LEBANESE AMERICAN UNIVERSITY

THREE-PHASE APPROACH FOR CURRICULUM-BASED COURSE TIMETABLING PROBLEM

By

HANAA SALEM EL-JAZZAR

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 2012

Thesis Proposal Form

Name of Student: Hanaa El-Jazzar I.D.#: 200202191

Division: Computer Science and Mathematics

On (dd/mm/yy) 21/06/2010, has presented a Thesis proposal entitled:

Scatter Search for Curriculum Based Course Timetabling

in the presence of the Committee members and Thesis Advisor

Nashat Mansour [redacted] 21/6/2010
(Name, signature, and date of the Thesis Advisor)

Sanaa Sharafeddine, [redacted] 21.06.2010
(Name, signature, and date of Committee Member)

Abbas Tarhini [redacted] 21.06.2010
(Name, signature, and date of Committee Member)

Comments/Remarks/Conditions to Proposal:

Date: 22/6/2010 Acknowledged by [redacted]
(Dean of Graduate Studies/School of .....)

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

Lebanese American University

School of Arts and Sciences - Beirut Campus

Thesis Defense Result Form

Name of Student: **Hanaa Salem El-Jazzar** I.D.#: **200202191**

Program: Master of Science in Computer Science

Department: Computer Science and Mathematics department

Date of thesis defense: **Friday, June 8, 2012**

Thesis Title: **Three-Phase Approach for Curriculum Based Course Timetabling Problem**

Result of Thesis defense:

- ☑ Thesis was successfully defended. Passing grade is granted
- ☐ 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:

Thesis Advisor: **Dr. Nashaat Mansour** [redacted] 8/6/2012

(Name and Signature)

Committee Member: **Dr. Sanaa Sharafeddine** 08.06.2012

(Name and Signature)

Committee Member: **Dr. Abbass Tarhini**

(Name and Signature)

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

Changes Approved by Thesis Advisor: _____ Signature:

Date: **Friday, June 8, 2012** Acknowledge by [redacted]

(Dean, School of Arts and Sciences - Beirut)

8/6/12

Cc: Registrar, Dean, Chair, Advisor, Student

Lebanese American University

School of Arts and Sciences - Beirut Campus

Thesis Approval Form

Student Name:	**Hanaa Salem El-Jazzar**	I.D. #: **200202191**
Thesis Title:	**Three-Phase Approach for Curriculum Based Course Timetabling Problem**	
Program:	Master of Science in Computer Science.	
Department:	Computer Science and Mathematics Department	
School:	School of Arts and Sciences - Beirut	

Approved by:

Thesis Advisor:	Dr. Nashaat Mansour	Signature : [redacted]
Member:	Dr. Sanaa Sharafeddine	Signature : [redacted]
Member:	Dr. Abbass Tarhini	Signature : [redacted]

Date: **Friday, June 8, 2012**

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, back¬up 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: **Hanaa Salem El-Jazzar**

Signature [redacted] Date: **June 8, 2012**

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: **Hanaa Salem El-Jazzar**

Signature 

Date: **June 8, 2012**

To my adorable and lovely family,
To my special loving and caring parents,
To my beloved LAU, my second big family

ACKNOWLEDGMENTS

I would like to sincerely thank my supervisor Dr. Nashaat Mansour who guided me and supported me during all my thesis work. His support for me over the years, since I entered LAU, was a great push for me to continue my thesis, and not to surrender easily to the external pressures of life.

I would like to thank greatly my family, my father, my mother, my fiancé, my sister and my brothers, for supporting me, standing by my side, and being so patient with me. Without them, I couldn't make it. I would like also to thank all my true friends whose words were really innovative for me.

I like to give unique thanks to Dr. Tarek Nawas who was like my second father and special thanks to Dr. Ramzi Haraty, Dr. Samer Habre, and Dr. Faisal Abukhzam since they were like our big brothers during our study at LAU. I would like to thank them from inside my heart because they helped me to fulfill my dreams and get both of my degrees, the BS and the MS, from LAU, my beloved university and my second family.

Finally, I would like to sincerely thank Dr. Sanaa Sharafeddine and Dr. Abbas Tarhini for being in my thesis committee and for their support for me during my study at LAU.

Three-Phase Approach for Curriculum Based Course Timetabling Problem

Hanaa Salem El-Jazzar

Abstract

University course timetabling is an important problem for scheduling courses into predefined periods and rooms over a week with a given set of constraints. This problem is NP-complete and, thus, heuristics are required to produce good suboptimal timetables. This work considers the curriculum based course timetabling problem (CCTP) and proposes three-phase heuristics algorithm that fulfils the requirements of zero hard constraints values and minimal values for soft constraints. The three algorithms are simulated annealing (SA), scatter search (SS) and a tuning heuristic (THEU). We have run our algorithm on subject problems listed at the international timetabling competition in 2007 (ITC2007) and we have compared our results with those of the winner of ITC2007, which is Muller's hybrid algorithm. Our results show that our approach produces better results than Muller's for larger or more complex problems.

Keywords: Metaheuristics; Scatter search algorithm; Simulated annealing; Timetabling; Course timetabling problem; ITC2007.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CHARTS

LIST OF ABBREVIATIONS AND ACRONYMS

SS: Scatter Search

SA: Simulated Annealing

THEU: Tuning Heuristic

CTP: Course Timetabling Problem

CCTP: Curriculum-Based Course Timetabling Problem

UCT: University Course Timetabling

UCTP: University Course Timetabling Problem

PECTP: Post-Enrollment Based Course Timetabling Problem

PATAT: International Series of Conferences on the Practice and Theory of Automated Timetabling

WATT: EURO Working Group on Automated Timetabling

ITC2007: Second International Timetabling Competition sponsored by *PATAT* and *WATT*.

OF: Objective Function

EF: Energy Function

GA: Genetic Algorithm

SSGA: Steady State Genetic Algorithm

MGA: Modified Genetic Algorithm

CGA: Cooperative Genetic Algorithm

EA: Evolutionary Approaches or Algorithms

LOP: Linear Ordering Problem

NN: Neural Network

TSP: Travelling Salesman Problem

MAX-SAT: Maximum Satisfiability Problem

HC: Hill Climbing

PSO: Particle Swarm Optimization

PCA: Particle Collision Algorithm

MPCA: Multi-Neighborhood Collision Algorithm

GD: Great Deluge

AGD: Altered Great Deluge

EGDA: Extended Great Deluge Algorithm

IFS: Iterative Forward Search

CBS: Constraint-Based Statistics

PSN: Problem Specific Neighborhood

RMO: Ranking Mutation Operator

IFO: Intersection Filter Operator

ACO: Ant Colony Optimization

MMAS: MAX-MIN Ant System

ACS: Ant Colony System

RR: Round Robin

RBH: Repair-Based Heuristic

DGM: Diversification Generation Method

IMP: Improvement Method

SGM: Subset Generation Method

RSUM: Reference Set Update Method

CHAPTER 1

INTRODUCTION

1.1 Timetabling Problems

Timetabling problems are one of the most challenging problems of time-based scheduling and combinatorial optimization nature that tend to be solved using a combination of search-based metaheuristic techniques and methods that usually leads to "*acceptable"* and "*good*" but "*sub-optimal"* solutions in a reasonable time. Timetable scheduling problems are known to be computationally *NP-complete*, hard and complex problems because there is no deterministic polynomial time algorithm that can solve them especially with a large number of instances (Chu and Fang, 1999). Moreover, using advanced search and constraint satisfaction techniques based on different meta-heuristic methods showed effectiveness in pruning the search space but it does not guarantee the foundation of an optimal solution since meta-heuristics work as general methodologies (templates) that need to be tailored to solve different timetabling instances, as Talbi (2009) defined them, as well as those techniques concentrate on solving hard constraints while it is more difficult to handle soft constraints (Chu, Chen and Ho, 2006). Thus, here we have to distinguish between two types of constraints: hard and soft. Hard constraints are rigidly enforced and they must be satisfied in the solution. *Feasible solutions* are the solutions that satisfy the hard constraints. On the other hand, the soft constraints are those that are desirable to satisfy, but they are not essential. In real-world timetabling problems, it is almost impossible to solve all of the soft constraints violations.

Timetabling problems can be categorized among one of the most important real life problems and can be divided into various types, like, for example, transportation timetabling, university timetabling (examination timetabling, courses timetabling, …), nursing timetabling, …etc, by considering different specific constraints, processes and resources. In each timetable scheduling problem, there are events that should be spread over certain times taking into consideration satisfying several problem constraints. However, it is difficult to come up with a general algorithmic method that can solve all instances of a timetabling problem because each particular problem instance has its own certain special constraints as well as because of the need to use some simplification assumptions to lessen the problem hardness. Besides

that, real-life timetabling problems often involve constraints that cannot be represented or stated (Chu and Fang, 1999). Performance evaluation for any algorithmic approach used is based on the time spent in producing a feasible timetable (solution) as well as on evaluating the quality of this solution that can be assessed on the basis of how well the soft constraints are satisfied.

1.2 University Course Timetabling

Among all timetabling problems, university course and exam timetabling problems have captured the attention over the years because of their importance in the academic administrative activities that almost all academic institutions need to deal with on regular basis on regular basis. In general, university timetabling is a combinatorial optimization problem of practical relevance defined as assigning a set of events (courses or exams) to a finite number of timeslots while trying to satisfy a set of hard and soft constraints (Qu and burke, 2009). University timetabling problems need much (human or computing) efforts in coming up with solutions that are both effective and of a high quality. University course timetabling problem (UCTP) can be categorized into two types: curriculum-based course timetabling problem (CCTP) and post-enrolment course timetabling problem (PECTP) as defined by ITC2007.

1.2.1 Post-Enrolment Course Timetabling

In general, post-enrollment course timetabling (PECT) problem is a combinatorial optimization problem based on real world course scheduling instances where the schedule is generated after the students enroll or register in different courses and events offered by the department or the institution seeking the timetable. This timetable construction method is usually favored by many educational institutions like universities because it provides the students with the capability of maximizing their choices while making sure that the available resources are used effectively and as efficiently as much as possible, though this construction method is quite a risky option since there is a small room for error in it.

1.2.1.1 Problem Formulation

The PECTP as explained by Di Gaspero, McCollum and Schaerf (2007), in the *ITC2007* competition, constitutes of the following:

1- Set of events/courses that need to be scheduled into periods (timeslots*days);
2- Set of rooms in which the courses will be given where each room has a specific capacity;
3- Set of students registered/enrolled in courses where each student is registered in specific number of courses;
4- Set of features and constraints that need to be satisfied by the rooms, required by the courses as well as the students. For example, some courses can be assigned to specific periods in the timetable.

Hard Constraints

In order to have a feasible timetable, all the following hard constraints of the produced solution for PECTP should be satisfied:

1- Each student is not allowed to be present at more than one course at a time;
2- Every room should not occupy a number of students more than its allowed capacity as well as it should satisfy the course requirements;
3- Courses' conflicts are not allowed; i.e. only one course is allowed to be scheduled at specific period in a specific room;
4- Each course has available periods in which it is allowed to be scheduled in;
5- Courses are scheduled in specific week.

Soft Constraints

On the other hand, when any of the following soft constraints takes place, a timetable will be penalized equally for each violation:

1- When a student is registered in a class at the day last timeslot.
2- When a student is registered in more than two consecutive classes.
3- When a student has only one class in one of days.

Solution Evaluation

To evaluate the solution, first of all, the validity of the hard constraints should be checked. If any of those constraints are violated in the problem solution, then this solution is ineligible. On the other hand, some courses are allowed to remain unassigned (unplaced). In such a case, a *distance to feasibility* measure should be calculated in the following way: Determine the number of students attending each unplaced course; then, the summation of these numbers will result in the distance to feasibilit*y* of the generated solution. After that, the soft constraints violations penalty value/score is calculated by, first, counting the number of occurrences for each

student having just one class on a day, then adding to them the number of occurrences for each student having more than two consecutive classes (e.g. three consecutive classes has penalty score 1, while four consecutive classes count as 2, ...etc), as well as adding the number of occurrences for each student having a class in the day last timeslot. Therefore, the best solution will be the one having the lowest "*Distance to Feasibility"* value, or the solution having the smallest value of soft constraints violations.

1.2.2 Curriculum-Based Course Timetabling

Di Gaspero et al. (2007), in ITC2007 competition, defined the curriculum-based course timetabling problem (CCTP) as a combinatorial optimization problem defined to be the weekly scheduling of courses' to a given number of rooms and time periods, which consist of days and slots, while satisfying the hard and soft constraints. CCTP constraints are mainly based on courses' conflicts that are based on the curricula determined by the university or by the educational institution and not on the post enrollment data-basis as described in details in *Chapter 3*.

1.3 Research Objectives and Contribution

ITC2007 is the *Second International Timetabling Competition* sponsored by *PATAT* and *WATT*. It was composed of three main tracks (adding to the previous two types, the Exam Timetabling problem) as described in the content of the *ITC2007* website: http://www.cs.qub.ac.uk/itc2007/. Although there is much overlap between them, these tracks correspond to different problems in the area of educational timetabling from the practical and the research point of view.

Our objectives in this work is to provide some knowledge about previously existing approaches and algorithms used to solve university course timetabling (UCT) problems, in general, and to solve curriculum-based course timetabling problem (CCTP) in specific, given its constraints and restrictions from *ITC2007* through which we noticed that many concentrated on developing faster solutions based on exhaustive search and the forgot that those approaches has to be applied on large-scale real life problems. Therefore, in our study we propose the use of *Three-Phase Heauristics* approach to solve CCTP using Simulated Annealing (SA), Scatter Search (SS) (Glover, 1977, 1998; Laguna and Marti, 2003) and a greedy Final Tuning Heuristic (THEU), test its performance over five benchmark datasets and aim to prove that our proposed approach is able to produce competitive results in

comparison to those of Muller's hybrid algorithm (2008, 2009) that participated in the competition and that we will talk more about it in *Chapter 2, section 2.3.1*. We are also targeting to have a scalable algorithm that can work on large-scale problems for this we chose to combine the effect of those three heuristics together. To the best of our knowledge, there is not any published and available work that is mainly based on Scatter Search algorithm to resolve CCTP along with the other algorithms used.

According to Di Gaspero *et al.* (2007), the description of CCTP model, which is proposed by the ITC2007 competition, contains many characteristics that exist in some Italian and European universities. Based on that specific problem modeling, the input instances used for testing purposes are real data from Udine University in Italy. Twenty one instances were available for the competitors and they were provided by the competition organizers. Not only this, but the competition organizers also provided the *solution validator* C++ source code that takes the input file and the output file to produce the overall solution evaluation with some detailed descriptions for all the violations of hard and soft constraints.

In this work, we designed and implemented the three heuristics algorithm in a way that takes into consideration the objectives of CCTP. Those objectives are represented as the *hard and soft constraints* that need to be satisfied along with specific weights. The summation of the multiplication of those weights with the accompanying constraints constructs the objective function for CCTP whose minimization is essential to generate good and feasible course timetable. Then, we carry out some experimental tests and runs of this algorithm and compare our results with those of Muller (2008, 2009).

1.4 Thesis Organization

In the rest of this work, a closer investigation for previously developed algorithms used to solve UCTPs in general and CCTPs in specific will be presented in *Chapter 2*. Then, CCTP description in details as it came in ITC2007 will be in *Chapter 3*. After that, in *Chapter 4*, we will talk about our proposed algorithm design used to solve CCTP with all its different phases and adapted methods. Furthermore, we will present the analysis of our empirical results in *Chapter 5*. After that, we will have the conclusion and some suggestions for further possible future work in *Chapter 6*. Finally, in *Appendix I*, we will give a brief description and overview of SS design template.

CHAPTER 2

PREVIOUS METHODS FOR UNIVERSITY COURSE TIMETABLING PROBLEMS

This chapter talks about some previous approaches and methods used to solve university course timetabling problems (UCTPs) and that quietly differ in their description. The main two types of course timetabling we are concentrating on are curriculum-based course timetabling problem (CCTP) as well as post-enrolment based course timetabling problem (PECTP) along with other general university course timetabling problems.

Various different methods have been previously applied to solve different versions of university course timetabling problems (UCTPs). In the following sections, we are going to see how those different methods have been used, modified and enhanced to solve the different types of course timetabling problems. Some of those algorithms used are like hill climbing (HC), tabu search (TS), local search, great deluge (GD), artificial intelligence heuristics, genetic algorithm (GA), particle swarm optimization (PSO), …etc. Moreover, many of those algorithms concentrate on producing feasible solutions (i.e. with no hard constraints violations) in the first stage, then in the next stage, they attempt to minimize (optimize) soft constraints violations.

2.1 General University Course Timetabling Problems

In this section, we will talk about various algorithms used to solve UCTPs with different formulations from different universities.

2.1.1 A Combination of Local Search with Particle Swarm Optimization

In 2009, Irene, Deris and Mohd Hashim defined the problem of course timetabling as the organization of a set of subjects into pairs of rooms (resources) and times (as timeslots) in order to satisfy a set of constraints. They used particle swarm optimization (PSO) along with Local Search algorithm to solve University Course Timetabling (UCT) while their work mainly concentrated on two major issues: The quality of the produced university course timetable, and the time spent in producing it.

In this UCT, as a hard constraint, no two subjects are allowed to be assigned in one specific room at the same timeslot. The university courses are distributed over five days per week where each day has nine timeslots from 8 AM till 5 PM (i.e. equal distribution of one hour for each timeslot). In the Course Timetable framework used by Irene *et al.* (2009), 1,2,…,n are the subjects where n is the maximum number of available subjects that need to be scheduled, the timeslots are numbered from 1 up to 45 (9 * 5) and they represent the y-axis while the rooms are numbered from 1,2,…,k where k is the maximum number of rooms and they represent the x-axis as shown in *Table 2.1.*

Table 2.1: Subjects Allocations for UCTP

Time 45	Subject 1		
.			
.			
Time 3		Subject 2	
Time 2			Subject n
Time 1	Subject 3		
	Room 1	…	**Room k**

In 1995, Kennedy and Eberhart were the first to introduce the *PSO* algorithm as a problem optimization heuristic is inspired by the intelligence of the swarm such as the fish schooling, the bird flocking, and the human social behavior. On the other hand, local search is a search iterative heuristic algorithm that move from a candidate solution *X* to another *X'* in the neighborhood according to certain criteria and to a problem-specific neighborhood structure as explained in Talbi's book (2009).

To solve university course timetabling problem (UCTP), Irene, Deris and Mohd Hashim (2009) proposed an algorithm that combines PSO and local search each representing a specific stage in their search approach. *Figure 2.1* demonstrates the flowchart of the algorithm that combines PSO with local search solve UCTP.

Irene *et al.* (2009) have tested the algorithm on three different datasets of different complexity levels. Timetable had five days from Monday to Friday and particle size of ten (in PSO). Irene *et al.* algorithm was tested five times with only a single clash with penalty 5000 points. They compared their work results with those produced by the hybrid *GA* Constraint-Based Reasoning (CBR) technique (as cited in Zalmiyah, 2001) and with standard PSO. The experimental results showed that the combination of PSO with local search has better results in solving UCTP.

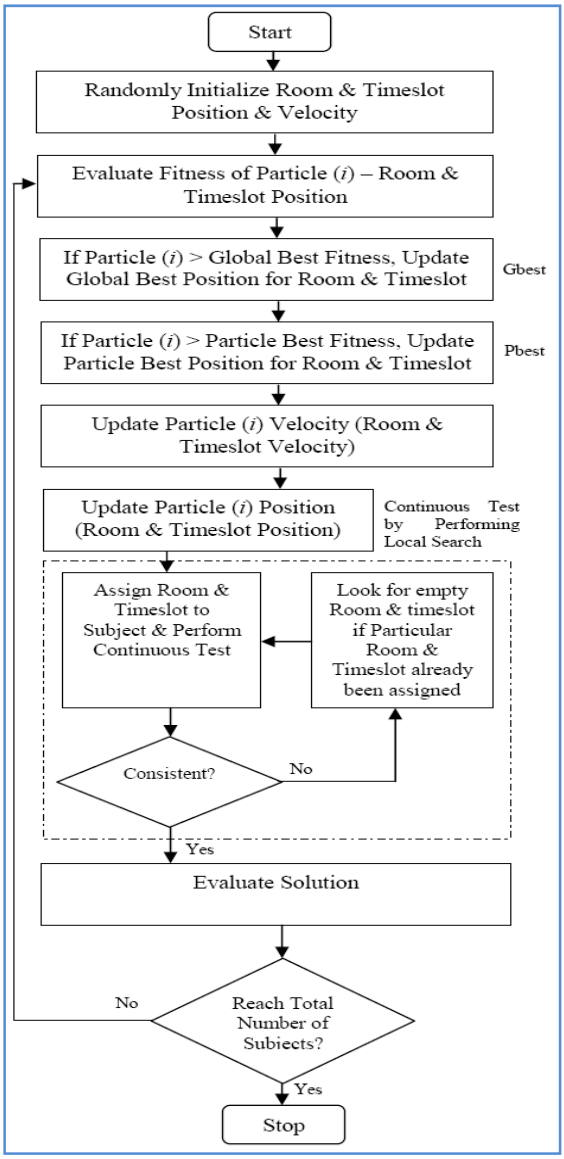

Figure 2.1: PSO Along with Local Search

In Irene *et al.* paper (2009), it is true that the combination of PSO and local search showed promising results by being more efficient in producing a feasible university course timetable in comparison with other published approaches on the same datasets, but they only had one hard constraint in their problem and the timeslots have fixed lengths over fixed number of days (i.e. over simplification and limitation of the problem) that leads some doubts regarding the scalability of the algorithm with more complex real-life timetabling problems. Therefore, more enhancements on the problem side should be made as well as on the algorithm side so that it will give better results for more complicated problems. Therefore, more constraints should be included (other soft and hard constraints) as well as further tuning of the PSO algorithm's coefficients and study of their effects. Not only this,

but also different datasets can be used to further validate the algorithm and apply this technique on different timetabling techniques.

2.1.2 Iterative Forward Search Algorithm to Solve a Hard Course Timetabling Problem

In 2007, Murray, Muller and Rudová discussed some solution and modeling methods that can be used to automate the construction of UCTs at the University of Purdue (a large university). It dealt with a complex real world course timetabling problem through the complex structure of courses. The problem is transformed into a constraint-satisfaction and optimization problem using a multi-layer structural approach applying IFS heuristic and dividing the complex problem into multiple sub-problems along with applying student sectioning and scheduling as well.

Murray *et al.* worked on the student sectioning and course timetabling problem at University of Purdue which applies to the following structure: Create course timetables that meet student course demands in a good way. Minimize conflicts by following a good approach to permit the students to be assigned to different courses. It is a large scale university-wide problem such that the problem includes 9,000 classes, 570 rooms, 39,000 students with 259,000 class requests. Decompose the problem into several problems (large lectures, departmental timetables). For example, the problem is divided into large lecture room (LLR), computer laboratory (LAB), and two selected departments (D1, D2) problems each with specific characteristics as shown in *Table 2.2.* Departmental schedule managers needed so that each is responsible for his own solutions.

Hard Constraints

This problem implements the following hard constraints:

- **Resource constraints**: Only one course or class can be taught by a teacher at any time, and only one room can be assigned for it at a time.
- **Distribution constraints**: This constraint expresses the must be there relations between different classes. For example, some classes should be given one after the other or two lectures of the same course cannot be given at the same timeslot.

Soft Constraints

The three implemented soft constraints are:

- Soft constraints on rooms and timeslots.
- Every student can register in different classes, and the algorithm should try to minimize the student conflicts' value among them.
- Soft distribution constraints that state the favored or the discouraged relations between different sets of classes.

Table 2.2: Characteristics of the Problems to be Tested (Spring 2007)

Problem	LLR	D1	D2	LAB
Number of classes	804	440	69	442
Avg. number of classes per type of instruction	1.25	3.52	1.50	4.8
Avg. number of hours per class	2.40	2.43	2.30	1.97
Avg. number of meetings per class	2.09	2.32	1.67	1.25
Avg. number distribution constraints per class	0.68	2.94	0.78	1.82
Number of rooms	55	25	6	36
Room sizes	$40-474$	$24-51$	$14-48$	$20-45$
Avg. room utilization [hours/week]	35.0	42.8	26.5	24.2
Average distance between rooms [m]	223.9	83.9	21.5	159.7
Number of students	27881	11992	1312	8408
Avg. number of classes per student	3.15	1.11	1.40	1.14
Classes with an instructor assigned [%]	69.8	33.9	60.9	13.35
Avg. number of classes per instructor	1.25	1.49	1.68	2.11

For each class, the information needed is the time and student requirements as well as the meeting pattern, the room necessities and preferences like the capacity, necessary equipment, room/building preference and building distances. Information about instructors…etc

System Architecture

Murray *et al.* modeled the problem using three-layer architecture*:*

- **Presentation Layer**: consists of the user interface, and contains course modeling structure, classes, instructors, rooms, constraints, and requirements as submitted by the users.
- **Timetabling (solver implementation) Layer**: contains the timetable solver data modeling, solver implementation as well as problem specific heuristics.
- **Constraint Satisfaction (solver abstract) Layer**: includes the implementation of the optimization solver. The solver is guided by a general set of heuristics without knowledge of any classes, rooms or other timetabling specific primitives. This will help in constructing a general framework that can be used on different timetabling problems.

