PEDS

There is 73.47% decrease in computational time.

Reading No.	Total No. of Iterations	
	PSO	GA
1	0:02:15	0:09:55
2	0:02:12	0:09:50
3	0:03:24	0:09:45
4	0:03:00	0:09:50
5	0:02:11	0:09:39
Average	**0:02:36**	**0:09:48**
Average (Sec)	**156.00**	**588.00**
Ratio	**3.77**	
Percent	**73.47**	

Table 4.14: Computational Time of PEDS.

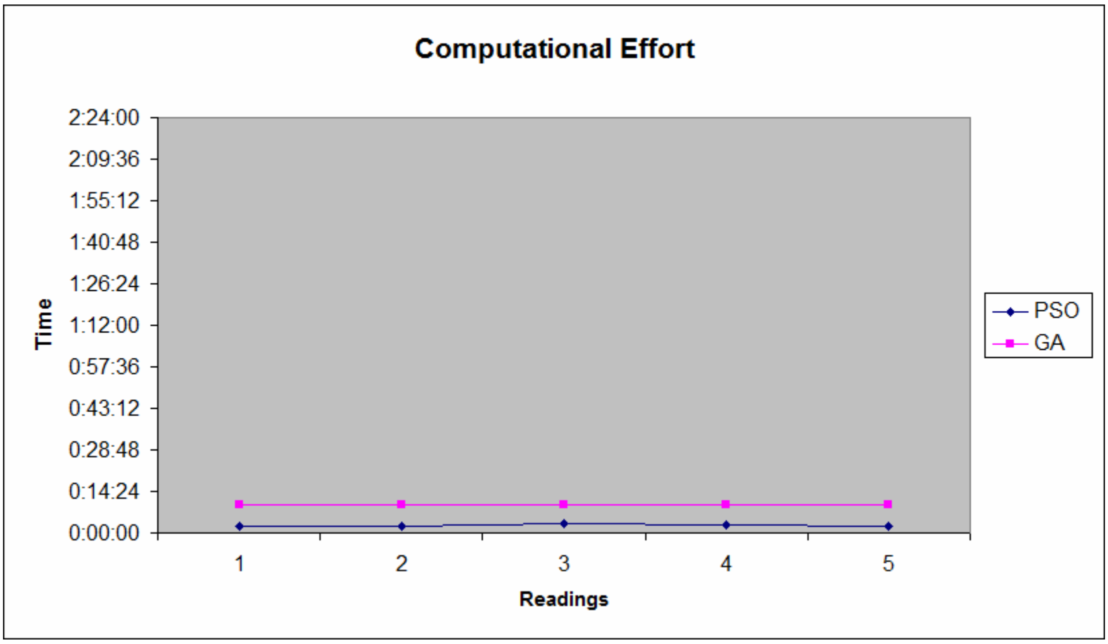


Figure 4.4: Graphical Representation of Computational Time of PEDS.

4.3.2.2 SAVT

There is 49.04% decrease in computational time.

Reading No.	Total No. of Iterations	
	PSO	**GA**
1	1:04:17	2:05:43
2	1:03:11	2:05:40
3	1:03:14	2:04:46
4	1:04:56	2:05:29
5	1:04:51	2:04:37
Average	**1:03:54**	**2:05:24**
Average (Sec)	**3834.00**	**7524.00**
Ratio	**1.96**	
Percent	**49.04**	

Table 4.15: Computational Time of SAVT.

