ASHESI UNIVERSITY COLLEGE

A PROCESS FOR AUTOMATED CLASS SCHEDULING AT ASHESI

GEORGE DONKOR

2014

Applied Project

ASHESI UNIVERSITY COLLEGE

A PROCESS FOR AUTOMATED CLASS SCHEDULING AT ASHESI

By

GEORGE DONKOR

Dissertation submitted to the Department of
Computer Science,
Ashesi University College
In partial fulfilment of Science degree in Computer Science

APRIL 2014

I hereby declare that this dissertation is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:……

Candidate's Name:……

Date:……..

I hereby declare that the preparation and presentation of the dissertation were supervised in accordance with the guidelines on supervision of dissertation laid down by Ashesi University College.

Supervisor's Signature:……..

Supervisor's Name:…….

Date:……..

ACKNOWLEDGEMENT

I want to acknowledge my Supervisor, Dr. Ayorkor Korsah for her valuable advice, Mr. Osafo Maafo for his opinions and Dr. Nathan Amanquah for suggestions. I am grateful to my esteemed colleagues Eben Ashiakwei and Nii Lous Nelson for their help and support through this journey. I thank my parents who have been a great source of hope and most importantly, I thank God for everything.

ABSTRACT

At the beginning of every semester, the registrar at Ashesi University goes through the laborious task of either manually or semi-automatically developing a course schedule. Very often, after the schedule has been developed, conflicts are realized in the various schedules. Conflicts are categorized into student, lecturer and room conflicts. An open source software, FET was recently used by the university to help develop schedules for the courses. This project is an attempt to review the ways in which the automation process can be enhanced in order to potentially reduce the conflicts faced.

At the heart of automated course scheduling is the algorithm being used. Any effort made at enhancing the scheduling process in Ashesi will require an efficient algorithm. This paper begins with a background on scheduling, an extensive research on existing approaches and algorithms follows. The algorithms reviewed include the Multi-Agent System approach, Sequential methods, Constraint Based Methods, Genetic Algorithms, Simulated Annealing, Particle Swarm optimization and Tabu Search. The algorithm used in the FET software is also reviewed. These techniques are compared based on their computational time, ease of implementation, solution quality and constraint handling. Based on the literature, it is realized that Particle Swarm Optimization is potentially the best algorithm with respect to the set criteria. A basic version of the Particle Swarm Algorithm is implemented and tested and the results compared with the results from testing the current FET software algorithm, recursive swapping. The outcome implies that recursive swapping, can produce good solutions but Particle Swarm Optimization is easier to implement.

Table of Contents

Chapter 1

1.1 Introduction

Scheduling, in the broadest sense, means setting an order for planned events. It is the placement of an event, or set of events to occur at a particular time. The scheduling problem is referred to as an NP-complete problem, i.e. No efficient optimal solution has been found [1].

Course scheduling at a university level is concerned with groups or classes of students following a particular defined pathway or course which has associated events that require the allocation of time and resources.

The development of a schedule involves a set of meeting times, a set of available resources (eg. staff, rooms and students), a set of available time slots and a set of constraints. The challenge is assigning resources and time slots to each given meeting, while ensuring that constraints are satisfied.

1.2 Manual and Semi-Automated Course Scheduling

Over the past years, the scheduling process at Ashesi University has been either manual or semi-automatic. The process is either done without any automation or with some level of automation using existing scheduling software systems. Below is a brief description of the manual used in the past and semi-automated process which is currently being used.

Each semester, there are sets of courses to be taken by different groups of students and taught by different lecturers. The manual scheduling process is basically assigning a lecturer to a course by hand. Most of the attention is given to

lecturer preferences (i.e. a preference could be that a lecturer prefers all his courses to be scheduled on Monday and Wednesday mornings). The process gets tedious as the number of courses to be assigned increases, lecturers are limited, less room space is available and student size keeps increasing. Due to this, software is applied to the process in order to automate it. The software is given information about the courses, students, lecturers and rooms in the University and it assigns each based on an algorithm.

Although the software reduces the difficulty in the process, it does not absolutely automate the process. After the software is done, the registrar has to screen the output and make a few changes to the assignments in order to make the solution applicable and effective.

Automation of the process has the potential to produce a great impact with respect to the time involved and the optimality of the final solution. With the right algorithm, the entire process could be simplified and the solution produced could be highly effective.

The scheduling problem has been studied for years and has been referred to as an NP-complete problem i.e. the problem cannot be solved optimally in polynomial time [2]. Hence even though automated scheduling has the potential to solve the issues of manual scheduling, it is on its own a very complex subject. The key to automating any process however lies in the algorithm used. A great deal of research has been conducted in this area. This paper seeks to discuss the possible algorithms that exist and in effect, implement and test the best algorithm or

combination of algorithms which can potentially solve the scheduling needs in Ashesi University College.

Chapter 2

Background and Existing Research

This section briefly explains NP-complete problems and discusses distinct approaches that exist in theory and practice to solving the course scheduling problem.

2.1 NP-Complete Problems

NP complete problems (nondeterministic-polynomial complete problems) are problems that cannot be solved in a polynomial amount of time. In theory, as well as practice, there is no optimal solution to NP complete problems that can be computed in polynomial time. The timetabling problem is a classic example of an NP complete problem. For years, researchers have tried different means of tackling the problem but no polynomial-time optimal solution has yet been discovered. Most of the earlier approaches have been based on heuristic algorithms to the problem [2].

2.2 General Timetabling Concepts, Terms and Constraints

University Course Scheduling is the process of assigning time slots, rooms, lecturers and students to courses under a set of constraints. The challenge is finding the optimal sequence for executing a finite set of operations under the defined set of constraints such that most, if not all the constraints are satisfied.

With respect to course scheduling at Ashesi University, the problem is defined as the process of assigning a set of events (courses) to time periods and to rooms, subject to certain conditions.

Resources include students, lecturers, rooms, courses and time.

2.3 Constraints

Typical constraints for scheduling at Ashesi include the following:

- Each full-credit course has two lecture sessions per week**.** Some full credit courses also have one discussion session per week but not all full credit courses have a discussion session. Half credit courses have only one lecture session each week.
- Students are grouped into different cohorts
- No student can be in more than one place (lecture) at a time
- No lecturer or faculty intern can lecture more than one class of students at a time
- Space is limited in Ashesi. We have 5 lecture halls, 2 laboratories and 1 multipurpose room
- Each session (lecture or discussion) has a duration of ninety minutes (e.g. 8:30-10:00).

In the subsequent sections, the paper will describe the Multi Agent System approach, Graph Coloring approach, Constraint Based approach, Genetic Algorithms, Simulated Annealing, Particle Swarm Optimization and Tabu Search technique.

2.4 Multi Agent System Approach (MAS)

This section discusses course scheduling using a Multi Agent System Approach.

Scheduling problems are iterative and time consuming. People who are involved in timetable scheduling processes encounter numerous conflicting preferences that make the search for an optimal solution an NP-hard problem.

The Multi Agent System approach (MAS) involves creating connections between a collection of autonomous intelligent agents (e.g. software agents) that work in an environment. In other words, autonomous intelligent agents are responsible for connecting the behavior of one software with that of another software.

A type of software agent that is commonly used in this approach is the expert assistant. The expert assistant is an intelligent software agent that performs certain tasks on behalf of humans. It helps automate certain manual tasks and works more efficiently. A physical example of an expert assistant is the daily organizer. The MAS approach is a highly complex system as compared to other software systems. Its success depends on a properly designed and well tested subsystem. [3]

2.4.1 The MAS architecture

The MAS process maps course timetabling in terms of autonomous intelligent agents. Each faculty[1] in the system has its own scheduler Multi Agent System which has to allocate courses to that faculty. The Main Scheduler Agent (MSA) is responsible for room allocation.

Because some lecturers teach courses to different faculty, every faculty agent has to communicate with the other scheduler agents in order to solve some critical situations that may arise.

Each specialization (major or minor) has an expert assistant that does all the activities connected to that specialization (evidence of students, course curricula

[1] A group of university departments concerned with a major division of knowledge.

preferred rooms, available lecturers, etc). This means the expert assistant maintains the data of each major or minor program. The main role of the expert assistant (or course assistant) is course timetabling (day and time scheduling).

2.4.2 How constraints are satisfied

All courses of a specialization must be taught for all groups of students. Conflicts are likely to arise when a particular course is supposed to be taught for more than one specialization. For example, at Ashesi, the Programming II course, although it is a computer science course, must be taught to management information systems students as well.

The autonomous agents can act on the professors' behalf. A best case scenario is when all preferences are accepted. However this is not usually the case. In reality, agents must quantify the professors' preferences. The system handles such critical situations using a persuasion protocol. This process is based entirely on the rationality of agents. The autonomous agent must meet some criteria of rationality (e.g. maintaining logical consistency).

The MAS approach uses four types of autonomous agents

- The Main Scheduler Agent (M.S.A)
- The Faculty Scheduler Agent (F.S.A)
- The Expert Assistant Agent (E.A.A)
- The Personal Agent (P.A)

2.4.3 Roles of each agent

M.S.A: The Main Scheduler Agent is the main scheduler at the university level. It is responsible for allocation of rooms and negotiation activities.

F.S.A: The Faculty Scheduler Agent is the scheduler at the faculty level. It handles the allocation of periods, integration of periods done at each faculty level and negotiation activities.

