Lebanese American University

Bee Colony Algorithm for Assigning Proctors to Exams

By

Mohamad Kassem Taha

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

School of Arts and Sciences

June 2013

Lebanese American University
School of Arts and Sciences

Thesis Proposal Form

Name of Student: Mohamad Taha I.D.#: 200402286

Division: Computer Science & Math

On (dd/mm/yy) 20/09/12, has presented a Thesis proposal entitled:

Bees Colony Algorithm for Proctor Assignment to exams.

in the presence of the Committee members and Thesis Advisor:

Nashat Mansour [redacted] 20/9/12
(Name, signature, and date of the Thesis Advisor)

Abbas Tarhini [redacted] 20/9/2012
(Name, signature, and date of Committee Member)

Azzam Mourad [redacted] 20/9/2012
(Name, signature, and date of Committee Member)

Comments/Remarks/Conditions to Proposal:

- more precise definition of variables
- use real data set for experimentation/evaluation
-

Date: 21/9/12 Acknowledged by [redacted]
(Dean of Graduate Studies/School of .....)

cc: Department Chair
Thesis Advisor
Student
Dean of Graduate Studies/School Dean

LAU LEBANESE AMERICAN UNIVERSITY

Lebanese American University

School of Arts and Sciences - Beirut Campus

Thesis Defense Result Form

Name of student Mohamed Taha I.D: 2[redacted]

Program / Department: Comp. Sc. & Math

Date of thesis defense: June 6, 2013

Thesis title: Bee Colony Algorithm for Assigning Proctors to Exams.

Result of Thesis defense:

X Thesis was successfully defended. Passing grade is granted

X Thesis is approved pending corrections. Passing grade to be granted upon review and approval by thesis Advisor

" Thesis is not approved. Grade NP is recorded

Committee Members:

Advisor: N. Mansour [redacted] June 6, 2013
(Name and Signature)

Committee Member: Azzam Mourad [redacted]
(Name and Signature)

Committee Member: Abbas Tarhini [redacted]
(Name and Signature)

Advisor's report on completion of corrections (if any): Done

Changes Approved by Thesis Advisor: N. Mansour Signature [redacted]

Date: 7 June 2013 Acknowledge by [redacted]

(Dean, School of [redacted])

Cc: Registrar, Dean, Chair, Advisor, Student

LAU Lebanese American University

LEBANESE AMERICAN UNIVERSITY

School of Arts and Sciences - Beirut Campus

Thesis Approval Form

Student Name: Mohamad Taha I.D. #: 200402285

Thesis/Project Title: Bee Colony Algorithm for Assigning Proctors to exams.

Program : Master of Computer Science

Department : Computer Science & Mathematics

School : **School of Arts and Sciences**

Approved by :

Thesis/Project Advisor: Nashat Mansour Signature : [redacted]

Member : Azzam Mourad Signature : [redacted]

Member : Abbas Tarhini Signature : [redacted]

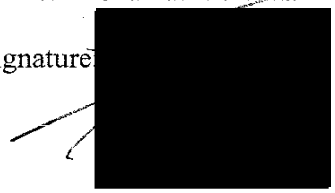
Date : June 6, 2013

Thesis Copyright Release Form

Lebanese American University Non-Exclusive Distribution License

By signing and submitting this license, you (the author(s) or copyright owner) grants to Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video. You agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. You also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Name: Mohamad K. Taha

Signature: 

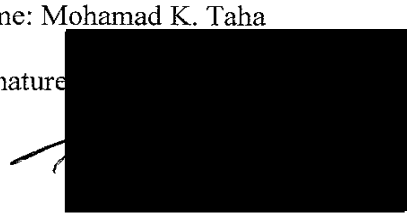
Date: 24/05/2013

Plagiarism Policy Compliance Statement

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Mohamad K. Taha

Signature 

Date: 24/05/2013

Acknowledgments

First of all I would like to thank Allah Almighty for giving me courage and support in order to accomplish the task of my Master's Thesis. Secondly I would like to thank my supervisor Dr. Nashat Mansour for his great support not only in my thesis but during my whole B.S and M.S periods. I am also grateful to Dr. Azzam Mourad and Dr. Abbas Tarhini and all of my teachers at the Computer Science and Math Department. Special thanks also go to my family and friends and to anonymous person(s) which I wish to be with them again.

To my loving parents, brothers, sister, family, and friends.

Bee Colony Algorithm for Assigning Proctors to Exams

Mohamad Kassem Taha

Abstract

Proctor assignment refers to assigning proctors to exams with the objective of having the appropriate number of proctors assigned to exams, subject to conditions such as minimizing the load of proctoring and preventing any conflicting assignments. This problem is intractable and, hence, heuristics algorithms are needed to provide good solutions. In this thesis, we propose a new solution for the proctor assignment to exams problem based on the Bee Colony meta-heuristic algorithm. The Bee Colony algorithm is a recent population-based search algorithm which imitates the behavior of the swarms of honey bees during the process of collecting food. The algorithm performs a neighborhood search combined with random search. We apply the Bee Colony algorithm to previously published data. Experimental results show good solutions that maximize the preferences of the proctors while preserving the fairness of the workload given to proctors.

Keywords: Proctors, Bee Colony Algorithm, School and University Exams, Constraint based Assignment, Meta-heuristics, NP-Complete Problem, Intractable Problem

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Most schools and universities struggle every year when scheduling exams and assigning to them the appropriate number of proctors without having any conflicts. This challenge gives rise to the problem of assigning proctors to exams which is of vital importance to schools and universities. Proctors have several time constraints because those who proctor can be students or teaching assistants (TAs). As such, they have a limited amount of time during which they are available to proctor their assigned or to be assigned exams. Moreover, we have to consider that the proctors who are still taking courses (in our case graduate students) have their own exams as an additional constraint.

The Proctor Assignment to Exams Problem (PAE) is considered to be an extension case of the Generalized Assignment Problem 'GAP', (Marti & Laurenco & Laguna, 2000). In fact, the main aim of the Generalized Assignment Problem is to assign tasks to agents while minimizing the cost of such assignment taking into consideration that an agent has limited or restricted resources. Furthermore, the Generalized Assignment Problem entails that only one agent is to be assigned to a certain task (Marti et al., 2000).

The proctors' assignment to exams is known to be a difficult chore due to the reason that the number of is usually large and the constraints that we need to respect are complex. As a result, the team responsible for assigning proctors to exams spends enormous amount of time in order to complete this task manually, but with unavoidable error rate which maybe high in some cases (Awad & Chinneck, 1998).

For the above mentioned reasons automating this process or chore would not only save time but would also achieve many of the desired goals that were very hard to accomplish using the manual procedure and mainly optimizing the proctors' preferences.

For illustration, assuming that our problem is to assign a number of proctors p for exams that should take place over a number of time slots t and for each of these slots there are a different assignments, then there will be $\frac{t \times p!}{(p-a)!}$ different combinations of assignments. As an example, if we take p=100, t=45, and a=30, we will get 3.5×10^{59} solutions (Awad et al., 1998). Taking into consideration all the issues discussed so far, enumerations and deterministic algorithms are not suitable and would fail for this kind of problems.

Two attempts were made to solve this problem through the usage of Scatter Search algorithms and Genetic algorithms. The authors who employed a Genetic algorithm (Awad et al., 1998) solution were only concerned with producing the minimal acceptable solution that meets the client needs as we can see they did a very basic Genetic based algorithm to solve the problem instead of using an improved version or even other methods that will give better results. However the authors of the scatter search solution (Marti et al., 2000) produced acceptable good results compared to other methods like IP algorithms. The details will be presented in Chapter 3.

For the sake of solving the PAE problem, we introduce a new solution based on the Bee Colony meta-heuristic algorithm. Choosing this algorithm was based on many criteria, most importantly its applications despite they are few but they are considered important and they produced good result.

The Bee Colony algorithm has been adapted to several NP-Complete and NP-Hard problems such as the fragment assembly problems and timetabling problems (Alzaqebah, 2011), (Saha, 2012). We also modified the scatter search algorithm (SS) of (Marti et al., 2000) to a multi-objective scatter search (MOSS) version applied to PAE. The two proposed meta-heuristics BCA and MOSS were applied to previously published data and were compared to SS and CPlex integer programming (IP) software. The obtained results showed that the BCA outperforms all of the algorithms previously tried to solve the problem, including the IP algorithm.

This thesis is organized as follows. Chapter two presents and describes the proctor assignment problem. It frames the objective functions of the PAE problem as well as lists all the constraints that directly related to the problem. Furthermore, a small discussion of the solution evaluation is presented. Chapter three presents the previous work done to solve the PAE problem using different meta-heuristic algorithms. In addition, it presents previous work that was done to solve similar assignment problems as well as some of the applications of the bee colony algorithm. Also, it includes a critical assessment for the proctor assignment problem where it shows the advantages and disadvantages of the prosed algorithmic solution. Chapter four provides an overview of the bee colony methodology, and mainly over its implementation for our main PAE problem. Additionally, the Multi-objective version of the scatter search and the Cplex software implementations are also introduced. Chapter five presents and discusses the experimental results that were obtained when testing our main algorithm in comparison with other algorithms using real-life data. Chapter six presents summary of my work and my recommendations for future work. Finally, appendix A introduces the software designed for the PAE problem, as well as the description of our data feed.

Chapter 2

Proctor Assignment to Exams Problem Description

2.1 Problem Definition

The proctor assignment problem is an NP-Hard problem, additionally there are a set of additional constraints that should be satisfied which also adds to the complexity of the problem. The constraints can be described as follows:

A- A specific number of proctors have to proctor each exam: Since the number of students differs per class, each exam has a different number of required proctors that need to be assigned in order to meet the proctoring requirement.

B- A proctor has a limited number of hours that he can devote to proctor exams: Not all proctors have the same amount of free time, some might be part time students, some might be full time students, some might be only teaching assistants, and some might be staff. So the assignment process should take into consideration this kind of time limitation per proctor.

C- A proctor can only be assigned to one exam at a time: proctoring assignment should not overlap with each other for the same proctor in order to avoid conflicts.

D- A proctor can only be assigned to an exam that does not conflict with his own exam: as mentioned in the list of constraints, proctors have to sit for their own exams, therefore it is expected that they are inclined to proctor

on particular days while trying to avoid others. As an example, proctors might prefer not to be assigned to an exam that is scheduled one day before some of his/her exams. Accordingly, one of the main objectives of the PAE problem is to assign proctors to exams in such a manner as to maximize these preferences.

E- Load balance between proctors: another consideration to be taken into account before assigning proctors to exams is to adopt a balanced approach towards the workload among all proctors since the unfairness in the workloads would result in a conflict between the proctors from one side and between the proctors and the administration office of the university or the school from the other side.

In order to measure the fairness of the workload that resulted from the assignments of proctors to exam, we could minimize the difference between proctors who have been given a huge amount of workload and the proctors who have been given a small amount of workload. To implement this idea, what we can do is that we can formulate our algorithm or model so that we maximize the minimum workload that is going to be assigned to each proctor. This workload can be calculated by dividing the number of hours assigned to each proctor by the number of available hours to him.

2.2 Problem Formulation

We assume that an exam day is split into two periods: the first period lies from 8:00 AM till 2:00 PM while the second period lies from 2:00 PM till 9:00 PM.

Therefore, if d is the number of exam days, then the number of exam periods will be $k = 2d$.