Timetabling Solver

Muller's solver is built on the basics of Iterative Forward Search (IFS) algorithm (2005) as shown in *Figure 2.2.* The algorithm is a combination of local search and backtracking techniques. It gradually produces feasible assignments. It is applicable to different timetabling problems and scenarios. It could be extended easily. And it uses Constraint-Based Statistics (CBS) technique (Muller, 2004, 2005).

The solver built the model of the problem and the constraints structure based on the difficulty of all the courses. Also, the problem model considers data consistency; For example, the solver has the capability of identifying and presenting any inconsistencies and possible problems that might happen in the input data.

```
procedure SOLVE(initial)                    // initial solution is the parameter
    iteration = 0;                          // iteration counter
    current = initial;                      // current solution
    best = initial;                         // best solution
    while canContinue(current, iteration) do
        iteration = iteration + 1;
        variable = selectVariable(current);
        value = selectValue(current, variable);
        UNASSIGN(current, CONFLICTING_VARIABLES(current, variable, value));
        ASSIGN(current, variable, value);
        if better(current, best) then best = current
    end while
    return best
end procedure
```

Figure 2.2: Pseudo-Code of the Search Algorithm.

IFS algorithm starts with a solution in which all the lectures are still unassigned. Then, at each iteration, IFS algorithm selects an unassigned lecture randomly or it finds this variable among those most difficult to assign depending on the selection criteria (*Figure 2.4*) and it is also assigned to a best value chosen from its specific domain (*Figure 2.5*). If any hard constraints violations occurred with this assignment, then the conflicting lecture will be unassigned (*Figure 2.6*).

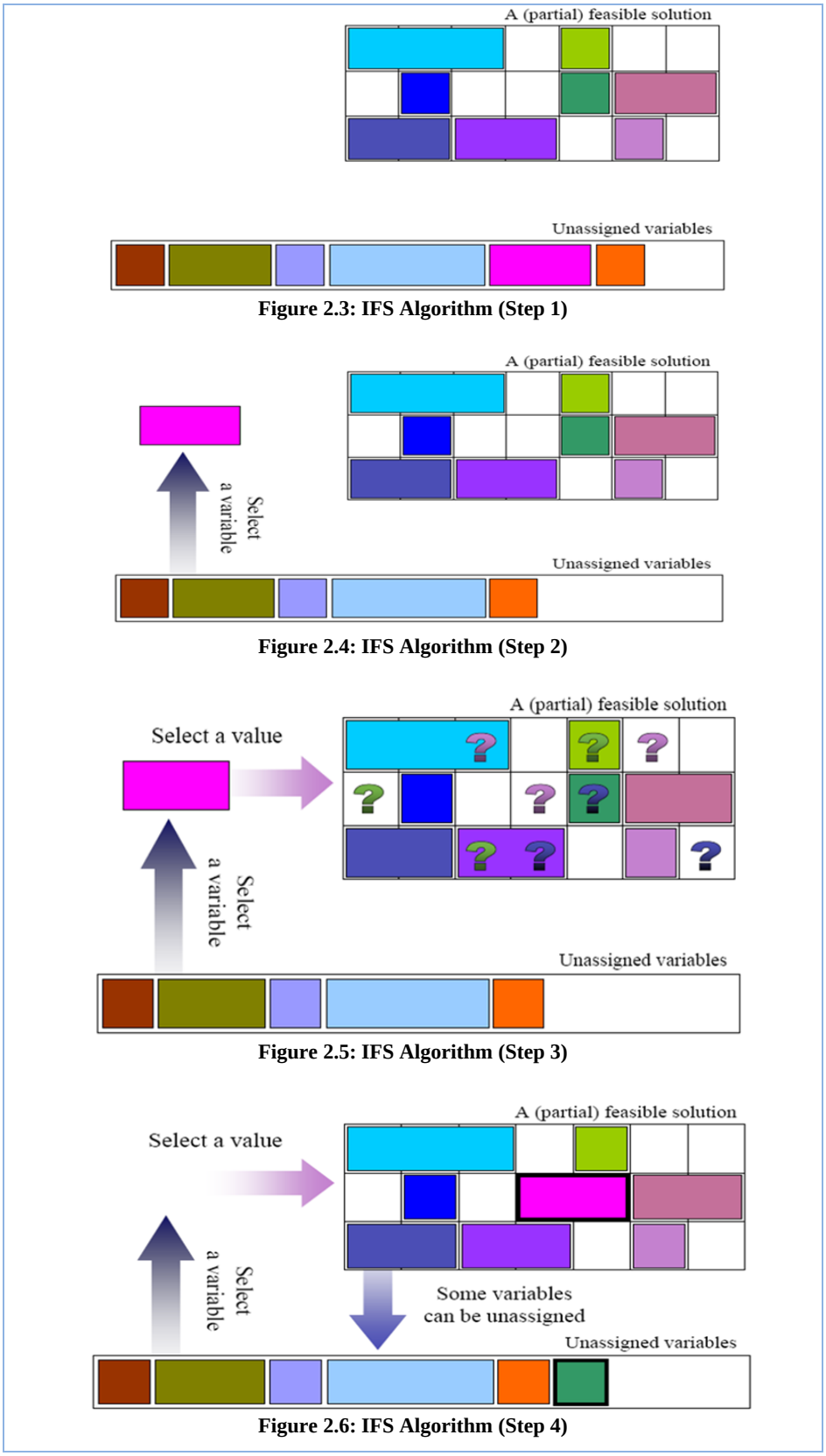

Figure 2.3: IFS Algorithm (Step 1)

Figure 2.4: IFS Algorithm (Step 2)

Figure 2.5: IFS Algorithm (Step 3)

Figure 2.6: IFS Algorithm (Step 4)

The value selection criteria depends on choosing the value whose assignment will decrease the solution overall cost by decreasing the number of those conflicting lectures that need to be unscheduled to take the solution back to its feasibility state.

At the first stage, the soft constraints violations can be ignore. Therefore, the value selection procedure will concentrate on selecting a value that minimizes the violations of hard constraints. The search stops when all the variables got assigned.

During those steps, Conflict-Based Statistics *(*CBS*)* (Muller *et al.*, 2004) has been applied in order to memorize the conflicting assignments and discourage their potential repetition. Each conflict is weighted by its occurrence in the past. Furthermore, CBS is a data structure that memorizes specifically hard conflicts that occur during the search procedure. It stores the conflicts frequency and the assignments values that caused them as shown in *Figure 2.7*. This does not mean that it prevent such assignment from happening again, but during the value selection procedure, each hard constraint violation is weighted by its frequency which means that it is evaluated by the number of past un-assignments of the conflicting variables caused by this assignment.

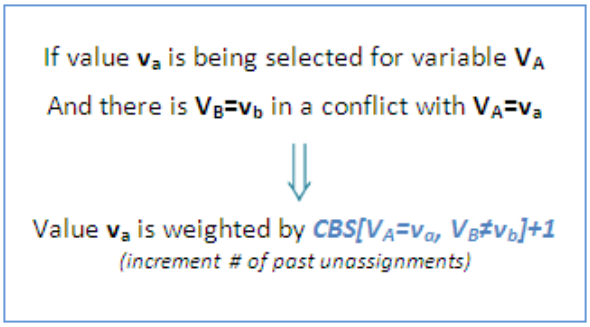

Figure 2.7: Constraint-Based Statistics Array Format and Example.

Student Sectioning

In course timetabling problems, students usually request courses and the system will determine the classes (sections). The system has to respect the course structure, reservations, and student choices.

The student sectioning procedure is divided into two major phases:

- **Initial sectioning** of students to classes takes place during timetabling. It aims at minimizing the potential student conflicts. This can be achieved by moving the students from one class to the other belonging to the same course throughout or following the search.
- **Final Sectioning (re-sectioning)** is done after the timetable for the whole university is created, or according to user's inquiry. The re-sectioning stands over local search algorithm with some neighborhood moves like, for

example, two students registered in the same course can exchange their class (lecture) assignment.

The system was tested on data from Spring 2007 semester from Purdue University. The departmental data was entered by separate departments' manager each responsible for the part of the system related to them. According to the results of the tests, it became clear that the complex scheduling problems can be solved by dividing them into smaller sub-problems and solving each separately at the start.

Murray *et al.* faced so many challenges during their work on a complex university course timetabling and student sectioning among which is the ability to understand the structure of the problem and be able to address the concerns that the users have along the basic solution methods. Developing the framework in a three-tiered structure was a good idea that helped in developing a flexible approach to solve different problems. Also, during this work, it has been proven that it is possible to build up some models that could be comprehensive enough on a broad set of problems. This is true in concept, but it has not been tested on different wide range of data from different universities with different complexity of the course timetabling problems with different complex course structures. In this paper, it has not been mentioned clearly how the verification process of the solution took place or how the solution verification procedure for such complex problems can be done. Also, considerable training has to be performed for scheduling managers because they have to be trained on the use of the servers and this might exhaust some time and some university resources. Thus, this issue should be taken care of seriously.

2.1.3 A Steady State Genetic Algorithm

In 2002, Ozcan and Alkan presented a steady state genetic algorithm (SSGA) to solve a multi-constraint UCTP. What characterizes this approach is that it makes no distinction between hard and soft constraints. Also, it introduced ranking mutation operator (RMO), which is a new mutation operator that showed to be successful, as well as the initial experimental results look promising. In addition, a configurable application has been coded, named TEDI using SSGA.

The algorithm is targeted for a specific university in Turkey, and hence it defines the university imposed constraints on the problem. Some of these constraints are for example:

- A regular undergraduate student must complete 5-8 courses depending on the curriculum.
- Lectures of a course must not be scheduled in the same day.
- Schedule courses that belong to the same curriculum and of a specific teacher in a way that assigned periods must not conflict.

First, the algorithm creates a random initial population of individuals in which mapping of timeslots to class periods of all available courses take place. The quality of these individual solutions are measured and determined using a fitness function, and by assigning penalties to those that do not satisfy a certain constraint.

The approach defines a new operator, RMO, which chooses a gene to apply the mutation operator on it by taking the fitness contributions of genes into considerations and it uses the strategy of ranking. And because using crossover and mutation can lead to an infeasible solution, Intersection Filter (IFO) algorithm is used as a filtering repair method. The process continues until the number of generations exceeds 200,000 or the best fitness is achieved.

Preliminary tests were performed to fine-tune SSGA parameters and test the different newly developed operators. The input data consists of 66 given courses, with 205 classes' hours need to be assigned to a timetable with 115 course gatherings. SSGA with one point crossover, RMO and IFO did the best. Ozcan and Alkan's algorithm (2002) defines its own constraints since it is built for a specific university; thus, abiding by the university rules. Although the results that were obtained are reported to be good by the paper, the approach does not differentiate between hard and soft constraints, and hence making it a bit more unusable for standard timetabling problems. Moreover, the ranking strategy that the defined operator uses is not explained, and the filtering algorithm was not elaborated on or referenced in the paper. It appears that the algorithm is a slight variation of the classical GA. As possible future work, a method should be developed that modifies that the penalties associated with the different constraints forming the fitness function and TEDI will be adjusted to control different classroom assignments along with high school class timetabling.

2.1.4 A Genetic Algorithm Optimizer

Ghaemi, Vakili and Aghagolzadeh (2007) used computational evolutionary algorithms like GA to solve timetabling problems. The main target was to optimize

the number of conflicts in each timetable. They applied a modified GA and cooperative GA.

The timetables they used for testing came from input data for undergraduate courses for the faculty of electrical engineering in Tabriz University. The students are divided into four groups. The day is divided into four timeslots and each one consists of 90 minutes lecture time and 30 minutes break. Each course is taught for one or two timeslots per week. Five main entities define the problem and they are the Teacher, Student, Subject, Classroom location and Timeslot. Those are the main objects that will need to be directly assigned in a complete schedule.

Hard constraints:

- Every teacher can teach one class at a time.
- Only one class can be taught in one classroom at a time.
- Conflicts are no allowed to happen between subjects for students belonging to the same group.

Soft Constraints:

- The lectures of the same subject should be scheduled on different days.
- The second subject time should not be scheduled on the next day of the first time.
- No subject should be assigned to a time not allowed for it by the head of the department.
- The course/subject should not be allocated to a time period that a lecturer does not demand.
- The course/subject should not be assigned to a time period not available for the teacher of the course.

Genetic Operators (Approach)

The approach uses the following crossover, selection and mutation operators:

- **Crossover:** They applied three crossover types. They are single point (SX), Uniform (UX), and modified uniform (MUX) crossover types.
- **Selection:** They chose to apply two selection methods. They are the roulette wheel weighting technique and the Tournament selection in which three candidate individual chromosomes are used to select one parent.
- **Mutation**

The repair function

The chromosome coming from the crossover of the mutation methods might lose its feasibility. Then, a deterministic repair method is used (cited by Michalewicz, 1996).

Algorithm Implementation

Ghaemi *et al*. (2007) applied Modified GA and Cooperative GA on the problem:

- **The Modified Genetic Algorithm (MGA):** The algorithm begins with *pop1* and *pop2,* two initial sets of chromosomes. They are randomly generated. The single chromosome consists of 40 genes. To create a complete chromosome, the algorithm randomly selects from *pop1* three candidate chromosomes. Then, each one of those selected chromosomes contributes with the chromosomes in *pop2* to produce a complete chromosome. This approach works on both populations sets in the same way. It has the advantage of lessening the single population size as well as the results can be now reached faster and at a shorter run time.
- **The Cooperative Genetic Algorithm (CGA):** Usually, GAs works slowly on large search spaces. Hence, for speeding up the execution, now apply the co-evolution method; thus, placing *pop1* and *pop2* respectively in *species1* and *species2*. Separate these two species as you are independently constructing those. Moreover, their contribution will lead to the reduction in objective function cost value. Hence, co-evolution of cooperative type is being used. Accordingly, another three candidate individual chromosomes are chosen randomly from *species1* and *species2*.

The testing results show that MGA improved considerably the algorithm's performance by simply using some adapted genetic operators. Therefore, the overall value of the solution as well as the overall behavior of the algorithms enhanced greatly with the help of such intelligent. Not only this, but also the overall performance of the algorithm enhanced remarkably by using CGA approach.

You should note that less number of bits is needed to represent such type of chromosomes. Moreover, now the algorithm will consume less time at each run. Also, the testing results showed that the use the use of smaller population size than the one used in GA for MGA and CGA lead to better results and speeded up the

developed algorithm. Also, you can observe that the number of violated constraints in CGA is less than that produced in MGA method. However, the algorithm needs to be tested on UCTPs and on larger scale real-life problems.

2.1.5 Multi Agents (Artificial Intelligence) Approach

In this paper, Nandhini and Kanmani (2009) solve course timetabling problem with multi agents by using a proposed steepest ascent hill climbing approach. The following two agents are used in this approach:

- ***CombinationGenerator*** which generates the maximum possible combinations for the input timetable.
- ***MinFinder*** which finds a timetable combination with minimum evaluation function for further examination.

They consider a course timetabling problem model followed by Pondicherry University. It has set of five teachers *T*, set of five Subjects *S* and two Practical sessions *P*. They suppose that each teacher need to be assigned one subject to teach; The lectures of each subject has to be distributed over four or five different periods per week while the practical sessions has to be allocated in three successive periods only in the afternoon two distinct days. The whole timetable should satisfy the limit of having maximum thirty hours per week to schedule the subjects and the practical sessions on.

Hard Constraints

- Lectures of a teacher for a subject must be scheduled at different time slots in a week.
- All timeslots in a week should be scheduled.

Soft Constraints

- Maximum number of time slots for a subject in a day could be two.
- There could be a min of two timeslots gap between same subjects in a day.
- At least once, each subject should come in the first timeslot of the week.
- A subject should be allotted in minimum of 3 days in a week.
- Maximum in two days of a week, a subject can come in the same time slot.
- Adjacent days of a week could not have same subjects in the same timeslots.

Agents Description

The agent named *CombinationGenerator* is developed to work on the hard constraints violations in collaboration with the suggested heuristic processes:

– Adjacent timeslots could not have the same subjects, and teacher's workload in a week should be equally distributed.
– When a timetable is assigned to an agent, it generates the various combinations satisfying the constraints and heuristics.

On the other hand, the agent named *MinFinder* is developed to work on the soft constraints and while trying to find the suitable collaboration with the minimum evaluation function. Weights are assigned to the soft constraints based on their importance. For all the combined solution generated by CombinationGenerator, their evaluation objective function value is measured by using the weights and costs of soft constraints violations. Then, the solution combination with the minimum evaluation function value is chosen by the MinFinder and it is passed through Hill Climbing for further as shown in *Figure 2.8*.

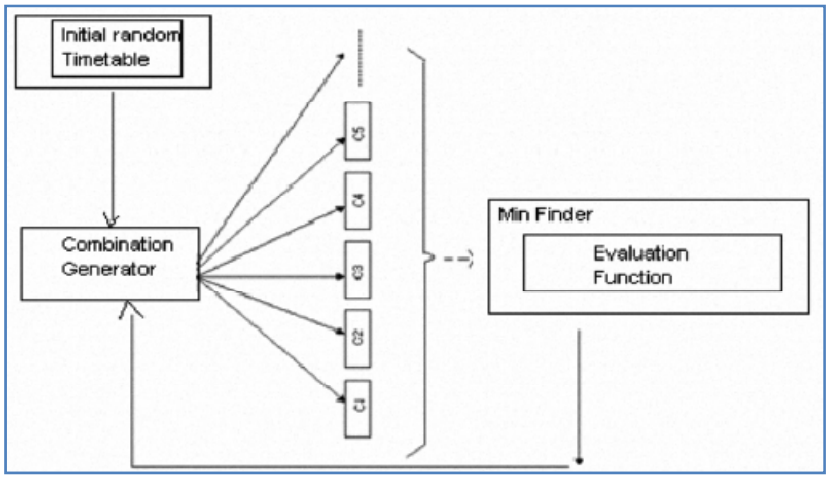

Figure 2.8: The Agents Communication

It is mentioned that the algorithm produces effective result, but the results data was not published in this paper. Applying those heuristics to generate different solution combinations helps tremendously in reducing the search space, and yielding the optimal solution in a faster way. The introduction of the agents, CombinationGenerator and MinFinder, helped in significantly decreasing the complexity of the problem since the work is now divided over them. The formulation of the first two soft constraints makes it hard to calculate their penalties since they are based on a probabilistic condition. Moreover, the solution encoding is not elaborated in this paper, and hence is left ambiguous. Also, unlike real world problems, their problem model only has two hard constraints which make it easier but, unfortunately, far from practical. They promised to extend this in their future

work. Lastly, the communication between the agents seems to be synchronous, and hence the agents will have to wait for each other's input to move on. Therefore, introducing some distinction between hard and soft constraints, and using some efficient heuristics to enhance the search process should have been there.

2.1.6 A Simulated Annealing Algorithm implemented with a new Structure of Neighborhood

Liu, Zhang and Leung (2009) presented in their paper a new neighborhood structure. This neighborhood is attained after applying a number of consecutive swaps of periods in contradiction with the standard neighborhood where only one move is applied at a time. Now, SA can work on solving the course timetabling problem while using the new neighborhood structure.

There is quite some difference between university course timetabling and high school course timetabling. Usually, in high schools, the students are distributed over different classes where each one has specific classroom assigned for it, and the teachers are given specific lesson assignments to teach for different classes. High School Timetabling problems are described as the assignment of timeslots to different lessons where taking into consideration a different set of constraints.

Hard Constraints

- A teacher conflict might occur if it is assigned to two or more lessons at the same period.
- Each class is not allowed to be assigned to two or more lessons at the same period.
- Each teacher must teach exactly the lessons specified for him.
- Each class must be present at the exact lessons assigned.
- Every teacher must teach in the available timeslots allowed for him.
- Each class must have consecutive class hours during the day.
- The class is allowed to have a no-class hour only at the last period of the day.

Soft Constraints

- The teaching hours of each teacher should be continuous during the day.
- Teacher's teaching hours should be balanced throughout the week.
- Each class should not attend the same lesson twice or more on one day.
- Teacher should not be allocated to timeslots s/he is unwilling to work in.

A two-dimensional matrix CXP is needed to represent the solution, where the rows represent the classes the school has and the columns represent the periods to be assigned to teachers. Therefore, when teacher *t* has a lesson in class *c* at time *p*, you will find *t* located in row *c* and column *p.* A feasible solution should have all the (teacher,class) assignments.

Neighborhood Structure

The standard neighborhood structure is used to be the set of solutions that can be reached from the currently used solution by applying only a single move (as cited by Avella *et al.*, 2007). However, in the new neighborhood structure, the neighborhood is constructed by applying a set of consecutive swaps between two periods and the decision whether this swap is accepted or not is taken by the SA algoritm.

Algorithm

This algorithm includes two phases. At the first phase, a feasible solution will be produced, while in the second phase, SA tries to improve the quality of the solution while keeping its feasibility.

- **Phase 1: A feasible timetable construction:** at this phase, only hard constraints are considered. A stochastic strategy is adapted at this phase in which for each SA loop, a period is selected randomly (hard constraints violations might occur) and choose another one also randomly from the rest of periods. After that, a set of consecutive swaps between these chosen periods will be generated.

- **Phase 2: Feasible solution optimization:** After finding a feasible solution, Liu *et al.* tries to decrease soft constraints penalty value while keeping the feasibility of the solution and by using a tabu search approach.

The testing of this approach was done with two benchmark data input instances based on real-world high schools scheduling problems, and the computational results showed that the *SA* algorithm that adopted the new neighborhood structure was able to compete with other existing methods. The results show that Liu's *et al.* algorithm is capable of producing high quality solutions that can compete with those produced by other algorithms. Though their performance was good so far, but they should try to adopt this new neighborhood with other heuristics and try to solve larger-scale CTPs.

2.2 Post-Enrolment Based Course Timetabling Problems

In this section, we will talk about various algorithms used to solve course timetabling problems with different formulations from different universities while being based on students' enrolment/registration in courses prior to scheduling them.

2.2.1 Hybrid Multi-Neighborhood Particle Collision Algorithm

In Abuhamdah and Ayob's (2009) paper, a multi-neighborhood collision algorithm (MPCA) is introduced. It improves the PCA method suggested by Sacco for policy optimization (cited by Sacco *et al*., 2005). The algorithm operates on university course timetabling, and the results prove that MPCA performed better that the other existing methods in some of the instances and that it is able to give good solutions in an acceptable time. The paper uses the following same hard and soft constraints that were set in ITC2002 for post-enrollment course timetabling.

Hard Constraints

Each student cannot attend more than one course at the same allowed period. Each room should be large enough so it can hold all the attending students and, at the same time, it should be able to satisfy all the features needed by the course. Only one course can be given in any room at specific timeslot/period.

Soft Constraints

In the last period of the day, no student should have any class. A student cannot have more than two classes assigned successively. A student should not have only one class per day.

The quality of the solution timetable is calculated by incrementing the penalty value for each kind of violations of soft constraints by one point. MPCA consists of two stages. First, the constructive stage which produces a feasible solution, and second, the improvement stage which tries to optimize the soft constraints penalty value.

Problem Moves

The algorithm uses the following stage 1 and 2 moves.

Stage 1 Moves:

- Select two courses randomly and try to swap their periods' values (and rooms if needed).

- Select randomly two periods and simply try to swap all the courses in one period with all the courses in the other period.
- Select randomly four courses and try to swap their periods (and rooms if necessary)

Stage 2 Moves:

- Select a course randomly, then select a feasible period and feasible room, and move the course to the new period assignment (and move the course to the new room if needed).

Constructive Algorithm

An initial solution is found using a constructive graph coloring heuristic. Next, a feasible solution is attained by using the following three steps successively:

- **Step1: Highest Degree Heuristic**: All the courses that do not conflict are assigned. Then, the courses that were left unassigned while having a high number of conflicts to timeslots are assigned randomly. Then different rooms are selected to be assigned to different courses by applying the maximum matching method for bipartite graph.
- **Step2: Local Search**: If the previous step does not give a feasible timetable still, the algorithm executes this step where two neighborhood moves (move 1 and 2) are used to produce a feasible timetable. If not feasible yet, the algorithm executes the next step.
- **Step3: Tabu Search**: TS is applied using only move 1. The local search is repeated and the Tabu search until a feasible solution is obtained.

The Algorithm

MPCA is almost similar to PCA. The major difference between PCA and MPCA is that MPCA does not depend on the user-defined parameters and it has an improvement phase which is considered to be the exploration phase in which MPCA differs from PCA mainly by the use of hill climbing as an “exploration” method to do the required local search. PCA performs local search in the construction and in the scattering stage, while MPCA performs local search based on hill climbing only in the scattering stage, and this gives it the capability of running away from getting stuck at the local optima. In addition to that, while PCA employs the same neighborhood in stage 1 and 2, MPCA uses different neighborhood structures in both phases.