E.A.A: The Expert Assistant Agent handles the scheduling at the specialization level. It allocates periods and if there are any conflicts, it handles the through a negotiation process.

P.A: The Personal Assistant is the lecturers own scheduler. It handles the scheduling process for each professor, allocates periods and deals with conflicts through negotiations with the other agents.

2.4.4 Possible Conflicts

1. At the faculty level, day and time schedule may conflict (two or more professors may have identical options).

   Solution: A negotiation process is started between the expert assistant of that specialization and the professors involved (the personal agents). A message is sent by the expert assistant to all the professors involved in the conflict and it will wait for a solution. If it receives an answer, it will do a rescheduling, otherwise if no answer is received, it will start a persuasion process of negotiating and suggesting a solution.

2. At the university level, a possible situation will be that no room is available for a particular course on a certain day and time.

   Solution: The Main Scheduler Agent starts a negotiation process between faculty agents involved in the conflict by giving some options. Each scheduler involved will pass the message to the corresponding expert assistant, or in some cases,

to the personal agents who will then negotiate directly. If no solution is found (e.g. some courses cannot be moved) the main scheduler will start a persuasion dialogue with faculty agents that are in conflict which will in turn transfer the problem to the lower level i.e. the expert assistants.

Advantage

The MAS approach uses several autonomous software agents to develop non-conflicting schedules for each major (specialization), professor and room. The system handles conflicts using persuasion and negotiation protocols between agents which prove to be effective.

Disadvantage

A drawback of this method is that it expensive and time consuming to implement, considering the fact that each faculty and specialization needs to have its own specialization agent [3].

2.5 Sequential Methods.

Sequential methods are absolutely different from the Multi Agent System approach explained earlier. Sequential methods order events[2] using domain heuristics and then assign the events sequentially into valid time periods such that no events in the period are in conflict with each other [2]. They are based on the idea that one

[2] Events are the courses to be scheduled.

stage will be implemented before the other with the aim that each stage will build on the previous one.

In sequential methods, timetabling problems are usually represented as graphs where events are represented as vertices and conflicts between the events are represented by the edges. The construction of a conflict free timetable can therefore be modeled as a graph-coloring problem.

2.5.1 Properties of Graph Coloring Methods

All graph-coloring techniques have two basic properties.

1. Each time period should correspond to a colour in the graph colouring problem.
2. The vertices of the graph are coloured in such a way that no two adjacent vertices are the same colour [2].

2.5.2 Existing Graph Coloring Methods

M.H Williams and K.T Milne compare several different graph-coloring algorithms in their paper "The performance of Algorithms for coloring Planar graphs" [4]. Below is a brief description of each algorithm.

The first is the Welsh and Powell algorithm, which works by assigning the least possible color to the vertices in a given order. The vertices of the graph are presented in order of descending degree. The algorithm assigns a color *x* to the first uncolored vertex of the graph which can be found. After, it assigns a color *x* to each uncolored vertex which is not adjacent to a vertex that has been already colored. The process is repeated with each color in turn until no uncolored vertices remain [4].

The Welsh and Powell algorithm was extended by Saaty and Kainen to sort the vertices of the graph on the basis of density calculations rather than simply by degree. The time taken to sort the vertices of the graph is considerably greater than the method used by Welsh and Powell due to the density calculations [4].

There is the Dsatur method which was based on the saturation degree of a vertex i.e. the number of different colors to which that vertex is adjacent at any point in the coloring process. The algorithm repeatedly chooses the vertex with the highest degree of saturation and assigns it the least possible color [4].

The Dsatur method was extended by the DSI method, which requires an interchange of colors whenever a new color must be introduced [4].

There is the Dutton and Brigham algorithm. This method is considerably slower than the other methods described but requires fewer colors. It attempts to merge non-adjacent vertices of a graph that can be colored with the same color until all the vertices are neighbors of each other [4].

The tree method is the simplest method which recursively tests partial solutions until an exact solution is found [4].

The second exact method is a modification of Randall Browns method, which starts by using the DSI method to produce an initial coloration of the map, and then by the use of cliques, seeks to reduce the number of colors needed [4].

There are reduction methods that operate by repeatedly selecting and removing one or more vertices of a graph until it is reduced to a size that can be colored with

only four colors. The vertices removed are then reinserted in reverse order and colored appropriately [4].

Based on the evaluation conducted by M.H Williams and K.T Milne, the performance of the different algorithms and concluded that exact methods such as the tree method and Randall Browns modified algorithm take a significantly long amount of time, but use fewer colors [4]. The other methods however manage larger graphs in less time, but have to deal with the trade-off of using more colors.

Advantage

- Graph colouring is simple to implement.

Disadvantages

- The method might be inefficient for a large set of students.

2.6 Constraint Based Methods

Constraint Satisfaction Problems are also known as CSP. They require a value, selected from a finite domain to be assigned to each variable in the problem, so that all the constraints are satisfied [5]. The timetabling problem is modeled as a set of variables (events) to which values (resources such as rooms and time periods) have to be assigned to satisfy a number of constraints [2].

2.6.1 About Constraint Based Methods

The whole idea is to assign values to a set of variables without violating the given set of constraints.

For most categories of Constraint Satisfaction Problems, an efficient algorithm is unlikely to exist (the problems are NP Complete). Hence, an algorithm that guarantees to find a solution that satisfies all constraints is said to be enumerative and therefore has an exponential time requirement in the worst case. It may be possible to find a solution, at reasonable computational expense that satisfies most of the constraints, especially if the problem contains soft constraints. If all or a majority of the constraints are satisfied, the solution is called 'exact', otherwise it is termed 'approximate' [5].

Approaches used to tackle Constraint Satisfaction Problems include integer programming techniques (like branch or bound and cutting plane methods) for exact solutions, local search methods (like simulated annealing, threshold acceptance, tabu search and genetic annealing) and neural networks for approximate solutions. There are also methods that use tree search combined with backtracking and consistency checking [5].

A Constraint Satisfaction Problem consists of:

- A set of variables X = {x1, x2, x3… xn}
- A finite set of possible values Di (the domain) for each variable
- A set of constraints to restrict the values that variables can simultaneously take.

Expected solutions are either feasible or optimal. A feasible solution is an assignment of a value from its domain to every variable in such a way that every constraint is satisfied. In such a case, the problem is satisfiable. If however, assignments of values to variables from their respective domains do not satisfy all constraints, the problem is termed unsatisfiable [5].

2.6.2 Search Algorithms

Most algorithms for solving CSP's search systematically through possible assignments of values to variables [5]. In other words, search algorithms are employed in the assignment of variables to values in an effort to handle possible conflicts. Three systematic search algorithms used for solving constraint satisfaction problems are back tracking, forward checking and MAC.

2.6.2.1 Backtracking Algorithm

The current variable is assigned a value from its domain. This assignment is checked against the current partial solution; if any of the constraints between this variable and the past variable is violated, the assignment is abandoned and another value for the current variable is chosen. If all values of the current variable have been tried, the algorithm backtracks to the previous variable and assigns it a new value. If a complete solution is found (a value has been assigned to every variable), and only one solution is required, the algorithm terminates. Otherwise, it continues to find new solutions. If there are no new solutions or all possible solutions have been considered, the algorithm terminates [5]. Backtracking is not used in practice because it is very inefficient. It only checks constraints between current variable and the past.

2.6.2.2 Forward Checking Algorithm

Forward-checking and MAC (Maintaining Arc Consistency) are lookahead algorithms, essentially more efficient than backtracking. They check constraints between current, past and future variables.

When a value is assigned to the current variable, any value in the domain of a future variable which conflicts with this assignment is (temporarily) removed from

the domain. The advantage is that if the domain of a future variable becomes empty, it is known immediately that the current partial solution is inconsistent. Another value for the current variable is tried or the algorithm backtracks to a previous variable; the domains of the future variables are restored to what they were before the assignment which led to failure [5].

The difference between the forward and the backtracking algorithm is that the backtracking algorithm will not have been able to detect the failure until the future variable was considered, at which point none of the values will be consistent with the current partial solution. Forward checking therefore allows branches of the search tree that will lead to failure to be pruned earlier than with simple backtracking.

2.6.2.3 Maintaining Arc Consistency Algorithm

The MAC algorithm does more work in looking ahead when an assignment is made, and hence is considered more efficient than both forward and backtracking algorithms. Whenever a new subproblem consisting of future variables is created by a variable instantiation (assignment of a value to a variable), the subproblem is made arc consistent[3]. This means that as well as checking the values of future variables against the current assignment, MAC checks the future variables against each other. Hence for each future variable, every variable that is not supported in the domain of some other future variable is deleted, as well as those that are not supported by the current assignment. [5]

[3] Local consistency conditions which are properties of constraint satisfaction problems related to the consistency of subsets of variables or constraints.

This removes further values from the domains of future variables in the hope that in doing additional work at the time of the assignment, overall computational time will be reduced.

2.6.3 Variable and Value Ordering

The order in which variables are considered for instantiation has a significant effect on the time taken to solve the CSP, as does the order in which each variable's values are considered. The ordering of the variables may either be static or dynamic [5]. Static ordering specifies the order of variables before the search begins and the order is not changed afterwards. For dynamic ordering, the choice of the next variable to be considered depends on the current state of the search.