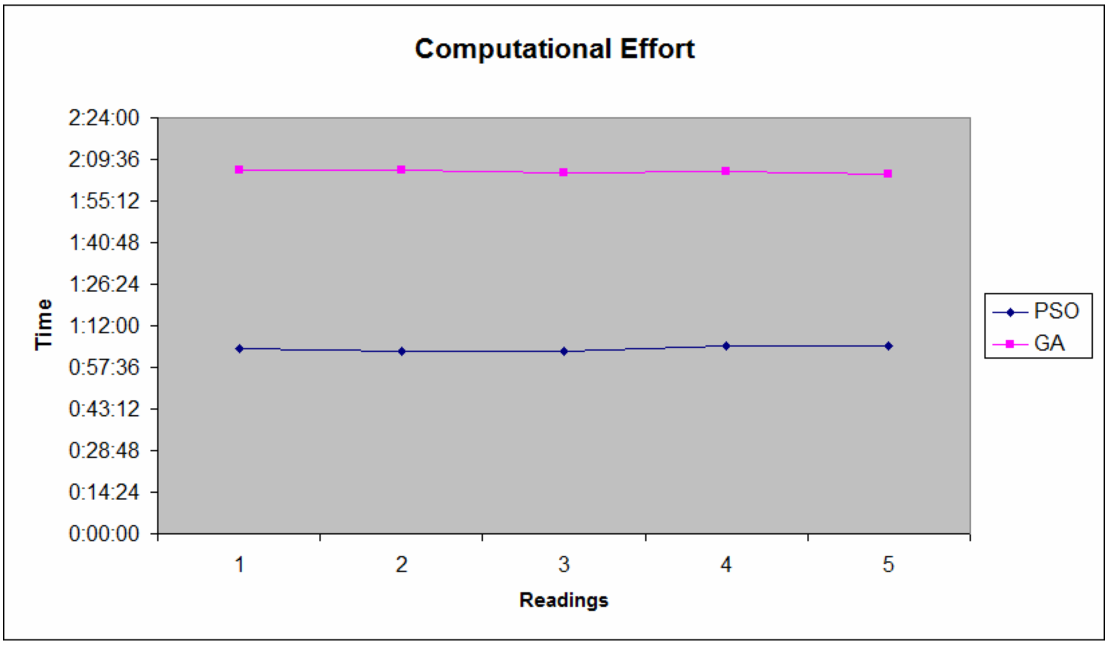


Figure 4.5: Graphical Representation of Computational Time of SAVT.

4.3.2.3 PLC

There is 77.26% decrease in computational time.

Reading No.	Total No. of Iterations	
	PSO	GA
1	0:10:43	0:42:21
2	0:10:01	0:43:24
3	0:09:28	0:43:22
4	0:09:58	0:44:42
5	0:09:47	0:45:39
Average	**0:09:59**	**0:43:54**
Average (Sec)	**599.00**	**2634.00**
Ratio	**4.40**	
Percent	**77.26**	

Table 4.16: Computational Time of PLC.