It should be noted that an exam is assumed to start and end in the same day, also it can take place over one or two periods. Furthermore, proctors choose their preferences for certain periods and these are successively interpreted as preferences for exams that falls under these periods. To illustrate this scenario, if a proctor has a high preference for a specific period, it is assumed that this high preference of the proctor will be applicable to all the exams that are scheduled within the given period. Regarding the exams that are scheduled over two periods, we will assume that the period with the lower preference value will be the same preference for this exam.

We denote by J the set of m exams ($j = 1, ..., m$) and I the set of n proctors ($i = 1, ..., n$), also the parameters for this problem are as follows:

a_i = maximum number of available hours for proctor i

bj = number of hours associated with exam j

tj = number of proctors required for exam j

cij = preference of proctor i for exam j, which is an integer between 0 and 2

Pi = the set of exams that overlap with any proctor i's exams

Tk = the set of exams scheduled in period k

d = number of exam days

The variable xij is a flag and can be described as follows:

xij = 1 if proctor i is assigned to exam j

otherwise $x_{ij} = 0$

The objectives of the (PAE) problem are (Marti et al., 2000):

$$\text{Max } f(x) = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \quad (1)$$

$$\text{Min } g(y) = y \quad (2)$$

Subject to:

$$\sum_{j=1}^{m} b_j x_{ij} \le a_i \qquad \text{i = 1….n} \quad (3)$$

$$\sum_{j=1}^{m} b_j x_{ij} - a_i y \ge 0 \qquad \text{i = 1….n} \quad (4)$$

$$\sum_{i=1}^{n} x_{ij} = t_j \qquad \text{j= 1….m} \quad (5)$$

$$\sum_{j \in T_k} x_{ij} \le 1 \qquad \text{i = 1….n; k = 1….2d} \quad (6)$$

$$x_{ij} = \{0,1\} \qquad \text{i = 1….n; j = 1….m} \quad (7)$$

$$x_{ij} = 0 \qquad \text{j} \in \text{P}_\text{i} \quad (8)$$

$$y \ge 0 \quad (9)$$

Formulas (1) and (2) represent the objective functions of the problem that needs to be maximized. Formula (1) represents the sum of the preferences of the proctors' assignments while formula (2) aims to minimize the proctors utilization which as mentioned earlier is defined as the ratio of the hours that a proctor is assigned to exams over his/her available hours. The purpose of the PAE problem is to satisfy these objective functions concurrently. Formulas from (3) till (9) represent the constraints we have.

Formula (3) ensures that the number of the assigned hours for each proctor does not exceed his/her total available hours. As for formula (4) it determines the minimum proctors' utilization. Formula (5) ensures that all exams are assigned the required number of proctors. Formula (6) averts a proctor to be assigned more than one exam during a specific period. Formula (7) implements the conditional flag on what is considered to be a decision variable. Formula (8) ensures that proctors are not assigned to exams that overlap with their own exams. Formula (9) confines the y-variable to have only positive values.

As described earlier in the PAE problem we have two objective functions, this forces us to combine them into one weighted objective function.

The weighted function for the PAE problem is:

$$h(x) = f'(x) + \alpha g(y) \tag{10}$$

where

$$f'(x) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} \frac{c_{ij}}{c_{\max}(j)} x_{ij}}{\sum_{j=1}^{m} t_j} \tag{11}$$

and

$$c_{\max}(j) = \max_i (1, c_{ij}) \tag{12}$$

The parameters of the weighted function *g(y)* and *f'(x)* are both restricted between 0 and 1 since the minimum preference value is zero. Considering $\alpha \geq 0$, we will be in a position preferring one of the terms of the combined objective function.

However, if we consider $\alpha > 1$, we will end up with assignments that have a uniformly distributed workload. Additionally, if we consider $\alpha < 1$, then assignments will be made is such a way to maximize the preference values of the proctors (Marti et al., 2000).

2.3 Solution Evaluation

We evaluate our solutions based on the weighted function $h(x) = f'(x) + \alpha g(y)$ that combines both objectives described below:

Max $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$

Min $g(y) = y$

To illustrate the issue, if a reasonable solution *M* has been dominated by another good solution *N* in having *H* weighted function, then we have $h_o(n) \geq h_o(m)$ for all objective functions and $h_o(n) \leq h_o(m)$ for a minimum of one objective function. In other words, for each proctor we initially consider all solution until one of those solutions dominates all being the one having Max *f(x)* and Min *g(y)* for any proctor *P*. So, an acceptable solution is said to be efficient if there is no other good solution that dominates it.

In case we reach a situation when there are several efficient solutions, we feed the decision-maker with all these options; thus, it would have the choice to select any solution from the pool of the founded good solutions (Marti et al., 2000).

Chapter 3

Bee Colony Algorithm for PAE Problem

3.1 Background

The Bee Colony Algorithm is an algorithm that imitates the naturally optimised behaviour of honey bees for searching for food sources in order to optimize prospect solutions (Pham, 2005). The algorithm require setting a number of parameters, namely: (*n*) the number of scout bees, (*m*) which is the number of sites selected out of *n* visited sites which is also the total number of scout bees, (*e*) which is the number of best sites out of the previously selected *m* sites, (n_{ep}) which is the number of bees recruited for the best sites *e* which was selected previously, (n_{sp}) which is the number of bees recruited for the other *(m-e)* selected sites, (n_{gh}) which is the initial size of patches that includes site and its neighbourhood and stopping criterion.

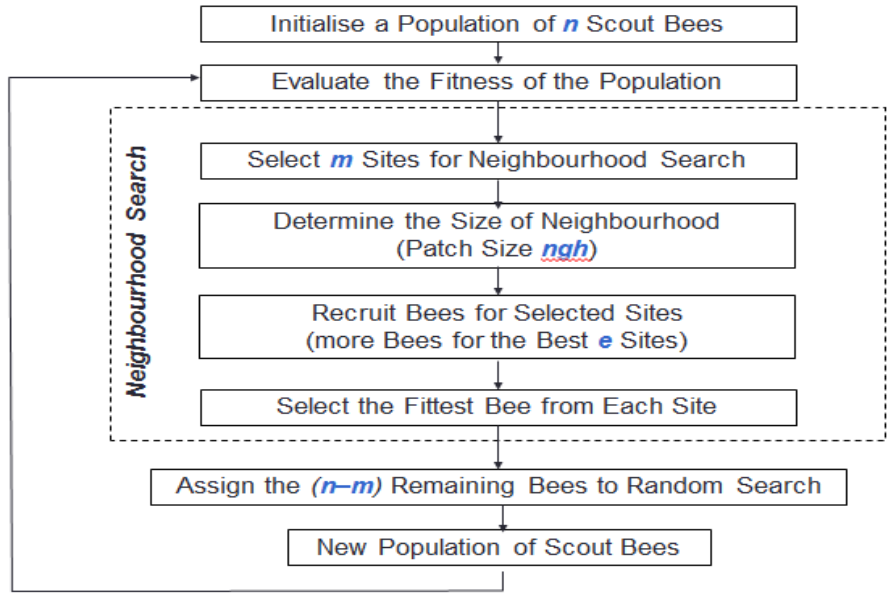


Figure 3.1 Steps of the Bee Colony Algorithm

The algorithm will start by placing *n* scout bees in the search space in a random fashion. Next, the algorithm will go through each of the sites that were visited by the scout bees and evaluate its fitness, and the bees that visited the sites that have the highest fitness are then chosen and considered as "selected bees". The sites that were visited by the "selected bees" are chosen for neighbourhood search and considered as "selected sites". Later on, my algorithm starts conducting searches in the neighbourhood of the selected sites, then assigns more bees to the search near to the best sites *e*. The bees can be chosen directly according to the fitnesses associated with the sites they are visiting. On the other hand, what I do is I use the fitness values in order to determine the probability of the bees being selected.

Since the best *e* sites represent the solutions that are more promising than the others, the searches in the neighbourhood of these *e* sites are more thorough. This is accomplished by assigning more bees to follow the *e* sites than any other sites, even among the "selected sites". Along with the act of scouting, this recruitment of more bees is a key operation of the BeeColony Algorithm. However, for each patch only the bee with the highest fitness will be selected to form the next bee population. This kind of restriction does not exist in nature. Furthermore, it is presented in my algorithm with the objective of reducing the number of points to be discovered. The remaining bees in the population will scout for new potential solutions as they are randomly assigned around the search space. The above mentioned steps are repeated until a stopping criterion is met. At the end of each iteration the colony will have two parts for its new population: agents from each selected patch and other scout bees assigned to perform random searches.

3.2 Bee Colony Algorithm Design

In our case we set the number of instructors (bees) who will try to find the fittest solution (exams to proctor) from the best e sites (pool of exam periods). The steps shown in Figure 3.2 are described below:

```
Initialize the set of n proctors with random solutions.
 Evaluate fitness of each solution based on the initial assignment
 While (we still got exams without proctors)
    //Forming new population.
 Select the best assignments so far
Evaluate their fitness and record any better solutions (override the inferior solutions)
If  parameter of  un-assigned proctors = parameters of assigned proctors
{
  Un-assigned proctors will check the obtained solutions from the assigned proctors and then
evaluate them
}

Else
{
 Check for random solutions based on the best solutions found so far
}
 Select the fittest solution of each proctor.
 Assign remaining proctors to search randomly and evaluate their fitnesses.
 End While.
```

Figure 3.2 Pseudo code of the PAE

3.2.1 Solution Representation

In our Bee Colony algorithm implementation (using Java), the basic data structure used to represent our solution is an array representing each proctor's assignment as a percentage value, noting that it should be the highest percentage obtained as our problem tends to maximization of the weighted function.

The actual solution is presented in a linked list where *I* denotes the set of *n* proctors ($i = 1, ..., n$) linked to their list of assigned exams J where *J* denotes the set of *m* exams ($j = 1, ..., m$). The initial lists of I and J are read from 2 input files in.

The element Proctor(i) represents the proctor that will be allocated to a chosen exam and period object denoted by (E,P) where P is the selected period allocated to an exam, while E corresponds to the exam list that ranges from {1,…,m}, noting that the preference of proctors and the exam periods are also read from two additional files..

The row data listed above is fed to our Bee Colony algorithm in order to start by generating an initial population and evaluating them, the details are listed in the following steps.

3.2.2 Initializing the Population

Our algorithm starts by initializing *n* scout bees where *n* represents the initial number of proctors we have in our problem. The *n* scout bees are then placed randomly in the search workspace.

3.2.3 Evaluating the fitness of the Population

The fitnesses of the sites visited by the scout bees are then evaluated as follows:

- The first scout bee is taken and trained with the data. (for example: if we get 200 proctors placed correctly without any time conflicts out of 1000 record, the bee will be given the evaluation of 20%).
- The second scout bee is taken and the same process is repeated and we may get 50%.

The processes will be repeated on the all scout bees and evaluated through the fittness function.

The evaluation of the *n* scout bees is stored in array as follow:

1	**2**	**3**	**4**	**5**	**6**	**..**	**..**	**..**	**n-1**	**n**
22%	50%	37%	60%	80%	10%	..	..	..	42%	70%

Then the array will be reordered based on the evaluation from the higher to the lower value.

3.2.4 Waggle Dance step

In searching for food sources, the bee colony has two major characteristics: waggle dance and forage (or nectar exploration). So we apply these to our PAP problem.