2.6.4 Application Constraint Based Methods to Timetabling

For timetabling problems, a feasible solution is often the main objective. The factors involved are qualitative (i.e. lecturer preferences or student preferences can have different weights showing their importance) and hence can easily be expressed as constraints. One approach could be to present each class as a variable whose domain is the set of available time periods. Another approach could be to block classes together and define the variables to be the starting time of each block. Though this approach uses fewer variables, it increases the complexity of the constraints [5].

Advantages

- Constraint Based methods are easy to implement

- They are flexible with constraints; they have the ability to handle more constraints.

2.7 Metaheuristic Techniques

A Metaheuristic is an iterative generation process that guides a subordinate heuristic by combining intelligently different concepts for exploiting and exploring a search space. Metaheuristic approaches make use of an initial solution, or an initial set of solutions and initiate an improving search, guided by certain principles. All metaheuristic algorithms have some properties in common.

This section outlines the common properties of metaheuristics, discusses different metaheuristic techniques and how they can be used in solving course scheduling problems

General properties of metaheuristics techniques.

- Metaheuristics are strategies that guide a search process.
- The goal of any metaheuristic technique is to efficiently explore the search space in order to find close to optimal solutions.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- The basic concept of any metaheuristic technique allows an abstract level of description.
- They make use of domain specific knowledge in the form of heuristics that are controlled by an upper level strategy.

All metaheuristic techniques have the same basic structure. Given the population of a solution or set of solutions, a candidate solution or set of solutions is selected and evaluated. The evaluation involves estimating the performance of the candidate

solution and comparing it with that of the current or other solutions in the population. The candidate may either be accepted or rejected based on this evaluation [6]. Given this framework, it can thus be said that all metaheuristic algorithms share the elements of selecting a candidate from the population and deciding whether to accept or reject the candidate. This means one or more of the elements needs to be specified and the rest have to adapt.

Below are different metaheuristic techniques that can be used in course scheduling.

2.7.1 Genetic Algorithms

Background

The idea of genetic algorithms is purely biological. It is based on the genetic model of chromosomes and genes. Each chromosome is a representation of a complete solution the scheduling problem. The genes represent the individual components of a scheduling solution. The individual organisms in the genetic algorithm are made up of single chromosomes. The chromosomes are made up of genes. By manipulating the genes, new chromosomes with different traits can be created. These manipulations occur through techniques known as crossover and mutation. Crossover is essentially the biological process of mating and mutation is a way of introducing new information to an existing population [7].

For an effective solution, we must determine a way to break the problem into individual components or genes.

Basic steps involved in creating a genetic algorithm

1. Create an initial population of chromosomes

2. Evaluate the fitness or suitability of each chromosome that makes up the population
3. Based on the fitness, select chromosomes that will mate
4. Crossover the selected chromosomes and produce offspring
5. Randomly mutate some of the genes of the chromosome
6. Repeat steps three though to five until a new population is created
7. The algorithm ends when a best solution has not changed for a preset number of generations

2.7.1.1The Initial Population

The initial population is the first thing the genetic algorithm must cater to. The entire population is made up of organisms and each organism is composed of a single chromosome. The genetic algorithm is responsible for creating the initial population that contains the possible solutions. After the generation of the initial population, each chromosome is evaluated for its fitness. The fitness is determined by a function and is usually problem specific [7].

2.7.1.2 Suitability to Mate

Mating is the creation of a new improved population. Not all chromosomes are suitable for mating hence it must be determined which chromosomes have the privilege to mate. The selection of which chromosomes will mate is based on the individual chromosomes fitness. If the chromosome is fit, then it is selected from the old population and crossed to form a new chromosome to join the new population [7].

2.7.1.3 How Mating Works

Mating works by taking a splice from the gene sequence of two parents. The splice effectively divides the chromosome into three gene sequences. The new chromosomes are built based on genes from each of these three sections. The process of mating takes traits from each parent to form the new chromosome. The problem here is that no new trait is introduced. To introduce new genetic material, the process of mutation is used [7].

2.7.1.4 Mutation

Mutation allows new genetic patterns to be introduced that were not already contained in the population. The mutation process introduces a new, random sequence of genes into the chromosome. As to whether the mutation will be desirable or not is completely unknown, but this does not affect the process in any way. As each chromosome is evaluated, the function checks if the fitness of the mutated chromosome is higher than the general population. If so, it is allowed to live and mate with other chromosomes. If not, then the algorithm ensures that the chromosome does not live to mate [7].

Advantage

1. Genetic algorithms are easy to understand and implement. Its application does not require knowledge of advanced mathematics.
2. They are good for exploring large solution spaces.

Disadvantage

1. Genetic algorithms cannot ensure constant optimization response time. The difference between the shortest and the longest optimization response time is significantly large.

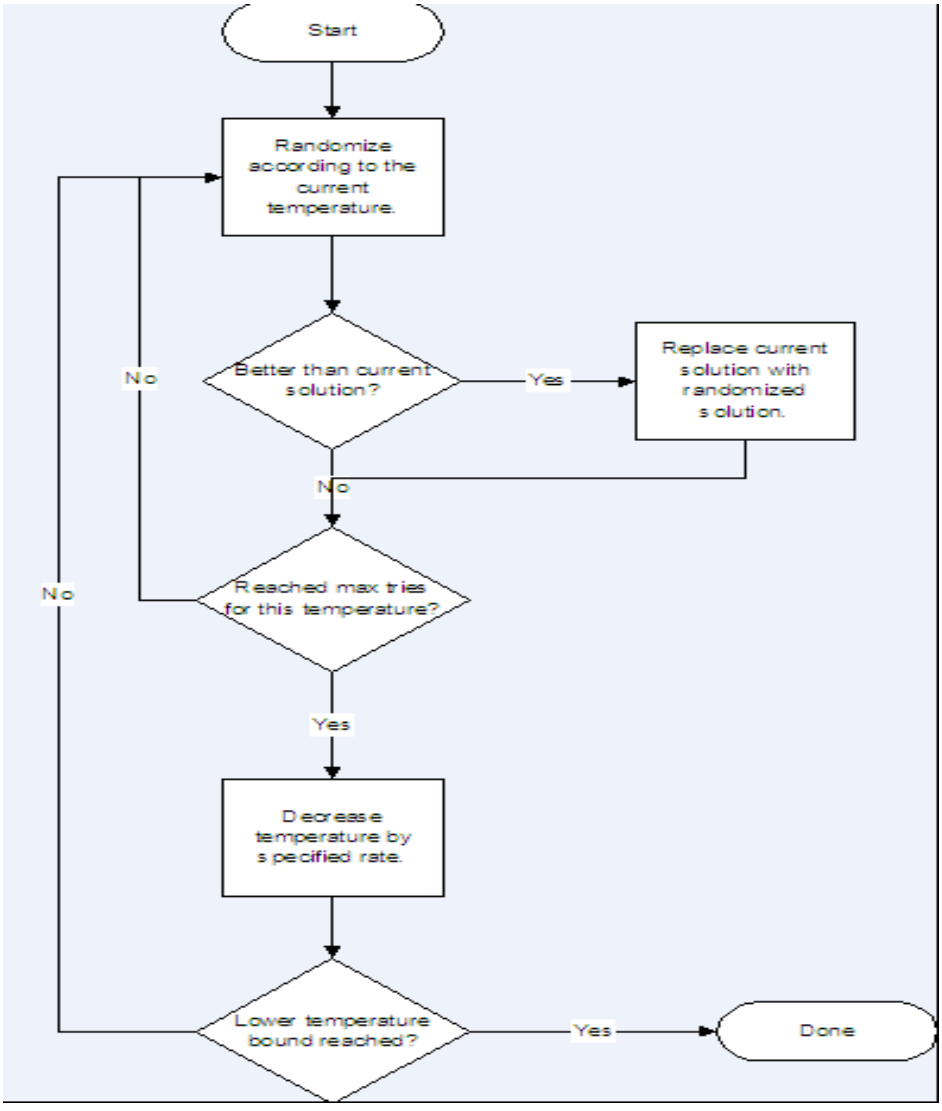
2.7.2 Simulated Annealing

2.7.2.1 Background

Simulated annealing is a process that mimics the process of annealing. Annealing is the process of heating up a solid (usually a metal) and cooling it slowly until it crystalizes. Atoms have very high energies at very high temperatures. This gives them a large amount of freedom in their ability to restructure themselves. As the temperature goes down at a slower rate, the energy of the atoms decrease. This allows a more consistent crystalline structure form and enhanced durability.

Simulated annealing emulates the process of annealing. It starts out at a very high temperature where the input values are allowed to assume a wide range of random values. As the training progresses, the temperature is allowed to fall, reducing the degree to which inputs may vary. This leads the algorithm to a better solution, just like the way a metal achieves a better crystal structure through the same process [7].

2.7.2.2 Structure of the Simulated Annealing algorithm



As demonstrated in the figure above, the process starts out by randomizing the inputs according to the current temperature. If the randomized solution is better than the current solution, then the current solution is replaced with the randomized solution. For each temperature, the simulated annealing algorithm runs through a number of iterations. A maximum number of iterations is set by the programmer. Until the maximum number of iterations for the current temperature is reached, the algorithm continues to randomize inputs and generate randomized solutions. If the maximum number of iterations for the particular temperature is reached, the algorithm decreases the temperature by a specified rate. It keeps decreasing until the lowest temperature is reached.