The criterion of accepting the solutions in the "Scattering" function are similar for both PCA and MPCA, where the approach is more probable to accept some bad solutions in order to escape local optima and solve the UCTP.

The experimental results conducted by Abuhamdah and Ayob (2009) indicate that MPCA has good performance using the multi-neighborhood structure and achieved a high quality solution for different datasets sizes. For small datasets, the results showed scores similar to those of already existing published methods. However, for medium datasets, their results were better than those previously published work. However, unlike other meta-heuristics in the literature that can be tuned towards solving more than one type of course timetabling problems and yet give acceptable results, the MPCA results proves that the approach competes with other existing methods in the literature which is not bad, but is only appropriate for solving CTPs. Some algorithms like the one for generating the initial solution were not elaborated on in this paper, or referenced, thus how to obtain an initial solution is not very clear. Therefore, the authors should look forward to combine MPCA with other optimization approaches and to enhance it with different neighborhood structures among a bigger set of neighborhoods and use it on ITC2007 input datasets.

2.2.2 A Hybrid Approach Using Simulated Annealing

In this paper, Abdullah and Hamdan (2008) considered the course timetabling problem with the following constraints:

Hard Constraints

There is no student who can register for more than one course at the same period. Each course might have specific characteristics and features that need to be taken care of when you try to assign a room for it. For example, the capacity of the room chosen to be assigned to the course must larger or equal to the number of students attending the course. And at last, there must not be any course assigned to a period and a room already assigned for another lecture.

Soft Constraints

The weights over the soft constraints penalties are equal. It happens when a course of a student got scheduled at the last timeslot of the day. Each student should have more than two successive courses. Each student should have only one course per day.

The Algorithm

Abdullah and Hamdan (2008) presented in their paper a hybrid approach that consists of three phases. In phase one, an initial feasible solution will be constructed by applying the construction heuristic. In the second phase, an improvement method that includes an iterative algorithm along with a complex neighborhood structure will be applied as well as it will apply SA as an acceptance criterion. In the third phase, a hill climbing technique will be used to further enhance the quality of the timetable.

Experiments has been conducted for the proposed approach using benchmark *CTP*s (cited by Socha *et al.*, 2002) which are divided into five small, five medium and one large datasets/problems. The tests showed that this method is capable of generating feasible solutions for all dataset instances. This paper proved that *SA* acceptance criterion could help in minimizing the objective function for this problem and showed better performance in comparison to the Monte-Carlo acceptance criterion (cited by Abdulla *et al.*, 2006) that only concentrates on the solution quality, while *SA*'s cooling schedule might accept some bad solutions with higher acceptance probability at the start of the search and this probability decreases with the decrease in temperature leading to intensifying the search to get a better local optimal. Therefore, this approach showed good results in comparison to other previously published approaches. However, only one large dataset was used for testing, so it cannot be said that the results are hundred percent reliable. Therefore, more tests need to be done with large datasets. Not only this, but also the data used for testing are benchmark data and not from the real-world course timetabling problem. Hence, testing on real-world data should be done.

2.2.3 Ant Colony Optimization Algorithm

This paper written by Socha, Sampels and Manfrin (2003) worked on solving a basic version for UCTP because they are targeting to perform a comparison between two ant approaches; ant-colony system and MAX-MIN ant system.

This paper uses a version of UCTP where each student has already been assigned a subset of events. In his problem, a feasible timetable has all its events assigned to a period and room assignment with the next hard constraints respected: each student is allowed to register for only one course at a time, the selected room for assignment must have a capacity higher than the number of students registered for that course, each course might have specific requirements and features that need to

be taken care of when you try to assign a room for it, and one course must be assigned to a room at specific period to avoid conflicts. In addition to those hard constraints, the weights applied over the soft constraints penalties applied over the feasible solution produced are equal. The soft constraints of the problem are as follows: when the student registers in a class at the last timeslot of the day, when the student registers in more than two courses in a row and they give one point of penalty for each attended class, and, at last, when the student got registered in only one course during one of the days of the week.

Ant algorithms

There ecists at least two variations of ant-colony optimization: The MAX-MIN ant system (MMAS) and ant-colony system (ACS). The basic operation idea is similar for both of the methods, but the main difference is in the way each update the pheromone.

Both Ant algorithms have exposed the ability to give good solutions for UCTP input data. A comparison has been done between ACS and other recent meta-heuristics, while the comparison of MMAS with some random restart local search methods showed that MMAS produced better outcome. However, both algorithms were never compared against each other.

Some Differences between the two algorithms

The main difference between those two algorithms is in their different ways of using the currently present information, of using the local search techniques, and of how they update their pheromone matrix structures. Not only this, but those two algorithms differ also in their distinct ways of dealing with the heuristic information itself that MMAS does not use at all, while the ACS tries to use it before every neighborhood move. MMAS and ACS have one more last difference in the way each one employs the local search methods. In MMAS, the solution with the minimum soft constraints violations is chosen to be passed through the improvement local search method, while, on the other hand, ACS uses local search technique over every solution it produces in order to optimize it further.

All the way through comparing these two ant algorithms against other heuristics on small instances, it was more than obvious that Simulated Annealing did much better than any of the above algorithms or better than even the Random Restart Local Search (RRLS) algorithm itself. Therefore, MMAS performed much better

than RRLS, while RRLS performed considerably better than ACS. However, on large instances, the testing experiments showed that MMAS performance is the best, while ACS performance is considerably worse. Comparing the ant algorithm against each, the statistical results show big difference in performance between MMAS and ACS where MMAS worked considerably better. Also, the paper does not give an affirmative answer on the influence of the parameters used in Ants algorithm on the algorithms' performance, and yet more research and experiments needs to be done on this issue. More additional examination, testing and additional analysis is required in order to examine more closely the effect of different parameters on the ant algorithm performance.

2.2.4 An Extended Great Deluge Algorithm

In this paper, McMullan (2007) extended the GD algorithm for CTP, targeting to get rid of the problem of premature convergence and being stuck in the local optima, while employing simple structure of neighborhood search methods to produce the timetables in a faster way. The paper presents great results over currently published techniques where they have up to 60 percent improvements in some of the test cases.

The hard constraints of this problem are the same as for the problem being discussed in *section 2.2.3*. On the other hand, the soft constraints are counted when a course got scheduled at the last period of any day, a student has two successive courses, and when a course is scheduled separately on one of the days of the week.

The Extended Great Deluge Algorithm (EGDA)

Generally, the termination stopping condition of GD or SA algorithms will be executed when no more enhancement is made over the solution for a specific predetermined time amount that is called idle time because at that moment, the farthest optimal solution will be achieved. Instead of stopping the algorithm, EGDA will apply a reheat technique in order to make the boundaries of the algorithm wider and that's specifically to permit the acceptance of some bad moves to the present solution. The cooling approach will carry on normally until the boundary got reduced at specific rate. The purpose behind this extension is first to enhance the pace of execution of the algorithm upon which an optimal solution is attained by employing relatively simple neighborhood moves, and second to use get use of the benefit of escaping getting trapped in local optima.

As shown in *Table 2.3*, EGDA has scored very good results for Medium-sized datasets, with around 60 percent of enhancement averagely over 10 testing executions. On the other hand, for Large-sized datasets, there is an exponential enhancement over the previous best output existed so far. Moreover, the run over small-sized datasets produced results that do not show significant improvement on the formerly existing published results. However, the authors noted as well that for smaller datasets, results with zero objective value were achieved more often than those of the other algorithms.

Table 2.3: Experimental Results for *EGDA*

Data Set	*EGD*		*CNS*	*VNS with Tabu*	*Local Search*	*Ant*	*Tabu HH*	*Grph HH*
	Best	*Avg*						
Small1	0 (x 4)	0.8	**0**	**0**	8	1	1	6
Small2	0 (x 2)	2	**0**	**0**	11	3	2	7
Small3	0 (x 4)	1.3	**0**	**0**	8	1	**0**	3
Small4	0 (x 5)	1	**0**	**0**	7	1	1	3
Small5	0 (x 8)	0.2	**0**	**0**	5	**0**	**0**	4
Medium1	80	**101.4**	242	317	199	195	146	372
Medium2	105	**116.9**	161	313	202.5	184	173	419
Medium3	139	**162.1**	265	357	77.5% Inf	248	267	359
Medium4	88	**108.8**	181	247	177.5	164.5	169	348
Medium5	88	**119.7**	151	292	100% Inf	219.5	303	171
Large	730	**834.1**	100 % Inf	100 % Inf	100% Inf	851.5	80% Inf 1166	1068

The results proved to be good only on medium and large datasets. However, for small datasets, results are not very promising. Moreover, the reason the evaluation is returning zero sometimes is not yet clear, and hence more investigation is yet to be done on this part. For this, more work still need to be done to consider these issues that appeared during the testing over small and large datasets.

2.3 Curriculum-Based Course Timetabling Problems

This section will talk about some of the previously existing algorithms that have been used to try to find solutions for CCTPs with mainly the formulation taken from Di Gaspero *et al.* in 2007 and proposed by track3 in the ITC2007 competition where the input instances used represent real data from University of Udine in Italy and explained in *Chapter 3*.

2.3.1 Muller's Hybrid Approach with Iterative Forward Search, Hill Climbing, Great Deluge and Simulated Annealing

In this paper, Muller (2008, 2009) presents a hybrid constraint-based solver (HCBS) used to solve the ITC2007 three main tracks (Di Gaspero *et al.*, 2007). We

will concentrate mainly in our discussion on how this Solver used mainly to solve *CCTP* described in details in *Chapter 3*.

Muller's Hybrid Approach

First of all, what differentiates Muller's hybrid approach (2008, 2009) from the other approaches, and what made us to chose comparing our work with it, is that it was applied and tested on the three important timetabling problem instances represented by all the three ITC2007 competition tracks with some negligible changes to reveal diverse problems' formulations (e.g. change in solver parameters, different problem modeling, use of problem specific neighborhoods, …etc). Also, some problems like Student Sectioning and University Timetabling (Muller, Murray and Schluttenhofer, 2007). Random Binary CSP (Muller, 2005) have been developed using the Constraint Solver Library.

Not only this, but it has been previously tested on different large-scale schedulin problems like UCTP applied at Purdue University (Murray, Muller and Rudová, 2007; Muller, Barták, and Rudová, 2005), and it proved successfully the viability of applying a single solution framework on different major scheduling problems with minimal changes. Furthermore, Muller's work (2008, 2009) was among the finalists in all three tracks of the ITC2007 competition (Di Gaspero *et al.*, 2007) and the winner of two of them.

Muller's CBS (framework) is built on different algorithms based on local search techniques (Muller, 2005, 2008, 2009). Then, we will talk about the problem-specific neighborhoods/moves created for this problem. The main algorithm this solver is based on is made up of four different phases as shown in *Figure 2.9*.

- Construction phase
- Hill climbing (HC) phase
- Altered great deluge (AGD) phase
- Simulated annealing (SA) phase (use of SA is optional where Muller (2008, 2009) had not used it for Exam Timetabling in Track 1 of ITC2007.

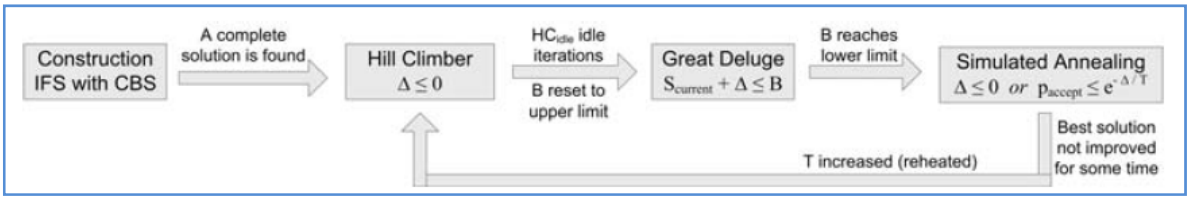

Figure 2.9: Schematic Diagram of CBS when SA is used

Construction Phase

In the construction phase, the IFS heuristic approach discussed previously (Muller, 2005) is used in order to produce a complete feasible solution while exploiting the CBS technique to avoid getting stuck in unstopping cycles (Muller *et al.*, 2004).

Hill Climbing (HC) Phase

After finding a complete feasible solution at the first phase, it will be the initial solution used at the second phase. During the second phase, HC algorithm (Talbi, 2009) is employed in order to discover local optimum (Muller, 2008, 2009).

Altered Great Deluge (AGD) Phase

The great deluge (aka Degraded Ceiling) algorithm was proposed by Dueck in 1993. It is a simulated annealing inspired heuristic algorithm and the purpose behind it was to enhance the speed of the search of SA algorithm without scarifying the solution quality. At the same time, it is differentiated from SA by its deterministic acceptance function of neighboring solutions. GD came from the concept of analogy with the direction a hill climber might take when he is looking for the highest point in a landscape (finding global optimum assuming maximization problem) to keep his feet dry while the rain keeps on increasing the level of water. *Figure 2.10* shows the great deluge algorithm with B as the upper bound in a minimization context (Talbi, 2009).

Input: Level *L*. /* *Operates somewhat like temperature in SA; Boundary Value; decreased monotonically during search* */
$s = s_0$ /* *Generate initial solution* */
Choose the rain speed *UP* /* *UP > 0; Only tuned parameter; decrement of B reduction* */
Choose the initial water level *B* /* *Initial Level B is equal to initial* $f(s_0)$ */
Repeat
  Generate a random neighbor s'
  If $|f(s')| < B$ *Then* $s = s'$ /* *Accept neighbor S' if f(s') < current boundary value* */
  $B = B - UP$ /* *update the water level* */
Until *Stopping criteria satisfied*
Output: Best solution found.

Figure 2.10: *GD* Algorithm Schema in Minimization Context

Muller (2008, 2009) used an altered great deluge (AGD) algorithm assuming a minimization problem where *B* is the bound set on the total value of the present solution. Altering GD will help the solver in widening the search and allow it to escape from a deep local minimum by oscillating of the upper bound *B*. *Figure 2.11* shows the schema of the AGD in minimization context and shows how AGD different from original GD. S_{best} is the total best solution value, GD_{ub} is a problem parameter, GD_{cr} is the cooling rate by which *B* is decreased, GD_{lb} is a given lower bound coefficient and *at* is a counter value that starts at one and increases by one each time *B* reaches its lowest limit and got augmented agan. *at* set to one again after the last best solution value is enhanced.

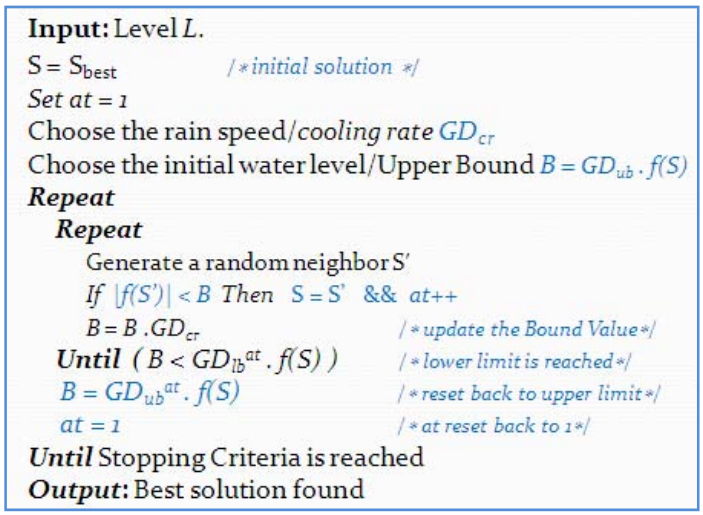

Figure 2.11: Altered Great Deluge Algorithm Schema in Minimization Context

Simulated Annealing (SA) Phase

The use of SA (Talbi, 2009) is optional in this algorithm for some problems. The control of the algorithm is given to the SA phase when the bound *B* of AGD reaches its lower limit. Similarly, the control is transmitted from *SA* phase back to HC phase directly after the reheat of system.

Problem Specific Neighborhoods (PSN) for CCTP

CCTP specific neighborhoods (PSNs) used by Muller during scheduling procedure are quite close to the ones we used (Chapter 4) with few changes. We have to always keep in our minds that the terms neighborhood and move are used interchangeably during this work but both refers to the same meaning.

*Table 2.*4 contains two different results sets for CCTP. The upper table part shows the best solutions Muller found within a certain time limit. For each input

instance, he made one hundred runs. On the other hand, the bottom part of the table shows the results produced by ten runs performed by the competition organizers for each solver. Muller's Solver won two out of the three tracks in the competition among them is CCTP.

Table 2.4: Top Table Shows Submitted results of Muller, Bottom Table shows the best recorded scores of finalists

Instance number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Room capacity	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Minimum working days	0	15	10	5	180	15	0	5	35	5	0	255	10	5
Curriculum compactness	0	28	62	30	114	26	14	34	68	4	0	76	56	48
Room stability	1	0	0	0	4	0	0	0	0	0	0	0	0	0
Overall value	5	43	72	35	298	41	14	39	103	9	0	331	66	53

Instance number	1	2	3	4	5	6	7	8	9	10	11
T. Müller	**5**	51	84	37	330	**48**	**20**	41	109	**16**	**0**
Z. Lu et al.	**5**	55	**71**	43	**309**	53	28	**49**	**105**	21	**0**
M. Atsuta et al.	**5**	**50**	82	**35**	312	69	42	**40**	110	27	**0**
M. Geiger	**5**	111	128	72	410	100	57	77	150	71	**0**
M. Clark et al.	10	111	119	72	426	130	**110**	83	139	85	3

Instance number	12	13	14	15	16	17	18	19	20	21	*rank*
T. Müller	333	66	59	84	**34**	**83**	83	**62**	**27**	**103**	12.9
Z. Lu et al.	343	73	**57**	**71**	39	**91**	69	65	47	106	16.7
M. Atsuta et al.	351	68	59	82	40	102	**68**	75	61	123	17.6
M. Geiger	442	622	90	128	81	124	**116**	107	88	174	38.2
M. Clark et al.	408	113	84	119	84	152	**110**	111	144	169	42.2

Previously, many of the proposed solutions to solve CCTP were problem specific. But, nowadays, there is a focus on creating general timetabling frameworks and solvers, like Muller's Constraint-Based Solver (2008, 2009), that can solve wide class of problems while having minimal problem specific modifications.

One of the major issues related to the competition was that the competition organizers limited the running time for some reasons such as removing one of the variability degrees from the scoring system. Therefore, more work should be done to study the effect of longer running times on the value of the produced solution and on the solver scalability as we are targeting in our work. Regarding the CBS implemented, more work should be done on tuning the problem specific parameters (neighborhoods such as creating new kinds of moves, tune parameters, …etc). Also, different combination of heuristics can be tested like the use of *Scatter Search* like the one we are working on.

2.3.2 A Hybrid Genetic Algorithm

In this paper, Massoodian and Esteki (2008) introduced a GA based approach for CCTP described in track3 of ITC2007 (*Chapter 3*).

The algorithm consists of two main stages, with a local search applied in every stage in different domains. The first stage solves completely the hard constraints violations (get feasible solution) while the second stage eliminates the soft constraints violations while keeping the feasibility of the solution by keeping the violations of hard constraints to zero. *Figure 2.12* shows the pseudo code for the hybrid GA used.

```
ReadData();
EncodeData();
sortCourses( CourseList );
Initialize();
for (All timetables)
      fitnessHARD(timetable);
Generations=1;
do{
      SelectBetterHalf();
      SelectOtherHalf();
      for (i=population_size/2; <population_size;i++)
      {
          Choose a random number a;
          if (a <= crossover rate)
              Crossover(parent1, parent2, parent1);
          if (feasible timetable has not found yet)
              fitnessHARD(i);
          else
              fitness(i);
      }
      if (max and average fitness of the population are
          very close)
          increase mutation rate;
      sortPool(pool);
      for(i=population_size/4; i<population_size;i++)
      {
          Choose a random number a;
          if (a<=mutation_rate)
                      mutation(i);
          if (feasible timetable has not found yet)
                      fitnessHARD(i);
          else
                      fitness(i);
      }
      if (max fitness of this gen > max fitness of the
          previous generation)
      {
          if (feasible timetable has not found yet)
                      Local search is applied to the best
          chromosome;
          else
                      Local search is applied to the
          columns of the best chromosome;
      }
}
while (time is not over && timetable with soft and
    hard constraints are satisfied is not found);
```

Figure 2.12: Pseudo-Code of the Hybrid GA

Encoding

The type of encoding used in this paper is permutation encoding where a Chromosome is represented as a sequence of integers starting from one up till the number of existing genes taking into consideration the assumption that there is not any lost or duplicated number (in other words all lecture are scheduled.

Initialization

First, the algorithm starts by generating an initial semi-random chromosomes population while taking care of not assigning a course lecture to a timeslot that is not allowed for it and try to keep satisfying this constraint in the subsequent generations. Then, the course list is sorted. Integer values that belong to courses with more constraints will be assigned first in the schedule.

Selection

At the selection stage, half the population with better quality will be sent directly to the subsequent generation while the remaining half is chosen by using a Tournament-*K* method, in which *K* chromosomes are randomly selected from the population, and it will copy to the subsequent generation the chromosome with the larger fitness value. This step is repeated until the required chromosomes number is copied into the subsequent generation.

Crossover

In this algorithm, the partially mapped crossover (PMX) technique is used. PMX works as follows: it takes two parents and breaks each one of them in two points. Then, it directly copies to the child the middle chunk between the two points from the first parent, while it copies the rest of the parts from the second parent such that it does not miss any integer number or overload with any extra value.

Mutation

The mutation operator works on exchanging randomly the values between each chromosomes pair at different randomly selected locations taking into account not to perform this swap of values unless the availability constraint will not be violated. In this approach, the mutation rate is not stable, and for each generation, a calculation of the maximum and the average fitness objective number values is performed. If those values were very close to each other, then the rate of mutation is augmented in order to avoid any possibility of premature convergence.

When local search was added to the algorithm, great enhancements were noticed. The first run of the algorithm that has no local search needed 786 seconds to come up with a feasible solution. On the other hand, the running of the program after adding local search and at the same seed took 99 seconds which is considered a great success. Therefore, the experiments showed that the addition of local search to optimize the value of fitness function enhanced the performance of the algorithm and speeded it up.

The fitness function used in this algorithm uses 0.999 coefficient for penalizing the hard constraint violations and a 0.001 coefficient for penalizing the soft constraint violations. Therefore, Massoodian and Esteki (2008) are still allowing many possible violations of soft constraint with this low coefficient. We believe this

number should be bigger still to have a much bigger limit on the violation of soft constraints.

2.3.3 Great Deluge Algorithm with Kempe Chain Neighborhood Structure

In this paper, Shaker and Abdullah (2009) present a hybrid algorithm of GDA and kempe-chain neighborhood (KCN) structure used together to solve the curriculum-based course timetabling problem, CCTP. The algorithms works in two different stages to solve the problem: First, to produce a feasible solution, a heuristic based on graph structure is employed. Second, the feasible timetable is carried out to the hybrid approach. In comparison with the results published through competition, this new suggested approach produced high expectations as well as promising output.

Neighborhood Structure

This algorithm uses the following neighborhood structures:

- Select a lecture randomly, and try to move it to a period that will decrease its penalty value while keeping its feasibility.
- Select randomly two lectures and try to swap their assignments.
- *KCN* - Select a course randomly as well as a random period. Then, use Kempe Chain by De Cesco, Di Gaspero and Schaerf (2008).

In the timetable, the KCN approach works on moving a subset of courses together to different selected timeslots whilst maintaining the hard constraints as shown in *Figure 2.13*.

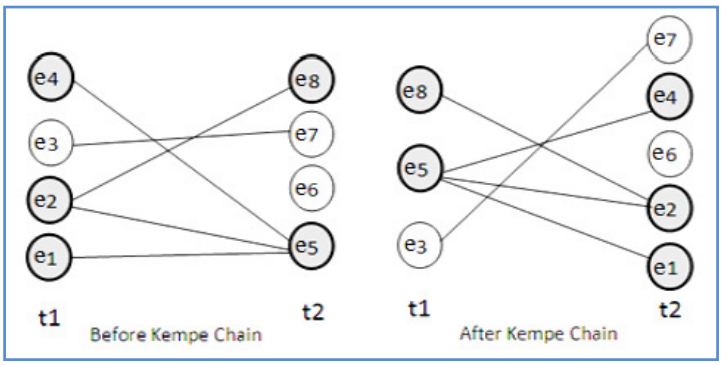

Figure 2.13: The Kempe Chain Move

Constructive Heuristic

This heuristics is based on a "large degree" heuristic starting from an empty timetable (Muller, 2008, 2009). Basically, the idea is to define a "degree" of a lecture *L* to be the number of lectures that *L* conflicts with. Then, the heuristic decreasingly

orders the lectures according to their degree value. Lectures with high degrees are hard to be scheduled and will be tackled first along with ignoring the soft constraints violations value, until the present timetable becomes feasible (no hard constraints violations).