In the waggle dance step, a forager bee denoted as fg_i when returning to the hive after doing its nectar exploration will attempt with probability p to make a waggle dance on the dance floor with duration D = diA, where di changes with profitability rating while A denotes waggle dance measuring factor. Additionally, it will try to attempt with probability ri to detect and follow a dance which is randomly selected. It is to be noted that the probability ri is dynamic and also changes with the rating. Continuing the next steps of my algorithm we reach a step where in case a forager bee chooses to track a selected dance, it will use the previously taken 'path' by the forager bee that is performing the dance to follow its direction for flower patches. We can then consider that the selected path as 'preferred path'. Forager's bee path is a series of landmarks from a source (hive) to a destination (nectar). For the proctor assignment, the profitability rating should be related to the objective functions:

$$\text{Max } f(x) = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}$$

$$\text{Min } g(y) = y$$

Describing the foraging approach, we start by a population of one group of forager bees that is to be defined in the colony. This group of foragers regularly constructs solutions to the PAP problem. Then they move along branches from one node to another node in the disjunctive graph resulting in path construction that will be a representation of the solutions.

It is to be noted that each forager bee must visit every node once and only once in the graph, where it starts from the source or the initial node and stops at the sink or final node, and this in order to construct a complete solution. The forager bee moves along the nodes in this manner, when it is at a given node, its next move can only be the next node that is defined and present in the allowed list of nodes.

3.2.5 Selecting sites for neighborhood search

Neighborhood or local search moves from an initial solution by a sequence of neighborhood changes, which improve each time the value of the objective function until a local optimum is found.

The *m* sites will be selected based on the best evaluation to *m* scout bees from *n*. For example the number of bees having the highest evaluation are 1000, so m = 1000.

1	**2**	**3**	**4**	**5**	**6**	**..**	**..**	**..**	**m-1**	**m**
92%	87%	83%	80%	77%	72%	..	..	..	70%	68%

Then we choose the best e site (scout bee) out of the m sites randomly. Then we record *e* and m-e, for example e = 600 then m-e = 1000 – 600 = 400.

3.2.6 Determining the size of the neighborhood

A neighborhood search sites of size *ngh* are selected which will be used to update the *m* bees declared in the previous step. This is important as there might be better solutions than the original solution in the neighborhood area. *ngh* takes a percentage value, for instance ngh = 0.5.

3.2.7 Recruiting Bees for the selected sites

Recruit Bees for the selected sites and evaluate the fitness of the sites. Number of bees *n2* will be selected randomly to be sent to *e* sites (example: *n2* = 300). Then choosing *n1* bees also randomly which their number is less than *n2***,** (example: *n1* = 200) to be sent to *m-e* sites.

3.2.8 Selecting the Fittest Bee from each site

In this step we choose the best bee from each site which is the one having the highest fitness in order to form the next bee population as follows:

- The first site will be taken (for example a site from *e* sites).
- An array contains *n2* = 300 bees will be constructed, where the value of each bee is equal to the value of the original scout bee with a little modification depending on the neighborhood *ngh*.
- The data will be trained on the 300 bees and evaluated through the fitness function.

- The results will be stored in temporary array where they are order in a descending order. Then the best value will be selected which will be the first element in the array.
- This step is repeated for all *m* sites. At the end we will get the best *m* bees and we will store them at the beginning of our main array.

1	**2**	**3**	**4**	**5**	**m**	**m+1**	**..**	**..**		**n**
92%	87%	83%	80%	77%	72%	..	..	..	70%	68%

3.2.9 Initializing new population

The remaining bees in the population will be assigned randomly around the search space (values from *m+1* to *n* in the previous array). The new population becomes as follow:

1	**2**	**3**	**4**	**5**	**m**	**m+1**	**..**	**..**		**n**
92%	87%	83%	80%	77%	72%	Random Values…				

3.2.10 Stopping criterion

The loop counter is reduced and the steps two to seven are repeated until the stopping criterion is met. (For example our stopping criterion will be the ending the number of the repetitions imax where imax=1000).

At the end of each iteration, the colony will have two parts to its new population representatives from each selected patch and other scout bees assigned to conduct random searches.

Chapter 4

Other Algorithms for PAE Problem

4.1 Multi-objective Scatter Search for PAE Problem

The multi-objective solution for solving the proctor assignment problem involves two heuristic algorithms scatter search and tabu search algorithms. Tabu search is used to generate an initial set of diverse solution, so our diversification generation method will incorporate tabu search. Then we incorporate scatter search algorithm to enhance these solutions.

The procedure starts with the diversification generator in order to build a collection P of diversified solutions. The dimension of this collection (PSize) should be large and is typically 10 times greater than the reference set (RefSet). Then the first reference set is constructed using the reference set update method by taking the best b distinct and diverse solutions from P. Here we employ tabu search that seclets the variables as a basis for defining the move attributes.This is employed in order to prevent moves that will produce close results to previously held ones. Specifically we consider a move as a tabu if it produces a solution that lies much close than a specified tabu defined distance to any previously obtained/visited solution. Furthermore, an approach based on a weighted sum is built in order to take a decision on picking solutions as we need to pick the good solutions.

This is done by first taking the most attractive b_1 solutions from P which are included into the reference set and removed from P. Then, the remaining solutions in

P are used to compute the least distance measure with respect to all solutions which make up the current reference set. The solution with the greatest minimum distance gains entry to the reference set and is removed from P and the least distances measures are computed again. This procedure is iterated b_2 times knowing that $b_2=b-b_1$. The resulting reference set would be made up of b_1 high-quality solutions and b_2 diverse solutions.

These solutions are sorted with respect to the goodness of their objective function whereby the solution with the highest utility function figures in the first position. Next the look-up begins with setting the Boolean variable NewSolutions to TRUE.

Afterwards, NewSubsets are constructed using the subset generation method constructed and NewSolutions is assigned the value of FALSE. The subset generation method in its simplest form generates all pairs of reference solutions. That is, $(b^2-b)/2$ NewSubsets of solutions are obtained. Then these pairs are subjected to the solution combination method to produce new trial solutions which are then subjected to the improvement method.

Then the reference set update method is run another time to get the new RefSet by taking the best solutions found from the current RefSet and the new trial solutions. These solutions are chosen according to their quality, i.e. according to their objective function. In case the RefSet changes at this stage, then NewSolutions is set to TRUE to indicate that a solution has gained membership in the reference set. The subset that was considered for combination is deleted from the collection of subsets

or NewSubsets. Then all sub-collections constituting NewSubsets are combined and if no better solution is added to the reference set, the scatter search is terminated.

It is worthwhile noting that after the combination method is applied, the reference set is updated in a static way. This is so because when trial solutions are generated by the means of the combination method, they are placed in a solution pool. Then the b most attractive solutions found in the union of the reference set and the pool generated are chosen to construct the new reference set.

```
1:   s ← s0
2:   sBest ← s
3:   tabuList ← null
4:   while (not stoppingCondition())
5:      candidateList ← null
6:      for(sCandidate in sNeighborhood)
7:           if(not containsTabuElements(sCandidate, tabuList))
8:                candidateList ← candidateList + sCandidate
9:           end
10:     end
11:     sCandidate ← LocateBestCandidate(candidateList)
12:     if(fitness(sCandidate) > fitness(sBest))
13:          sBest ← sCandidate
14:          tabuList ← featureDifferences(sCandidate, sBest)
15:          while(size(tabuList) > maxTabuListSize)
16:               ExpireFeatures(tabuList)
17:          end
18:     end
19:  end
20:  return(sBest)
```

Figure 4.1 MOSS main method

4.2 CPLEX

The IBM ILOG CPLEX software has the ability to solve different problems listed below (www.ibm.com):

- *Integer programming problems.(Used in our paper)*
- Extremely big linear programming problems.

- What is known as quadratic programming non-convex and convex problems.
- Convex constrained quadratic problems.

An integer programming problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers. Integer programming is NP-hard. A special case, 0-1 integer linear programming, in which unknowns are binary, is an NP-Complete problem (www.mit.com).

Chapter 5

Experimental Results

We applied the PAE algorithms to real world data sets from two universities, university of Barcelona (Awad et al., 1998) and Carleton University (Marti et al., 2000). The data used is comprised of 11 subject cases described in Table .1. The first column represents the case number, the second column represents number of proctors we have in the respective case, and while column three represents the number of exams we have in the respective case. It is worthy of note that we managed to make cases fluctuate between two conditions:

- Cases where the number of proctors are more than the number of exams.
- Cases where the number of exams are more than the number of proctors.

Table 5.1: PAE subject problems

Case	# of Proctors	Exams
1	23	21
2	59	52
3	59	44
4	25	21
5	46	42
6	150	300
7	230	690
8	370	1480
9	520	2600
10	740	4440
11	1500	15000

We ran 5 algorithms on these data: Bee Colony (BC), Scatter Search (SS), Multi-Objective Scatter Search (MOSS), Partitioned Cplex, and Cplex. Each algorithm was executed 10 times on each of the data sets and the worst, average, and best results were computed. The results of these runs are shown in Tables 5.2-5.4 for SS, MOSS, and BC respectively. The entries presented in these tables are the values of the weighted function ($h(x) = f'(x) + \alpha g(y)$).

Table 5.2 below shows the 10 different runs for the scatter search algorithm over the 11 cases we have, the first column represents the case number, columns 2 till 11 represents the values of the weighted function in each run, column 12 records the worst result obtained from columns 2 till 11, column 13 records the average result among the 10 runs we have, while column 14 records the best result obtained among the 10 runs we have.

Table 5.2: 10 Different runs using Scatter Search Algorithm

Case	1	2	3	4	5	6	7	8	9	10	Worst	Average	Best
1	0.759	0.752	0.713	0.750	0.750	0.764	0.763	0.763	0.763	0.761	0.713	0.753	0.764
2	1.000	1.000	0.979	0.991	1.000	0.981	0.994	1.000	0.994	0.989	0.979	0.992	1.000
3	0.994	0.997	1.000	0.997	0.991	0.989	1.000	0.988	0.995	0.999	0.988	0.995	1.000
4	0.754	0.733	0.749	0.737	0.739	0.751	0.750	0.744	0.746	0.742	0.733	0.744	0.754
5	0.951	0.947	0.952	0.950	0.949	0.945	0.948	0.951	0.930	0.951	0.930	0.947	0.952
6	0.653	0.649	0.650	0.646	0.644	0.651	0.649	0.641	0.645	0.651	0.641	0.647	0.653
7	1.023	1.020	1.012	1.018	1.015	1.022	1.014	1.011	1.022	1.019	1.011	1.017	1.023
8	1.041	1.039	1.039	1.036	1.039	1.033	1.032	1.040	1.038	1.033	1.032	1.037	1.041
9	0.812	0.831	0.824	0.823	0.828	0.827	0.830	0.833	0.833	0.822	0.822	0.826	0.833
10	1.151	1.144	1.150	1.149	1.145	1.148	1.142	1.153	1.151	1.147	1.142	1.148	1.153
11	1.278	1.260	1.266	1.269	1.262	1.281	1.253	1.257	1.279	1.275	1.253	1.268	1.281

Table 5.3 below shows the 10 different runs for the multi-objective scatter search algorithm over the 11 cases we have, the first column represents the case number, columns 2 till 11 represents the values of the weighted function in each run, column 12 records the worst result obtained from columns 2 till 11, column 13 records the average result among the 10 runs we have, while column 14 records the best result obtained among the 10 runs we have.