Once the number of iterations for each temperature has been completed, the algorithm checks if the lowest allowed temperature has been reached. If the temperature has not reached its minimum, then it is lowered and another iteration of randomizations will occur. If the algorithm is lower than the lowest minimum allowed, then the process is complete.

As inferred from the description above, randomization of the input values is crucial to the process of simulated annealing. It is what causes the algorithm to alter the input values that it is seeking to minimize.

2.7.2.3 How inputs are randomized

There is no exact method on how to randomize the inputs. The exact nature depends on the problem being solved. The randomization process usually takes the previous values of inputs and the current temperature as inputs. The input values

are then randomized according to the temperature. A higher temperature will result in more randomization and a lower temperature will result in less randomization.

2.7.2.4 Temperature Reduction

There are two common methods used for temperature reduction. The first simply reduces the temperature by a fixed amount through each iteration. The second is to specify the beginning and ending temperature. In the second method, a ratio is calculated at each step in the annealing process. The equation used must guarantee that the step will cause the temperature to fall to the ending temperature in the number of cycles requested.

Advantages

1. Relatively easy to implement, even for complex problems
2. Simulated annealing is an attractive option for optimization problems where heuristic methods are not available.

Disadvantages

1. The method cannot tell when an optimal solution has been found. It requires the help of another method such as the branch and bound.
2. The method has potentially expensive computational cost.
3. Simulated Annealing is not efficient in exploring large solution spaces.

2.7.3 Particle Swarm Optimization

2.7.3.1 Background

Particle Swarm Optimization (PSO) is another optimization technique inspired by the flocking and schooling patterns of birds and fish. While searching for food, birds are either scattered or go together before they locate the place where they can find

food. While the birds are searching for food from one place to another, there is always the bird that can smell the food very well, i.e. the bird is perceptible of the place where the food can be found, hence has better food resource information. Because they are all transmitting information about where the food is (by chirping at each other), conducted by the information, birds will eventually flock to the place where food can be found [8].

Just like how birds swarm, the PSO algorithm works in the same manner. The solution swarm (the population containing possible schedules) is compared to the bird swarm, the movement of birds from one place to another is compared to the development of the solution swarm, good information is equivalent to the most optimistic solution before the search is over and the food resource is equivalent to the most optimistic solution obtained after searching through the entire swarm.

As long as the individual particles in the swarm cooperate with each other, a feasible solution can be found. Over a number of iterations, a group of variables have their values adjusted closer to the member whose value is closest to the target at any given moment. The algorithm is fairly simple and easy to implement.

The PSO algorithm keeps track of three global variables:

- The target value which represents the best solution.
- Global best (gBest) value indicating which particle’s data is currently closest to the target value or solution.
- Stopping value indicating when the algorithm should stop if the target is not found.

Each particle in the swarm is made up of at least three items

- The data representing a possible solution
- A velocity value indicating how much data can be changed.
- A personal best (pBest) value indicating the closest the particle's data has ever come to the target.

The particle's data could be anything. If the data is a pattern or sequence, the velocity would describe how different the pattern is from the targets' and how much it needs to be changed to match.

Note that the pBest value is only an indicator of the closest the data has ever come to the target since the algorithm started. The gBest value will change when the pBest value comes closer to the target than the current gBest.

Advantages

- PSO is simple to understand and easy to implement
- It requires less computational time and less iterations as compared with genetic algorithms.

2.7.4 Tabu Search

2.7.4.1 Background

Tabu search is an algorithm under the meta-heuristic family of algorithms. The fundamental approach is to avoid movement into bad solutions by penalizing or forbidding moves which take that solution in the next iteration to points in the solution space previously visited. This method is partly based on human behavior. Human beings operate with a random element that lead to inconsistent behavior in

particular situations. The tendency to deviate from a chartered course might be regretted as a source of error but might also prove to be a source of gain. The tabu method operates in this way with the exception of the use of random operations.

2.7.4.2 How the Algorithm Works

The algorithm begins with the assumption that there is no point in accepting a poor solution unless it is to avoid a path that has already been investigated. This ensures that new regions of the solution space will be investigated with the desired goal of obtaining an optimal solution.

The search makes use of one or more tabu lists to record moves it makes in order to avoid repeating moves. This is done using some form of memory structure to hold the data. The role of the memory can change as the algorithm proceeds. When the algorithm starts, the goal is to examine the solution space, this stage is known as diversification. As the search progresses, candidate solutions are identified and the focus is geared towards achieving local optimal solutions. The second phase is known as intensification.

2.7.4.3 Basic tenets of Tabu Search Algorithm

- The search is a local search strategy with a flexible memory structure
- The search has two prominent features; an adaptive memory and responsive exploration strategies
- The main feature is to always move to the best available neighbourhood solution point, even if it is worse than the current solution point.
- Maintains a tabu list. A tabu list is a list of solution points that must be avoided or a list of move that are not allowed. This list is updated based on some memory structure

- Aspiration criteria; this represent exceptions from the tabu list if such moves lead to promising solutions.
- Diversification and intensification

As previously stated, tabus are stored in short term memory and only a fixed amount of information is recorded. The most common used tabus involve recording the last few transformations performed on the current solution and prohibiting reverse transformations

2.7.4.4 Aspiration Criteria

The aspiration criteria allows the algorithm to explore attractive solutions that are prohibited i.e. on the tabu list. The simplest and most commonly used aspiration criterion, which is found in almost all

TS implementations, consists in allowing a move, even if it is tabu, if it results in a solution with an objective value better than that of the current best-known solution (since the new solution has obviously not been previously visited) [9].

2.7.4.5 Termination Criteria

Theoretically, any tabu search algorithm could go on forever. In practice however, the search must be terminated at some point. The most commonly used criteria for terminating a tabu search algorithm include

- Including a fixed number of iterations after which the function should stop
- After a given number of iterations, if there is no improvement in the objective function, the process will terminate
- When the objective function reaches a specified threshold value

2.8 Evaluation of Approaches

The goal of this project is to develop a software to handle the course scheduling process in Ashesi University College. The scheduling algorithms described above have been used in numerous Universities for the same goal however, each university may have its unique constraints and requirements hence one particular algorithm may not work for every situation. For this reason, the evaluation criteria for each algorithm is based on the following criteria:

- Ease of implementation
- Flexibility to handle constraints
- Computation time
- Solution quality

2.8.1 MAS

The first algorithm described was the multi agent system approach (MAS). The MAS approach makes use of numerous autonomous agents representing different departments of the university to automatically generate non-conflicting course schedules. Although an optimal solution is most likely guaranteed with the MAS approach, the resources and time required to implement it is significantly high. Each department and lecturer will have an autonomous agent to handle the scheduling process. The development and implementation of these autonomous agents, if not outsourced, will be very expensive with respect to time, skill and money. This makes the MAS approach difficult to implement and time consuming. The literature does not state how much computational time is involved in generating the schedule, but it is clear that a lot of overhead will be incurred during the process.

The system handles conflicts through a series of back and forth negotiations amongst the autonomous agents. Communication between the agents induce a heavy load on the system. One way to handle this issue could be the use of high speed drives (hardware).

Essentially, although the MAS approach may be able to handle constraints and conflicts efficiently to produce good solutions, the cost of implementation and computation time potentially outweigh the benefits the system provides.

2.8.2 Sequential Methods

After the MAS approach, sequential methods were described next. Sequential methods make use of domain heuristics to assign events sequentially into valid time periods such that no events in the period are in conflict with each other. As stated above, sequential methods tend to be represented by graph coloring techniques hence are sometimes referred to as graph coloring methods. The basic properties of sequential methods are that each time period should correspond to a color in the graph coloring problem and secondly, the vertices of the graph should be colored in a way that no two adjacent vertices should have the same color.

Based on the existing literature, sequential methods are easy to understand and implement however the solution quality is largely dependent on the problem being solved i.e. they tend to not work well with problems that have large solution spaces. One of the basic properties of this method (coloring of vertices should be done in a way that no two adjacent vertices have the same color) will most likely not work well for certain courses that have to be scheduled next to each other.

Basically, although sequential methods are easy to implement, they do not pose the potential to handle constraints effectively, solve problems with large solution space in less computation time or produce good solutions under tight constraints.

2.8.3 Constraint Based Methods

Constraint based methods assign values to variables based on a set of constraints. With these methods, the time table is created with focus on the constraints. Scheduling is done in a way that constraints are not violated. Compared to the other methods under review, constraint based methods are not as easy to implement. They require some level of advanced mathematical knowledge to facilitate their implementation. Their performance with respect to solution quality and computation time is dependent on the problem being solved. Constraint based methods perform better when there is a significant amount of constraint propagation i.e. the instantiation of each variable should allow a reduction in the domain of other variables, reducing the search space eventually. This means that with large solution spaces, if the constraint propagation technique being used is not enhanced, large regions of the solution space will be left unexplored.

Although constraint based methods are not good for exploring large solution spaces, they can be combined with other metaheuristics to prune search trees to help search large neighborhoods in local search methods. If they are however to be used alone, they will be more efficient for exam scheduling.