Improvement Algorithm

Shaker and Abdullah's paper (2009) incorporates the GDA which begins with feasible initial solution fed to it from the constructive heuristic. Then, successive HC iterations are applied with the aim of reducing the penalties of soft constraints before allowing the acceptance of a new solution.

Preliminary experiments has been conducted and by comparing their outputs with those of other approaches, the initial comparisons shows that this approach is able to compete with other methods and produce better output values.

In many test cases, the produced solutions and the overall performance of the algorithm prove to be analogous to the performance of tabu search and mathematical programming approaches. Most of the work in this paper seems to be based on improving Muller's approach to CCTP by introducing a new type of neighbor, the kempe chain one. Although some of the results obtained in this approach improve on other existing approaches, the amount of improvement looks to be fairly minor, and on specific cases only. The comparisons were very preliminary and more experiments need to be done, as stated by the authors as part of their future work. Shaker and Abdullah (2009) think that if they introduced some intensification and diversifications techniques, this will give a great push to the algorithm to make more enhancements over the produced solutions' quality. Also, more future research should be focused towards testing this approach on the other ITC2007 available datasets.

2.3.4 A Repair-Based Heuristic Search

In 2008, Clark, Henz and Love's paper presented a software program called QuikFix used to solve timetabling problems in general, and applied on ITC2007 CCTP in specific. QuikFix applies a repair-based heuristic (RBH) to solve timetabling problems. Clark *et al.* presented a high level object-oriented model built specifically for timetabling problems that imposes certain structural constraints and important neighborhood moves in the problem domain of search. QuikFix allows the

generation of repair moves that will be selected and used by specific constraints in order to fix any of their violations.

Clark *et al.* did their tests on the data presented by ITC2007 competition specific for track 3 (CCTP). The only difference is that QuikFix Solver added the ability of assigning the teachers to the courses as well as courses lectures' to rooms and periods. Therefore, the solution in QuickFix is the set of assignments of courses' lectures to specific (Teacher *T*, Period *P*) and (Room *R*, Period *P*) tuples and two basic timetabling constraints for this problem are:

- Only one lecture is allowed to occupy a distinct (Room, Period) at a time.
- Each distinct (Teacher, Period) can only be assigned to one lecture at a time.

Constraints

Negligible distinction exists between hard and soft constraints in QuikFix. The distinction mainly occurs by giving hard constraints weight larger than the weight of the soft constraints.

Initial Assignment

The construction of the initial solution is done randomly by going over all the courses' lectures and assign each (course *C*, lecture *L*) pair to an available randomly selected (*R*, *P*) and (*T*, *P*). If all the (*T*, *P*) choices are all exhausted, virtual teachers and rooms are used.

Swap Moves

A move is made up of two independent changes, *TeacherPeriodSwapMove* and *RoomPeriodSwapMove*, each for a different tuple *(Room, Period)* or *(Teacher, Period).* The move strategies make up the core of the repair-based heuristic method where it selects a violated constraint lecture and generates a swap move to solve it after applying it conditionally, or those constraints will be used to guide the moves that are performed to try to solve other violated constraints.

Solver Main Structure and Heuristics

When no moves exists to enhance the best solution found so far value and it looks to be trapped, the solver uses the *Move Strategies* along with using a *climbprobability* constant that allows the tuning of the probability of selecting hill climbing (HC) as an escape method from local minima.

QuikFix Solver uses two main *tabu lists*: *list of recent moves* (help in escaping local minima) and *list of bad moves* (help avoiding the moves that decreases the solution quality and thus works as an optimization technique).

Not only this, the solver also uses Strategic Oscillation which is a mechanism used to escape from local minima by constraints weights modifications to explore feasible and infeasible boundaries.

At the end, Clark *et al.* used Rapid Restarts during which the solver starts with a randomly chosen assignment and apply previously mentioned repair-based search in order to reach a feasible region with the help of strategic variation techniques with different starting points from the current best solution found so far.

Table 2.5 shows the performance of running and testing QuikFix solver on the ITC2007 data sets. QuickFix was run for one hundred times, and the average values resulted from those runs has been calculated and they were based on the competition stopping criteria. "Feasible" column shows the feasible solutions percentage produced from the number of runs for each specific instance. The remaining three columns show the best, median, and worst solution values produced.

Table 2.5: Results of QuikFix Tests.

Instance	Iterations	% Feasible	Best	Median	Worst
comp01	6630000	100	9	16.0	92
comp02	2550000	100	103	139.0	183
comp03	2600000	100	101	137.5	187
comp04	3040000	100	55	84.0	107
comp05	1330000	58	370	510.5	762
comp06	2590000	100	112	146.0	187
comp07	2640000	100	97	142.0	203
comp08	3040000	100	72	95.5	128
comp09	2880000	100	132	150.0	171
comp10	2630000	100	74	107.0	152
comp11	7720000	100	1	5.0	9
comp12	1410000	100	393	452.5	555
comp13	3050000	100	97	124.0	141
comp14	2810000	100	87	109.0	129

The performance of QuikFix, after testing it on the ITC2007 competition track3 (CCTP) as well as on a class scheduling of K-12 international school in Singapore, showed that the results are very good on those problems. This can pave the way of using repair-based heuristic on timetable scheduling. The major advantage in this algorithm is that it allows the exploration and evaluation of smaller number of

neighbors. Another advantage of QuikFix is that it allows the assignment of Teachers to Courses not only Courses to Rooms and Periods. However, the main disadvantage here is that it allows the assignment *(Teacher, Period)* to have a different period than *(Room, Period)* to the same *(Course, Lecture number)* which increases the number of violations of hard constraints by causing many conflicts. Not only this, but also the use of virtual *(Teacher, Period)* combinations lead to the use of non-real data that need to be explored and fixed on later stages. Thus, it could be a possible future work to test the applicability of producing feasible initial solutions to be used at the following iterations and try to assign to *(Course, Lecture Number)* the value *(Teacher, Room, Period)*. On the other hand, the object-oriented design (OOD) of QuikFix solver proved that it is easy to come up with a simple high-structured model for the timetabling problem that can be customized to adapt the needs and requirements of the problem.

2.3.5 A Round Robin Strategy over Multi-Algorithms

In 2011, Abdullah, K. Shaker and H. Shaker's paper studied again the university course timetabling problem and they investigated the use of a round-robin (RR) approach as a control method over three chosen algorithms that need to be applied, GD, SA and HC, so that it can chose which algorithm to be used at the current stage. Abdullah *et al.* did their tests on two benchmark datasets presented by ITC2007 competition specific for track 2 (ECTP) and for track 3 (CCTP) and they compared their results with a set of methods coming from the literature.

Abdullah *et al.* proposed an algorithm composed of two stages, constructive phase in which the initial solution will be constructed, and the improvement stage in which the algorithm will try to minimize the number of soft constraints violations.

Constructive Phase

For CCTP, the constructive algorithm starts with an empty solution and starts with scheduling the lectures with the largest possible conflicts number (highest number of hard constraints) and they are randomly scheduled on different periods and rooms without taking into consideration the soft constraints violations. For example, if some of them cannot be set to an exact room, they will be scheduled in any other randomly chosen room. When a feasible solution is found, the algorithm stops; otherwise, some neighborhood moves are applied.

Improvement Algorithm

HC, GD and SA are ordered in sequence where their application is controlled using the Round Robin approach. A time quantum is given to each of the algorithms in equal values in a circular order and it is set to fifteen mins for medium and large datasets, while it is set to ten seconds for small datasets. *Figure 2.14* shows the pseudo code of the improvement algorithm.

```
Set the initial solution Sol by employing a constructive
heuristic;
Calculate initial cost function f(Sol);
Set best solution Solbest ← Sol;
Set quantum time, q_time;
Set initial value to counter_qtime;

 do while (not termination criteria)
      Set a sequence algorithms in a queue which is ordered
as
      ALGi where i ∈ {1,...,K} and K = 3;
       do while (q_time not met )
       Select an algorithm ALGi in the queue where i ∈
{1,...,K};
  A:  Apply ALGi on current solution, Sol to generate new
       solution, Sol*;
       if there is an improvement on the quality of the solution
then
          update Solbest, Sol;
          repeat label A
       else
          reset ALGi parameters;
          insert ALGi into the queue;
```

Figure 2.14: Improvement Method Pseudo-code

Neighborhood Moves

Abdullah *et al.* used two neighborhood moves in their approach:

- N1: a move that chooses randomly a lecture and try to move it to a feasible timeslot that can generate the lowest soft constraints penalty cost.
- N2: select two lectures assigned at the same room and try to swap only their timeslots and check if this move decreases the soft constraints penalty.

Experiments were done on course timetabling problems' track 2 and track 3 datasets from ITC2007. The preliminary tests showed that Abdullah's *et al.* approach has a high competition in comparison with other methods exist in the literature and was able to give the best test results on *comp05, comp21, DDS2, DDS3, DDS4, DDS5, DDS6* datasets as shown in *Table 2.6*. However, those results are not

comprehensive enough because they do not show a complete and sufficient probabilistic description of the results and they are only showing the best run result coming from Abdulla's *et al.* approach and from even all the existing approaches in the literature.

Table 2.6: The Best Run Result of Abdulla's *et al.* Approach in Comparison with Other Approaches.

Dataset	Our method	M1	M2	M3	M4	M5	M6	M7	M8
Comp01	5*	5*	13	5*	5*	9	5*	5*	5*
Comp02	43*	43*	43	75	34	103	108	50	60
Comp03	77	72	76	93	70*	101	115	82	81
Comp04	38	35*	38	45	38	55	67	35	39
Comp05	311*	298	314	326	298	370	408	312	321
Comp06	44	41*	41	62	47	112	94	69	45
Comp07	19	14*	19	38	19	97	56	42	21
Comp08	44	39*	43	50	43	72	75	40	41
Comp09	108	103*	102	119	99	132	153	110	102
Comp10	13	16	14	27	16	74	66	9	17
Comp11	0	0	0	0	0	1	0*	0*	0*
Comp12	339	331	405	358	320*	393	430	351	349
Comp13	69	66	68	77	65	97	101	68	73
Comp14	60	53	54	59	52*	87	88	59	59
Comp15	76	84	-	87	69*	119	128	82	82
Comp16	48	34	-	47	38	84	81	40	49
Comp17	91	83	-	86	80*	152	124	102	81
Comp18	84	83	-	71	67*	110	116	68	79
Comp19	71	62	-	74	59	111	107	75	67
Comp20	42	27	-	54	35	144	88	61	30
Comp21	103*	103	-	117	105	169	174	123	110
Dds1	143	-	132*	1024	-	-	-	-	158
Dds2	0	-	0*	0	-	-	-	-	0*
Dds3	0	-	0*	0	-	-	-	-	0*
Dds4	24	-	68	233	-	-	-	-	28
Dds5	0	-	0*	0	-	-	-	-	0*
Dds6	4	-	4*	11	-	-	-	-	4*
Dds7	0	-	0*	0	-	-	-	-	0*
Test1	227*	-	-	234	-	-	-	-	-
Test2	16*	-	-	17	-	-	-	-	-
Test3	83*	-	-	86	-	-	-	-	-
Test4	89*	-	-	132	-	-	-	-	-

M1: A constraint-based solver by Müller (2009); M2: Integer programming by Lach and Lübbecke (2010); M3: The dynamic tabu search by Cesco et al. (2008); M4: Adaptive tabu search by Lü and Hao (2010); M5: A repair-based timetable solver by Clark et al. (2008); M6: Threshold accepting met-heuristic by Geiger (2010); M7: Incorporating tabu search and iterated local search by Atsuta et al. (2008); M8: Great Deluge approach with Kempe Chain by Shaker and Abdullah (2009); *The best results.

CHAPTER 3

CURRICULUM-BASED COURSE TIMETABLING PROBLEM

3.1 Problem Description and Main Entities

Curriculum-based course timetabling problem (*CCTP*), as described by Di Gaspero, McCollum and Schaerf (2007) and as university timetabling, is a combinatorial optimization problem defined as the weekly assignment of courses of the university to a known set of rooms and time periods consisting of days and slots, while satisfying the problem hard and soft constraints sets. CCTP constraints are mainly based on conflicts that occur between different courses belonging to the same or to different curricula and these are issued by the university. Therefore, CCTP is not based on enrollment issues.

Although CCTP formulation is somewhat simplified with respect to real world course timetabling problems in order to reach a certain generality level. This description, in general, is based on the timetabling strategies followed by the University of Udine in Italy and followed by many other Italian and European international universities. The following entities build the CCTP as stated by Di Gaspero *et al.*:

- **Timeslots, Days and Periods**: We have a number of *working days* during a single week and usually it is either five or six for any educational institution. Each working day is broken down into a fixed number of equal duration *timeslots*. The combination of day and a timeslot together makes up a *period*. The overall number of periods available to be assigned to different lectures came from the multiplication of the total working days number by the number of timeslots per day (*DxTD*).
- **Teachers and Courses**: Every course in the timetabling problem has a predetermined number of lectures that need to be assigned to different periods. A certain number of students are going to register for and attend the course's lectures and each course should have one teacher to teach it. Also, each course has a minimum number of days on which its lectures' assignments should be spread. On the other hand, there are

also lists of unavailable periods in which the assignments of the course's lectures will not be allowed (unavailability constraints).

- **Rooms**: We have a predetermined capacity of the existing number of seats available in every room. Rooms with enough capacity (if large enough to hold the number of students of a specific course) are equally suitable.
- **Curricula**: A curricula is composed of more than one curriculum set by the university. A curriculum consists of a set of courses in such a way that any couple of courses belonging to this curriculum has common students. Some soft and hard constraints are based on curricula as we will see in the next constraints section.

3.2 Problem Constraints

The final problem solution represents the assignment or the mapping of every lecture, belonging to any course, to an available period (day and timeslot) and available room keeping in mind the satisfying of the available constraints set. The set of CCTP constraints as described as Di Gaspero *et al.* are as follows:

3.2.1 Hard constraints

- **Lectures:** All lectures of a course must be assigned to different periods. If a lecture is not scheduled, a violation for this hard constraint takes place.
- **Room Occupancy**: it is not allowed to schedule two different lectures in same room and at same period. Therefore, "one violation" occurs if any two lectures take place in same room at same period. Any additional lecture scheduled in the same assignment value (period and room) will count as one more extra violation.
- **Conflicts**: Courses' lectures that belong to same curriculum or given by same teacher must be all assigned to distinct periods. If any two lectures belonging to same curriculum or given by same teacher got assigned to the same period timeslot, they conflict and represent "one violation". If three lectures are assigned at same period and they conflict, the change in OF value will be increased by three extra violations (one penalty point or violation for each pair).

- **Availabilities**: Each lecture scheduled at a period not available for its course is counted as one more extra violation.

3.2.2 Soft Constraints

- **Room Capacity**: For every course, the available seats number (capacity) of all the rooms that hosts course's lectures must be greater than or equal to the students number registered in the course; thus, each extra student attending the course and above the course's capacity counts one extra penalty point.
- **Minimum Working Days**: For each course, its lectures must be distributed over a given number of minimum working days number and each day less will be counted as five extra penalty points.
- **Curriculum Compactness**: For each curriculum, the lectures that belong to it should be scheduled consecutively to each other (i.e., in adjacent periods). When any lecture for a given curriculum is not adjacent to any other lectures along the same day (i.e. when a lecture is isolated), this is considered as a violation for this constraint and it will count as two extra penalty points.
- **Room Stability**: All lectures belonging to the same course should be taught in the same suitable room. If not all the course's lectures can be assigned to the same room, then each room different from the first one will count as one extra penalty point.

3.3 Description of Instances and File Formats

According to Di Gaspero *et al.*, the above explained problem description contains many of the characteristics that exists in some Italian and European universities and based on that specific problem modeling proposed by ITC2007 competition, the input instances used for testing purposes are real data from Udine University. Twenty one instances were available for the competitors distributed as seven instances for each set among three main sets (early, late, and hidden). For each instance, there exists at least one feasible solution (i.e. a solution that does not have any hard constraints violations but is not necessarily optimal with respect to the soft constraints violations).

To represent some situations in which the timeslots number is not the same among all week days, for example, no classes should be taught on Saturday

afternoon, then some periods (like Saturday afternoon timeslots) are going to be unavailable for all courses. In addition to that, no two curricula are allowed to have exactly the same set of courses.

3.3.1 Input Structure

Di Gaspero *et al.* explained the format of input files and said that each input file holds one instance and they are named as comp01.ctt, comp02.ctt, ..., comp21.ctt. Each file has two main sections: the header and the content section. The header section provides some scalar information to the problem, while the content section is made up of four main parts which represents the arrays for each specific entity of the problem like courses, rooms, curricula, and finally constraints. *Figure 3.1* shows the data input format while the header and the other sections should be provided in their precise order as shown in *Figure 3.2* shows a sample data input.

```
Courses:   <CourseID> <Teacher> <# Lectures> <MinWorkingDays> <# Students>
Rooms:     <RoomID> <Capacity>
Curricula: <CurriculumID> <# Courses> <MemberID> ... <MemberID>
Unavailability_Constraints: <CourseID> <Day> <Day_Period>
```

Figure 3.1: Input Format

```
Name: ExampleProblem
Courses: 4
Rooms: 2
Days: 5
Periods_per_day: 4
Curricula: 2
Constraints: 8

COURSES:
SceCosC Ocra 3 3 30
ArcTec Indaco 3 2 42
TecCos Rosa 5 4 40
Geotec Scarlatti 5 4 18

ROOMS:
A 32
B 50

CURRICULA:
Cur1 3 SceCosC ArcTec TecCos
Cur2 2 TecCos Geotec

UNAVAILABILITY_CONSTRAINTS:
TecCos 2 0
TecCos 2 1
TecCos 3 2
TecCos 3 3
ArcTec 4 0
ArcTec 4 1
ArcTec 4 2
ArcTec 4 3
END.
```

Figure 3.2: Sample Input Data

As suggested above, the unavailability constraint, as shown in line “ArcTec 4 0” as example, says that course ArcTec cannot be scheduled at day 4 and timeslot 0.

3.3.2 Output Structure

Di Gaspero *et al.* talked also about the output and said that it should be produced in a file in which each single line represents one assignment of a lecture to a selected room and period. *Figure 3.3* shows the output lines format and we have a solution for the *ExampleProblem* we talked about in the previous section.

```
<CourseID> <RoomID> <Day> <Day_Period>
```

```
SceCosC B 3 0
SceCosC A 3 1
SceCosC A 4 0
ArcTec B 0 1
ArcTec B 1 1
ArcTec B 1 2
TecCos B 0 0
TecCos A 0 1
TecCos B 2 2
TecCos B 4 2
TecCos B 4 3
Geotec A 2 2
Geotec A 2 3
Geotec B 3 0
Geotec A 3 1
Geotec A 4 2
```

Figure 3.3: Output Line Format and Sample Solution for *ExampleProblem* in Figure 16

3.4 Standard Solution Validation

The solution validator C++ source code was provided by competition organizers and it takes input file and output file to produce the overall solution evaluation with complete descriptions for all the hard and soft constraints violations. This *validator* was used also by the organizers to evaluate the competitors' solutions and prepare the finalists. We used it to evaluate our work's results in order to compare them with those of Muller. *Figure 3.4* shows a sample of the validator's output messages and results that represent the violations of hard constraints, and the cost of soft constraints violations as well as the total cost and violations as shown by Di Gaspero *et al.* paper.

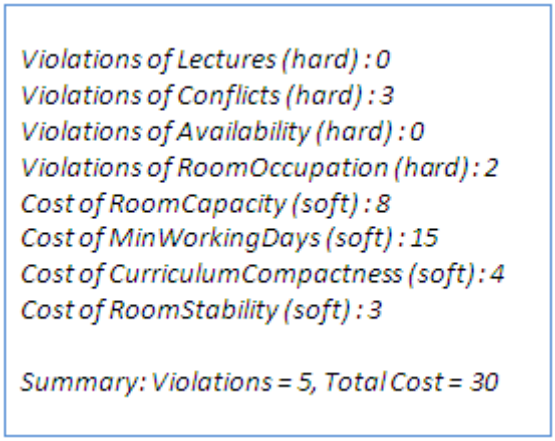

Figure 3.4: Validator's Result for the *ExampleProblem*

3.5 Useful Problem Specific Notations and Objective Function Formulation

In this section, we will describe some of useful symbols and notations used in our problem modeling and we will use them in the formulation of our objective function (Energy function).

3.5.1 Problem Specific Notations and Counters

- Let *Solution* represents a single solution.
- Let *Course* represents a single course object.
- Let *Curriculum* represents a single curriculum object.
- Let *Lecture* represents a single variable object.
- Let *PeriodRoomAssignment* represents a single value object.
- Let *Room* represents a single room object.
- Let *Teacher* represents a single teacher object.
- Let *CCTPModel* represents the problem model solution.
- Let *WorkingDaysNumber* represents the problem's available working days.
- Let *SlotsPerDay* represents the number of timeslots per day.
- Let *RoomsNum* represents the number of Rooms of the problem.
- Let *TotalPeriodsNum* = *WorkingDaysNumber* * *SlotsPerDay* represents overall set of Periods for the whole problem.
- Let *ConstraintsNum* represents the number of Availability constraints.
- Let *CoursesNum* represents the number of courses.
- Let *LecturesNum* represents the total number of lectures for all courses.

- Let *CurriculaNum* represents the number of university curricula.
- Let *CourseCapacity* represents the number of students per course.
- Let *RoomCapacity* represents the capacity of each room.
- Let $MinWorkingDaysNum_i$ represents the minimum number of days on which the lectures of $course_i$ should be scheduled.
- Let *AssignedDays* represents the number of working days each course is spread on.
- Let *NrAssignedVariables* represents the number of assigned lectures/variables per solution.
- Let *OFCalls* represents the number of OF calls during the execution.

Our algorithm starts by initializing the counters and the parameters of CCTP by reading from the input file the required information like the number of rooms *RoomsNum*, the number of courses *CoursesNum*, the number of university working days *WorkingDaysNumber,* the number of timeslots per day *SlotsPerDay,* the number of university curricula *CurriculaNum*, the number of availability constraints *ConstraintsNum* as well as it initializes the total number of lectures (variables to be assigned) for all courses *LecturesNum*.

3.5.2 Problem Specific Hard Constraints Parameters

- Let S_{UL} represents the total number of unassigned lectures per solution (hard constraint violation of *Lectures*);
- Let S_{ROC} represents the total number of lectures (decremented by 1) assigned to rooms with conflicts (hard constraint violation of *Room Occupation*);
- Let S_{TC} represents the total number of lectures assigned with teacher conflict (hard constraint). Let φ represents the weight related to the significance of S_{TC} in OF and let φ=50;
- Let S_{CC} represents the total number of lectures assigned with curriculum conflict (hard constraint). Let ψ represents the weight related to the significance of S_{CC} in *OF* and let ψ =1;
- Let S_{Conf} is equal to $\frac{((S_{TC}+S_{CC})^2-(S_{TC}+S_{CC}))}{2}$ and it represents the number of counted violations for the *Conflicts* hard constraint. This is used at the last

stage when we want to compare our results with those coming from ITC2007 solution validator;

- Let S_{Avail} represents the total number of Lectures scheduled at a period not available for it (Availability constraint violation). Let ω represents the weight related to the significance of S_{Avail} in OF and let ω =1.

3.5.3 Problem Specific Soft Constraints Parameters

- Let S_{ExtStd} represents the total number of extra students attending a course lecture and are above the course's capacity. This represents the penalty/cost of violating the Room Capacity soft constraint (γ=1).
- Let S_{MWD} represents the difference between the *MinNumDays and AssignedDays*. If *MinNumDays - AssignedDays > 0*, then a α penalty points is given for each day below the min working days; In our problem, α=5.
- Let $S_{CurrComp}$ represents the total number of isolated lectures in the all curricula. For each isolated lecture, we count β penalty of points for Curriculum Compactness; In our problem, β=2.
- Let $S_{RoomStab}$ represents the total number of rooms assigned to course's lectures and different from the first room assigned to each course. This forms the penalty/cost of violating Room Stability soft constraint (μ=1).

3.5.4 Objective Function

The summation of the hard constraints number of violations along with the cost of the soft constraints number of violations multiplied by the weights associated with each one of those constraints will result in the problem specific objective function that will be used during the different stages of our approach. Therefore, our objective function is as follows:

$$OF = Total\ Violations + Total\ Cost$$

$$Total\ Violations = \varphi * S_{TC} + \psi * S_{CC} + \omega * S_{Avail}$$

where φ=50, ψ=1, and ω=1.

$$Total\ Cost = \gamma * S_{ExtStd} + \alpha * S_{MWD} + \beta * S_{CurrComp} + \mu * S_{RoomStab}$$

where γ =1, α=5, β=2, and μ=1.