Table 5.3: 10 Different runs using Multi-Objective Scatter Search Algorithm

Case	1	2	3	4	5	6	7	8	9	10	Worst	Average	Best
1	0.896	0.873	0.877	0.886	0.889	0.879	0.883	0.894	0.892	0.888	0.873	0.885	0.896
2	1.000	0.997	0.984	0.992	0.995	0.976	0.999	0.989	0.987	0.984	0.976	0.990	1.000
3	1.171	1.173	1.165	1.168	1.163	1.169	1.162	1.167	1.170	1.171	1.162	1.167	1.173
4	0.955	0.952	0.948	0.950	0.947	0.944	0.954	0.955	0.951	0.949	0.944	0.950	0.955
5	0.963	0.958	0.952	0.957	0.961	0.956	0.949	0.960	0.955	0.953	0.949	0.956	0.963
6	0.771	0.769	0.766	0.768	0.763	0.755	0.761	0.766	0.762	0.765	0.755	0.764	0.771
7	1.112	1.110	1.109	1.110	1.102	1.101	1.105	1.100	1.105	1.107	1.100	1.106	1.112
8	1.147	1.138	1.141	1.145	1.142	1.148	1.139	1.144	1.141	1.143	1.138	1.142	1.148
9	0.958	0.952	0.955	0.951	0.947	0.942	0.949	0.954	0.950	0.946	0.942	0.950	0.958
10	1.159	1.145	1.156	1.150	1.142	1.151	1.147	1.153	1.155	1.143	1.142	1.150	1.159
11	1.301	1.299	1.282	1.289	1.297	1.294	1.291	1.288	1.284	1.291	1.282	1.291	1.301

Table 5.4 below shows the 10 different runs for the bee colony algorithm over the 11 cases we have, the first column represents the case number, columns 2 till 11 represents the values of the weighted function in each run, column 12 records the worst result obtained from columns 2 till 11, column 13 records the average result among the 10 runs we have, while column 14 records the best result obtained among the 10 runs we have.

Table 5.4: 10 Different runs using Bee Colony Algorithm

Case	1	2	3	4	5	6	7	8	9	10	Worst	Average	Best
1	1.107	1.092	1.104	1.099	1.093	1.101	1.097	1.091	1.093	1.098	1.091	1.097	1.107
2	1.234	1.227	1.221	1.230	1.228	1.220	1.229	1.231	1.225	1.222	1.220	1.226	1.234
3	1.311	1.304	1.307	1.299	1.312	1.301	1.306	1.301	1.304	1.311	1.299	1.305	1.312
4	1.189	1.183	1.180	1.185	1.178	1.190	1.177	1.184	1.188	1.181	1.177	1.183	1.190
5	1.204	1.193	1.191	1.197	1.201	1.194	1.198	1.206	1.199	1.203	1.191	1.198	1.206
6	0.933	0.927	0.922	0.931	0.934	0.925	0.928	0.935	0.929	0.923	0.922	0.928	0.935
7	1.323	1.314	1.310	1.317	1.321	1.309	1.311	1.319	1.324	1.320	1.309	1.316	1.324
8	1.563	1.560	1.551	1.557	1.560	1.566	1.553	1.559	1.553	1.552	1.551	1.557	1.566
9	1.411	1.403	1.400	1.409	1.412	1.416	1.407	1.410	1.402	1.413	1.400	1.408	1.416
10	1.272	1.261	1.264	1.253	1.260	1.266	1.271	1.268	1.275	1.269	1.253	1.265	1.275
11	1.426	1.411	1.417	1.423	1.414	1.407	1.419	1.423	1.413	1.417	1.407	1.417	1.426

Table 5.5 shows the best, average, and worst results obtained from the five algorithms. We note that CPlex is deterministic and yields the same results for all of the 10 runs. Based on these results we make the following comments:

- The Bee Colony algorithm produced better results than the other four algorithms used in 8 out of the 11 cases in terms of the average values, in 9 out of 11 cases in terms of the best values and in 8 out of 11 cases in terms of the worst cases. The average parentage of improvement over the 11 cases yielded by the Bee Colony algorithm in comparison with CPLEX is 5% in terms of the average values over 10 runs.
- CPlex in its partitioned version were also able to solve all of the cases and produced better results than the scatter search and multi-objective scatter search. The degree of improvement in the results of CPlex with respect to the results of the multi-objective scatter search was around 27% in terms of the average values.
- Scatter search was able to find results for PAE but since the nature of the problem tends to be more multi-objective, as discussed in Chapter 2, we find that the multi-objective version of scatter search yielded slightly better results. The degree of improvement between both algorithms was around 9.2% on average for all of the 11 test cases in terms of average values of 10 runs.
- However CPlex in its non-partitioned version the values produced from the first 6 cases were the same the partitioned version. The quality of the results started to decrease as the input data is increasing until it crashed in the last three cases.

Table 5.5: Best, average, and worst case results of the 5 algorithms used in PAE											
Cases	**Bee Colony Algorithm**			**Multi-Objective Scatter Search**			**Scatter Search**			**CPlex**	
	Best	**Average**	**Worst**	**Best**	**Average**	**Worst**	**Best**	**Best**	**Best**	**Partitioned**	**Non Portioned**
1	**1.107**	**1.097**	**1.091**	0.896	0.885	0.873	0.764	0.764	0.764	0.953	0.953
2	1.234	1.226	1.220	1.000	0.990	0.976	1.000	1.000	1.000	**1.393**	**1.393**
3	**1.312**	**1.305**	**1.299**	1.173	1.167	1.162	1.000	1.000	1.000	1.165	1.165
4	**1.190**	**1.183**	**1.177**	0.955	0.950	0.944	0.754	0.754	0.754	0.815	0.815
5	1.206	1.198	1.191	0.963	0.956	0.949	0.952	0.952	0.952	**1.268**	**1.268**
6	**0.935**	**0.928**	**0.922**	0.771	0.764	0.755	0.653	0.653	0.653	0.901	0.901
7	**1.324**	**1.316**	**1.309**	1.112	1.106	1.100	1.023	1.023	1.023	1.297	1.280
8	**1.566**	**1.557**	**1.551**	1.148	1.142	1.138	1.041	1.041	1.041	1.542	1.423
9	**1.416**	**1.408**	**1.400**	0.958	0.950	0.942	0.833	0.833	0.833	1.308	Crashed
10	**1.275**	1.265	1.253	1.159	1.150	1.142	1.153	1.153	1.153	**1.266**	Crashed
11	**1.426**	**1.417**	**1.407**	1.301	1.291	1.282	1.281	1.281	1.281	1.395	Crashed

Chapter 6

Related Work

6.1 Methods for Proctor Assignment to Exams

As a previous work that was done to solve the proctors assignment problem two papers were published that tackles the problem using Scatter Search Heuristic Algorithm, and Genetic Algorithm combined with some problem-specific heuristics consequently.

6.1.1 Assigning Proctors to Exams with Scatter Search

(Marti et al., 2000) presents a heuristic algorithm to assign in generic terms teaching assistants, known as proctors, to monitor university students during the final exams. In order to achieve this goal, they classified the given problem as an integer program (IP) with a weighted objective combining two functions: a preference and a workload-fairness. Then they developed a scatter search approach to tackle the problem. By applying the steps of the scatter search algorithm in order to solve the PAE problem they came up with the following procedure:

Step 1– Preprocessing: In order to simplify things, we may consider that the professors will proctor to the exams that correspond to the courses that they teach and that have only a small number of students enrolled in them. In this case, the data

related to the exam and the proctor are usually updated in order to reflect these assignments.

Step 2 – Generate random population: Here the authors apply the diversification generator method of the scatter search to produce varied solutions.

Step 3 – Construct the reference set R: First, this step adds the top $r1$ solutions in population P to the reference set R. Second, it adds $r2$ solutions that were generated by using the diversification method from population P in order to construct the set R having $|R| = r1 + r2$ solutions.

Step 4 – Maintain and update the reference set R: In this step we apply the subset generation method (Glover, 1998) in order for us to be able to combine solutions from the reference set R. Then we update R by adding solutions with the objective of improving the quality of the poorest population in the set.

In order to implement the algorithm, the authors start with the diversification method which is initialized by a population of solutions through a method called a diversification generator. The authors selected to implement a generator by developing the idea of constructing solutions using a set of modified frequencies. The objective of the method is to produce a variety of solutions good-quality; Figure 6.1 shows the algorithmic steps of this method.

```
Initialization
        Solutions = 0;
        f_ij=0  for all i,j

While (Solutions < PopSize )
{
        For k=1 to 2d
        {
                Order the exams in period k by decreasing number of required TA's.
                For each exam j in period k
                {
                        Construct the list of TA's that can proctor j.
                        Order the list according to c_ij
                        Assign the first t_j TA's to proctor exam j.
                        Update TA and exam information.
                }
        }
        Add the solution to the population.
        Make Solutions = Solutions + 1;
        Update the corresponding f_ij.
}
```

Figure 6.1 Diversification Generator

The next step falls under updating and maintaining the reference set. It is to be noted that the reference set *R* is made up from some of the high quality and diverse solutions residing in the population set *P*. So we can say that a solution can only become part of the reference set when it is not dominated or if it is diverse or its distance is huge compared to the solutions that are currently residents of the reference set.

Finally we move to the solution combination step, in this step the scatter search will combine the already existing solutions in the Reference set *R in order to* produce new solutions. A main consideration in this step is that *the reference set* is not static, since it changes as new solutions are added and replacing the lower value and less fortunate old ones. This step is based on a voting scheme as shown in Figure 6.2.

For each exam *j*

{

Find the assignment *xij* with the largest vote.

Assign the exam *j* to TA *i*.

}

Figure 6.2 Voting Mechanism

If during one of the iterations a new solution became a member of the reference set, then the process is repeated having the set of non-dominated solutions in the reference set. Regarding the rest of the population they are generated using the diversification generator and later on the whole scatter search process is repeated. Finally, the scatter search ends when there are no more new solutions (Marti et al., 2000).

Finally the empirical results shows Scatter search algorithm gave lower quality values than those resulted from the Cplex. However, the good side of using the Scatter Search algorithm is that it is able produce very good quality solutions which can be later on handled to the person in charge for the final selection of the most suitable one. This kind of flexibility is suitable and acceptable when the decision is based on constraints that are subjective similar to our problem of assigning proctors to exams, where we have each of our proctors express their preference to a specific period or exams.

6.1.2 Proctor Assignment at Carleton University

(Awad et al., 1998) developed a computer program in order to replace the current manual system based on combining problem-specific heuristics under a genetic-algorithm frame.

In order to automatically create an initial assignment of a proctor, the authors tend to measure the quality the assignment step. In the normal case when there is an assignment that is satisfying all of the predefined constraints it will be placed as the top quality classification. However, an assignment of this nature is far from occurring due to the fact that some constraints actually have conflicting results. In order to solve this issue, the authors tend to correct and weigh the quality of a specific assignment by imposing penalty points for failing to comply with any one of the predefined constraints. In other words, they convert the constraints into an objective function which they will try to minimize it.

From the implementation point of view, the number of possible assignments is huge. In general we have *p* proctors, *t* time slots for examinations, and *a* possible assignments in each time slot. Looking into our worst case, we will be having (*t* 2! / (*p* 1 *a*)!) possible assignments.

The first thing to do in planning the GA is to look for a viable way of programming a solution. The solution representation is done through a matrix with exam time slots as columns and proctors as rows. Every cell includes a code representing the building where a specific exam will take place.