2.8.4 Genetic Algorithms

Genetic algorithms belong to the metaheuristic family of algorithms. They emulate the biological process of genes and chromosomes. With respect to the criteria listed

and the analysis done, it is safe to say they are easy to understand and implement. They do not require significant knowledge of advanced mathematics. They are also good for exploring large solution spaces. Although genetic algorithms seem good, they are drawn back by their response time. This means that they may take either a significantly large amount of time or significantly small amount of time to compute the solution. They do not have a constant response time. Theoretically, they use a large amount of time to search large solution spaces.

But despite these draw backs, they are still useful for implementing automated schedules.

2.8.5 Simulated Annealing

Another algorithm under the metaheuristic set is the simulated annealing algorithm. Unlike genetic algorithm that mimics the biological process of genes, simulated annealing algorithms emulate the process of annealing metals. The entire process is explained in detail in the previous section. It is worthy to note that the process is relatively easy to implement, even for complex problems (difficult constraints) however it is not efficient in exploring large solutions spaces because of the randomization technique it employs in generating possible solutions. It also tends to be computationally expensive.

Though this method may not be suitable for course scheduling, it could potentially work best for exam scheduling.

2.8.6 Tabu Search Algorithms

The tabu search algorithm makes use of a tabu list in ensuring that the algorithm avoids movement into bad solutions by penalizing or forbidding moves which take

that solution in the next iteration of the algorithm. Tabu search, like simulated annealing is best suited for exam scheduling problems. When presented with large solution spaces, it fails to explore the entire search area due to its nature. It is however easy to implement and can handle constraints fairly well, but its solution quality and computation time will not perform well with the large solution spaces course scheduling problems deal with.

2.8.7 Particle Swarm Optimization (PSO)

Particle swarm optimization is based on the flocking patterns of birds when searching for food. PSO is potentially the best algorithm yet, to be used for course scheduling. PSO techniques, like genetic algorithms, are easy to implement and understand. They produce feasible solutions with less iterations and computation time as compared to genetic algorithms.

All the other algorithms, with the exception of Genetic Algorithms and Particle Swarm Optimization have been ruled out based on the evaluation criteria. An extensive comparison of genetic algorithms and Particle Swarm Optimization would prove useful here but that is beyond the scope of this project. Limiting the selection criteria to the four categories listed above, the best algorithm to implement this project, based on literature is the Particle Swarm Optimization algorithm [10].

Below is a table summarizing the evaluation of the algorithms. The checked boxes indicate the presence of a particular quality for the algorithm in question.

Table 1

	Ease of implementation	Solution Quality	Constraint Handling	Computation time
MAS		✓	✓	
Sequential	✓			
Constraint Based		✓	✓	
Genetic	✓	✓	✓	
Simulated	✓	✓		
Tabu Search	✓	✓		
P.S.O	✓	✓	✓	✓

2.9 FET Scheduling Software

2.9.1 What is FET?

FET is a free scheduling software that is available on the internet for download. It is a generic software meant to handle High School, Secondary and University scheduling. It deals with all possible distinct constraints, such as lack of rooms or

limited time, faced by each institution [11]. The software was recently used by Ashesi in scheduling the courses for all classes of students for the 2014 fall semester and proved itself to be very efficient. The following sections explain the software in further detail.

The first version of the FET software made use of a genetic algorithm. It was slow and had difficulty in solving difficult data sets which could be solved by other similar software's. This made it very inefficient since most timetabling problems have difficult data sets [11].

The genetic algorithm used was later revised and a much more heuristic approach, which the author called "recursive swapping" was used. This algorithm was faster and highly efficient as compared to the previous genetic algorithm used.

2.9.2 How FET works

The first thing to do is to download and install the FET file. After setting up, enter the necessary data. Information entered must include the institution information, days and hours or periods per day, subjects, lecturers' names, student years, student groups and available spaces or rooms.

After entering the necessary data, you will have to construct lessons by assigning the different student groups to lecturers. After the assignment process, click on the generate button to allow the software automatically generate the timetable. The time required in generation a solution is highly dependent on the data set entered. Difficult datasets will typically take a longer time than simple datasets. After the timetable is generated, FET allows you to either print the results or export to some other format such as a csv or HTML file.

That is the basic idea of how the software works.

2.9.3 The FET Algorithm

The program was written in C++ and makes use of a heuristic algorithm known as recursive swapping. The full algorithm is provided below.

1) Sort activities, most difficult first. Not critical step, but speeds up the algorithm maybe 10 times or more.

2) Try to place each activity (A_i) in an allowed time slot, following the above order, one at a time.

Search for an available slot (T_j) for A_i, in which this activity can be placed respecting the constraints.

If more slots are available, choose a random one. If none is available, do recursive swapping:

2 a) For each time slot T_j, consider what happens if you put A_i into T_j. There will be a list of other activities which don't agree with this move (for instance, activity A_k is on the same slot T_j and has the same teacher or same students as A_i). Keep a list of conflicting activities for each time slot T_j.

2 b) Choose a slot (T_j) with lowest number of conflicting activities. Say the list of activities in this slot contains 3 activities: A_p, A_q, A_r.

2 c) Place A_i at T_j and make A_p, A_q, A_r unallocated.

2 d) Recursively try to place A_p, A_q, A_r (if the level of recursion is not too large, say 14, and if the total number of recursive calls counted since step 2) on A_i began is not too large, say 2*n), as in step 2).

2 e) If successfully placed A_p, A_q, A_r, return with success, otherwise try other time slots (go to step 2 b) and choose the next best time slot).

2 f) If all (or a reasonable number of) time slots were tried unsuccessfully, return without success.

2 g) If we are at level 0, and we had no success in placing A_i, place it like in steps 2 b) and 2 c), but without recursion. We have now 3 - 1 = 2 more activities to place. Go to step 2) (some methods to avoid cycling are used here).

The software was made to solve general scheduling problems, not tailored to a particular institution. It includes several features such as availability of buildings in different cities and break times which are not relevant for an institution such as Ashesi. In an effort to enhance the scheduling process at Ashesi, the rest of this project makes an effort to eliminate the irrelevant features and develop a software similar to FET tailored specifically to suit the scheduling needs of Ashesi.

Chapter 3 Implementation

This section describes how the PSO algorithm was used to implement a simple course schedule.

3.1 Background of Implementation and Model used

The foundation of this implementation was derived from James McCaffery's article on Particle Swarm Optimization. It describes the implementation of a very simple PSO algorithm [12]. The fundamental idea is to find approximate solutions to numeric maximization or minimization problems, otherwise known as objective functions.

In Ashesi, courses are preferred to occur at the same time, every other day. This means if a course e.g. Pre-calculus, occurs on Monday at 8:30, it will be preferred for the same Pre-calculus course to occur on Wednesday at 8:30. Given that there are only 5 time periods in a day and each course will be preferred to occur every other day, it will be best to schedule courses for the first two days of the week, i.e. Mondays and Tuesdays, and copy the results for Wednesdays and Thursdays. For this reason, 10 time slots are used in the entire process. The 10 time slots represent the 5 time periods for courses taken on each day (Mondays and Tuesdays) and each time slot represented by an array index.

Each course is randomly assigned a position in an array called course position. The size of the course position array is the number of courses times the number of time

slots. Hence if there are 5 courses, the course position array will be of size five times ten i.e. fifty. This gives each course a chance to occur at any time in the defined time slot.

3.2 Particle Definition

```
public static class Particle{

    int [] coursePosition; //equqivalent to course-Values and/or solution
    double fitness;
    double [] velocity; // best position found so far by this Particle

    int [] bestCoursePosition;
    double bestFitness;
```

The particle class is made up of five fields. A course position array of type int, which represents the position of each course of a particle; a fitness value of type double which holds a value for how good or fit the particular schedule is; a velocity value to help search for, and update the best position found so far; a best course position array of type int to keep track of the best course position; and a best fitness value which represents the associated fitness of the best course position.

```
public Particle(int [] coursePosition, double fitness, double [] velocity, int [] bestCoursePosition,
    double bestFitness)
{
   this.coursePosition = new int [coursePosition.length];
   System.arraycopy(coursePosition, 0, this.coursePosition, 0, coursePosition.length);

   this.fitness = fitness;

   this.velocity = new double [velocity.length];
   System.arraycopy(velocity, 0, this.velocity, 0, velocity.length);

   this.bestCoursePosition = new int [bestCoursePosition.length];
   System.arraycopy(bestCoursePosition, 0, this.bestCoursePosition, 0, bestCoursePosition.length);

   this.bestFitness = bestFitness;
}
```

The particle class has a constructor that accepts the five parameters that correspond to each of the particles defined above. The purpose of the constructor is simply to copy each parameter value into its corresponding data field.