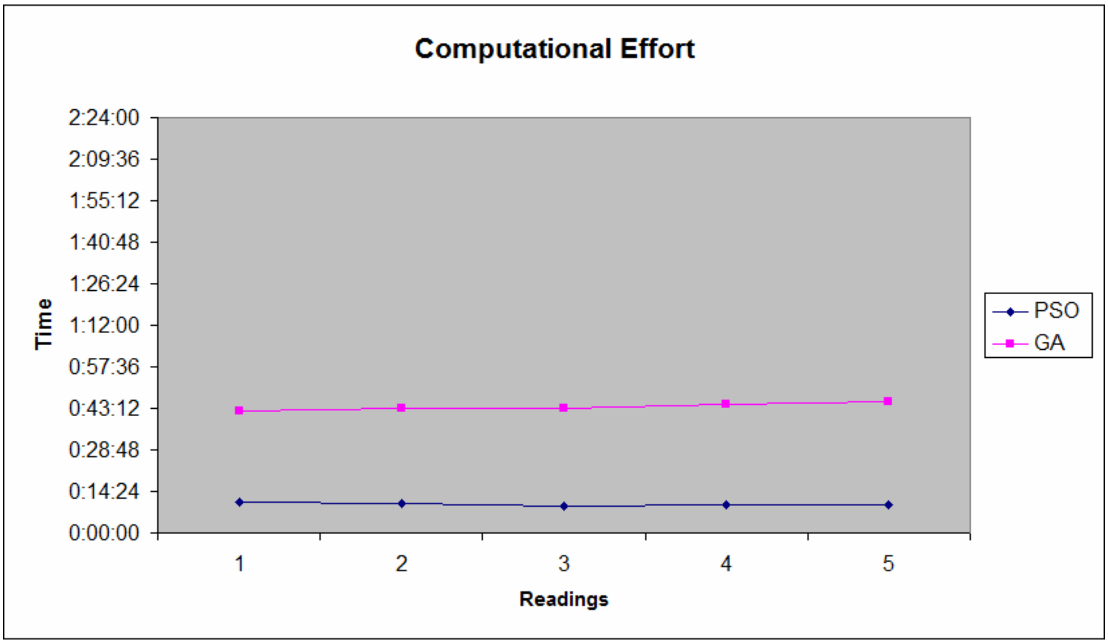


Figure 4.6: Graphical Representation of Computational Time of PLC.

It appears that PSO offers more computational savings because PSO do not have genetic operators such as crossover and mutation. PSO has few parameters to adjust. According to test results, computational time is reduced by 49% ~ 77%.

4.4 SUMMARY

To fully test PSO it has been evaluated from various aspects like fitness values, computational time, and rate of convergence. The results have been obtained on three test systems for both PSO and GA under the same environment variables. Analysis of results shows that PSO is very stable and requires little computational time to operate.

Chapter 5

CONCLUSIONS & FUTURE WORK

Software clustering is an NP-Hard problem. It cannot be solved in real time. Search Based Software Engineering (SBSE) is an approach to Software Engineering in which search based optimization algorithms are applied to Software Engineering problems. Particle Swarm Optimization (PSO) is an evolutionary heuristic search method based on biological behaviours and can be used to solve NP-hard problems. This thesis provides a framework for solving software clustering problem using PSO.

5.1 CONCLUSIONS

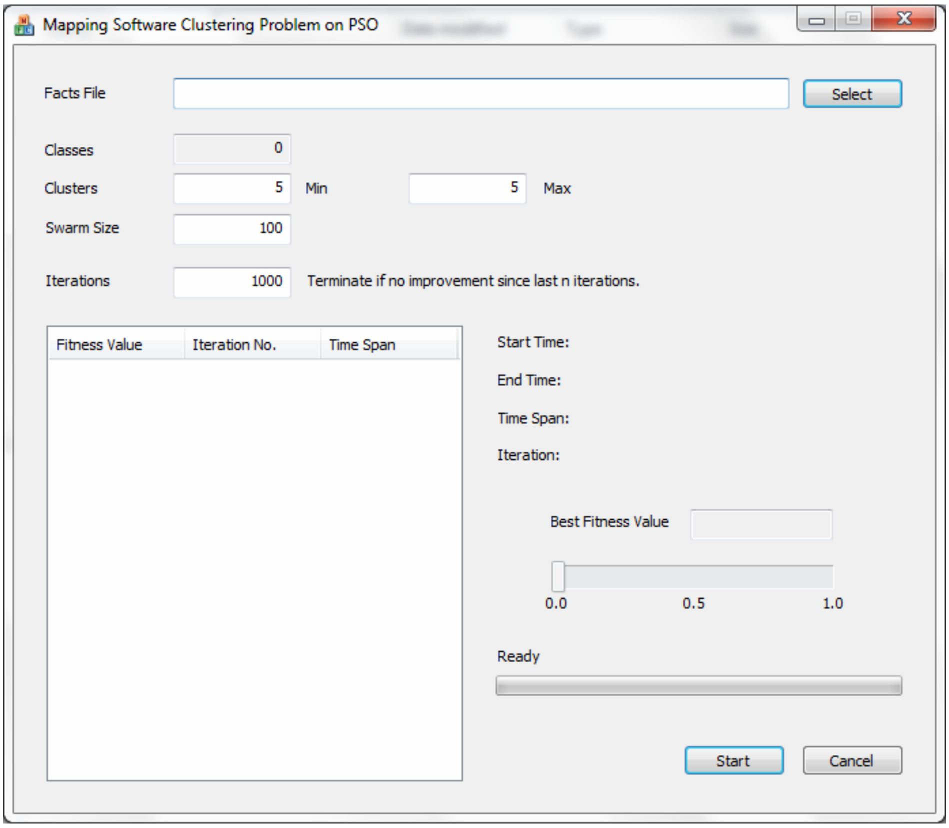
In this thesis, PSO approach is used to solve software clustering problem. Results of Particle Swarm Optimization and Genetic Algorithms are compared. Simulation results show that the PSO approach requires small computational time as compared to GA. Binary version of PSO is applied due to discrete nature of software clustering problem. Solution Quality of PSO is better than GA. In GA, reproduction can eliminate good solutions while good solutions always survive into the next iteration in PSO. PSO does offer a less expensive approach than the GA in general. It appears that PSO offers more computational savings because PSO do not have genetic operators such as crossover and mutation. According to test results, computational time is reduced by 49% ~ 77%.