The second thing is to produce an initial population of different solutions randomly. This is done by taking into consideration some rule in order to reduce the number of poor assignments:

1- Only subscribe proctors to the previously agreed time slots of their choice.
2- Only subscribe proctors to the previously agreed rooms of their choice.
3- All the slots that are already assigned are preserved.

The first rule is accomplished by creating a write-protect the elements in the matrix that corresponds to the unavailable periods of the proctor. The same is done for the elements that falls under the third rule.

Then they move to use the previously described quality measure for each generated solution in the population in order to compute its fitness function. Basic crossover and mutation is then applied to the pool of schedules for the purpose of creating the next generation.

The final solution that is based on the basic GA provided somehow fairly good assignments but it violated the requirements for reserving a room in an unacceptable way. This is why a heuristic approach is used to help in this matter.
Moving on to the heuristics used we have them as follow:

1- The first heuristic: It is used to generate a random and an initial list of proctors. The list is used for each time slot, taking into consideration that the

authors randomly choose a new starting point at the beginning of every time slot, while the list is being treated as a circular queue.

2- The second heuristic: In this section the proctors are ordered from the most to the least constrained based on their lists of possible assignments (to be noted that proctors who have a short lists they are considered to be the most constrained). As a result we will be reducing the number of the empty time slots, of course to avoid hiring more proctors.

3- The third heuristic: This kind of variation gives greater priority to bigger number of senior and flexible proctors. Now ordering of the proctors will be from the most to the least seniority. Afterwards, proctors who have the same seniority are sub-ordered according to their list of available time slots from the longest to shortest one. (Awad et al., 1998). Figure 6.3 shows a common algorithm for the three heuristics.

```
For each time slot:
  Order the list of buildings from largest number of proctors needed to smallest number of proctors
    needed.
  For each building needing proctors:
    n = number of proctors needed in this building during this time slot.
    Number Assigned = 0
    For each proctor:
      If (proctor is available for work during the time slot) and
      (time slot is not masked for this proctor) and
      (proctor not already scheduled during this time slot) and
      (building is on the list of possible assignments for the proctor) then:
          Assign proctor to this building during this time slot.
          Increment Number Assigned.
          If Number Assigned = n, then exit the proctor loop.
      End if.
    Next proctor.
  Next building.
Next time slot.
```

Figure 6.3: This Stage 2 algorithm is common to all of the heuristics, which differ only in the ordering of the list of proctors.

In conclusion this approach gave a good and successful solution to the problems afflicting the current manual system. The used approach which is based on combining the basic version of genetic algorithm along with problem-specific heuristics resulted in an effective and good solution. It is to be noted that a better solution might result when using an advanced GA or different algorithm. The authors see that the trade-offs in the complexity of implementation and keeping up with the maintenance are not worth any further effort in development and maintenance.

6.2 Methods for problems similar to PAE

In this section we list some of the approaches used to tackle assignment problems similar to the proctor assignment using different heuristic algorithms.

6.2.1 Structured Neighborhood TS for Assigning Judges to Competitions

A meta-heuristic approach is introduced in (Ferland & Lamghari, 2007) and that it features three different stages for assigning the judges for a special competition. The authors selected such an approach due to the intricacy of the mathematical formulation standing behind the rules that the assignment process of judges follow. In the beginning, the two different tabu search methods in stages one and two respectively are combined with a diversification strategy.

The mathematical model can be summarized as follows:

$$\sum_{j=1}^{M}(1-a_{ij})x_{ij}=0 \qquad i=1,\ldots,N \qquad (3)$$

$$\sum_{i=1}^{N}l_i x_{ij}\geq 1 \qquad j=1,\ldots,M \qquad (4)$$

$$\sum_{i=1}^{N}x_{ij}=3y_3^j+5y_5^j \qquad j=1,\ldots,M \qquad (5)$$

$$y_3^j+y_5^j=1 \qquad j=1,\ldots,M \qquad (6)$$

$$x_{ij}=0 \text{ or } 1 \qquad \begin{matrix} i=1,\ldots,N \\ j=1,\ldots,M \end{matrix} \qquad (7)$$

$$y_3^j, y_5^j=0 \text{ or } 1 \qquad j=1,\ldots,M \qquad (8)$$

$$Min \sum_{j=1}^{M}\sum_{k=1}^{K}\max\left\{\left(\sum_{i=1}^{N}e_{ik}x_{ij}\right)-1,0\right\}-5M\sum_{j=1}^{M}y_5^j \qquad (1)$$

$$Subject\ to \sum_{j=1}^{M}x_{ij}\leq 1 \qquad i=1,\ldots,N \qquad (2)$$

In this paper the authors considered two alternative processes to produce a preliminary solution. In the first process denoted as ‘Random’, the authors extracted an initial solution, while in the second process they have used the HLA-HOA heuristic method in a particular strategy in order to assign a lead judge to each individual competition. First, the method starts by assigning lead judges who have the most common expertise k’ and who are allowed to contribute in fewer individual competitions.

For stage one of assignment, and with the objective of reducing the number of individual competitions with 1 or 3 judges assigned, a structured tabu search is

done by reassigning pairs of judges. As a result, the algorithm will reassign a pair of judges (i,r) from its current assignment to the given competition j to another different individual competition I, and therefore it will be generating the neighborhood of a feasible solution x. The algorithm will use the solution that was generated in the first stage as the initial point for the tabu search of the second stage . The objective in this stage will be to reach the point where the fields of expertise of the judges assigned to each competition are much more diversified. Hence, the neighborhood of a good and appropriate solution x is generated by exchanging two judges r and i currently assigned to different individual competitions I and j respectively.

The numerical results are included to as a proof of the capability of approach described above to produce very good and efficient solutions.

6.2.2 Using the Bees Algorithm to Assign Terminals to Concentrators

The main objective of this problem is to assign terminals to concentrators from their given sets. (Bernardino & Bernardino & Sánchez-Pérez & Gómez-Pulido & Vega-Rodríguez, 2010) proposed a way to perform the assignment job of terminals to concentrators using the Bee Colony Algorithm. The proposed algorithm implements a sort of neighborhood search and uses a local search method for its quest in locating the global minimum.

As mentioned earlier, the suggested solution is a join of two algorithms. First, it uses the elementary form of the Bee Colony algorithm that is based on the Bee Colony Algorithm proposed by Pham. Hence it applies a neighborhood search to discover around the selected sites. Second, it incorporates a LS algorithm proposed

by Bernardino in order to improve the performance of the first algorithm. The LS algorithm will enhance the quality of the individual solutions included in the population generated by the Bee Colony algorithm.

$$fitness = 0{,}9 * \sum_{c=1}^{M} bal_c + \quad (1)$$

$$0{,}1 * \sum_{t=1}^{N} dist_{t,c(t)} + \quad (2)$$

$$Penalization \quad (3)$$

c(t) = concentrator of terminal t
t = terminal
c = concentrator
M = number of concentrators
N = number of terminals

$$bal_c = \begin{cases} 10 & if\left(total_c = round\left(\frac{N}{M}\right)+1\right) \\ 20*abs\left(round\left(\frac{N}{M}\right)+1-total_c\right) & \end{cases} \qquad total_c = \sum_{t=1}^{N} \begin{cases} 1 & if(c(t)=c) \\ 0 & \end{cases}$$

$$dist_{t,c(t)} = \sqrt{(CP[c(t)].x - CT[t].x)^2 + (CP[c(t)].y - CT[t].y)^2} \qquad Penalization = \begin{cases} 0 & if(Feasible) \\ 500 & \end{cases}$$

Figure 6.4 Fitness Function of the problem

Moving on to the LS algorithm, it works on applying a partial neighborhood inspection. Then the algorithm continues by generating a neighbor by exchanging two terminals between two concentrators c1 and c2 (this is chosen randomly). The algorithm seeks somehow better solution in the initial set of neighbors. In case we reached an improvement in the actual solution from the previously selected better neighbor, then the LS algorithm changes the actual solution by replacing it with the better neighbor. In case there is no improvement, the algorithm generates a new set of neighbors.

Experimental results showed that the Bee Colony Algorithm that this paper is porposing is considered to be a good and competitive algorithm in creating satisfactory results measured in terms of both the quality of the proposed solution as well as the execution time.

6.2.3 A Tabu search algorithm for assigning teachers to courses

(Parrefeno & Alvarez-Valdes & Tamarit, 2002) tackle the problem of assigning teachers to courses or sections in a secondary school. The problem appears when the school begins, at that time the school will be having task to build a timetable while the teaching assignments are not yet fixed. The authors proposed and developed a tabu search algorithm to tackle these types of problems. It is to be noted that the parameters to be used in the algorithm are a result of estimation of several regression techniques.

The assignment problem of teachers can be split into two parts follows:

- Simple courses assignment
- Teacher's assignment to a block courses.

First, the authors assign the block courses and at later stage come the assignment of the simple courses. During the assignment process of teachers to a block of courses the authors try to select a group of teachers who have their sets of non-available time slots are as much similar as possible, and this is due to the idea that the set of the already reserved periods for the block will usually include the union of those sets. Moving to the second step, as mentioned earlier they assign the simple courses. After that they define what is called a "frequency vector" for every class. The idea will be that the vector whose x^{th} element shows the number of time slots or periods, will be having x available teachers. The size of the vector that corresponds to each class is simply the number of the corresponding courses.
The basic ingredients of the suggested algorithm are as follows:

- *X - The solution space:*

  It corresponds to the set of assignments that satisfy all the predefined constraints of the problem.

- *f - The objective function*:

  The objective of this problem corresponds to minimizing the sum of *f(c)* over the set of classes c where (c ϵ C), in which, for every class c having a frequency vector *V*:

  f(c) 106v[112+105v[212+..+10v[512+v[612+v[712/2+..+v[n]2/(n-5)

  The coefficients, 10 6, 105,... 10, 1, 0.5, 0.33,..., correspond to the significance that is specified for the first components of the vector, which should be having a very low value or zero. The values of the vector, *v[i]*, are squared for the reason of looking for a balance among classes. As an example, our preference would be tending towards a solution having two classes, i and j, with *v[i]* = 1 and *v[j]* = 1 to another with *v[i]* = 2 and *v[j]* = 0.

- *The neighbourhood N(s):*

  Any solution *s'* is considered to be a neighbour of another solution *s*, if *s'* can be generated from *s*. The generation process can be defined as follows: An assumption is made considering that inside the solution *s*, a teacher *P* is assigned to a class *c* also another teacher *p'* is assigned to another class *c'*. The solution *s'* can be generated by exchanging the two assignments

of *P* to *c'* and of *p'* to *c*. Here there may be three possible exchange moves as follows:

- Changes among teachers assigned to the set of simple courses.
- Changes between a teacher assigned to a simple course and another teacher assigned to a block.
- Changes among teachers assigned to blocks.

- *The tabu list:*

  The tabu list constitutes of the original assignments of teachers to classes. Before the move is done and in case we have a solution *s* having a teacher *P* assigned to class *c,* also if we have another teacher *p'* that is assigned to another class *c'*, the list will then include two pairs *(P, c)* and *(p', c')*. Regarding the move, we consider it a tabu if the resulting assignment from includes any pair from the tabu list.

Finally the computational results show that the developed procedure have the ability to produce somehow similar or in some other cases better results than the experts assignments. Consequently, the developed concept can be blended with another timetabling program in order to tackle the whole teacher assignment problem for every academic year.