3.3 PSO Algorithm Explanation

The PSO algorithm is fairly simple but it needs to be well understood in order to successfully implement it. After initializing each particle, in the main processing loop, the algorithm is used to update each particle's current velocity based on the particles current velocity, its local information and global swarm information. After, the particle's position is updated using the particles new velocity. A mathematical representation of the algorithm is as follows:

$$\boldsymbol{V}(t+1) = (w * \boldsymbol{v}(t)) + (c_1 * r_1 * (\boldsymbol{p}(t) - \boldsymbol{x}(t)) + (c_2 * r_2 * (\boldsymbol{g}(t) - \boldsymbol{x}(t))$$

$$\boldsymbol{X}(t+1) = \boldsymbol{x}(t) + \boldsymbol{v}(t+1)$$

The first equation updates a particle's velocity. The term $\boldsymbol{v}(t+1)$ means the velocity at time $t+1$. The $\boldsymbol{v}$ indicates that velocity is a vector value and has multiple components such as {1.55, -0.33}, rather than being a single scalar value. The new velocity depends on three terms. The first term is $w * \boldsymbol{v}(t)$. The w factor is

called the inertia weight and is simply a constant like 0.73; $\boldsymbol{v}$ (t) is the current velocity at time t. The second term is $c_1 * r_1 * (\boldsymbol{p}(t) - \boldsymbol{x}(t))$. The c_1 factor is a constant called the cognitive (or personal or local) weight. The r_1 factor is a random variable in the range (0, 1), which is greater than or equal to 0 and strictly less than 1. The $\boldsymbol{p}$ (t) vector value is the particle's best position found so far. The $\boldsymbol{x}$ (t) vector value is the particle's current position. The third term in the velocity update equation is $(c_2 * r_2 * (\boldsymbol{g}(t) - \boldsymbol{x}(t))$. The c_2 factor is a constant called the social—or global—weight. The $r_2$factor is a random variable in the range (0, 1). The $\boldsymbol{g}$ (t) vector value is the best known position found by any particle in the swarm so far. Once the new velocity, $\boldsymbol{v}$ (t+1), has been determined, it's used to compute the new particle position $\boldsymbol{x}$ (t+1) where $\boldsymbol{x}$ (t) represents the particles current position and $\boldsymbol{v}$ (t+1) represents the new velocity [12]. As shown in the equation above, the new position is simply the sum of the current position and the new velocity.

For this project, each particles position is made up of numbers that correspond to a particular timeslot for each course. The idea is that the timeslot with the largest number is the one that will be chosen for the particular course. This simply means that when the particles position is being updated, the timeslot for the given course is being changed. In this manner, the PSO algorithm is able to explore various possible schedules in the solution space.

3.4 Declaring and Initialization Variables

In the algorithm class, a random object is used to generate the cognitive and social random numbers, as well as the initial velocities and positions of each particle.

```
final int NUMBER_OF_LECTURERS = 5;
final int NUMBER_OF_COHORTS =1;
final int NUMBER_OF_ROOMS=5;
final int NUMBER_OF_COURSES=5;

final int NUMBER_OF_PARTICLES=10;
int numberIterations = 1000;
int iteration = 0;

final int DIM = 10;//number of x-values or positions or time slots

int coursePositionSize = NUMBER_OF_COURSES *DIM;
int lecturersSize = NUMBER_OF_LECTURERS *DIM;
int cohortsSize = NUMBER_OF_COHORTS *DIM;
int roomsSize = NUMBER_OF_ROOMS *DIM;

int lecturerNumberOfConflicts = 0;
int cohortNumberOfConflicts = 0;
int roomNumberOfConflicts = 0;
//int updateConflicts=0;

int numberOfConflicts=0;

int [] randomCoursePosition = null;

//Random numbers will be limited from 1 to 5
int minX = 1;
int maxX = 5;
```

After instantiating the random object, key PSO variables are declared and assigned. As an initial test of the program, 5 lecturers, 1 student cohort, 5 rooms and 5 courses are used in the process. 10 particles are used for 1000 iterations of the processing loop. The Dim variable represents the number of time slots in an array/schedule. The size of each array is the number of elements in the array times the number of factors being scheduled. The result of this multiplication is held in an integer variable which would later be assigned to an array as its size. The minX and

maxX values are used to limit the range within which random numbers are assigned as positions.

A swarm array of particle objects is created with its size as the number of particles. An array to hold the best global position of any particle is declared along with the associated best global fitness. The global fitness holds the value of the highest possible double. The minV and maxV values are set to -0.5 and 0.5 respectively. This helps to ensure that the velocity is not arbitrarily huge.

3.6 Initialization

3.6.1 Particle Position Initialization

```
System.out.println("\nInitializing swarm with random positions/solutions");
for(int i = 0; i<swarm.length; i++)// initialize each Particle in the swarm
{
    randomCoursePosition = new int[coursePositionSize];//Dim

    for(int j=0; j<randomCoursePosition.length; j++)//initializing courses w
    {
       // int lo = minX;
       // int hi = maxX;
        //randomCoursePosition[j] = (hi - lo) * ran.nextInt() + lo;
        randomCoursePosition[j]= (int)(Math.random()*5);
        //System.out.println("course at position " + j + " has value " + ran
    }
```

The initialization stage begins with assigning random values to the timeslot windows for every course. An array called random course position is used to hold the random values of all the courses given.

The random course array of size fifty is split up into smaller blocks of size ten which represent the course schedule for each course, also known as a course particle. Each course can be identified with its course code. After the initialization stage, the

time slot with the highest number for a given course is selected as the time slot for that course. The corresponding timeslot must be blocked out for the lecturer and the student cohort and a room must be selected for that course to be held in at that time slot. In the lecturer array, the lecturer teaching a particular course, identified by the particular course code is selected. If the particular lecturer's time slot is empty at the time selected for the course to occur, the lecturer's time slot (the particular index) is filled with the course code for that course. If the time slot already has a value, meaning the lecturer has already been booked for that period, the next empty slot is selected. If that slot is also booked then the next empty slot after the previously selected slot is checked. The process of checking for the next slot iterates in a while loop until the end of the course length (the tenth index) is reached. If it finds an empty slot, it assigns it the value otherwise, a conflict is assumed and the conflict array is updated with a defined number representing a conflict in the lecturer schedule. The same process is repeated for the student cohort and room schedules.

3.6.2 Fitness

After initializing the schedules for the lecturer, courses, rooms, and student cohorts, the fitness value is computed. The fitness value is based on the number of conflicts that occur in assigning courses to a particular schedule. The objective function for computing the fitness values sums up the number of conflicts that occur in each schedule and assigns the value as a fitness for the schedule. The ultimate goal is to find the schedule with the least fitness.

```
public static int ObjectiveFunction(int[]a){
    int lConflict = 0;
    int cConflict = 0;
    int rConflict = 0;
    //int updateConflict = 0;
    final int LECTURER_CONFLICT_VALUE=77;
    final int COHORT_CONFLICT_VALUE=88;
    final int ROOM_CONFLICT_VALUE=99;

    for(int i=0; i<a.length; i++){
        if(a[i]==LECTURER_CONFLICT_VALUE){
            lConflict++;
        }
        if(a[i]==COHORT_CONFLICT_VALUE){
            cConflict++;
        }
        if(a[i]==ROOM_CONFLICT_VALUE){
            rConflict++;
        }
    }
    return (lConflict + cConflict + rConflict);
}//objective function
```

Each conflict in each schedule is assigned a particular value. If a conflict occurs in any of the schedules, the arbitrary value for that schedule is placed in the conflict array. The objective function simply searches through the array for the number of times a particular conflict occurs. At the end, it sums up the number of times each conflict happened and returns an integer value as the fitness of the particle in question.

After initializing the fitness, a random velocity is assigned to each particle in the nth dimension. This means each course particle will be assigned a random velocity of some value.

The initialized particles are added to the swarm and the global best position is set to the current position. It is worth noting that at the initialization stage, the current position and fitness are equal to the global best position and best fitness.

3.7 Updating

The updating process makes use of local and global constants in determining the new velocity and position. The figures used in the diagram below were recommended by a research paper that investigated the effects of various PSO parameter values on a set of benchmark minimization problems [13]. The update loop iterates a thousand times and on each iteration, new values are generated.

```
double w = 0.729; // inertia weight. see http://ieeexpl
double c1 = 1.49445; // cognitive/local weight
double c2 = 1.49445; // social/global weight
double r1, r2; // cognitive and social randomizations
```

3.7.1 Updating Velocity

```
for (int i = 0; i < swarm.length; i++) // each Particle
{
   Particle currP = swarm[i];

  for (int j = 0; j < currP.velocity.length; j++) // each x value of the velocity
  {
    r1 = ran.nextDouble();
    r2 = ran.nextDouble();

    newVelocity[j] = (w * currP.velocity[j]) +
      (c1 * r1 * (currP.bestCoursePosition[i] - currP.coursePosition[j])) +
      (c2 * r2 * (bestGlobalPosition[i] - currP.coursePosition[j]));

    if (newVelocity[j] < minV)
    {
      newVelocity[j] = minV;
    }
    else if (newVelocity[j] > maxV)
    {
      newVelocity[j] = maxV;
    }
    //System.out.println("Updated velocity = " + newVelocity[i]);
  }
   System.arraycopy(newVelocity, 0, currP.velocity, 0, currP.velocity.length);
```

At the beginning of the iteration, a new array is created to hold the value of each new velocity. For each particle object in the velocity array, random number variables are generated and the velocity is then updated for each particle in the array. As seen from the image above, the velocity is updated using the update formula and assigned to the new velocity array. After computing the velocity, a check is made to ensure that the new velocity is within the range of the maximum and minimum velocity. This check is done to prevent the position from spinning out of bounds. The current particle objects velocity is updated using Java's array copy method.

3.7.2 Updating Positions

The updating of positions requires that the new velocity be added to the current position to generate a new position. For the purpose of the project and the definition of the objective function, this step has been slightly amended.