5.2 FUTURE WORK

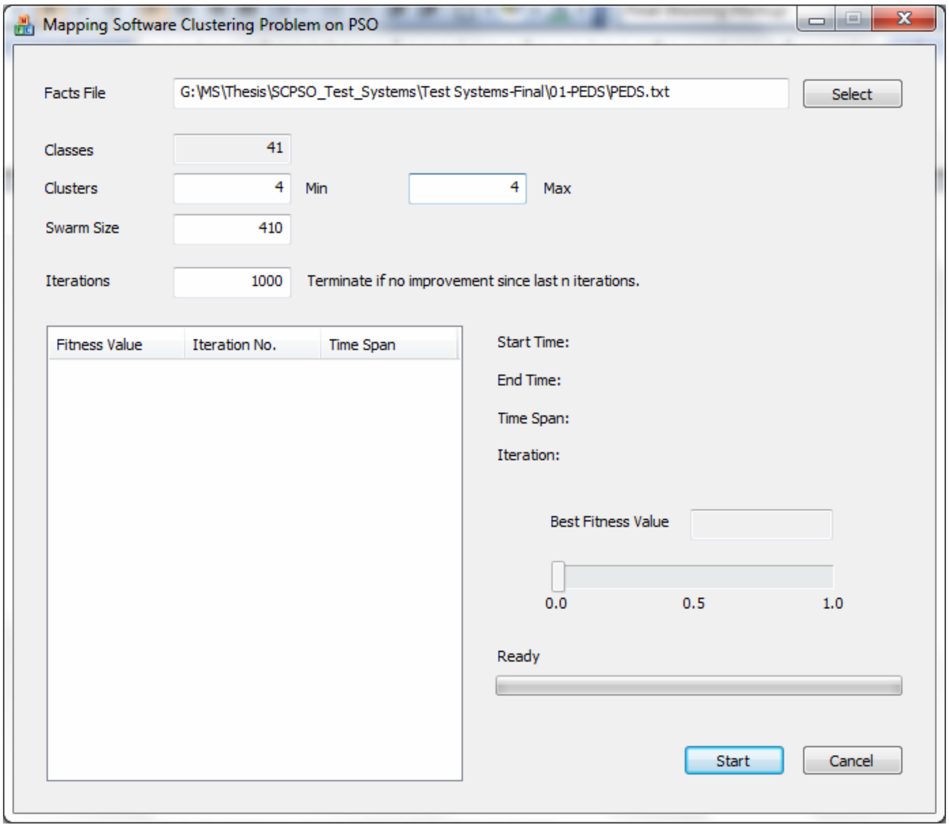
Although the PSO has been tested on three test software systems with different sizes, but there is still room for testing on much bigger test systems. The Binary PSO can be enhanced by applying inertia weight [25] and meta-optimization [27] while updating velocities. A novel PSO approach [24] can also be applied in which two new probability vectors are introduced to change bits of the particle.