In our approach, we first aim to have a feasible solution in which all the hard constraints should be satisfied and its *Total Violations* should be equal to zero. Then, we start minimizing the *Total Cost* of our solution (minimizing soft constraints violations). You can notice that in our *Total violations* (Hard constraints violations) S_{uvc} and S_{roc} are not included because they will be satisfied during the generation of our first initial solution where we are not allowed to continue unless all the lectures are assigned to different periods and rooms while satisfying all the possible rooms' conflicts. Also, in our hard constraints Total Violation function, more weight is given to the teachers' conflicts S_{tc} because it is the most important violation among the remaining violations even over the time availability limitation for the following:

- If a time availability violation occurred and a lecture was scheduled in a period not allowed for it, this can be manageable in some worst case scenarios and some exceptions can be made for it.
- Also, if a curriculum conflict occurred where two lectures belonging to the same curriculum got scheduled at the same period (timeslot of the day), the students (in worst-case scenario) can postpone one of the two courses to the next semester (term).
- However, if any of the lectures of a course was given in a time in which the teacher will not be able to give at the chosen time, usually this conflict will make the university either obliges the teacher to teach that course at that specific time or it will get another instructor for the course as a worst-case scenario. Therefore, the teacher conflict is a problem that cannot be solved or the teacher will have to quit teaching one of the courses and sometimes more than one course which is really not affordable (especially for fulltime faculty).

During our approach's execution, our algorithm should keep track of the number of objective function (energy evaluation) calls *OFCalls,* the number of assigned variables (lectures) *NrAssignedVariables,* the capacity of each course *CourseCapacity*, the capacity of each room *RoomCapacity*, the minimum working days for each course *MinNumDays*, and the number of days assigned to each course *AssignedDays*; so that we can easily follow the objective function value of each solution in our approach as well as keep track of the solution conflicts' detection.

CHAPTER 4

THREE-PHASE APPROACH DESIGN FOR SOLVING CURRICULUM-BASED COURSE TIMETABLING

4.1 Overview of the Three-Phase Heuristics Algorithm

In this chapter, we propose to study the effectiveness of implementing a three-phase algorithmic approach employing simulated annealing (SA), scatter search (SS) and a final tuning heuristic (THEU) to solve CCTP given its constraints and restrictions from ITC2007 (Di Gaspero *et al.*, 2007). We designed and adapted the implementation of each of the three algorithms in a way that takes into consideration the hard and soft constraints of CCTP that needs to be satisfied along with their assigned violation's weights.

We are not only trying to have competitive results and better feasible solutions than those produced by the hybrid technique that Muller (2008, 2009) used to solve the ITC2007 three main tracks (Di Gaspero *et al.*, 2007), but we are also aiming to produce more scalable solutions than those of Muller.

We start our approach by constructing the first initial empty solution according to the structure and representation explained in *section 4.2* and we then produces a semi-random infeasible solution in which only the room conflicts and time availability hard constraints are satisfied. After that, this solution became the initial solution with which we will start the first phase of our approach. In the first phase (*section 4.3*), we aim to produce a feasible solution in which all the hard constraints are satisfied using SA algorithm and the perturbation move while ignoring completely the soft constraints violations. SA is used at this stage since it has been previously proven in literature that SA is very effective in producing scalable solutions for large datasets, has a simple design choices and it is even faster than Genetic Algorithm (Mansour and Timani, 2007). Then, the best solution produced at phase one will be utilized to generate the hundred initial feasible solutions used to form the initial population of SS algorithm that is used along with different problem specific neighborhoods/moves created specifically for this problem (Muller, 2008, 2009) to optimize the soft constraints violations. This formed our algorithm's second phase discussed in *section 4.4*. At the last phase of our approach, the best solution we get from *SS* will be passed to a local search heuristic developed to be the final

possible tuning step for the soft constraints of our final solution (discussed in *section 4.5*).

Actually, the second phase of the algorithm represents the vital heart of our approach where the most important and the most effective improvements in the soft constraints' violations take place through the use of an adapted implementation of *SS*. The first question that might come to your minds is that why we chose to work on the implementation of SS and not on any of the other techniques we previously discussed and talked about in *Chapter 2*. First of all, Scatter search is an interesting population-based evolutionary algorithm that proved to achieve interesting results for different optimization, constraint-based, linear, and non-linear problems as mentioned in *Appendix I,* and according to our knowledge, there is no published work related to using Scatter Search to any course timetabling problem. Not only this, but we also chose to implement an approach with different structural phases because it helps in developing a flexible algorithm in which each phase is used to solve specific problem. We were also careful while choosing which algorithm to be used at each phase because we were targeting to guarantee the scalability of the solution produced and the scalability of the algorithm itself over large-scale real-life problems. Therefore, in the following sections, we will start talking and describing in details the different adapted algorithms used in our approach.

4.2 Solution Representation and Initial Solution Construction

4.2.1 Solution Representation

Throughout the implementation of our algorithmic approach (using Java), the basic data structure used to represent our solution is a vector *Lectures(L)* where *L* is the total number of courses' lectures objects initialized when *CCTPModel* object read the required information from the input file. The element Lectures(i) represents the lecture that will be allocated to a chosen *PeriodRoomAssignment (P,R)* object where *P* is the selected period allocated to lecture *i* and it ranges from {1,…, *TotalPeriodsNum*}; while *R* corresponds to the room from {1,…, *RoomsNum*} in which lecture *i* will be taught. *Figure 4.1* shows the structure of our solution and it shows for each lecture *i*, we have to keep track of the course of the lecture, the teacher giving the course, the curriculum to which the course belongs, the availability of the selected period for the lecture, the number of minimum working days that should be met for the course's lecture, and the chosen room capacity.

The algorithm starts by creating an empty initial solution, and it copies the essential parameter values from the main problem model as well as copy the list of *Courses*, list of *Teachers*, list of *Curriculas* and list of *Rooms* into initial solution. Then, the model will compute the whole domain of assignments (search space) for the initial solution by applying the Cartesian Product (*PeriodsXRooms*) where *Periods* is from {1,…,*TotalPeriodsNum*}, and *Rooms* from {1,…, *RoomsNum*}.

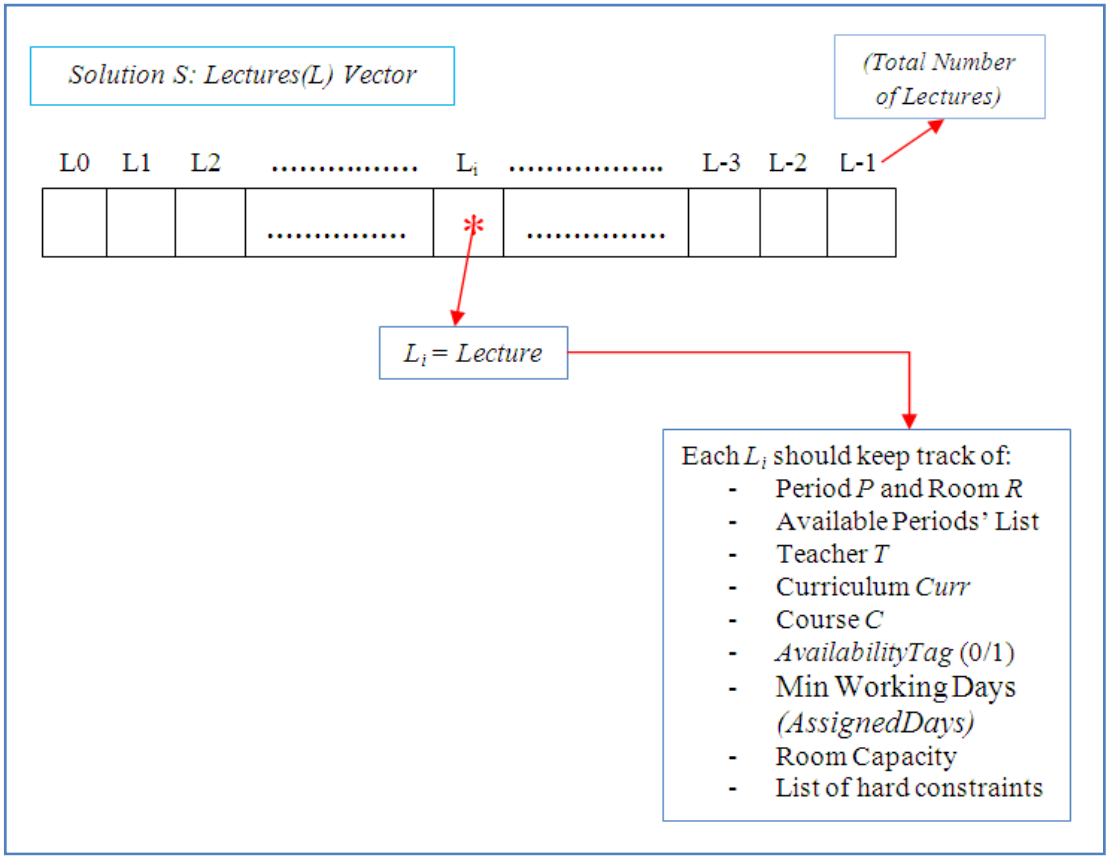

Figure 4.1: Solution Representation

4.2.2 Initial Solution Construction

The algorithm starts by generating a semi-random initial solution using controlled randomization technique by, first, trying to assign the list of unassigned lectures having time/availability constraints where we guarantee their locations in the schedule where we select such a lecture *i* and we select a *(Pi,Ri)* from the domain *(PeriodsXRooms)* that respects its time limitations. After that, *(Pi,Ri)* is removed from the search space so that it will not be selected again. At this stage, if a suitable assignment cannot be found for one of the lectures, a swap of assignments between lectures might be needed (or new assignment might need to be selected). What are left will be the lectures with no such time constraints and they can be scheduled

randomly from the search space with no worries. *Figure 4.2* shows the pseudo code for the generation of the initial solution.

1. Create New Empty Solution ***InitialSolution***
2. Copy the Model Data **Model.copyModelData(InitialSolution)**
3. Create Initial Solution Domain (Periods x Rooms)
   InitialSolution.computeSolutionDomainAssignments()
4. **for** (each Lecture ***L*** …. *unassignedLecturesWithTimeConstraint*) **do**
   a. Choose a lecture to assign
   b. **for** (each Assignment ***A*** … *AssignmentsDomain*) **do**
      i. Choose assignment ***A*** randomly
      ii. **if** (assignment is not available for this lecture) **then**
          ignore it and select another
          else if (period of assignment ∈ to another lecture ∈ to same course) **then** ignore it
          else Assign ***A*** to ***L*** and Remove ***A*** from the Solution Domain
          end if;
          end if;
      end for;
   c. **if** (no suitable Assignment ***A*** found to Lecture ***L***) **then**
      i. find a swap for it from the list of assigned lectures and we might unassign one of them
      end if;

   end for;
5. **for** (each Lecture ***L*** …. *unassignedLecturesWithNOTimeConstraint*) **do**
   a. Choose a lecture to assign
   b. Choose assignment ***A*** randomly
   c. **if** (assignment is not available for this lecture) **then**
      ignore it and select another one
      else
      Assign ***A*** to ***L*** and Remove ***A*** from the Solution Domain
      end if;

   end for;

Figure 4.2: Semi-Random Initial Solution Generation Using Controlled Randomization Technique

In the initial generated semi-random solution, we will be left with the teachers' conflicts and curriculum conflicts which will remain unsolved along with all the soft constraints violations. On the other hand, this solution will have the following hard constraints violations respected and solved:

- All the lectures of the solution are assigned;
- No room conflict is going to be there because we created a vector of *(PxR)* combinations (values) and once a value is chosen from it, it is removed from the domain, so that, it will not be selected again;
- The time/availability constraints are respected for all the assigned lectures.

4.3 Phase One: Simulated Annealing Algorithm

4.3.1 Simulated Annealing Algorithm and Energy Function

Originally, the idea behind the foundation of simulated annealing, SA, stands on ideas came from physics and it resembles the idea of the solid physical annealing (Kirkpatrick, Gelatt and Vecchi, 1983). But why such a physical problem would be even of an interest? To know the answer, understand the mechanism of coercing a solid into a low energy-state where the solid became in a highly ordered state. To achieve this, the material is heated (annealed) and then cooled very slowly and very carefully coming into a thermal equilibrium at each state (temperature level) until it freezes and reaches its good state. Similarly, simulated annealing, at each stage (temperature level), uses the Metropolis algorithm that simulates the behavior of the controlled cooling operations to solve any possible optimization problem and gives a solution that is taken from a bad, unsolved, and unordered state into a highly optimized, good and desirable state.

In other words, the SA algorithm behaves like the natural phenomenon by searching and exploring (perturbing) the solution search space (energy landscape) looking for optimizing the energy function (cost/objective function) of the problem. SA starts working on an initial solution by setting an initial high (artificial) temperature and, during its work, the temperature is reduced gradually until it reaches the freezing point (Mansour and Timani, 2007).

Therefore, the initial semi-random solution, that we ended up with after performing the constructive generation step using controlled randomization, will be the initial solution our adapted SA algorithm will start with in order to fix the hard conflicts only while ignoring completely the soft constraints. In our approach, SA is used to solve the teacher conflicts (TC), the curriculum conflicts (CC) as well as the time availability conflicts which we allow their violation again at this step in order not to restrict or limit the search space for the correction of TC and CC. The main

reason because we are targeting to have a solution that will be more scalable than the one Muller (2008, 2009) got using Iterative Forward Search and not only to get a feasible solution for small problems. Our algorithm will work on large problems as well as on small problems; unlike Muller's work in which he is simply doing an exhaustive search that will not work for large problems (2008, 2009). *Figure 4.3* shows our simulated annealing algorithm that is adapted to solve CCTP. Let energy function for SA is:

$$OF{=}E = \varphi * S_{TC} + \psi * S_{CC} + \omega * S_{Avail}$$

where φ=50, ψ=1, and ω=1.

1. Calculate the initial temperature T_0 and set $T_i = T_0$;
2. Calculate the freezing temperature T_f ;
3. Set ***bestSolution = initialsolution***;
4. Set ***currentSolution*** = null;
5. **while** ($T_i > T_f$ and not converged) **do**
   a. Save *bestSolution* in *currentSolution*
   b. **repeat** (10**LecturesNum* WorkingDaysNumber * SlotsPerDay*) times
      i. Set energy value of *currentSolution* in $E_{current}$
      ii. Perturb();
      iii. Save solution in *perturbedSolution* having energy value $E_{perturbed}$
      iv. Calculate $\Delta OF = E_{perturbed} - E_{current}$
      v. **if** (ΔOF < 0) **then**
         update() & *currentSolution= perturbedSolution*; /* accept */
         else if (0 < randomNum() < $e^{-\Delta OF/Ti}$) **then**
         update() & *currentSolution= perturbedSolution*; /* accept */
         else
         reject_purturbation();
      end repeat
   c. save_best_solution_found_sofar() in *bestSolution*;
   d. $T_i = \theta * T_i$;
   end while
6. Final solution = best solution found so far;

Metropolis Algorithm

Figure 4.3: SA Algorithm for solving hard constraints violations in CCTP

4.3.2 The Metropolis Algorithm and the Perturbation Function

At each iteration of the Metropolis step, the function *perturb()* is called. The perturbation method is performed by selecting two lectures randomly from the

current solution and it swaps their content on condition that the selected lectures have different periods' values and they belong to different courses as shown in *Figure 4.4* because the teacher conflicts and the curriculum conflicts are affected by the periods room and not by the room and if we swapped two lectures with the same period, this will not affect he hard constraints penalties; Swap (P_1, R_1) of lecture L_1 with (P_2, R_2) of lecture L_2. We accept this swap only if it does not violate the room hard constraints and leave no lecture unassigned. We allowed the violation of time availability hard constraint again as we previously mentioned.

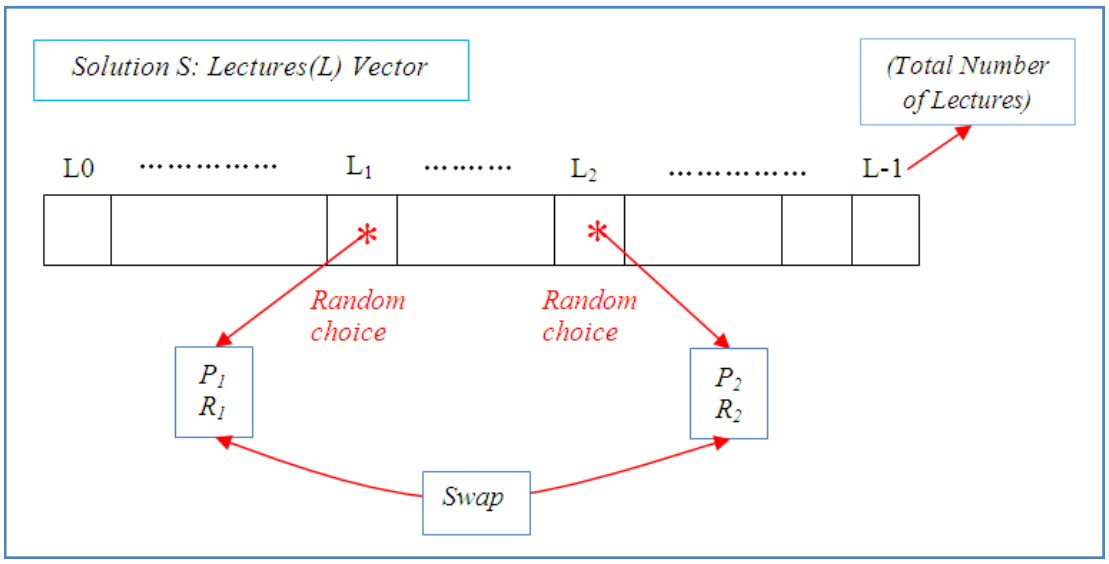

Figure 4.4: Perturbation Operation of our Simulated Annealing algorithm

Once the *perturb()* method is called, the change in OF (Energy) value is calculated and the acceptance criterion is checked. If the new change decreased the OF value (the hard constraints violations), then the perturbation move is accepted and the current solution is set to the newly perturbed solution. On the other hand, if the move lead to the increase of ΔOF, it will be accepted only with am probability $e^{-\frac{\Delta OF}{Ti}}$. This will lead to accepting a number of uphill moves helping the system in escaping any premature convergence into a bad local minimum-energy state (the advantage of using Monte-Carlo algorithm). As temperature T_i goes smaller, the probability of accepting uphill moves will go less as well and at near-freezing temperatures, uphill moves will not be accepted any more (Mansour and Timani, 2007). This procedure is repeated 10**LecturesNum***WorkingDaysNumber** *SlotsPerDay* times for each change in temperature and after it a thermal equilibrium is believed to be reached. Our perturbation method accounts only for solving the

infeasibility problem of our first solution and after this phase, the output will be a feasible solution.

4.3.3 The Cooling Schedule

Let T_0 to be the initial temperature that leads to a high initial probability for accepting uphill moves which is equal to 0.93 (Mansour and Timani, 2007). On the other hand, the freezing temperature, T_f, is the point at which the probability of accepting those moves become very small (2^{-30}) and almost make them impossible to be reached and it allows only downhill moves; $T_f = \frac{-0.1}{\log 2^{-30}}$. In this work, the cooling schedule to be used is: T(i+1)= θ*T(i) where θ=0.95 according to Mansour and Timani in 2007. *Figure 4.5* shows the pseudo code for the algorithm used to generate T_0.

1. Set ***total_ΔE*** = 0;
2. Set ***NumAccepUpHillMoves*** = 0;
3. Set $\mathbf{E_0}$ to be the initial energy value of the *initialSolution*;
4. **repeat** (50**LecturesNum*) times
    a. perturb() on *initialSolution* without saving the change by choosing randomly two lectures and swap them.
    b. Calculate $\mathbf{\Delta E = E_{new} - E_0}$
    c. **if** (ΔE > 0) **then** /* uphill moves */
        total_ΔE += ΔE;
        NumAccepUpHillMoves++;
    end if

    end repeat
5. Calculate ***Average_ΔE*** = *total_ΔE/ NumAccepUpHillMoves*;
6. ***T_0 = - Average_ΔE/log 0.93***;

Figure 4.5: Pseudo Code for Calculating T_0

During the work of the algorithm, the best solution found so far (the one with the smallest *OF*) is continuously saved in order to make sure that the heuristic approach is keeping track of the best solution regardless of the temperature level at which it is found, or at which the algorithm ends. In our case, the convergence is detected when the algorithm find the feasible solution it is aiming to reach, or when the temperature reached the freezing point.

4.3.4 Partial Calculation of Objective Function

To be able to have better and faster performance for this algorithm, it is very important to develop an approach to calculate the partial change in objective value (energy function) at each iteration for each executed move rather than calculating the overall energy value each time. Therefore, to be able to calculate the change in energy value we need to apply the following formula:

$$\Delta OF = \Delta E = \varphi * \Delta S_{TC} + \psi * \Delta S_{CC} + \omega * \Delta S_{Avail}$$

where φ=50, ψ=1, and ω=1.

4.3.4.1 Time Availability Hard Constraint Partial Penalty Value

For each lecture L_x, to calculate partially the change in time availability violations penalty value (ΔS_{avail}), it is important to keep track of:

- The list of periods (*LP*) allowed for lecture L_x.
- The period P_x assigned to L_x.
- A 0/1 tag value (*AvailableConflictTag*);

Whenever a new Period value P_x is assigned to *Lx*, P_x will be first checked if it belongs to the list of allowed periods *LP* of L_x or not. If it belongs, then the tag value *AvailableConflictTag* will be set to zero and the change in availability penalty value (ΔS_{Avail}) will be decremented by one if the previous *AvailableConflictTag* was one, and L_x will move from the availability constraint violation status to the no violation status, or, ΔS_{Avail} might remain the same if the old tag value was set zero (there was already no violation). On the other hand, if the new P_x is not listed in *LP*, then *AvailableConflictTag* will be set to one and ΔS_{Avail} will be increased by one if the old tag value was zero or it will remain the same if the old tag was one (already in violation and continues). Therefore, ΔS_{Avail} value will be either:

- decremented by -1
- or incremented by +1
- or remains the same

4.3.4.2 Teacher Conflicts Hard Constraint Partial Penalty Value

To calculate the change in the teachers' conflicts penalty value ΔS_{TC}, a new two-dimensional array data structure is needed to save how many lectures are assigned for each teacher at specific period (time); For example, when we have

lectures L_1 and L_2 such that L_1 has teacher T_1, period P_x and room R_x, and L_2 has teacher T_2, period P_y and room R_y, and both of them has conflicts with other lectures, as shown below in *Figure 4.6,* and the algorithm tries to swap their values, this swap move will be accepted only if it solves partially or completely those conflicts and $\Delta S_{TC} < 0$.

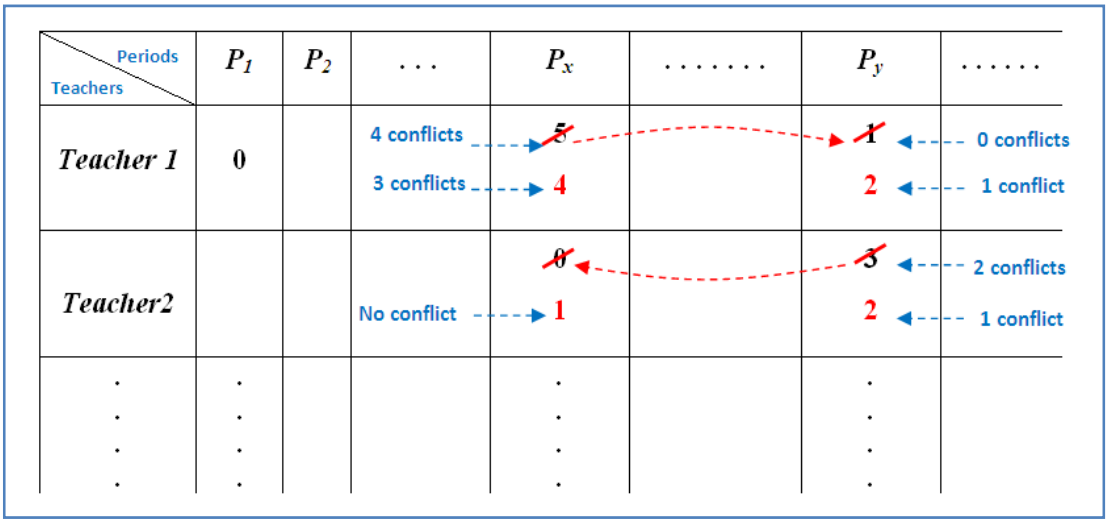

Figure 4.6: The Change in Teachers Conflicts Penalty When a Swap Move is Applied.