For the position update, a new velocity is computed. This velocity is added to the current position to get a new position. After the new position is obtained, the lecturer and student cohort schedules are updated to reflect the change. If there is no value in the new position that has been selected, the schedules are updated without any issues. If there is a value already in the particular position, meaning a conflict occurs while trying to update the lecturer and student cohort schedules (probably because that position is occupied), the conflict array is updated. A new position is searched for again and the process is repeated.

The new position is then copied to the current particle's position and the objective function is called to assess the fitness of the schedule. The current particle's best fitness and the best global fitness are then sought for by comparing their values to the new fitness obtained. If they are found, their values are then copied to the current particles best fitness and best global position.

Chapter 4 Results

4.1 Initialization of Particles

This section explains the results obtained from the program using ten particles and an iteration limit of 1000. This example assumes that there are five time slots in which lectures can occur in a day and the time for a lecture is preferred to be the same for any other day on which the lecture must occur. It also assumes that a lecture is preferred to happen every other day. This means that if Pre-calculus is taught at 8:30 on Mondays, it will be preferred that it is taught at the same time on Wednesdays.

The database used has 5 courses, 5 lecturers and 5 rooms and 1 student cohort. The diagrams below show the initialization of each particle.

Figure 4.1 The Schedule corresponding to the initial random particle position 0

```
=============================================================================================================
Particle: 0 at 0 :Statistics-a is taught by Mrs. Awuah in room Room 216  to cohort 2017a at time 1
=============================================================================================================
=============================================================================================================
Particle: 0 at 1 :Precalculus-a is taught by Joseph Mensah in room Room 217  to cohort 2017a at time 17
=============================================================================================================
=============================================================================================================
Particle: 0 at 2 :Written & Oral-a is taught by Oduro Frimpong in room Room 218  to cohort 2017a at time 20
=============================================================================================================
=============================================================================================================
Particle: 0 at 3 :Text & Meaning-a is taught by Andrew Nunekpeku in room Room 115  to cohort 2017a at time 30
=============================================================================================================
=============================================================================================================
Particle: 0 at 4 :Calculus-a is taught by Anthony Spio in room Room 116  to cohort 2017a at time 40
=============================================================================================================
```

It can be observed from the diagram that the first particle has statistics occurring at time 1 (10:10-11:40 on Mondays), pre-calculus at time 17 (10:10-11:40 on Tuesdays), Written and Oral at time 20 (8:30-10:10 on Mondays), Text and Meaning at time 30 (8:30-10:10 on Mondays) and calculus at time 40 (8:30-10:10 on Mondays).

Figure 4.2 The Schedule corresponding to the initial random particle position 1

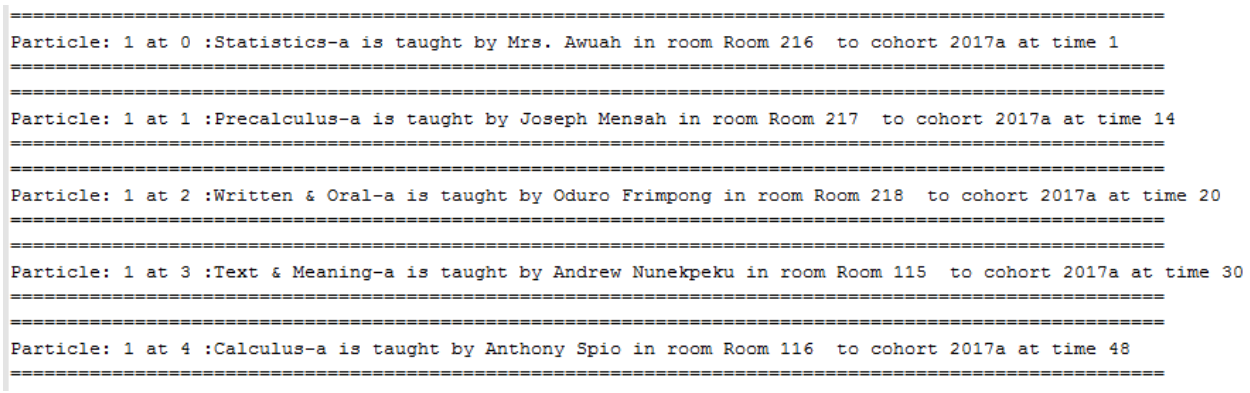
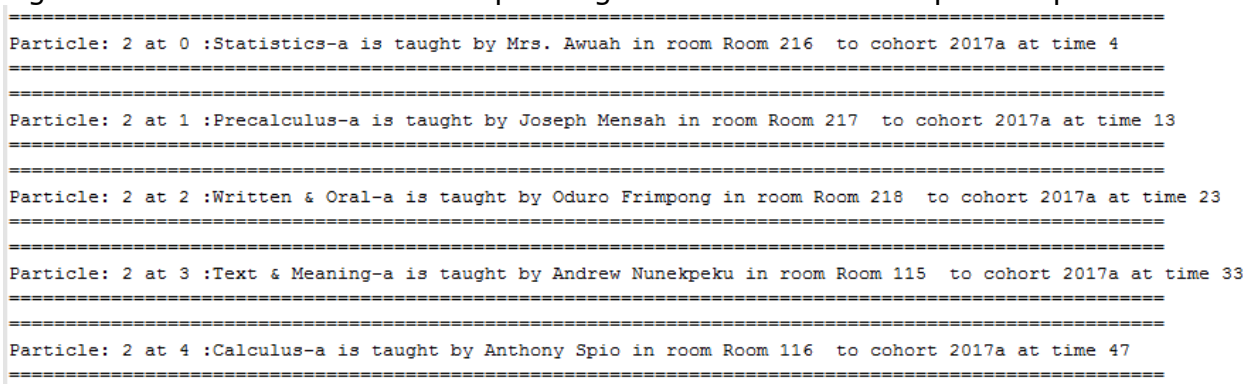


Figure 4.3 The Schedule corresponding to the initial random particle position 2



4.2 Fitness and Velocity

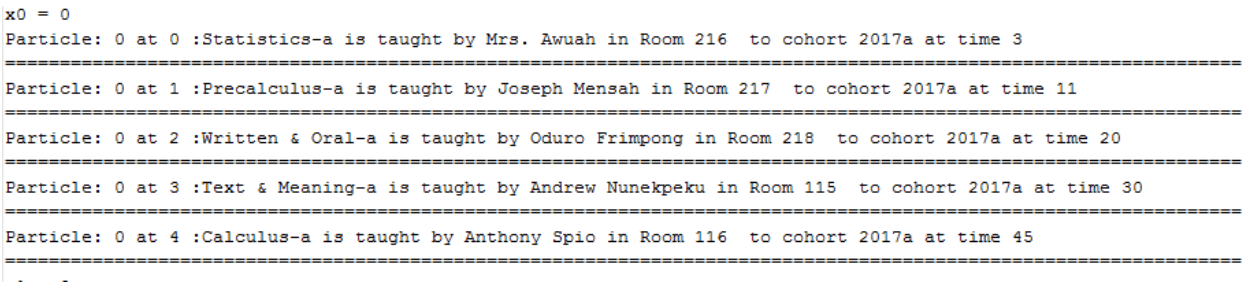
As the particles are being assigned positions, they are given initial velocities because the algorithm assumes that the particles are moving in the solution space in search for the best solution. A fitness value is also computed for each particle

based on the number of conflicts that occur in assigning time slots to the lecturers, courses and rooms. The fitness tells how good a particular solution is. Since the idea is to minimize the number of conflicts that occur, the solution with the least fitness value represents the best solution.

After the initialization stage is complete, the best fitness of each particle is computed to get the best initial position.

The initial fitness computed has a value of 30. Each particle is assigned the best schedule in the swarm to get the initial best solution. Figure 4.11 shows the best initial schedule assigned to each particle in the swarm.

Figure 4.11 The Schedule corresponding to the initial best particle position



4.3 Updating

After the best initial solution is found, the program enters the main Particle Swarm Optimization loop where each particles current position is updated based on its current position, a computed velocity, local and global swarm information.

4.3.1 Updating Velocity

Figure 4.12 The Velocity Update Computation

```
newVelocity[j] = (w * currP.velocity[j]) +
  (c1 * r1 * (currP.bestCoursePosition[i] - currP.coursePosition[j])) +
  (c2 * r2 * (bestGlobalPosition[i] - currP.coursePosition[j]));
```

As discussed in the previous section, updating the position requires the computation of a new velocity. Figure 4.12 shows how the new velocity is computed to facilitate the position update in the program. The weights w, c1 and c2 are constants that represent the velocity, local and global weights respectively. R1 and r2 are random numbers used to represent local and global randomizations.

Figure 4.13 shows the best final schedule for each particle. The detailed definition of this computation has been explained in chapter 3.

4.3.2 Updating Positions

Figure 4.13 The Position Update Computation

```
newPosition[j] = currP.coursePosition[j] + (int) newVelocity[j];
```

The new position of each particle is simply the sum of the new velocity and the current position of the particle. This is what causes particles to move around in search of the best position or schedule.

After each particle has been updated with a new position and fitness, the best fitness is sought for by comparing the different fitness values in the swarm to each other.

When the best fitness is found, its position is copied to a best Global Position array and the schedule at that fitness is returned as the best schedule. Figure 4.14 shows the best schedule in the swarm.