SNAPSHOTS

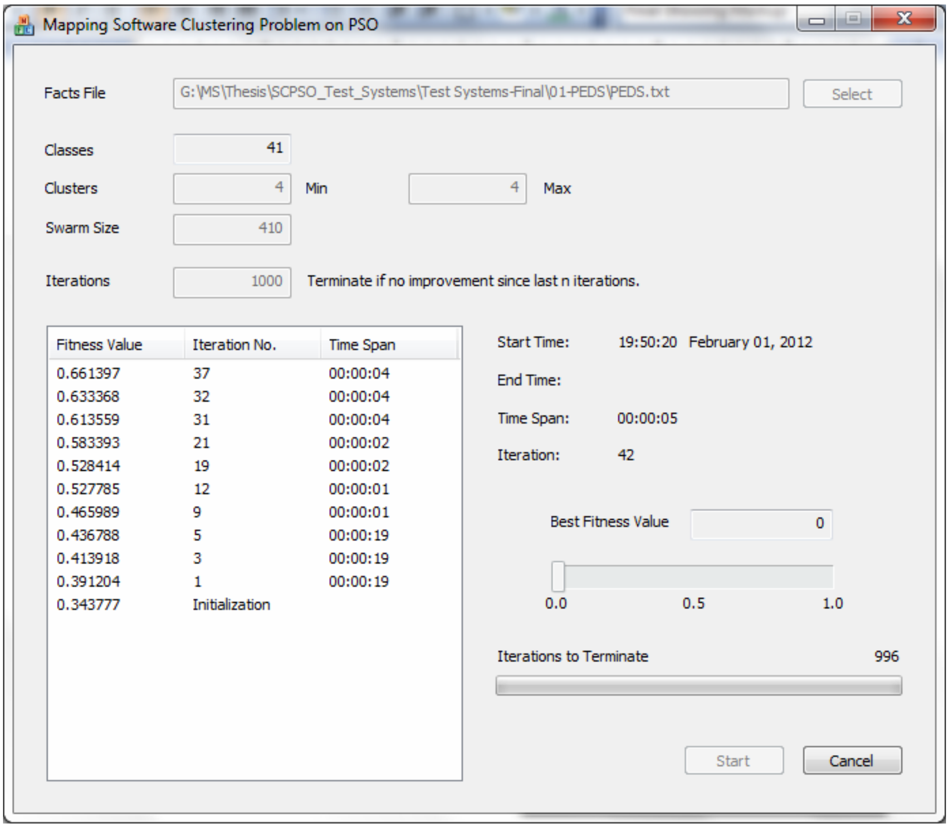
A.1 SCPSO SCREEN



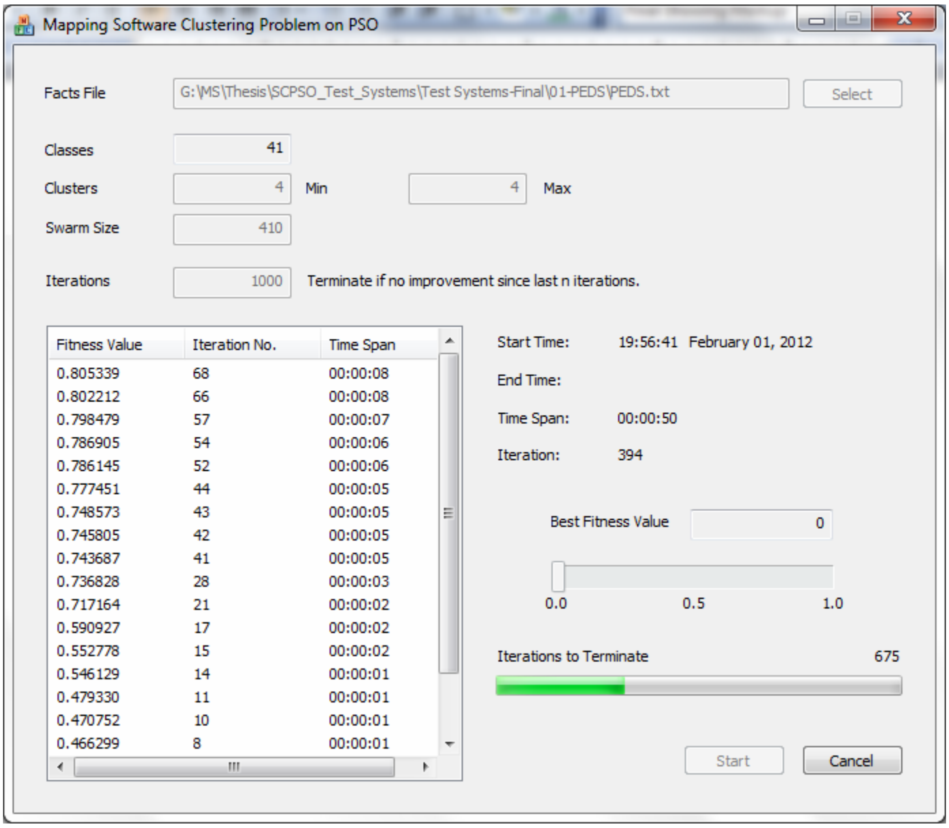
A.2 FACTS FILE LOADED



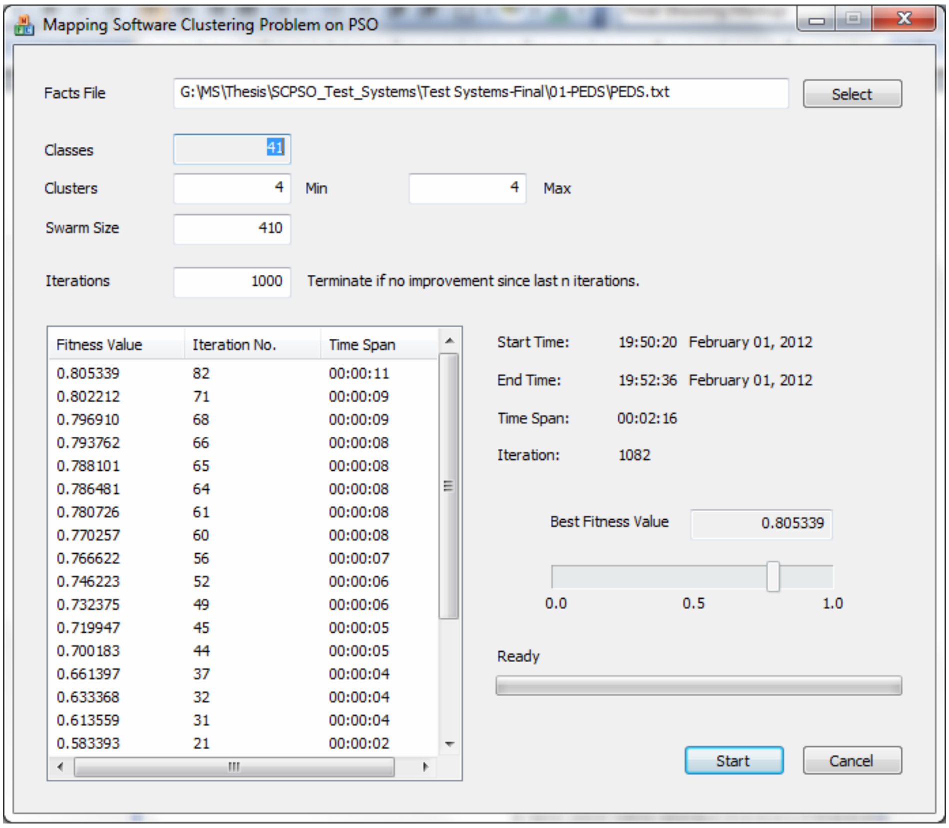
A.3 PSO EXECUTION



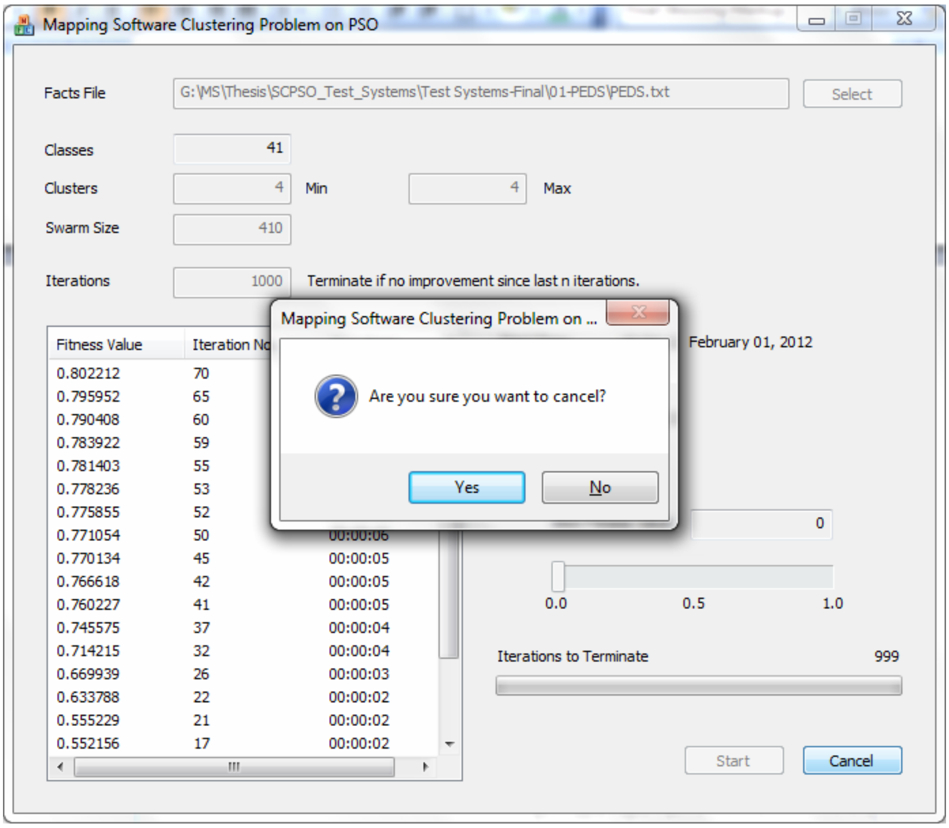
A.4 PSO GOING TO TERMINATE



A.5 RESULTS ON PROCESS COMPLETION



A.6 CONFIRMATION BEFORE CANCEL



REFERENCES

[1] IEEE Std 610.12−1990, IEEE Standard Glossary of Software Engineering Terminology.

[2] R. Kazman, S.G. Woods, and S.J. Carriere, “Requirements for Integrating Software Architecture and Reengineering Models: Corum II,” Proc. Fifth Working Conf. Reverse Eng., pp. 154-163, 1998.