Therefore, $\Delta S_{TC} = \Delta S_{TC1}$ for $T_1 + \Delta S_{TC2}$ for T_2, where:

– ΔS_{TC1} = *New conflicts After Swap from* $P_x \rightarrow P_y$ – *Existing Old Conflicts*

$$= \left(\begin{array}{l} \text{If existing } (T_1, P_x) > 1 \\ \quad \text{new conflicts} = (T_1, P_x) - 2 \\ \text{Else} \\ \quad \text{new conflicts} = 0 \end{array} \right) + \left(\begin{array}{l} \text{If existing } (T_1, P_y) = 0 \\ \quad \text{new conflicts} = 0 \\ \text{Else} \\ \quad \text{new conflicts} = \text{existing } (T_1, P_y) \end{array} \right)$$

$$- \left(\begin{array}{l} \text{If existing } (T_1, P_x) > 1 \\ \quad \text{old conflicts} = (T_1, P_x) - 1 \\ \text{Else} \\ \quad \text{old conflicts} = 0 \end{array} \right) + \left(\begin{array}{l} \text{If existing } (T_1, P_y) > 1 \\ \quad \text{old conflicts} = (T_1, P_y) - 1 \\ \text{Else} \\ \quad \text{old conflicts} = 0 \end{array} \right)$$

– ΔS_{TC2} = *New conflicts After Swap from* $P_y \rightarrow P_x$ – *Existing Old Conflicts*

$$= \left(\begin{array}{l} \text{If existing } (T_2, P_y) > 1 \\ \quad \text{new conflicts} = (T_2, P_y) - 2 \\ \text{Else} \\ \quad \text{new conflicts} = 0 \end{array} \right) + \left(\begin{array}{l} \text{If existing } (T_2, P_x) = 0 \\ \quad \text{new conflicts} = 0 \\ \text{Else} \\ \quad \text{new conflicts} = \text{existing } (T_2, P_x) \end{array} \right)$$

$$- \left(\begin{array}{l} \text{If existing } (T_2, P_y) > 1 \\ \quad \text{old conflicts} = (T_2, P_y) - 1 \\ \text{Else} \\ \quad \text{old conflicts} = 0 \end{array} \right) + \left(\begin{array}{l} \text{If existing } (T_2, P_x) > 1 \\ \quad \text{old conflicts} = (T_2, P_x) - 1 \\ \text{Else} \\ \quad \text{old conflicts} = 0 \end{array} \right)$$

Check what you can observe from the following three examples for calculating ΔS_{TC1} for lecture L_1 with T_1 while studying different kinds of moves:

- If $\begin{array}{lll} & P_x & P_y \\ \text{Old:} & 5 & 1 \\ \text{New:} & 4 & 2 \end{array} \Rightarrow \Delta S_{TC1} = (3+1)-(4+0) = 0$. Nothing changed because L_1 is removed from a conflicting period to another conflicting one.

- If $\begin{array}{lll} & P_x & P_y \\ \text{Old:} & 5 & 0 \\ \text{New:} & 4 & 1 \end{array} \Rightarrow \Delta S_{TC1} = (3+0)-(4+0) = -1$. This is a good move because it removes L_1 from a conflicting period to a period that has no teacher conflicts; thus, improving ΔS_{TC1}.

- If $\begin{array}{lll} & P_x & P_y \\ \text{Old:} & 1 & 5 \\ \text{New:} & 0 & 6 \end{array} \Rightarrow \Delta S_{TC1} = (0+5)-(0+4) = +1$. This is a bad move where ΔS_{TC1} is getting worse because L_1 is removed from a non-conflicting period to a conflicting period.

Therefore, the value of ΔS_{TC1} will be in the range [-1, 1] and this will give the conclusion that for the perturb move of *SA* algorithm, the value of ΔS_{TC} of swapping the assignments of L_1 and L_2 ($\Delta S_{TC} = \Delta S_{TC1} + \Delta S_{TC2}$) will be between the range [-2, 2]

4.3.4.3 Curriculum Conflicts Hard Constraint Partial Penalty Value

To measure the change in the curricula's conflicts penalty value ΔS_{CC}, a two-dimensional data structure similar to that of teachers' conflicts should be built in which we can keep track of the number of lectures belonging to the same curriculum and assigned to the same period and follow the same calculation methods we used in *section 4.3.4.2*.

4.3.4.4 Overall Conflicts Penalty Value

To be able to compare our objective function value with that of the value coming from ITC2007 standard solution validator (Di Gaspero, McCollum and Schaerf, 2007), we have to write another method that goes in parallel with our formula and that would calculate the overall conflicts value that counts the conflicts between the courses. For example, assume:

- Curriculum_1 has courses *C1, C2* and *C3*;
- Curriculum_2 has courses *C1, C2*, and *C4*

- Teacher$_1$ teaches courses *C1* and *C2*.

If it happen that *C1* and *C2* has a conflicting lecture L at Period$_1$, then, in our method, we count the following three conflicts:

- Period$_1$ at Curriculum$_1$: L_1 of *C1* conflicts with L_2 of *C2*.
- Period$_1$ at Curriculum$_2$: L_1 of *C1* conflicts with L_2 of *C2*.
- Period$_1$ at Teacher$_1$: L_1 of *C1* conflicts with L_2 of *C2*.

On the other hand, the standard solution validator counts those as one conflict between *C1* and *C2*. Therefore, a new data structure is needed to be developed to keep track of possible conflicts between courses and calculate the objective value in the validator's way. This can be done by building from the start a set of possible conflicts' objects where each one includes two courses that might have the same curriculum or the same teacher in common. And, we should have a two-dimensional array of conflicts similar to the one we built in *section 4.3.4.2*, that counts the number of conflicts happened at each period.

4.4 Phase Two: Scatter Search Algorithm

From phase one, SA, in our approach, the best solution found will be a complete feasible solution in which all the hard constraints have been solved and which will be the input for the second phase that uses Scatter Search as our search technique that tries to optimize the value of soft constraints violations. As previously mentioned, scatter search (SS) is an interesting population-based evolutionary algorithm that has been applied successfully on different hard optimization problems and achieved important results (Glover *et al.,* 2000, 2002, 2003) and according to our knowledge, there is no published work related to using scatter search to course timetabling problem. In *Appendix I*, we gave an introductory background about *SS* and we explained how SS works on a set of solutions belonging to what is called *Reference Set* which is relatively smaller in size than those of other evolutionary approaches (Manrour, Isahakian and Ghalayini, 2011; Laguna and Marti, 2003). Our *SS* is made up of different methods and different local search techniques used at each method represented by the neighborhoods/moves created specifically for this problem using some ideas from Muller's work (2009, 2008). In the following sections we will describe in details the implementation of *SS* methods as they are adapted in the context of CCTP. Those methods are:

- Diversification Generation Method;
- Improvement Method;
- Reference Set Update Method;
- Subset Generation method;
- Solution Combination Method.

SS uses different techniques for search diversification and intensification that confirmed its effectiveness in different optimization problems. It diversifies the search space by operating on *RefSet* and combines its solutions to produce new ones by not merely relying on randomization (Laguna and Marti, 2003; Glover *et al.*, 2003, 2004; Marti *et al.*, 2005). Not only this, but intensification also is supported by allowing the solutions with good quality to be added to *RefSet. Table 4.1* shows a summary of SS used notations and *Figure 4.7* shows the general Scatter Search pseudo-code used in our implementation.

Table 4.1: Summary of Useful Notations and Symbols Used in SS Algorithm

Notations and Used Symbols	Description
PopSize	*SS* population size.
Pop	*SS* initial population (set of solution objects).
RefSet	Reference Set of solution objects
b	Reference Set size (10 for testing and 20 solutions for final results).
b1	Size of the High Quality Sub Reference Set (*b1=b/2*).
b2	Size of the Diverse Sub Reference Set (*b2=b-b1*).
HQRefSet	The High Quality Reference Set of solution objects.
DivRefSet	The Diverse Reference Set of solution objects.
SubsetsType	The subset type used 1, or 2.
Subsets	The Solutions Subsets used in the combination method.
Pool	Set of candidate trial solutions produced by the combination method *CM*.
idleIterations	The number of idle iterations in which best Solution did not change.
bestSolution	The best solution found so far in *SS* iterations
p	The Solutions index in Pop.
Sol	The Solutions index in RefSet.
Subset	The subset index in set of subsets.
Course	The index in Courses set.
L	The Lectures index.
Curr	The Curricula index.
PRA	The (period, room) assignment index.
R	The Rooms index.
T	The Teachers index.

1. Start with the initial Feasible Solution ***initSolution*** coming from *SA*;
   Start with Population Set *Pop* = ∅;
2. Apply the improvement method on *initSolution* and add it to Pop;
   While (|*Pop*| < *Pop*Size-1) **do**
   Create a solution *x* by using diversification generation method by applying random *swap moves* on *initSolution* while keeping its feasibility.
   3. Use the improvement method on solution *x* to produce the improved solution x'
      if (*x'* ∉ *Pop)* **then**
      Pop = *Pop* ∪ *x'*
      else
      Discard x'
      end if;
   end while;
4. Order Solutions in *Pop* based on *OF* values. By using reference set update method, construct *RefSet* = {x^1, …, x^b} where b=b1+b2 by selecting "best" b solutions in *Pop* and forming *HQRefSet* and *DivRefSet*;
   Set *HQRefSet* = ∅;
   for each *i (0 … b1-1 in Pop)* **do**
   HQRefSet[i] = *Pop*[i];
   end for;
   Order *HQRefSet in increasing order based on OF values;*
   Set *DivRefSet* = ∅;
   for each *x* ∈ *Pop-HQRefSet* **do**
   for each *y* ∈ *HQRefSet* **do**
   Calculate dissimilarity distance *d(x, y);*
   end for;
   Find $d_{min}(x) = MIN_{y \in RefSet}\ \{d(x,y)\};$
   end for;
   Order solutions in *Pop-HQRefSet* in decreasing order based on min distance values;
   for each *i (0 … b2-1 in Pop-HQRefSet)* **do**
   DivRefSet[i] = *Pop*[i];
   end for;
   ⇒ *RefSet = HQRefSet* ∪ *DivRefSet*
   Set *Iteration* for each solution added to *RefSet* to 0;
5. Set *CurrentIteration* = 1, *idleIterations*=0, *bestSlution= HQRefSet[0];*
   while (*idleIterations<MaxIdleIterationsNum*) **do**
   6. Use subset generation method to generate *Subsets* of type 1 or 2 skipping no subsets even for those whose elements did not change from previous iterations.
   7. *idleIterations*++.
   while (*Subsets* ≠ ∅) **do**
   Select the next subset *S* in *Subsets;*
   8. Use the solution combination method over *S* to obtain the trial solution *X;*

```
            if(X != Feasible) then
                Pass X through a repair heuristic to maintain its feasibility;
                if(X remains infeasible) then
                    ignore X and continue;
                end if;
            end if;
            Use the improvement method on solution X to produce X';
            Apply the reference set update method to add X' to RefSet;
            if (bestSolution.OFvalue != HQRefSet[0].OFValue) then
                idleIterations = 0;          /*reset idle iterations counter */
            end if;
            Delete subset S from Subsets.
        end while;
    9.  CurrentIteration ++;
    end while;
GET THE BEST SOLUTION FOUND IN REFSET AS THE FINAL SOLUTION
```

Figure 4.7: General Scatter Search Pseudo-code used in our implementation

4.4.1 Partial Calculation of Objective Function for Scatter Search

Since *SS* will concentrate only on optimizing the soft constraints penalty value while maintaining the feasibility of the solution. Therefore, let the objective function for *SS* be as follows:

$$\text{OF} = \text{Total Violations} + \text{Total Cost where Total Violations}=0$$

$$\Rightarrow \text{OF} = \text{Total Cost} = \gamma * S_{ExtStd} + \alpha * S_{MWD} + \beta * S_{CurrComp} + \mu * S_{RoomStab}$$

where $\gamma = 1$, $\alpha = 5$, $\beta = 2$, and $\mu = 1$.

As mentioned discussed, it is very important to calculate the partial change in objective function value for each executed move. And in this case we need to apply the following formula:

$$\Delta\text{OF} = \gamma * \Delta S_{ExtStd} + \alpha * \Delta S_{MWD} + \beta * \Delta S_{CurrComp} + \mu * \Delta S_{RoomStab}$$

where $\gamma = 1$, $\alpha = 5$, $\beta = 2$, and $\mu = 1$.

4.4.2 The Diversification Generation Method

4.4.2.1 Diversification Generation Method

The feasible solution produced from SA algorithm will be used to generate the *N* initial solutions which will form the initial population for SS by applying random and controlled-random accepted feasible swaps on this feasible solution while keeping the feasibility of the solutions. All the solutions produced in the

diversification generation method (DGM) are going to be feasible while we are trying to get a balance between diversification (randomization to generate half the initial population) and intensification (controlled randomization to generate the second half of the initial population). The number of accepted feasible swaps upper limit is determined through testing whether it will be equal to half or third *PopSize*. *Figure 4.8* shows the diversification generation method pseudo code.

---

1. Set *P=0*, *nrAcceptedFeasibleSwaps* = 0;
   Set *cFreq*[iFreqDivisionFactor] as Frequency Complement; */ *iFreqDivisionFactor=5*/*
   Set iDGMMaxAcceptedFeasibleSwapsLoop = TotalLecturesNum /3;
   Set iFreq[iDGMMaxAcceptedFeasibleSwapsLoop][iFreqDivisionFactor];
   Set *PopSize* = 100;
2. **while** (P < PopSize/2) **do** */*generate half the Pop initial random solutions */*
   while (*nrAcceptedFeasibleSwaps* <= *iDGMMaxAcceptedFeasibleSwapsLoop*) **do**
   createEmptySolution(*X*);
   copyDataFromSASolution(*X);*
   Perform a random swap move on *X* between randomly two selected lectures;
   if (swap keeps the feasibility of the solution) **then**
   acceptSwap() ;
   nrAcceptedFeasibleSwaps++;
   else
   rejectSwap();
   end if;
   end while;
   Use the improvement method on solution *X* to produce *X';*
   if (*X'* ∉ Pop) **then**
   Pop = Pop ∪ *X'*;
   else
   Discard *X';*
   end if;
   Set nrAcceptedFeasibleSwaps=0;
   end while;
3. **while** (*P* < *PopSize*) **do** */* second half of Pop initial semi-random solutions */*
   while (*nrAcceptedFeasibleSwaps* <= *iDGMMaxAcceptedFeasibleSwapsLoop*) **do**
   - createEmptySolution(*X*);
   - copyDataFromSASolution(*X);*
   - Select a sub-Range of Lectures *LRange1*= GetLectureSubRange();
     Update frequency *iFreq[AcceptedFeasibleSwaps][LRange1]++;*
   - Select another sub-Range of Lectures *LRange2*= GetLectureSubRange();
     Update frequency *iFreq[AcceptedFeasibleSwaps][LRange2]++;*

L1 = $\left(LRange1 * \frac{TotalLecturesNum}{\text{iFreqDivisionFactor}}\right) + \left(random() * \frac{TotalLecturesNum}{\text{iFreqDivisionFactor}}\right)$;
L2 = $\left(LRange2 * \frac{TotalLecturesNum}{\text{iFreqDivisionFactor}}\right) + \left(random() * \frac{TotalLecturesNum}{\text{iFreqDivisionFactor}}\right)$;

a. Perform a swap between these two selected lectures;
if (swap keeps the feasibility of the solution) **then**
acceptSwap() ;
nrAcceptedFeasibleSwaps++;
else
rejectSwap();
end if;
end while;
Use the improvement method on solution *X* to produce *X';*
if (*X'* ∉ Pop) **then**
Pop = Pop ∪ *X'*;
else
Discard *X';*
end if;
Set nrAcceptedFeasibleSwaps=0;
end while;

Figure 4.8: Pseudo-code for the Diversification Generation Method of Scatter Search

4.4.2.2 Solution Equality

Note that before the new generated solution *X* is added to the initial population *Pop* of SS, it is a must to check if this solution belongs to *Pop* or not. To do this, *X* has to be compared with all the currently existing solutions in *Pop* and the comparison should be based on the objective value only because deep equality checking between solutions lecture by lecture will be very computationally costly; which means that if *X* has an objective value different from all solutions in *Pop*, then we add *X* to Pop; Otherwise, we ignore *X* and generate new solution.

4.4.3 The Improvement Method

4.4.3.1 Improvement Method

The improvement method (IMP*)* follows the diversification generation method to improve the solution before it is added to *Pop* as well as it follows the subset combination method to enhance the quality of the generated candidate solution before trying to add it to *RefSet*. The aim behind this improvement method is to produce better solutions by trying to minimize the objective function, based on the soft constraints, as much as possible while preserving the hard constraints that have been previously corrected in each of the produced solutions. Thus, the improvement

method input is a feasible solution that tries to improve its quality by trying to minimize the cost of violating soft constraints by applying on it some problem specific defined moves (neighborhood search moves). IMP goes over the solution with *iIMPMaxAcceptedFeasibleMoves* times. *iIMPMaxAcceptedFeasibleMoves* is equal to the total lectures' number multiplied by two or three depending on the testing results. In our case, it is two to avoid premature convergence of the solution. *Figure 4.9* shows the pseudo-code for the improvement method.

```
1. Set iIMPMaxAcceptedFeasibleMovesLoop = LecturesNum*2;
   Let NeighborSelection[totalMoves] be the vector of neighborhood moves.
   Set Move =0;
2. For each Move (0… iIMPMaxAcceptedFeasibleMovesLoop ) do
       update_moves_probabilities_according_to_solution(Sol);
       NeighbourSolution S = generate_Move_IMP(Sol);
       Calculate ΔE = E_S – E_Sol
       if ( S != null and S.Feasible  and ΔE < 0 ) then          /* downhill moves */
               acceptMove();
               Set Sol = S;
       else
               ignoreMove();
               ignoreNewSolution(S);
       end if;
   end for;
3. GET THE IMPROVED SOLUTION FOUND SO FAR
```

Figure 4.9: Improvement Method Pseudo-code

4.4.3.2 Problem Specific Moves

Two major types of improvement moves exist; *First Move* and *Best Move*. In the First Move, we select any element randomly from the input solution and attempt to move it to the first location that improves its objective function value, while, on the other hand, in the Best Move, the elements are moved to the place that ensures the optimum enhancement in the quality of the input solution while minimizing the objective function value. In our case, we will adapt the First Move type of improvement method with all our developed methods since the main drawback of the Best Move method is that it needs a longer execution time than the other move.

This section discusses the curriculum-based course timetabling problem specific moves (PSMs) used during improvement procedure. The terms moves or

neighborhoods are used interchangeably during this report but both refers to the same meaning. At the beginning, when the improvement method used at DGM, all neighborhoods are chosen with the same probability except for curriculum compactness move which will be chosen with 0.3 probability P. But, when IMP is used in the combination method, the probabilities of selecting the moves are the same for time move, lecture move, room move and swap move while it is dynamically changing for room stability move, curriculum compactness move and minimum-days move according to the change in room stability, room capacity, curriculum compactness, and min working days soft constraints penalty values. Those moves will not allow IMP to be merely random and some of the ideas of these moves come from Muller's work in 2008 and 2009. The different PSMs used for CCTP are:

a. **Time Move:** Select a lecture randomly, and change its period by assigning to it the first non-conflicting randomly chosen period.
b. **Room Move:** Select a lecture randomly, and change its room by assigning to it the first non-conflicting randomly chosen room.
c. **Lecture Move:** Select a lecture randomly, and select a new value (new time and new room) for it by considering only the times available for the chosen lecture course. If the new value does not result in any conflicts with other previously scheduled lectures, then assign it directly to the chosen lecture (return the assignment); otherwise, look for another choice, or ignore this move.
d. **Swap Move:** Select any two lectures randomly, and attempt to swap their values on condition that the feasibility of the solution is preserved. If the swap improves the overall objective function value, then the swap is accepted.
e. **Room Stability Move:** The aim of this move is to decrease the room stability constraint violations. Select randomly a course and a room and then attempt to move all the lectures of the selected course into the selected room. If this conflicts with a lecture that already exists in this room at specific time, then the conflicting lecture will be moved to the room of the chosen lecture. This move will be tried until a number of maximum trials will be reached.

f. **Minimum Working Days Move:** The aim of this move is to decrease the minimum number of working days penalty. Select randomly a course that has a positive min working days' penalty, select the day in which two or more lectures of that course are taught and finally attempt to move one of those lectures of that day into a day in which none of the course's lectures are taught.

g. **Curriculum Compactness Move:** The aim of this move is to decrease the curriculum compactness penalty. Select a curriculum randomly, select an isolated lecture in that curriculum and attempt to place it into an available period adjacent to another lecture that belongs to the same curriculum (if this assignment does not lead to any conflicts). This might lead to the assignment of a new room to the chosen lecture if its older room assignment is not available along at the chosen period.

4.4.4 Reference Set Update Method

Using reference set update method (RSUM), a reference set of solutions of size b (called *RefSet*) is constructed by combining two smaller sub reference sets. The first sub-set is composed of the best b_1 solutions called *HQRefSet* while the second one is composed of the farthest diverse b_2 solutions called *DivRefSet*. *RefSet* solutions are used to create new solutions by the way of applying Combination Method. RSUM is divided into mainly two phases/cases (Laguna and Marti, 2003):

- Initial construction of reference set *RefSet* from the initial *Pop* population.
- Updating the reference set with the best, in terms of good quality and good diversity, improved trial solutions resulting from the Improvement Method after being generated from the Combination Method.

First of all, after the construction of the initial population Pop using DGM and IMP methods, we need to determine the size of the reference set to start creating it. Usually, *RefSet* size is relatively smaller than the size of the initial population Pop by ratio of 0.1 or 0.2. In our case, we are going to use $|RefSet| = 0.2 * |Pop|$ (Mansour *et al.*, 2009). Also, *RefSet* is made up of high quality reference set of size b_1 and farthest diverse reference set of size b_2; $RefSet = HQRefSet \cup DivRefSet$ where $|RefSet| = b_1+b_2 = b$. To initially create *HQRefSet* from the initial population *Pop*, order the solutions in Pop in ascending order according to their OF values, and select the best b_1 solutions from Pop having the lowest values of objective function and add them to *HQRefSet* and remove them from the initial population *Pop*.

On the other hand, to construct *DivRefSet*, we select the farthest solutions in *Pop-HQRefSet* to the solutions in *HQRefSet*. To do this, we need to sort *HQRefSet* solutions in ascending order with respect to their objective function values, and then we compute the minimum distance of dissimilarity between all remaining solutions *x* in *Pop* and all solutions *y* in *HQRefSet* as shown in the following equation:

$$d_{min}(x) = MIN_{y \in RefSet} \{d(x,y)\}$$

Now, to find the b_2 diverse solutions, we need to sort in descending order the solutions in *Pop-HQRefSet* based on the minimum distances $d_{min}(x)$ and then select the b_2 solutions having the largest minimum distance values and insert them into *DivRefSet* after calculating their OF values. Now, by combining *HQRefSet* and *DivRefSet*, we will have our final initial *RefSet*. The distance between two solutions is the measure of dissimilarity between them. Thus, $d_{min}(x)$ represents the smallest dissimilarity between *x* and one of the y solutions in *HQRefSet* and it is found by counting the different lecture to room and period assignments between the examined solutions. Then, selecting the solution with maximum distance among the other solutions means that we are selecting the more distinct or the farthest diverse solutions to *HQRefSet* solutions. Step number 4 in *Figure 4.7* shows the pseudo-code for the construction of the initial reference set.

In the second case, where *RefSet* needs to be updated with trial solutions that have been generated from Combination and Improvement Methods, and that might be possibly added to *RefSet* to be used in the subsequent subset generation and solution combination cycle. The Reference Set Update Method is mainly responsible for the update and maintenance of the reference set *RefSet* by following the subsequent steps and as shown in *Figure 4.10* (Haidar, 2009):

- Check whether the candidate solution belongs to *RefSet* by comparing the *OF* values of this solution with every other solution in *RefSet* and check their equality. If none matched its value, then it does not belong to *RefSet*.
- Keep *HQRefSet* sorted in increasing order with respect to the objective function values of its solutions where we have x_{worse} of *HQRefSet* as the solution having the worst (i.e. highest) value of *OF* and is ordered the last in the list of solutions.
- Keep *DivRefSet* sorted in decreasing order of with respect to the Minimum Distance values of its solutions where we have x_{worse} of *DivRefSet* as the

solution having the worst (i.e. lowest) distance value and is ordered the last in the list of solutions.