6.2.4 Solving the Teacher Assignment Problem by Two Meta-heuristics

Talking about the timetabling problem it is composed of assigning of teachers to their corresponding courses as well as to their corresponding course sections as

described in this paper. The authors here try to frame the problem as a mathematical programming model. To solve this problem two different algorithms have emerged, the first one is based on simulated annealing, while the other one is based tabu search.

The suggested algorithms consist of two stages. The first stage comprises of allotting the teachers to their corresponding courses as well as figuring out the number of courses that are to be assigned for all teacher. The second stage comprises of allotting the teachers to the corresponding sections for one main goal which is balancing the teachers' load. Evaluating the performance of the two algorithms is done using two sets of live data and some problem instances that are randomly generated. The computational results revealed that tabu search outperforms the simulated annealing and the rest of the different approaches done in all previous work. Regarding the live data sets, the results show that both of the used algorithms are able to produce better solutions when compared to manual assignment.

As mentioned earlier TAE problem is expressed as a mathematical programming model. As described in that paper, the proposed mathematical model has a non-linear objective function as well as some constraints. The idea of solving this model in order to reach an optimal solution is neither easy nor straightforward, especially when we have huge amount of data to be processed (Tamarit et al, 2002).

Two phases are presented to tackle this problem; Phase 1 has one main goal which is to find an acceptable solution out of the pool of the already obtained good solutions. Its main motivation is regarded as follows; the variation of the teaching load can be minimized if and only if the total number of courses *R* taught by every

teacher is limited to a certain value. This states that the difference in the number of courses taught between the teachers is small. This consideration would pave for us the road to allot teachers to their corresponding course sections in the coming phase in order to get a minimal variation value. We are stating here that Phase 1 focuses on the allotting all of the teachers to their corresponding courses based on their capacities. Moving on to phase 2, the main goal here is to find the maximum number of courses that could be allotted to every teacher taking into consideration that teachers are unable to teach more than *R* courses, where *R* corresponds to the maximum number of courses that could be taught by every teacher. The other main goal in Phase 2 is characterized by minimizing the total variance of the weigthed load that corresponds to the set of available teachers.

Experiments were performed in order to evaluate the behaviour and the obtained results from the two used algorithms. Two sets of live data were the main ingredient of the experiment as well as some data sets that are randomly generated. Regarding the live data sets, the obtained results show that both of the used algorithms have the ability to generate enhanced solutions as compared to previous work done as well as manual allocation techniques.

6.2.5 Solving the Class Responsibility Assignment Problem Using Meta-heuristic Approach

A new assignment problem is tackled in this paper summarized as assigning responsibilities to classes. This problem is considered to be among the first and the

most disputably list of important steps when designing software that is purely object oriented. This stage relies on the decision as well as the experience of human.

In this paper the main goal of the authors is to automate class responsibility assignment through the usage of meta-heuristic algorithms. What the authors did is that they used four heuristical algorithms (Particle Swarm Optimization, Genetic Algorithm, Hill Climbing, and Simulated Annealing), along with class combination and unity metrics. Eventually, as per the obtained results generated from the four implemented algorithms, deductions on the behaviour of the algorithms and their improvements are outlined.

For the reason of making the problem more fit for application of search-based optimisation algorithms, two prerequisites are emerged:

1. The way the problem will be denoted and then encoded.
2. Fitness function definition.

The authors have chosen the easiest and most observable approach to signify the nominee solution. Regarding every responsibility existing in the system it will be represented in the chromosome by one gene. Defining the chromosomes in this problem, they correspond to integer value arrays, for which it is to be noted that the location of the gene in the chromosome indicates the responsibility of the system while the integer value that corresponds to the defined genes signifies the responsibility of a specified class. This kind of formation certifies that no classes are left empty and most importantly every responsibility is allotted to one class only.

Regarding the fitness function what the authors did is that they have chosen multi-objective optimization, normalizing and combining three various coupling along with cohesion measurements all of which are joined as one single grouped fitness function.

Moving on to working with the different cohesion actions and coupling approaches the proposed tactic is based on them, on top of that they also present measurement for assignment of abstractly somehow associated responsibilities. Implementation wise three procedures are used modularization quality, positive cohesion of methods and abstract data cohesion. For the fitness function it was designed as a linear mixture of the above measures. Explaining the *Modularization quality* denoted as (MQ) it has been introduced for one reason which is clustering the problem. Its ingredients are both of coupling and cohesion measures and they are named *inter-connectivity* corresponds to the coupling procedure while the *intra-connectivity* term corresponds to the cohesion procedure. For any class *c* presented in the problem its intra-connectivity is defined as showed below:

$$A(c) = \frac{\sum_{r \in R(c)} \sum_{s \in R(c)} uc(r,s)}{uc_{average} |R(c)|^2}$$

While for the two classes *c*1 and *c*2 their inter-connectivity is defined below:

$$E(c_1, c_2) = \frac{\sum_{r \in R(c_1)} \sum_{s \in R(c_2)} uc(r, s)}{2uc_{average} |R(c_1)| |R(c_2)|}$$

The MQ here is presented as the difference between the intra-connectivity value which is scaled averaged corresponding to all classes and the on corresponding to the all of the pairs of classes.

The designed solution revealed that some of the used meta-heuristic are able of producing designs that is considered to be acceptable when comparing to the considerations of human judgments.

6.3 Applications of Bee Colony Algorithm

In this part of my literature review I will try to introduce some of the application of the bee colony algorithm that has been previously introduced in different papers.

6.3.1 A Bee Colony Optimization Algorithm to Job Shop Scheduling

Starting by describing the nature of the problem, Job shop scheduling is considered to be an important chore for the manufacturing industry when speaking of enhancing the utilization of the existing machines as well as minimizing the cycle time. Though, the scheduling problem here is considered to be an NP-hard problem

having no easy solution. This paper introduces a solution for solving this problem by using the Bee Colony algorithm.

The scheduling problem of the Job shop has a main concern which is finding a consecutive distribution of competing resources that is able to optimize a certain objective function (Jain et al, 1999).

A disjunctive graph is considered to be a good way of representing the job shop scheduling problem. In this representation each operation will be designated as a node. Additionally, we will be having two more nodes, named as source node that corresponds to the initial operation and a sink node corresponds to the final operation. The processing time of any corresponding operation is denoted by the positive weight of every node.

Indulging more in the problem, we picture the source as a connected node to the first operation of every scheduled job while we picture the last operation of every scheduled job as a connected node to the sink. We also have a group of directed conjunctive arcs for the purpose of representing the higher level constraints of the jobs we have. On the other hand, we also have a group of disjunctive arcs for the purpose of representing the capacity constraints in order to guarantee that no two operations treated by the same machine are performed simultaneously.

In order to obtain the solutions to any job shop scheduling problem the algorithm directs the disjunctive arcs of the disjunctive graph in accordance with the permutations generated by the machine. In other words, what the algorithm tries to

do is to make bidirectional arcs transformed into some sort of unidirectional arcs. Figure 6.5 illustrates a good solution representation in terms of graphs for the job scheduling problem.

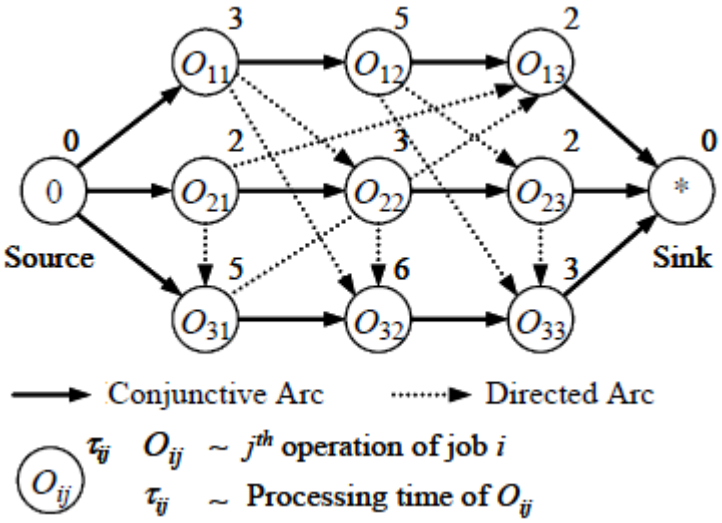


Figure 6.5: Feasible Solution for the JS problem

Implementation wise we have two approaches: operation centric approach and machine centric approach. Regarding the operation approach, a list of qualified operations is maintained during the process of job scheduling.

The operations that we have in the list are to be compared to the most recently scheduled ones that are on the same machine and this is in order to categorize if the 'edge' between the two operations exists in the desired path. If the operation with edge is found in the path, it will be assigned a higher rating *ij* ρ. All scheduled operations will be removed from the list while the following operations will be merged into the list.

Moving on to the machine centric approach, in order for the authors to complete operation scheduling a technique based on discrete-event simulation is used. The list of events is sorted in an increase manner and it is maintained during the whole process. The process starts at time 0.0 were the events are inserted in the list. Later on they are picked and processed according to their scheduled time. In case of tie, a random picking procedure is executed.

The results showed that tabu search algorithm beats other two algorithms used. However, since the bee algorithm used is the basic one there are a lot of room for enhancements which might lead to a better performance especially when using the artificial version.

6.3.2 ABC Algorithm and Its Application to Generalized Assignment Problem

The Generalized Assignment Problem (GAP) focuses on allocating a group of chores to a set of proxies taking into consideration the minimum cost. Every agent signifies one resource having limitation in its capacity. Assigning tasks can be a tough step where we have to assign for each agent a task in a one to one relationship, taking into consideration that the task requires a specific amount of the agent's resource.

GAP has many application domains, for example, location problems, CC networks, vehicle routing, and many more. In this paper the main trend is to solve the GAP which are considered an NP-Hard problems using the Artificial Bee Colony

algorithm noting that the GAP can be expressed as an Integer problem as described below:

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}$$

subject to

$$\sum_{i=1}^{n} a_{ij}x_{ij} \le b_j \qquad \forall j, \quad 1 \le j \le m$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i, \quad 1 \le i \le n$$

$$x_{ij} \in \{0,1\} \qquad 1 \le i \le n \quad \forall i, \quad 1 \le j \le m \quad \forall j$$

The main steps of the suggested ABC algorithm are presented in Figure 6.6.

1. *Initialize parameters*
2. *Construct initial Employed Bee Colony solutions by using Greedy Randomized Adaptive Search Heuristic (GRAH)*
3. *Evaluate fitness value for each bee*
4. *I=0*
5. *Repeat*
6. *N=0*
7. *repeat*
   a. *Apply Shift neighbourhood*
   b. *Apply DoubleShift neighbourhood*
   c. *Calculate probabilities related to fitness values*
   d. *Assign Onlooker Bees to Employed Bees according to probabilities*
   e. *For all Onlooker Bees*
      i. *Ejection -Chain Neighbourhood*
   f. *Find best Onlooker, replace with respective Employed Bee iffit(Best Onlooker)<fit(Employed)*
   g. *Find best Feasible Onlooker, replace with Best solution, if fit (BestFeas Onlooker)<fit(Best)*
   h. *N=N+1*
8. *Until (N=Employed Bee)*
9. *I=I+1*
10. *Until (I=MaxIteration)*

Figure 6.6 ABC Algorithm for GAP

Regarding the bee algorithm, the preliminary version of it is modeled using the GRAH algorithm (Lourenço et al, 2001). The greedy heuristic builds a solution as follows:

• A new task is selected at every step.