[3] R. Koschke, “Atomic Architectural Component Recovery for Program Understanding and Evolution,” PhD dissertation, Univ. of Stuttgart, 2000.

[4] D.R. Harris, H.B. Reubenstein, and A.S. Yeh, “Reverse Engineering to the Architectural Level,” Proc. 17th Int’l Conf. Software Eng., pp. 186-195, 1995.

[5] B.S. Mitchell and S. Mancoridis, “On the Automatic Modularization of Software Systems Using the Bunch Tool,” IEEE Trans. Software Eng., vol. 32, no. 3, pp. 193-208, Mar. 2006.

[6] Joel Huselius, “Reverse Engineering of Legacy Real-Time System” Ph.D. Thesis 2007, Malardalen University Press.

[7] Bernd Bruegge, Allen H. Dutoit, “Object Oriented Software Engineering, Conquering Complex and Changing Systems". Prentice Hall.

[8] Brian S. Mitchell, Spiros Mancoridis and Martin Traverso, “Search Based Reverse Engineering”. Department of Mathematics & Computer Science, Drexel University, Philadelphia, PA, USA, 2002.

[9] Len Bass, Paul Clements, Rick Kazman, “Software Architecture in Practice”. 2003. The SEI Series in Software Engineering.

[10] M. Saeed, O. Maqbool, H.A. Babri, S.Z. Hassan. S.M. Sarwar, “Software Clustering Techniques and the Use of Combined Algorithm”. Computer Science Department. Lahore University of Management Sciences, 2003.

[11] Onaiza Maqbool and Haroon A. Babri, “Hierarchical Clustering for Software Architecture Recovery”, 2007.