- Now, we need to check if the *OF* value of the trial solution x is less than the OF value of the worst solution x_{worse} in *HQRefSet.* If it is, then we need to replace the x_{worse} solution in *HQRefSet* with x and we need to resort *HQRefSet* in increasing order again based on OF value.
- Else, if OF value of x is larger than that of x_{worse} of *HQRefSet*, then we need to calculate the Minimum Distance value of the candidate solution x with respect to all solutions y in *HQRefSet*; $d_{min}(x) = MIN_{y \in RefSet} \{d(x,y)\}$. Then, we need to check if $d_{min}(x)$ is greater than the distance value of x_{worse} of *DivRefSet.* If it is, then replace x_{worse} solution in *DivRefSet* with x and resort *DivRefSet* in decreasing order based on the distance.
- If the minimum distance of the candidate solution x is less than that of the solution with the minimum distance in *DivRefSet*, then discard x.

```
/* Apply Reference Set Update Method on candidate improved combined solution X*/
1. Order HQRefSet in increasing order based on OF values;
2. Order DivRefSet in decreasng order based on Distance values;
3. if ( X ∉ RefSet ) then
   4. if ( f(X) < f(X_worse_b1) ) then
         Insert X into proper position in HQRefSet by replacing X_worse_b1 of HQRefSet;
         Reorder HQRefSet in increasing order based on OF value;
         Set X.Iter = CurrentIteration;
      else
        5. for each y ∈ HQRefSet do
              Calculate distance d(X, Y)
           end for;
           Find d_min(X) = MIN_{y∈RefSet} {d(X,Y)}
           if ( d_min(X) > d_min (X_worse_b) ) then
              Insert X into proper position in DivRefSet by replacing X_worse_b of DivRefSet;
              Reorder DivRefSet in decreasing order based on distance values;
              Set X.Iter = CurrentIteration;
           else
              Discard X;
           end if;
      end if;
   end if;
```

Figure 4.10: Reference Set Update Method Pseudo-code (2nd case)

4.4.5 Subset Generation Method

4.4.5.1 The Adapted Subset Generation Method

Subset generation method (SGM) for scatter search, in its general form, consists of generating subsets of different sizes and not only limited to subsets of size two. The main purpose behind the use of SGM is to produce subsets of reference set solutions to be as an input for the Combination Method. Scatter search in general uses commonly four different types (Laguna and Marti, 2003; Glover, 2002):

- **Subset Type 1:** consists of all 2-elements subsets derived from combining all possible pairs of solutions in *RefSet*, (x_1, x_2), that have not been combined before.
- **Subset Type 2:** consists of all 3-elements subsets derived from the 2-solutions subsets by adding to each two-solutions subset (x_1, x_2) the best solution $x_3 \notin \{x_1, x_2\}$.
- **Subset Type 3:** Consists of all 4-elements subsets derived from the 3-solutions subsets by adding to each (x_1, x_2, x_3) the best solution $x_4 \notin \{x_1, x_2, x_3\}$.
- **Subset Type 4:** the subsets containing the best i solutions of *RefSet*, for $i=5$ to b.

Although SGM can be customized to go with different problems, the solution combination method should be specifically developed and implemented for each problem as we will see in the following section. As mentioned by Laguna and Marti (2003), for subsets of size two, there is a maximum $SubsetsSize = b!/(2!\,(b-2)!) = (b^2-b)/2$ where b is the size of *RefSet*, while for subsets of size 2 and 3 together, there is a maximum (Mansour *et al.,* 2009; Isahakian, 2006; Laguna and Marti, 2003):

$$SubsetsSize = (b^2-b)/2 + (b^2-b)/2 - (b-1)$$
$$= b!/(3!\,(b-3)!) = (b^2-b)(b-2)/6.$$

In our algorithm, we will create subsets of type 1 and type 2 (size three) by adding to each two-solutions subset the best solution does not belong to it taking into consideration that we do not want to skip the subsets for which all its elements did not change from the preceding iterations because our combination method relies on many random moves and choices making the change of the resultant solution in the following iterations possible.

4.4.5.2 Partially Dynamic Subset Generation Method

We decided to use this type of subsets because most of the new solutions are generated by using subsets of sizes two and three because subsets of larger sizes is computationally time consuming and need to be used with care and with more advanced techniques. Therefore, for subsets of size two, we start by combining all possible pairs of solutions, (x_1, x_2), from the reference set *RefSet* generated from the Reference Set Update Method. After that, apply IMP method to the trial solutions generated at this step, and add them to *Pool* of candidate solutions that are waiting to be later added to *RefSet* using the Reference Set Update Method. This is called a *static update approach*. Then, for subsets of size three, we expand the two-solutions subsets created at the former step by adding the best solution x_3 in *RefSet* with the best objective function value such that $x_3 \notin \{x_1, x_2\}$. And repeat the same steps for the trial solutions generated here. *Figure 4.11* shows the pseudo-code of the static Subset Generation and Update Method for subsets of type 1 and 2.

```
/* Apply Subset Generation Method after the creation and maintenance of RefSet of size b;
   Start Combining Solutions of RefSet */
1. Set Pool = ∅, SubsetSet = ∅;
2. for i (0 … b-1) do
       for j (i+1 … b-1) do
           SubsetSet = SubsetSet ∪ generate_Type1_subset(Sol[i], Sol[j], 1);
           SubsetSet = SubsetSet ∪ generate_Type2_subset(Sol[i], Sol[j], HQRefSet[0], 2);
       end for;
   end for;
3. while (SubsetSet ≠ ∅) do
       if (subset=Type1) then
           TrialSolution = CombineSolutions(subset .Sol1, subset .Sol2 );
       else
           TrialSolution = CombineSolutions(subset .Sol1, subset .Sol2, subset .Sol3 );
       end if;
       Use improvement method on TrialSolution to produce TrialSolution'
       if (TrialSolution' ∉ Pool) then
           Pool = Pool ∪ TrialSolution';
       else
           Discard TrialSolution';
       end if;
   end while;
4. while (Pool ≠ ∅) do
```

```
    Pick up solution x from Pool;
    Use reference set update method to add each x in Pool to RefSet at CurrentIter
    Remove x from Pool;
end while;
```

Figure 4.11: Static Subset Generation Method Pseudo-code for Subset Type 1 and 2

A faster approach is that after the generation of each trial solution, the *RefSet* is updated directly before the next combination method takes place (Laguna and Marti, 2003). The main benefit of this *dynamic update approach* is that if the reference set has solutions of low quality, they will be rapidly replaced and the outlook combinations are made with the new improved solutions. But its main drawback is that some potential good quality combinations might be eliminated before they can be considered. Also, in the static update, the order of the combined solutions is not important, while now we have to modify the algorithm to take into consideration the dynamic update of *RefSet*.

We came into a compromise between both approaches by allowing the dynamic update of reference set with the candidate trial solutions generated by the combination of subsets of solutions previously added to *RefSet* at iterations less than that of *CurrentIteration* while do not allow the use of those newly added solutions to *RefSet* at the Subset Generation Method in *CurrentIteration* and leave them to be used till next iteration. The good in this partially-dynamic approach is that we are making use of replacing the solutions in *RefSet* with inferior quality, so that they will not be used in future combinations. In this case, we will eliminate the disadvantage of extra computational time needed because of the use of the static approach or because of the complexity of the implementation of dynamic update approach though it is faster than the static update one. *Figure 4.12* shows the implementation of our partially dynamic approach for subsets of Type 1 and 2.

```
/* Apply Subset Generation Method after the creation and maintenance of RefSet of size b;
   Start Combining Solutions of RefSet  */
1.  Set C = CurrentIteration;
    for i (0 … b-2) do
        for j (i+1 … b-1) do
            /*****************  SubsetType 2 ****************/
            2.  if ( ( Sol[i].Iter == C-1 || Sol[j].Iter == C-1 ) &&
                ( Sol[i].Iter != C   && Sol[j].Iter != C    ) )   then
                    TrialSolution = CombineSolutions(Sol[i], Sol[j] );
```

```
                Use improvement method to TrialSolution to produce x;
                Use reference set update method to try to add x to RefSet;
                if (RefSet = RefSet ∪ x) then
                        Set x.Iter = C;
                end if;
        end if;
        /***************** SubsetType 3 *****************/
        if ( i!=0 && j!=0 ) then
                if ( ( Sol[i].Iter == C-1 || Sol[j].Iter == C-1 || Sol[j+1].Iter == C-1 ) &&
                   ( Sol[i].Iter != C  && Sol[j].Iter != C  && Sol[j+1].Iter != C  ) )
                 Then
                        TrialSolution = CombineSolutions(Sol[i], Sol[j], Sol[0] );
                        Use improvement method to TrialSolution to produce x;
                        Use reference set update method to try to add x to RefSet;
                        if (RefSet = RefSet ∪ x) then
                              Set x.Iter = C;
                        end if;
                end if;
         end if;
    end for;
end for;
```

Figure 4.12: Partially Dynamic Subset Generation and RefSet Update Methods Pseudo-Code for Subsets of Type 1 and 2

4.4.6 Combination Method

4.4.6.1 The Combination Procedure

The combination method (CM) combines together two or more feasible solutions of every subset generated by the previously discussed SGM to create new feasible combined trial solutions. Laguna and Marti (2003) stated that this method might result in infeasible solutions; for this, every combined trial solution will be subjected to a greedy repair heuristic that aim to make the solution feasible again. Then, when the infeasible trial solution got its feasibility again, it will be subjected to the improvement method. After that, the reference set update method is called to try to add it to the reference set.

While designing the Combination Method, we should take care of the main goal behind it which is achieving a balance between intensification and diversification of the produced trial solutions. Therefore, this method will be implemented by a greedy heuristic method that adapts a scoring technique based on

calculation of partial objective function values as well as on randomization in selecting (Period, Room) assignments as a tie-breaker in case we got the same scores (Mansour *at al.*, 2009).

First of all, this method starts with an empty solution *S* to which (*period, room*) assignments will be assigned to its lectures (of courses). The heuristic begins by selecting the lectures one by one from solution *S* starting with *lecture i* for *i*=1,2, …, *LecturesNum*, and tries to assign to it one of the corresponding assignments of the same lecture *i* in the feasible solutions S_1 and S_2, in case of Type 1 two-elements subsets, and in S_3 in the case of three-elements subsets. The assignment values of lecture *i* in the examined subset solutions are assessed, and the one that will result in the lower partial objective function value will be accepted and chosen to be assigned to lecture *i* in *S*. The partial objective function value will take into account all the choices previously made so far for lectures j starting from 1,…, *i-1*. In case ties in scores for all the 2 (or 3) assignments of lecture *i* in S_1 and S_2 (and S_3) happened, we can break this tie by randomly selecting one of the assignment values and assign it to lecture *i*.

Choosing and adding the best assignment among the two (or three) solutions in subsets to the new solution will lead to increasing the combined solution intensification, while combining solutions using both *SubsetTypes* 1 and 2 will offer more distinct combined solutions. However, diversification is not much decreased by the intensification tendency because the CM goes over the lectures of the solution (one lecture at a time) and attempt to assign to it the (period, room) assignment value that will lead to the best partial objective function value (Mansour *et al.*, 2009). The assignment value that might give the best partial objective value for lecture *i* in *S* among the other assignments of lecture *i* in solutions belonging to *subset1*, might not give the best objective function value among other solutions of *subset2*. In other words, this will result in a larger pool of combined trial solutions that can be used later to update the reference set *RefSet*.

However, there are some cases in which some exceptions in choice can be done. For example, a case might happen where one of the choices causes a violation in to the previously fixed hard constraints while the other choice do not (in case of Type 1 subsets), then accept the first choice regardless of the change in partial objective function value (whether it is increased or decreased). Moreover, another case might happen where both choices cause hard constraints violations; in this case,

accept the one with the smaller partial OF value and raise the *HCflag* to one, to know that this trial solution is infeasible, and keep a list of lectures/courses where the hard constraints are violated and keep the number of violations in parameter *violationsNum*. *Figure 4.13* shows the pseudo-code for the *Combination Method* used by our *SS* approach for Subset Types 1 and 2.

```
/* Apply Solution Combination Method */
for subset (0 … SubsetsSize-1) ∈ Subsets do
    Set S = ∅;                                          /*new empty solution */
    for lecture (0 … LecturesNum-1) ∈ S do
        SubsetSize = GetSubsetSize(subset);
        BestAssignment = null, BestBadAssignment = null;
        BestObjValue = 9999999999, BestBadObjValue = 9999999999;
        BestSolindex = -1, BestBadSolindex = -1;
        Count1 = 0, count2 = 0, count3=0;
        SameScoreSol = ∅, HCviolationSet = ∅;
        for Sol (0 … SubsetSize-1) ∈ subset do
            AssignmentValue = GetAssignment(Sol, lecture);
            ObjValue = GetPartialObjValue(S, S.ObjValue, lecture, AssignmentValue);

            /* if assignment causes any hard conflicts, then add it to conflicts set */
            if (AssignmentValue cause HC violation) then
                count3++;
                if (ObjValue == BestBadObjValue ) then
                    Count1++;
                    if (Sol ∉ HCviolationSet) then
                        HCviolationSet ∪= (Sol, AssignmentValue);
                    end if;
                    if (BestBadSolindix ∉ HCviolationSet) then
                        HCviolationSet ∪= (BestBadSolindex, BestBadAssignment);
                    end if;
                else
                    /* Select Assignment with best score */
                    if (ObjValue < BestObjValue ) then
                        BestBadObjValue = ObjValue;
                        BestBadAssignment = AssignmentValue;
                        BestBadSolindex = Sol;
                    end if;
                end if;
            else
                /* If assignment causes no conflicts, check if scores are equal for the
                different solutions*/
                if (ObjValue == BestObjValue ) then
                    Count2++;
                    if (Sol ∉ SameScoreSol) then
                        SameScoreSol ∪= (Sol, AssignmentValue);
                    end if;
```

```
                        if (BestSolindix ∉ SameScoreSol) then
                            SameScoreSol ∪= (BestSolindex, BestAssignment);
                        end if;
                    else
                        /* Select Assignment with best score */
                        if (ObjValue < BestObjValue ) then
                            BestObjValue = ObjValue;
                            BestAssignment = AssignmentValue;
                            BestSolindex = Sol;
                        end if;
                    end if;
                end if;
            end for;

            /* Use Randomization as tie-breaker between solution not causing conflicts*/
            if ( count1==0 && Count2>0 &&  BestSolindex ∈ SameScoreSol) then
                rand = SelectRandomSol(SameScoreSol);
                if (rand.index != BestSolindex ) then
                    BestAssignment = rand.AssignmentValue;
                    BestSolindex = rand.index;
                end if;
            end if;

            /*If all solutions causes conflicts, go over them and select the one with the
            smallest partial OF value */
            if(count3 == SubsetSize) then
                if(count1>0 && count2 ==0 && BestBadSolindex ∈ HCviolationSet) then
                    rand = SelectRandomSol(HCviolationSet);
                    if (rand.index != BestBadSolindex ) then
                        BestBadAssignment = rand.AssignmentValue;
                        BestBadSolindex = rand.index;
                    end if;
                end if;
                BestAssignment = BestBadAssignment;
                BestSolindex = BestBadSolindex;
                S.HCflag = 1;              /* meaning that this trial solution is infeasible*/
            end if;

            /* Assign the best assignment found so far */
            AssignValueToSolution(S, lecture, BestAssignment, BestObjValue);
        end for;
        Subsets = Subsets - {subset};
        if(S.HCflag=1) then
            repair(S);
        end if;
        improvement_method(S);
        reference_set_update_method(S, subset);
    end for;
```

Figure 4.13: Combination Method Pseudo-code for Subset Types 1 and 2

4.4.6.2 The Repair Heuristic

To solve the problem of infeasibility of some of the produced candidate trial solutions, each generated infeasible solution will be sent to a repair heuristic (called by CM) which will be a greedy algorithm that works as shown in *Figure 4.14*.

```
1. Set count = 0, RepeatedViolationLoopConstant =100;
2. Set the max number of non changing attempts to
   maxNumUnchanged=violationsNum* RepeatedViolationLoopConstant;
3. Loop1: for each i (0 … violationsNum) do
      4. Loop2: while (violation remains for i && count < maxNumUnchanged) do
            Make a ViolatedLectureMove to a new (P,R) assignment that solves this
            violation; or make a PeriodSwapMove of this violated lecture with the period
            of another selected lecture to solve the violation;
            if( violation i solved) then
                  accept _move();
                  count = 0;
                  break Loop2;
            else
                  reject_move();
                  count++;
                  continue Loop2;
            end if;
      end while;
   end for;
```

Figure 4.14: Reference Set Update Method Pseudo-code (2nd case)

4.4.7 Termination Condition

In the standard scatter search algorithm design template, the algorithm stops when all the subsets' combinations do not produce any solution that can be added to *RefSet.* However, in our case, this is not possible because if at a specific iteration no update to *RefSet* happens, at the next iteration we might have such successful combinations because the procedure of generating the trial solutions goes through different stages at which each one relies mainly on random selections, swaps, and random neighborhood moves and this might lead to the change in the produced solution at the next iteration as shown in *Figure 4.15.* For this, even if the same solutions were combined together at different iteration, we have a high probability to have new different candidate solutions.

Therefore, our termination condition suggests the stopping of the SS algorithm cycle when the best solution available in *RefSet* does not change for 20 consecutive iterations.

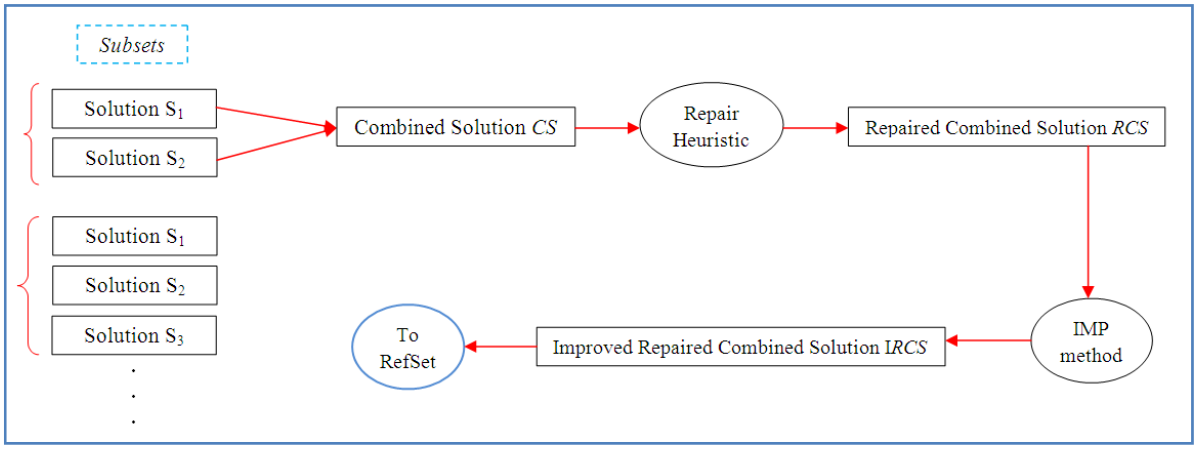

Figure 4.15: The Cycle of Generating Feasible Trial Solutions

4.5 Tuning Heuristic Phase

In some instances, we might still be eager to have three or four percent of extra improvement in the overall solution value without recalling the SA or SS again. For this, a greedy tuning heuristic is implemented to operate on the best feasible solution found so far by SS. The tuning heuristic targets directly the lectures having soft constraints violations, and tries to enhance them. It works on the room capacity constraint, room stability constraint, and minimum working days constraint as well the curriculum compactness constraint.

The tuning heuristic starts by initializing the maximum number of unchanging state moves to be equal to the number of lectures *LecturesNum* multiplied by a tuning heuristic parameter *sConstCoefficient* (which is 2 or 3 depending on the testing).The algorithm sorts the soft constraints penalties in decreasing order and it selects the constraint with the highest penalty value and tries to work on improving the lectures' number of violations related to the chosen constraint by applying moves constructed specifically to handle it while taking into consideration the preservation of the feasibility of the solution. If the number of maximum unchanging moves reached, selects the second soft constraint penalty value in the list and repeat the same process. The algorithm will repeat these steps for minimum ten times before it terminates, and give us the best final tuned feasible solution.

CHAPTER 5

EMPIRICAL RESULTS AND DISCUSSIONS

5.1 Experimental Procedure

In this chapter, we present the results produced by the application of our three-phase heuristic algorithms approach on the curriculum based course timetabling problem (CCTP) and compare them with those produced by Muller's hybrid approach (Muller, 2008, 2009) because Muller's algorithm combined IFS, HC, GD and SA and he was the one who won the ITC2007 competition that determined the structure, constraints and restrictions of CCTP (Di Gaspero *et al.,* 2007).

We tested the performance of our three-phase heuristics algorithm and Muller's hybrid approach by applying them to five benchmark datasets/subject problems (SPs) obtained from ITC2007 (based on real data coming from Udine University in Italy) whose characteristics are shown in Table 5.1. Three runs have been done for each subject problem by each algorithm and the results are compared and evaluated in terms of four main metrics: the hard constraints violations value (HCV) which has to be zero for all, the soft constraints violations value (SCV), the number of objective function (OF) evaluations (OFEvals), and the execution time (ETime). Also, we keep track of the best value, worst value, average value, and the standard deviation value (SD) for each of the above metrics. To make sure that those results for both heuristics are correct, they are evaluated by the standard solution validator we talked about in Chapter 3, section 3.4, and which is provided by ITC2007 (Di Gaspero *et al.,* 2007).

In our approach, for the simulated annealing phase, we use *CoolingRate*=0.95, $T_f = \frac{-0.1}{\log 2^{-30}} = 0.011$, *UpHillMovesProbability*=0.93, *RepeatInnerLoopCoefficient*=10, *RepeatTInitialCoefficient*=50. For the scatter search phase, we use |*PopSize*|=100, *b*=|*RefSet*|=20, b_1=|*HQRefSet*|=10, b_2=|*DivRefSet*|=10, *DGMCONS*=2, and *IMPCONT*=2. Also we use, as problem-specific moves, the *SwapMove* for the diversification generation method, the *SwapMove*, *LectureMove*, *RoomMove*, *TimeMove*, *CourseRoomChangeMove*, *CourseMinDaysMove* and *CurriculumCompactnessMove* for the improvement method, and, the *ViolatedLectureMove* and *PeriodSwapMove* for the combination method. Also, the

following values for the coefficients of OF has been chosen: φ=50 for (S_{TC}), ψ=1 for (S_{CC}), ω=1 for (S_{avail}), γ =1 for (S_{ExtStd}), α=5 for (S_{MWD}), β=2 for ($S_{CurrComp}$), and μ=1 for ($S_{RoomStab}$) that have been set in accordance with the CCTP objectives where the teacher conflicts occupies the highest level in the order of importance among the hard constraints, and the minimum working days constraints occupies the highest level in the order of importance among the soft constraints.

The complexity of a subject problem does not come only from its number of lectures (L), but it also comes from: how many curricula (C) these lectures belong to, the number of unavailability constraints (A) that restricts the scheduling of some lectures over some periods (which are directly proportional to complexity), and the number of rooms (R) these lectures should be taught in (which is inversely proportional to complexity). Thus, the subject problem complexity is equal to $\frac{L*C*A}{R}$. For example, as shown in Table 5.1, *comp05* represents the hardest dataset among all the other subject problems although it has the smallest number of lectures, but it has the highest number of availability constraints as well as the highest density of courses/curricula; thus allowing us to assume that most of the complexity in this subject problem comes from curriculum compactness soft constraint. This measure of complexity of the problem instance helps in measuring the scalability of the approaches as the complexity metric of the subject problems increases. Then, comes the *comp02* subject problem where it comes in the second position with respect to complexity because it has a higher number of curricula and availability constraints than those in *comp03* and *comp04* where those came in third and fourth place according to their computed complexity value respectively. *Comp01* is the simplest subject problem we have.

5.2 Results and Discussions

Table 5.2 shows the results of our three-phase approach along with the results of Muller's hybrid approach for the five subject problems using the above mentioned parameters.

Table 5.1: Input Instances (Subject Problems) Properties.