• The task from step one is assigned an agent.

• The above steps are repeated until all agents have tasks.

In GRAH the means of choosing is totally based on the probability method. This method is updated at every iteration through the usage of good solutions as shown in Figure 6.7.

1. *Let* $S_j = \emptyset \quad \forall j = 1,\ldots,m$ *(S_j is the set of task assigned to agent j)*
2. *Construct a list of agents for each task, L_i, initially Li={1,…,m} ∀i.*
3. *Consider any order of the tasks, i=1.*
4. *While (not all tasks have been assigned) repeat*
   - 4.1 *Choose randomly an agent j^* from L_i following the probability function that depends on the resource of agent j and the resource need by task i:*

$$p_{ij} = \frac{a/b_{ij}}{\sum_{l \in L_i} a_l/b_{il}}, \quad j \in L_i$$

   The agent with minimal cost has greater probability to be chosen.
   - 4.2 *Assign task i to agent $j^* : S_{j^*} = S_{j^*} \cup \{i\}$. Let i=i+1 and if $\sum_{i \in S_{j^*}} b_{ij^*} > a_{j^*}$ remove j^* from any list. Repeat step 4 (Note that the capacity constraint can be violated).*
5. *Let* $\sigma(i) = j$ *if* $i \in S_j$

Figure 6.7 GRAH Algorithm

All the preceding works conclude that the algorithms inspired by the Bee Colony have a very encouraging potential for forming and solving very complex optimization problems. However, it has been equally noted that the potential of the bee algorithms if fully reached only when too many steps introduced for enhancement. Here the authors attempted to solve the GAP problems which they are known as NP-Hard ones by using an ABC algorithm. They found that small and medium sized GAP problems are tackled easily by the ABC algorithm. Moreover, it

was able to easily final all the optimal solutions compared to the other 12 algorithms that did not find better solutions for most of the issues.

6.3.3 Artificial Bee Colony Algorithm for Vehicle Routing Optimization Problem

This paper takes the Bee Colony Optimization algorithm and tries to solve the problem of the travelling salesman. The bee behavior strategy is found to be very good in terms of quality and execution time when applied to path planning problems.

The success of the solutions or paths in our case has been assessed with the parameters like the tour length, or the bee traveling using the ABC algorithm. The authors in this article optimize the TSP-VR problem through the usage of the nearest neighbor function.

The main structure of the ABC algorithm is as follows:

1- Bee Initialization Phase
2- Set the Loop
3- Employed Bee Phase
4- Onlooker Bee Phase
5- Scout Bee Phase

Memorize the best solution found so far until the loop is terminated (Bhagade et al, 2012).

$$P_i = \frac{F(\theta_i)}{\sum_{k=1}^{s} F(\theta_k)}$$

$$x_{ij}(t+1) = \bar{\theta}_{ij}(t) + \emptyset(\theta_{ij}(t) - \theta_{kj}(t)$$

The vital control parameters in the ABC algorithm are, the number of food sources that is equal to the number of onlooker bees, the working to onlooker bee rate, the value of the limit (L) and the number of cycles or the number of iterations (MCN) that are required to terminate the program (Bhagade et al, 2012).

Moving to the implementation phase, in the initial step we set the control parameters, for instance (the size of the colony size, the number of iterations, and the rate of the working bee to onlooker bee. Moving on to the next phase, the route map is given as an input to the vehicle/bee along with the number of locations it should visit them. Then we obtain a reference path through the use of the nearest neighbor function. Then in order for the bee to start we assign a random node, then the probability is computed and the bees will work and pull the next node in order for them to find the path, noting that they will memorize the best solution.

In the final step the bees become scout bees and we update the number of working bees. We stop the loop when the numbers of iterations are completed and the best result is attained. Again the scout bees start their search for a new path.

Finally the authors developed a simulation framework in order for them to obtain performance assessments with the other method. The simulation showed that

the optimal distance attained by ABC Algorithm is way smaller as well as it is error free compared to the on without ABC. Therefore, they concluded that the ABC Algorithm is capable of solving the Travelling Salesman Problem in a very good way as it is considered to be highly flexible when compared to other heuristics.

6.3.4 Artificial bee colony search algorithm for examination timetabling problems

This paper works on solving the examination timetabling problems that are considered to be NP-Hard problems using the ABC algorithm. The authors here deal with allocating exams to a number of time slots bearing in mind that they are limited in number and that there is a set of constraints to be satisfied.

In this paper, the authors separate the problem into two chunks:

1- Problem 1: The problem has been previously proposed by Carter et al. in 1996, at that time the authors had around 13 benchmark datasets taken from different educational establishments, it is known as a non-capacitated examination timetabling problem in which a room capacity is not taken into consideration.

2- Problem 2: The authors take datasets from the International timetabling competition (ITC, 2007), these datasets include three tracks. The authors only consider the first track that signifies an exam timetabling design which has a number of live worldwide constraints.

The improvement algorithm presented in this paper is shown in Figure 6.8. We start by the constructive phase where the algorithm starts with practical preliminary solutions that are generated by heuristic algorithm. As a next step the employed bees starts working on random solutions and apply a neighborhood structure which is also random on all the initial solutions. Then the solutions are ordered based on a penalty method employing some cost, which later the probability for all the solutions is generated.

```
Initialisation:
Initialize the initial population and Evaluate fitness;
Calculate the initial cost function value, f(Sol);
Set best solution, Solbest ← Sol;
Set maximum number of iteration, NumOfIte;
Set the population size;
//where population size = OnlookerBee = EmployeedBee;
iteration ← 0;
Improvement:
do while (iteration < NumOfIte)
   for i=1: EmployeedBee
Select a random solution and apply random
    neighborhood structure;
 Sort the solutions in ascending order based on the
       Penalty cost;
    Determine the probability for each solution, based
       on the following formula :
            p_i = ( Σ (1/fit_i))^-1 / fit_i
   end for

   for i=1: OnlookerBee
     Sol* ← select the solution who has the higher
        probability;
     Sol** ← Apply a random Nbs on Sol*;
     if (Sol** < Solbest)
       Solbest=Sol**;
     end if
   end for
   Scoutbee determines the abandoned food source
     and replace it with the new food source.
   iteration++

  end do
```

Figure 6.8 ABC algorithm pseudo-code

Noting that the onlooker bees only subject their work on the solutions having the highest probability, where they attempt to enhance through the application of random neighborhood structure for one main reason which is reducing the soft constraints violation. The other bees which are the scouts locate the bad food sources and replace them with better ones.

In conclusion, the obtained results are comparable and even better if compared to the previous work done. So the ABC algorithm is considered better and also it has more room for improvement.

6.3.5 Parallelization of the Artificial Bee Colony Algorithm

This paper brings on some modifications to the ABC algorithm with a flavour of parallelization. The parallel approach here allows for both more speedy execution due to the multicore processors we will be having that have the ability to utilize better the algorithm as well as it allows for better performance for another reason which are the some changes that are introduced. Separate threads are allocated to separate bees, as well as a proposition of different types of communications among them. This kind of communication among the bees/swarms gives a space for additional alteration of the misuse and exploration ratio.

Due to the fact that the ABC algorithm has thousands of cycles, which makes their creation and synchronization very slow, the authors resorted to some tweak in their implementation from ABC than using the original version, what they did is listed below:

1- First they did some independent parallel executions.
2- Using multiple bees/swarms to get one top solution.
3- Using multiple bees/swarms to get a list of best solutions from all of the bees.

$$f_1(x) = \sum_{i=1}^{n} X_i^2$$

$$f_2(x) = \sum_{i=1}^{n} \frac{x_i}{4000} - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1$$

$$f_3(x) = 10n + \sum_{i=1}^{n} (X_i^2 - 10\cos(2\pi X_i))$$

$$f_4(x) = \sum_{i=1}^{n-1}[100(x_i^2 - X_{i+1})^2 + (X_i - 1)^2]$$

In the first step, the authors prefer to run a kind of a population based heuristics multiple times, due to the fact that they do not give precise result but instead they produce a kind of approximation as an end result. So it will be good to have all the iterations run concurrently to save time (this is the parallel approach), taking into consideration that in this step there are no communication at all among the existing threads.

Regarding the last two steps the authors have based their algorithmic tweak on using multiple swarms' strategies. The mere reason for this is the ability to have more than one bee/swarm working on the same search space. The swarms here are able to communicate their so far obtained results. Each communication is followed by formation of a new solution matrix which is based on the best solution generated so far. Using multiple swarm technique can be very helpful in finding better solutions faster and therefore narrowing the search space.

Detailing the second step, we collect the top solution so far at the end of a definite number of cycles from all the single bees/swarms. After that we take the best solution among the top one already selected and then distribute it among all the swarms in order for them to replace one of their solutions in their current solution matrix.

The last step defined above; take another route in terms of working out a solution. Here all of the bees/swarms drop their current solution matrix and replace it with another one that is created after a defined number of cycles. The contents of the new solution matrix represent that top solution so far from every swarm.

Finally this approach have proven that the usage of a variety of approaches in employing parallelism can reach results that are considered to some extend better than the original algorithm. Also the independent parallel runs approach has shown significance speed compared to the original method that uses a sequence of serial runs.

Chapter 7

Conclusion and Future Work

We have proposed a Bee Colony algorithm to produce good suboptimal assignments of proctor to exams subject to the conditions of maximizing the assignment of the appropriate number of proctors to exams as well as preserving the fairness of the workload given to proctors. In order to validate the quality of the results of our algorithm we have used real university data from two universities. Our experimental results show that the Bee Colony algorithm outperformed the scatter search and multi-objective scatter search algorithms a well as the CPLEX. In particular it produced an average improvement of 5% in comparison with the second best algorithm partition CPLEX.

Future work can focus on improving the bee colony algorithm especially in terms of processing time by using a parallel version of the algorithm.

References

Abdullah, S., & Alzaqebah, M. (2011). Artificial bee colony search algorithm for examination timetabling problems. *International Journal of Physical Sciences, 17,* 4264-4272.

Alvarez-Valdfis**,** R., Parreno, F., & Tamarit, J. (2002). A tabu search algorithm for assigning teachers to courses. *Journal of the Spanish Society of Statistics and Operations Research, 10,* 239-259.

Awad, R. & J. Chinneck (1998). Proctor assignment at Carleton University. *Interfaces, 28*(2), 58-71.

Baykasoglu, A.,Ozbakir, L., & Tapkan, P. (2007). Artificial bee colony algorithm and its application to generalized assignment problem. In F. Chan, & M. Tiwari (Eds.), *Swarm intelligence, focus on ant and particle swarm optimization* (pp. 113-144). Croatia: I-Tech Education and Publishing.

Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Gómez-Pulido, J., & Vega-Rodríguez, M. (2010). *Using the bees algorithm to assign terminals to concentrators*. Paper published at the 23rd international conference on Industrial engineering and other applications of applied intelligent system, Berlin, Germany.

Bhagade, A., & Puranik, P. (2012). Artificial bee colony (ABC) algorithm for vehicle routing optimization problem. *International Journal of Soft Computing and Engineering, 2*(2), 329-333.

Chong, C. (2006). A bee colony optimization algorithm to job shop scheduling. *Proceedings of Winter Simulation Conference.* Doi: 10.1109/WSC.2006.322980

Gunawan, A., & Ng, K. (2011). Solving the teacher assignment problem by two metaheuristics. *International Journal of Information and Management Sciences, 22*(2), 73-86.