Figure 4.14 The Schedule corresponding to the final best particle position

```
Particle: 0 at 0 :Statistics-a is taught by Mrs. Awuah in Room 216  to cohort 2017a at time 1
=============================================================================================================
Particle: 0 at 1 :Precalculus-a is taught by Joseph Mensah in Room 217  to cohort 2017a at time 17
=============================================================================================================
Particle: 0 at 2 :Written & Oral-a is taught by Oduro Frimpong in Room 218  to cohort 2017a at time 21
=============================================================================================================
Particle: 0 at 3 :Text & Meaning-a is taught by Andrew Nunekpeku in Room 115  to cohort 2017a at time 31
=============================================================================================================
Particle: 0 at 4 :Calculus-a is taught by Anthony Spio in Room 116  to cohort 2017a at time 42
=============================================================================================================
```

4.3.3. Second Test of PSO Algorithm

Another example was conducted to test the programs potential to handle complex situations. The database used had 2 student cohorts, 5 rooms, 5 lecturers and 10 courses. The diagrams below show the initial and final solutions computed by the program. Each course has 10 time slots, hence if there are 5 courses, there will be a total of 50 timeslots (i.e. 5 times 10) divided into 5 groups of 10 slots per group. This simply means that the first course will have timeslots 0 to 9, the second 10 to 19, the third 20 to 29, the fourth 30 to 39 and the fifth 40 to 49. For each course, the first five timeslots represent courses happening on the first day of the week, from 8:30 through to 4:40. The second five represent courses happening on the second day from 8:30 through to 4:40. On a typical day in Ashesi, the first time slot is from 8:30 to 10:00. The second is from 10:10 to 11:40. The third is from 11:50 to 1:20. The fourth is from 1:30 to 3:00 and the fifth is from 3:00 to 4:40.

Figure 4.15 The Schedule corresponding to the initial best particle position

```
x9 = 3
Particle: 9 at 0 :Statistics-a is taught by Mrs. Awuah in Room 216  to cohort 2017a at time 2
================================================================================================================================
Particle: 9 at 1 :Precalculus-a is taught by Joseph Mensah in Room 217  to cohort 2017b at time 14
================================================================================================================================
Particle: 9 at 2 :Written & Oral-a is taught by Oduro Frimpong in Room 218  to cohort 2017a at time 0
================================================================================================================================
Particle: 9 at 3 :Text & Meaning-a is taught by Andrew Nunekpeku in Room 115  to cohort 2017b at time 31
================================================================================================================================
Particle: 9 at 4 :Calculus-a is taught by Anthony Spio in Room 116  to cohort 2017a at time 43
================================================================================================================================
Particle: 9 at 5 :Statistics-b is taught by Mrs. Awuah in Room 216  to cohort 2017b at time 50
================================================================================================================================
Particle: 9 at 6 :Precalculus-b is taught by Joseph Mensah in Room 217  to cohort 2017a at time 0
================================================================================================================================
Particle: 9 at 7 :Written & Oral-b is taught by Oduro Frimpong in Room 218  to cohort 2017b at time 77
================================================================================================================================
Particle: 9 at 8 :Text & Meaning-b is taught by Andrew Nunekpeku in Room 115  to cohort 2017a at time 0
================================================================================================================================
Particle: 9 at 9 :Calculus-b is taught by Anthony Spio in Room 116  to cohort 2017b at time 0
================================================================================================================================
```

Figure 4.15 shows the initial best solution computed by the program. It can be observed that for cohort 2017a, there are a few conflicts in the schedule. For Written and Oral-a (Particle 9 with course id 2), Pre-calculus –b (Particle 9 with course id 6) and Text and Meaning –b (Particle 9 with course id 8) taken by 2017a, they happen at the same time. One of the fundamental constraints this projects works with is that no student or lecturer can be in the more than one places at the same time. This problem is solved in the final best solution shown figure 4.16.

Figure 4.16 The schedule corresponding to the final best particle position

```
Particle: 0 at 0 :Statistics-a is taught by Mrs. Awuah in Room 216  to cohort 2017a at time 1
================================================================================================================================
Particle: 0 at 1 :Precalculus-a is taught by Joseph Mensah in Room 217  to cohort 2017b at time 15
================================================================================================================================
Particle: 0 at 2 :Written & Oral-a is taught by Oduro Frimpong in Room 218  to cohort 2017a at time 21
================================================================================================================================
Particle: 0 at 3 :Text & Meaning-a is taught by Andrew Nunekpeku in Room 115  to cohort 2017b at time 31
================================================================================================================================
Particle: 0 at 4 :Calculus-a is taught by Anthony Spio in Room 116  to cohort 2017a at time 40
================================================================================================================================
Particle: 0 at 5 :Statistics-b is taught by Mrs. Awuah in Room 216  to cohort 2017b at time 50
================================================================================================================================
Particle: 0 at 6 :Precalculus-b is taught by Joseph Mensah in Room 217  to cohort 2017a at time 61
================================================================================================================================
Particle: 0 at 7 :Written & Oral-b is taught by Oduro Frimpong in Room 218  to cohort 2017b at time 70
================================================================================================================================
Particle: 0 at 8 :Text & Meaning-b is taught by Andrew Nunekpeku in Room 115  to cohort 2017a at time 82
================================================================================================================================
Particle: 0 at 9 :Calculus-b is taught by Anthony Spio in Room 116  to cohort 2017b at time 90
--------------------------------------------------------------------------------------------------------------------------------
```

As observed, the conflicting times have been resolved. The new schedule solves the conflicts that occurred in the previous schedule.

Chapter 5

Conclusion

The FET software recently used by the Ashesi in the scheduling of courses utilizes a genetic and heuristic method known as recursive swapping. The algorithm efficiently solves for the most difficult timetabling cases since it was made to accommodate the general scheduling needs of faced by universities and high schools.

It is observable from the implementation that the algorithm was simple to understand. It did not involve any complex mathematical equations like most of the other existing scheduling algorithms. The velocity and position calculation can potentially solve more complex optimization problems. The implementation of Particle Swarm Optimization does not involve any mutation or overlapping calculations like that of genetic algorithm. The search for the optimal solution is carried out by the speed of the particle and only particles with good information (fitness values) are allowed to transmit information to other particles.

The parameters used to test the PSO algorithm were applied to the FET software to compare both algorithms based on the solution quality, ease of implementation, constraint handling and computation time. It was realized that FET performs better with respect to the solution quality. The PSO algorithm on the other hand is easier to implement as compared to the recursive swapping algorithm used in FET. The recursive swapping technique is fundamentally a combination of a genetic and heuristic algorithm which requires advanced mathematical computations. Both approaches however handle constraints well.

In conclusion, although FET produces a much better solution, the implementation is more complex than PSO. For a small institution such as Ashesi, the PSO algorithm will work just fine.

References

[1] E. Burke, K. Jackson, J. Kingston and R. Weare, "Automated University Timetabling: The State of the Art," *The Computer Journal,* vol. 40, no. 9, pp. 565-571, 7 July 1997.

[2] E. K. Burke and S. Petrovic, "Recent Research Directions in Automated Timetabling," *European Journal of Opreational Reasearch,* vol. 140, no. 2, pp. 266-280, 16 July 2002.

[3] M. Oprea, "MAS_UP-UCT: A Multi-Agent System for University Course Timetable Scheduling," *International Journal of Computers, Communications and Control,* vol. 2, no. 1, pp. 94-102, January 2007.

[4] M. H. Williams and K. T. Milne, "The Performance of Algorithms for Colouring Planar Graphs," *The Computer Journal,* vol. 27, no. 2, pp. 165-170, 1984.

[5] S. C. Brailsford, C. N. Potts and B. M. Smith, "Constraint Satisfaction Problems: Algorithms and Applications," *European Journal of Operational Research,* vol. 119, pp. 557-581, October 1998.

[6] S. Ólafsson, "Metaheuristics," in *Handbooks in Operations Research and Management Science VII*, Iowa, 2006, pp. 633-654.

[7] J. Heaton, "Heaton research: Understanding Genetic Algorithms," 23 December 2007. [Online]. Available: http://www.heatonresearch.com/articles/8/page3.html. [Accessed 24 February 2014].

[8] Q. Bai, "Analysis of Particle Swarm Optimization Algorithm," *Computer and Information Science,* vol. 3, no. 1, pp. 180-184, February 2010.

[9] M. Gendreau and J. Y. Potvin, "Tabu Search," in *Hand Book of Metaheuuristics*, 2010, pp. 41-59.

[10] D. Adrianto, "COMPARISON USING PARTICLE SWARM OPTIMIZATION AND GENETIC ALGORITHMS FOR TIMETABLE SCHEDULING," *Journal of Computer Science,* vol. 2, no. 10, pp. 341-346, 2014.

[11] L. Lalescu and V. Dirr, "FET Free TimeTabling Software," [Online]. Available: http://lalescu.ro/liviu/fet/doc/. [Accessed 11th January 2014].

[12] J. McCaffery, "Artificial Intelligence: Particle Swarm Optimization," August 2011. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/hh335067.aspx. [Accessed 11 February 2014].

[13] K. E. Parsopoulos and M. N. Vrahatis, "Particle Swarm Optimization Method for Constrained Optimization Problems," *Frontiers in Artificial Inteligence and Applications,* vol. 76, no. Intelligent Technologies-Theory and Application, pp. 214-220, 2002.