[12] Mark Harman, “Search Based Software Engineering for Program Comprehension”, 2007.

[13] Mark Harman, “The Current State and Future of Search Based Software Engineering”, 2007.

[14] John Clarke, Jose Javier Dolado, Mark Harman, Robert Hierons, Bryan Jones, Mary Lumkin, Brian Mitchell, Spiros Mancoridis, Kearton Rees, Marc Roper, Martin Shepperd, “Reformulating Software Engineering as a Search Problem”, 2003.

[15] Abdul Qudus Abbasi, “Application of Appropriate Machine Learning Techniques for Automatic Modularization of Software Systems”. M-Phil Thesis 2008, Quaid-i-Azam University Islamabad.

[16] Randy L. Haupt, Sue Ellen Haupt, “Practical Genetic Algorithms”, 2nd Edition, A John Wiley & Sons Inc., Publication.

[17] Bilal Khan, Shaleeza Sohail, M. Younus Javed, “Evolutionary Strategy Based Automated Software Clustering Approach”. Department of Computer Engineering, College of E & ME, National University of Sciences and Technology, 2008.

[18] Bilal Khan and Shaleeza Sohail, “Using ES Based Automated Software Clustering Approach to Achieve Consistent Decompositions”. Department of Computer Engineering, College of E & ME, National University of Sciences and Technology, 2008.

[19] Frans Van Den Bergh, “An analysis of Particle Swarm Optimizers”, Doctoral Dissertation, University of Pretoria Pretoria, South Africa, South Africa © 2002.

[20] Maurice Clerc, © ISTE Ltd, 2006, “Particle Swarm Optimization”.

[21] Changhe Li and Shengxiang Yang, “A Clustering Particle Swarm Optimizer for Dynamic Optimization”, © 2009 IEEE.

[22] Thomas Wiese, “Global Optimization Algorithms - Theory and Application”, Second Edition, Chapter 8 & 9, Version 2009-06-26.

[23] Dr. Karl O. Jones, “COMPARISON OF GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION”, International Conference on Computer Systems and Technologies – CompSysTech 2005.

[24] Mojtaba Ahmadieh Khanesar, Mohammad Teshnehlab and Mahdi Aliyari Shoorehdeli, “A Novel Binary Particle Swarm Optimization”. Proceedings of the 15th Mediterranean Conference on Control & Automation, Athens – Greece, July 27 – 29, 2007.

[25] Rania Hassan, Babak Cohanim, Olivier de Weck, “A Comparison Of Particle Swarm Optimization And The Genetic Algorithm”. American Institute of Aeronautics and Astronautics, 2004.

[26] Koza, John R, “Genetic Programming: On the Programming of Computers by Means of Natural Selection”. Cambridge, MA: The MIT Press 1992.

[27] Magnus Erik Hvass Pedersen, “Good Parameters for Particle Swarm Optimization”. Technical Report no. HL1001, 2010.

[28] McCarl B.A. and T. H. Spreen. 1997. “Applied Mathematical Programming Using Algebraic Systems”. Chapters 15 and 16. *http://agecon2.tamu.edu/people/faculty/mccarl-bruce/books.htm*

[29] Paul A. Jensen, Jonathan F. Bard, “Operations Research Models and Methods”, 2002.

[30] Kennedy, J. Eberhart, R., In Proc. IEEE International Conference on Neural Networks, vol.4, pp. 1942 – 1948, 1995.

[31] Hesam Izakian, Behrouz Tork Ladani, Ajith Abraham, Vaclav Snasel, “A Discrete Particle Swarm Optimization Approach for Grid Job Scheduling”, International Journal of Innovative Computing, Information and Control, Volume 6, Number 9, September 2010.

[32] J. Kennedy and R. Eberhart, “A discrete binary version of the particle swarm algorithm”. In Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics, pages 4104–4108, Piscataway, NJ, USA, 1997.

[33] El-Ghazali Talbi, “METAHEURISTICS From Design to Implementation”, Copyright © 2009 by John Wiley & Sons, Inc. 2009.