Glavas, G., & Fertalj, K. (2011). Solving the class responsibility assignment problem using meta-heuristic approach. *Journal of Computing and Information Technology, 19*(4), 275-283.

Lourenco, H.R. & Serra, D. (1998, June). *Adaptive approach heuristics for the generalized assignment problem (Economics working paper, 288).* Retrieved from Social Science Research Network website: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=98650

Lamghari, A. (2007). Structured neighborhood tabu search for assigning judges to competitions. *Proceedings of Computational Intelligence in Scheduling.* Doi: 10.1109/SCIS.2007.367696

Marti, R., Lourenco, H. & Laguna, M. (2000). Assigning proctors to exams with scatter search. *Journal of Computer Science Interfaces, 12*(37), 215-227.

Narasimhan, H. (2009). Parallel artificial bee colony (PABC) algorithm. *Proceedings of Nature & Biologically Inspired Computing.* Doi: 10.1109/NABIC.2009.5393726

Pham, D.T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., & Zaidi, M. *The bees algorithm* (*Technical note).* Retrieved from Max Planck Institut Informatik website: https://svn-d1.mpi-inf.mpg.de/AG1/MultiCoreLab/papers/Pham - The Bee Algorithm.pdf

Pham, D.T., Castellani, M., & Ghanbarzadeh, A. (2007). A preliminary design using the bees algorithm. *Proceedings of the Eighth International Conference on Laser Metrology.* Retrieved from slideshare website: http://www.slideshare.net/amritkaur77920/bees-algorithm

Trick, M. (2002). Integer programming. In R. Fourer, D. M. Gay, & B. Kernighan (Eds.), *AMPL: A modeling language for mathematical programming* (pp. 272-319). USA: Duxbury Press.

Xinyi, L., Zunchao , L., & Liqiang, L. (2012). An artificial bee colony algorithm for multi-objective optimisation. *Proceedings of Intelligent System Design and Engineering Application.* Doi: 10.1109/ISdea.2012.711

Yamashita, D.S., Armentano, V.A., & Laguna, M. (2006). Scatter search for project scheduling with resource availability cost. *European Journal of Operational Research, 169*(2), 623-637.

Appendix A

PAE Problem JAVA Design

A.1 Class Diagram

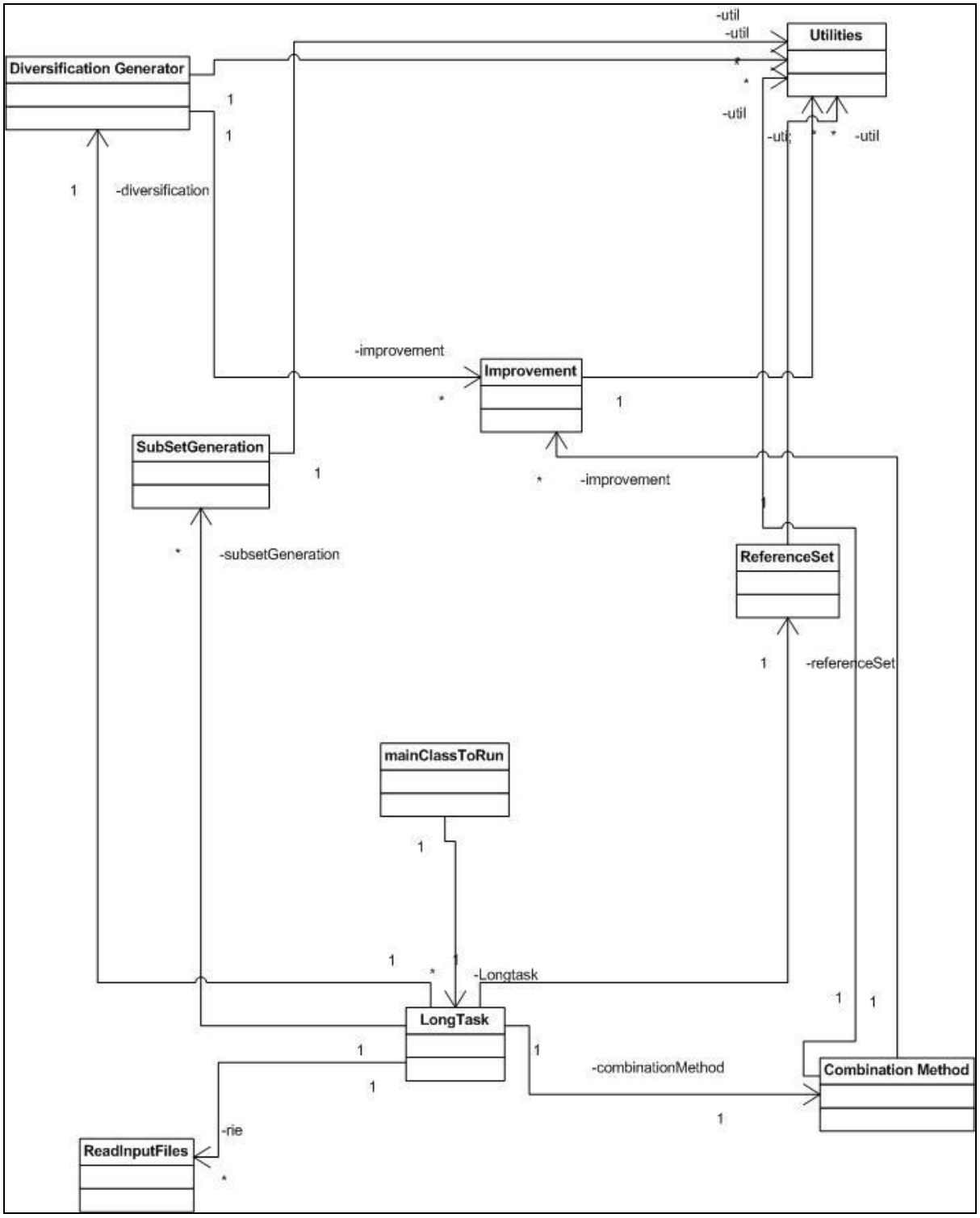


Figure A.1 Class Diagram for PAE

A.2 User Interface

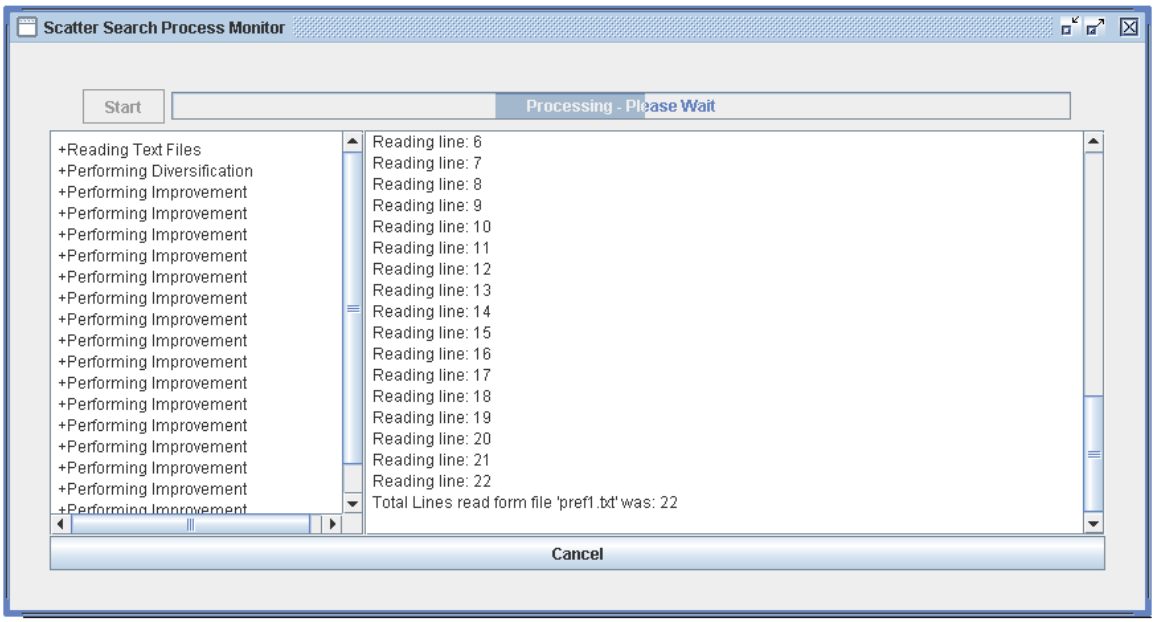


Figure A.2 User Interface

A.2.1 Input Data

We data feed the implemented software through text files. What the software does is that it reads them and them stores them into arrays. We have four types of input files in our implementation *exam.txt*, *pref.txt*, *ta.txt* and *tk.txt*.

Our first file is '*exam.txt'*; it includes the main data for the exams. Line 1 in the file corresponds to the number of exams we have in total, while the rest of the lines correspond to specific exam related data. Each exam has the following data:

- The number of needed proctors.
- Month and day of exam.
- The duration of the exam (starting and ending time).

```
2600
4    6    23    900    1100
2    6    23    900    1100
2    6    23    1130   1330
4    6    23    1130   1330
4    6    23    1500   1700
2    6    23    1530   1730
2    6    23    1730   1930
2    6    23    1900   2100
3    6    25    900    1100
3    6    25    1130   1330
```

Figure A.3: Exams input text file format

Our second file '*pref.txt'* includes the proctors' preferences for each exam. The rows in the file contain the data for each proctor, while the columns contain exam related data.

```
     0    1    3    5    ……..
     1    -1   0    0    ……..
     0    1    4    -1   ……..
     0    0    0    1    ……..
     -1   4    1    0    ……..
……..
……..
……..
……..
```

Figure A.4: Preferences input text file format

Our third file 'ta.txt' comprises the data related to all the proctors. The first line lists that number of proctors that are ready to be assigned. The other lines comprise specific data as follows:

- The first number corresponds to the available hours for each proctor.

- The second number corresponds to the number of exams that must be proctored.
- The third and the fourth numbers correspond to the day and month when the proctor has exams.
- The fifth and the sixth numbers correspond to the start and end time of the reserved exam.

```
520
24  2  6  23  1000  1200  6  25  1000  1200
20  2  6  23  1600  1800  6  25  1000  1200
24  1  6  25  1700  1900
12  1  6  25  1000  1200
16  2  6  23  1000  1200  6  26  1000  1200
24  1  6  26  1600  1800
24  2  6  23  1000  1200  6  25  1000  1200
```

Figure A.5: Proctors input text file format

Our fourth file '*tk.txt'* includes the distribution of the exams over the available periods. The first row corresponds to the number of available periods, while the rest of the lines list the exams that are allotted to the same period.

```
6
4  1 4 2 3
4  5 6 7 8
3  9 10 11
1  11
7  12 15 13 14 16 17 18
3  19 20 21
```

Figure A.6: Exam Distribution input text file format

A.2.2 Output Data

Our output file ‘solution.txt’ will contain a list of records; each list will be having some kind of identifier. The rows and columns in the text file correspond to the exams.

X = 0

0 0 0 0 ……..
1 0 0 0 ……..
0 1 0 0 ……..
0 0 0 1 ……..
0 0 1 0 ……..
……..
……..
……..
……..

X = 1

0 0 0 0 ……..
1 0 0 0 ……..
0 1 0 0 ……..
0 0 0 1 ……..
0 0 1 0 ……..
……..
……..
……..
……..

Figure A.7: Solution.txt output text file format