Subject Problem	Name	# of Courses	# of Lectures	# of Teachers	# of Rooms	# of Days	Periods /day	# of Curricula	# of Unavailability Constraints	Complexity Measure
comp01	Fis0506-1	30	160	23	6	5	6	14	53	19,786.67
comp02	Ing0203-2	82	283	70	16	5	5	70	513	635,158.13
comp03	Ing0304-1	72	251	60	16	5	5	68	382	407,498.50
comp04	Ing0405-3	79	286	69	18	5	5	57	396	358,644.00
comp05	Let0405-1	54	152	46	9	6	6	139	771	1,809,965.33

Table 5.2: Three-Phase Heuristic Approach and Muller's Hybrid Approach Testing Results for CCTP

Subject Prob. (input)	Evaluation Metrics	3-Phase Approach (SA+SS+THEU) Output				Muller's Hybrid Approach (HC+GD+SA) Output			
		Worst	Best	Average	SD	Worst	Best	Average	SD
comp01	HCV	0	0	0.00	0.00	0	0	0.00	0.00
	SCV	5	**5**	**5.00**	0.00	5	**5**	**5.00**	0.00
	OFEvals	49,208,874	45,273,827	47,177,107.67	**1,970,667.44**	9,704,719	**4,892,767**	**6,777,867.00**	2,569,563.37
	ETime (sec)	3060.65	2513.50	2,853.42	296.73	11.00	6.50	9.31	2.45
comp02	HCV	0	0	0.00	0.00	0	0	0.00	0.00
	SCV	56	**45**	**51.00**	5.57	75	54	63.33	10.69
	OFEvals	75,545,222	**65,338,057**	**70,065,506.33**	**5,144,995.90**	199,636,884	80,475,784	122,422,738.33	66,952,767.13
	ETime (sec)	103,029.50	99,391.98	101,310.69	1,826.98	1,689.47	325.68	915.13	700.44
comp03	HCV	0	0	0.00	0.00	0	0	0.00	0.00
	SCV	83	**79**	**81.33**	2.08	93	81	87.00	6.00
	OFEvals	101,127,927	95,862,114	98,784,060.33	**2,680,079.98**	115,258,722	**35,692,114**	**65,553,460.00**	43,336,439.96
	ETime (sec)	37,432.54	30,218.78	34,587.56	3,840.71	698.58	88.42	424.41	309.74
comp04	HCV	0	0	0.00	0.00	0	0	0.00	0.00
	SCV	42	**35**	**38.33**	3.51	59	39	46.33	11.02
	OFEvals	159,362,140	145,127,927	151,450,727.00	**7,248,860.14**	212,033,452	**58,901,766**	**132,100,544.00**	76,787,627.22
	ETime (sec)	75,433.78	54,633.35	66,059.48	10,550.92	824.60	340.30	518.16	266.53
comp05	HCV	0	0	0.00	0.00	0	0	0.00	0.00
	SCV	326	**316**	**320.00**	5.29	352	324	338.67	14.05
	OFevals	178,463,113	**164,018,886**	**172,093,783.67**	**7,371,610.81**	405,172,537	278,725,430	328,250,583.67	67,528,927.40
	ETime (sec)	156,215.82	134,716.74	145,637.21	10,753.62	2,638.24	1,270.54	1,858.56	703.71

Chart 5.1, Chart 5.2, Chart 5.3, Chart 5.4, Chart 5.5, Chart 5.6, Chart 5.7 and Chart 5.8 show line and bar charts for the best and average values of soft constraints violations and for the best and average values of the number of OF evaluations. Based on those results, we make the following findings and observations:

a. It is clear that both algorithms (the three-phase approach and the Muller's Hybrid approach) produce solutions with zero hard constraints violations leading to completely feasible solutions.

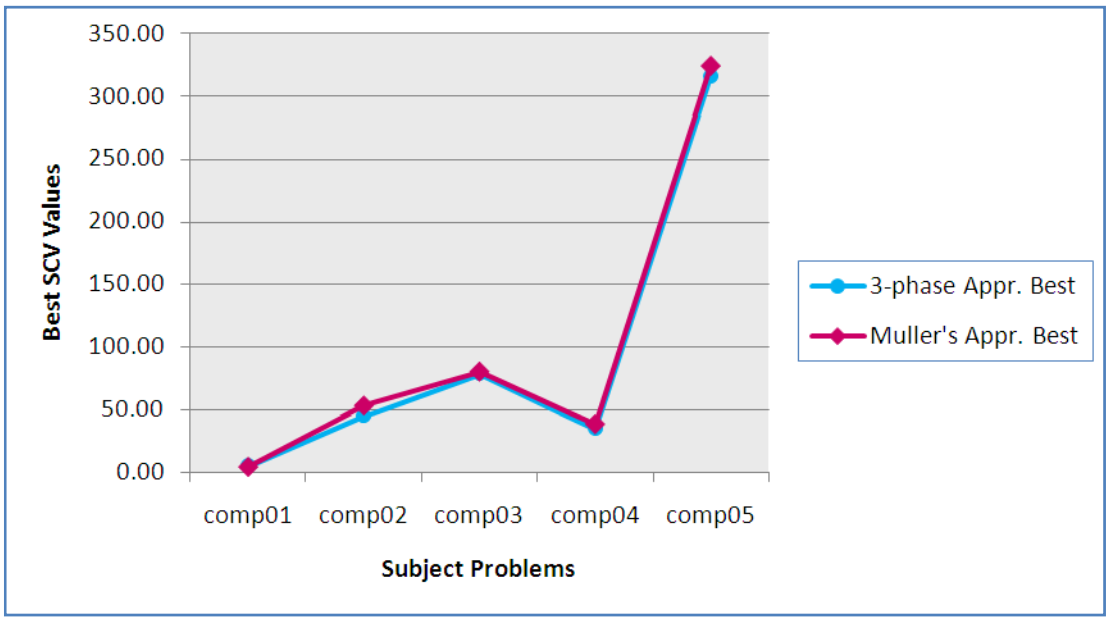

Chart 5.1: Line Chart for Best Soft Constraints Violations Values for Both Approaches

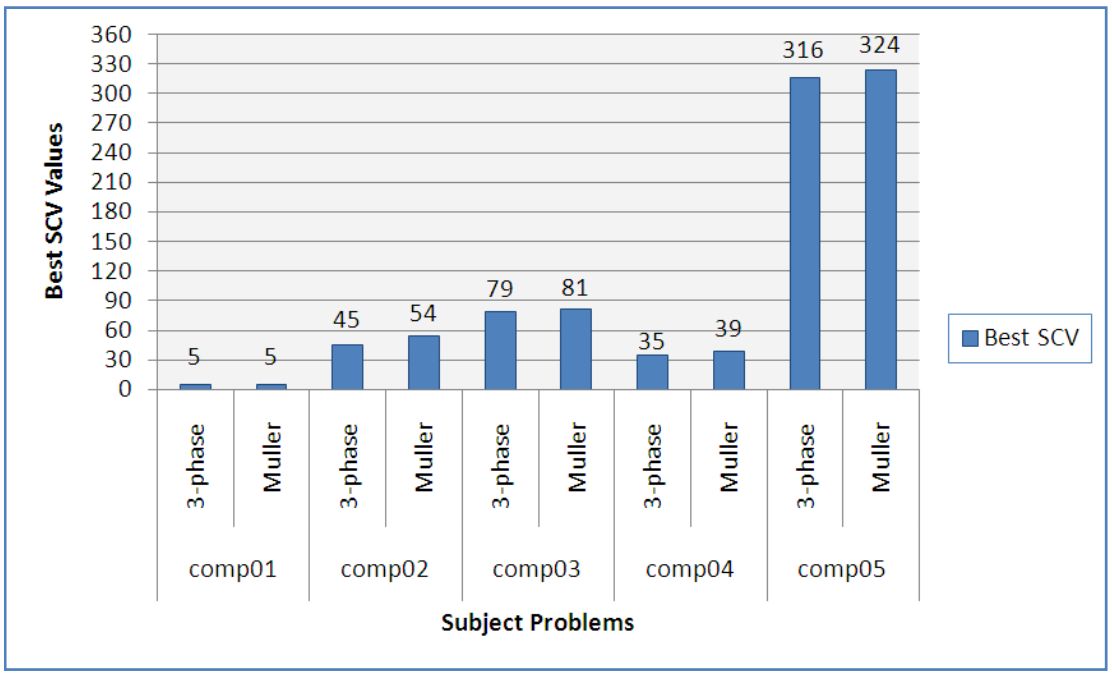

Chart 5.2: Bar Chart for Best Soft Constraints Violations Values for Both Approaches

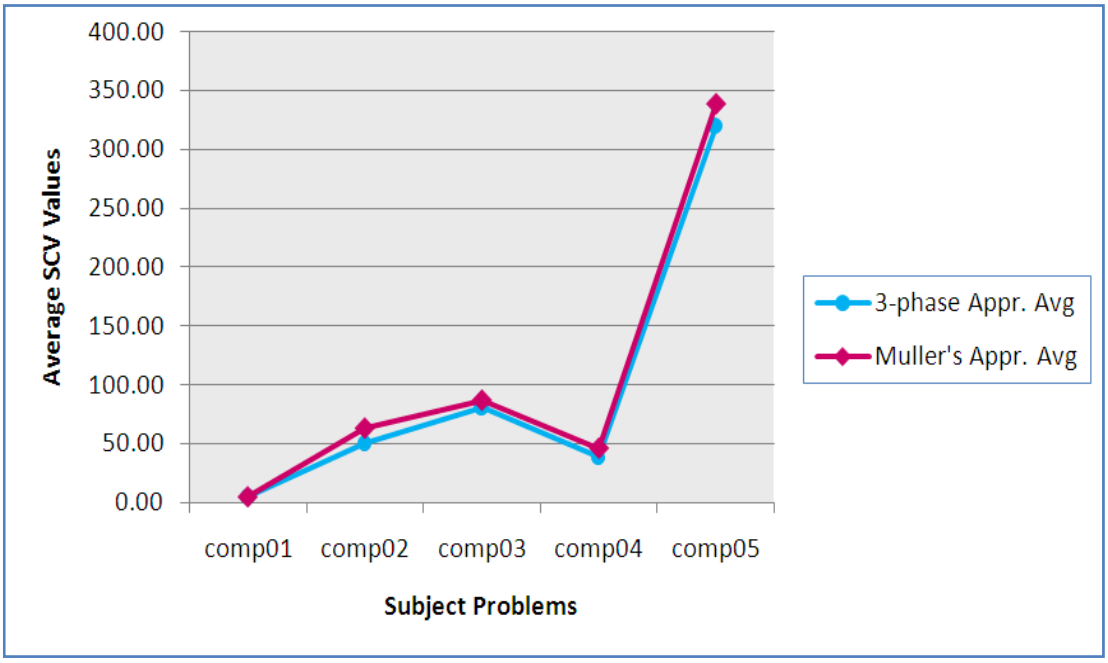

Chart 5.3: Line Chart for Average Values of Soft Constraints Violations for Both Approaches

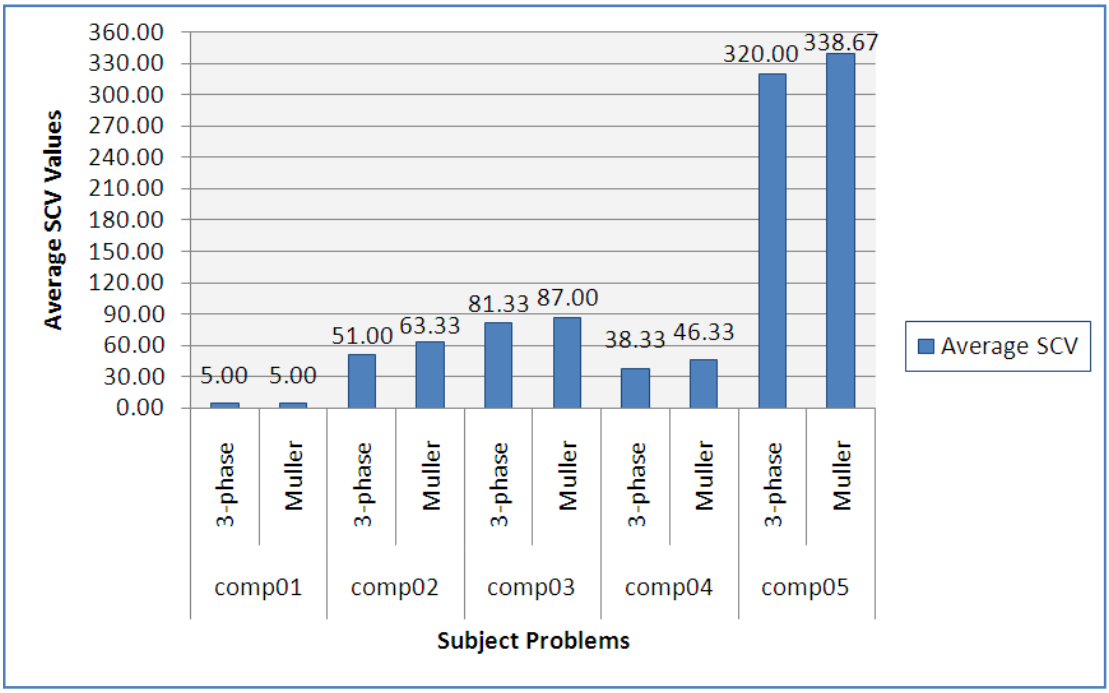

Chart 5.4: Bar Chart for Average Values of Soft Constraints Violations for Both Approaches

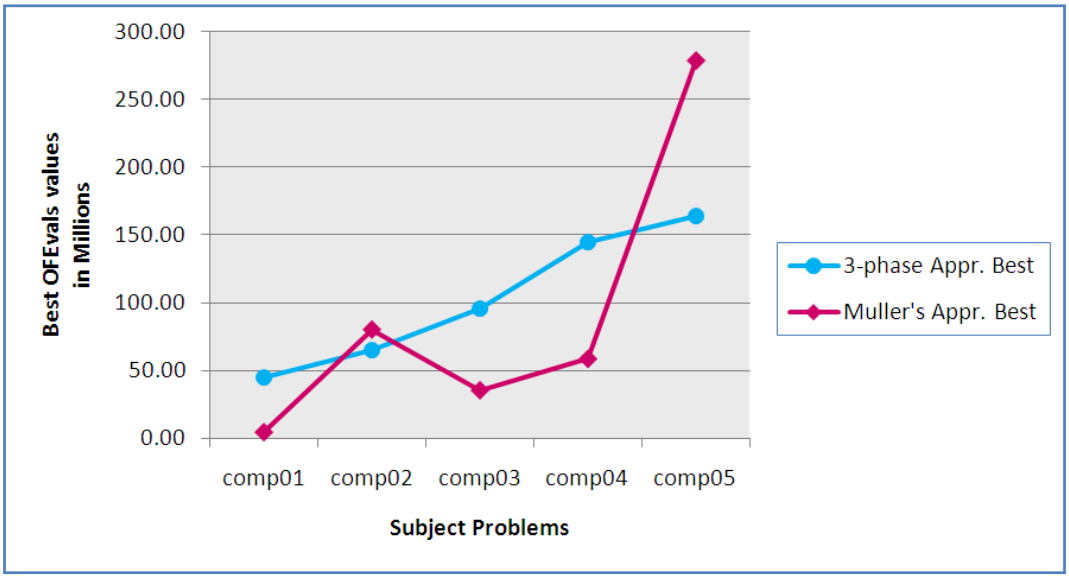

Chart 5.5: Best Values of OFEvals for Both Approaches

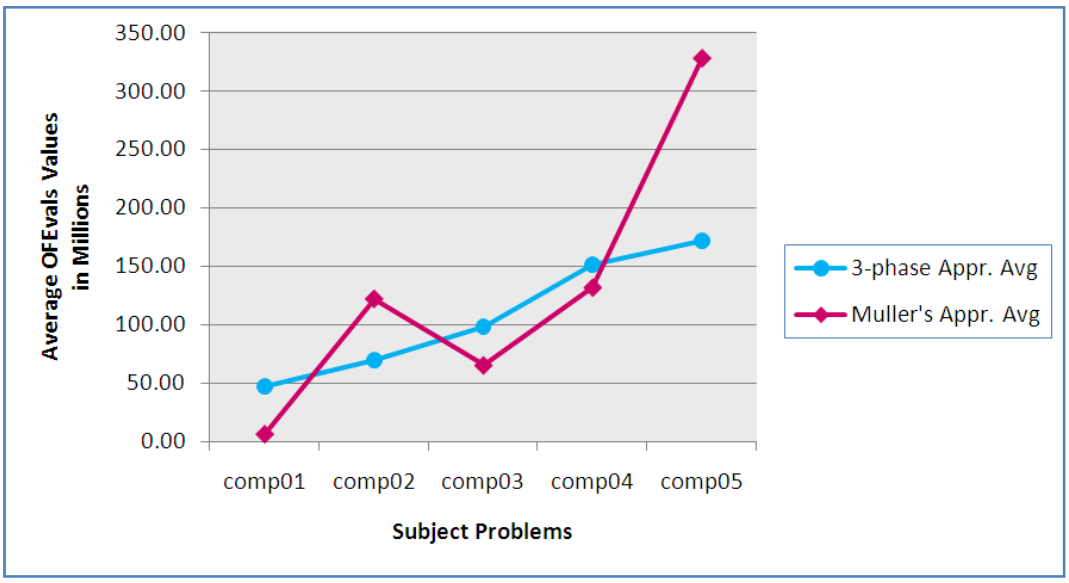

Chart 5.6: Average Values of OFEvals for Both Approaches

b. Our three-phase approach produced good timetables. This observation is based on the low values obtained in the evaluation metrics (HCV and SCV) in comparison to those exists in literature (as mentioned in Chapter 2, section 2.3) and specifically in comparison with those of Muller's approach (Chapter 2, section 2.3.1). Our approach shows an improvement by an average of 9.75% in the soft constraints violations value (SCV) for the five subject problems over Muller's hybrid approach where it improved the average SVC value of *comp02* by 19.47%, of *comp03* by 6.51%, of *comp04* by 17.27%, and of *comp05* by 5.51%; while for *comp01* both algorithms produced the same results values as we can see in Chart 5.2. From Chart 5.1,

we can clearly see that our algorithm did also better in four out of the five subject problems in terms of the best SCV value; For example, our approach improved the best SVC value of *comp02* by 116.67%, of *comp03* by 2.47%, of *comp04* by 10.26%, and of *comp05* by 2.47%; while for *comp01* both algorithms has the same best value.

c. On the other hand, our three-phase approach has a worse execution time by an average of 99.1% over that of Muller's approach. But this is implementation-dependent and can be alleviated using good partial calculation methods of the soft constraints OF at the second phase in our approach, as well as by parallelizing the algorithm. But, what is more important in this comparison is to consider the number of OF evaluations.
d. *Chart 5.3* shows that our approach has a number of OF evaluations better in two subject problems (*comp02* and *comp05*) which are the hardest subject problems we have. Thus, we notice that as the subject problem gets more complex, our three-phase approach gives a better and an improved number of OF evaluations which is 47.6% less in the average *OFEvals* for *comp05* subject problem than the result of Muller's approach as shown in Chart 5.4. We also got a 42.8% decrease in the average *OFEvals* for *comp02* over Muller's average *OFEvals*. However, as the subject problem gets simpler, the average value of *OFEvals* of our approach gets larger and exceeds that of Muller's by 85.6% in the case of *comp01* and by 12.7% for *comp04*. These observations help in showing the scalability of our three-phase algorithm for medium and high complex problems. Moreover, if we examine the overall number of compared runs, our approach performs better in *OFEvals* by 76.67% of the time and worse in 23.33% of the time.
e. The calculation of standard deviation (SD) helps in estimating how much the produced values are far from the mean average. A small SD indicates that the values studied tend to be close to the mean value, while high SD shows that the values are spread over a large range of values. From Chart 5.5 and Chart 5.6, we infer the stability of our algorithm in comparison with that of Muller. Though this still cannot be guaranteed with only three runs, the preliminary testing results showed the great performance of our algorithm especially with respect to the SD calculated over the *OFEvals*. The high SD values of Muller's algorithm shows that this approach might at

some runs produce very good values, while in the other runs might produce very bad values.

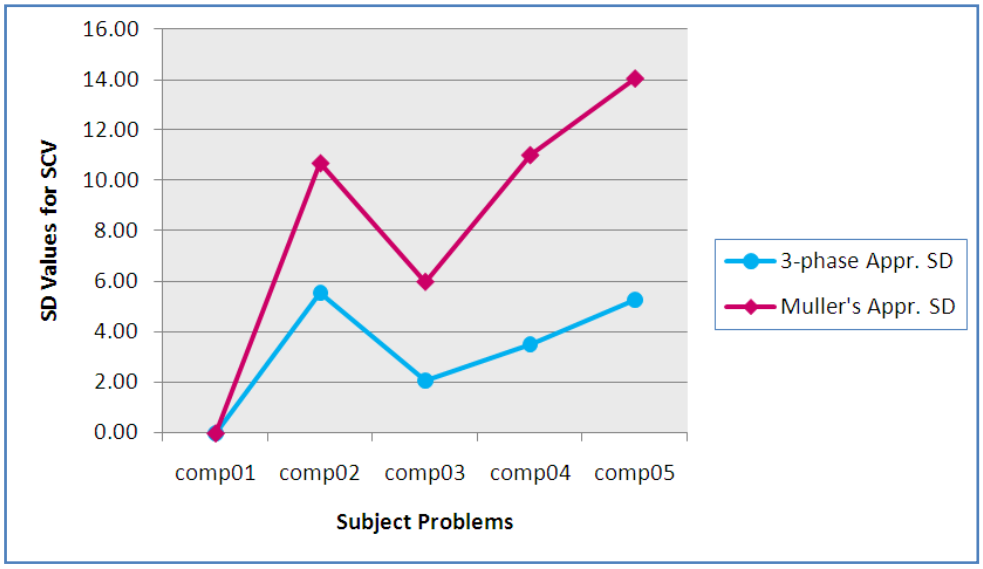

Chart 5.7: SD Values for SCV

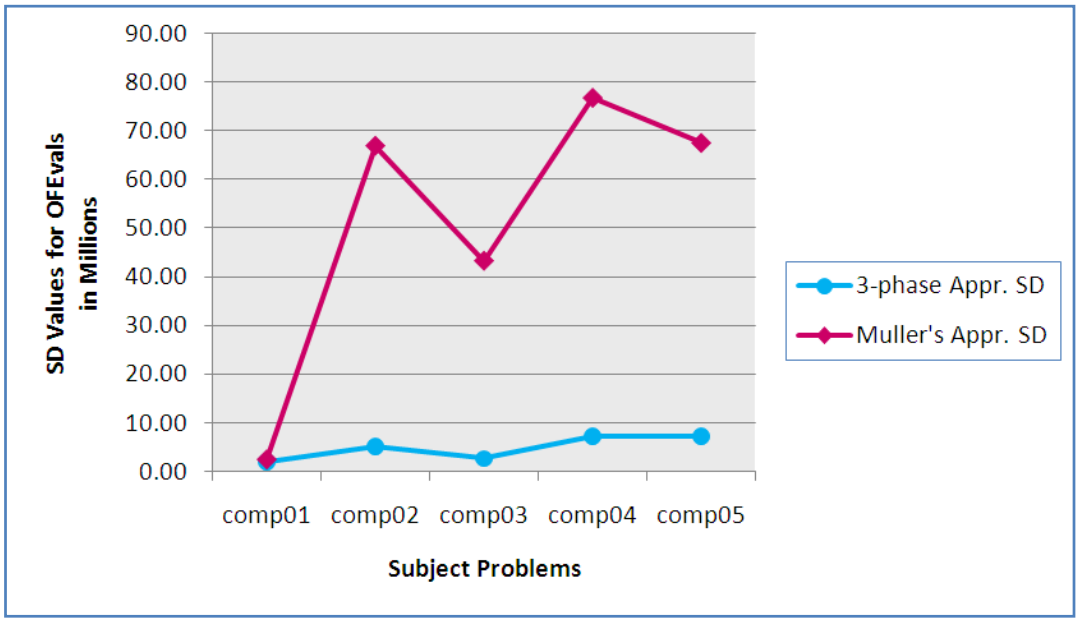

Chart 5.8: SD Values for OFEvals

CHAPTER 6

CONCLUSION AND FUTURE WORK

We have proposed a three-phase heuristics approach to produce good suboptimal timetables for curriculum based course timetabling problems. We have tested this approach on five benchmark datasets whose constraints and objectives were provided by the international timetabling competition, ITC2007. Our results are compared with those of the ITC2007's winner, Muller's Hybrid approach, proposed to solve the same problem. Our experimental results show that our method produces feasible and good course timetables that fulfill the requirements of zero hard constraints violations and minimal values for soft constraints. Also, our results were found to be better than Muller's for larger or more complex problems with respect to the soft constraints violations values and the number of OF evaluations. Our method yields lower soft constraints score on average for four problems out of five and for four problems in terms of the best out of the three runs. Also, our method yields lower score of objective function evaluations on average for two problems out of five and for two problems in terms of the best number of OF evaluations out of the three runs. The results of the three runs reveal more stability in our method as shown by the smaller standard deviation values produced from it.

Future work would explore the following research directives:

a. Studying the effects of changing our heuristics' parameters and constant coefficients over the overall solution value like the number of maximum iterations needed for the improvement method;
b. Improving the way in which the problem specific moves (PSMs) are selected in the improvement method aiming to make our approach more systematic by making their work to be directed to solve the conflicts directly;
c. Enhancing the required time of execution of our algorithm by improving the methods used to partially calculate the soft constraints penalty value as well as parallelizing the work of the heuristics specifically for scatter search which costs 90% of the running time.
d. Increasing confidence in the results of our algorithms by performing at least ten experimental runs for each of the twenty one subject problems provided by ITC2007.

Another major issue related to the ITC2007 competition is the simplification of the CCTP formulation. Di Gaspero *et al.* (2007) said that this oversimplification was due to their desire to get a certain level of generality and not to burden the competitors with understanding all the features of the problem and to let them concentrate on the core issue. However, by doing so, they missed some features that might need to be included in every real world course timetabling problem and they over simplified the problem in a way that we cannot be sure of how far the scalability the submitted solutions can be. For example, in university course timetabling problems, there might be some teacher preferences over time and room, but these were not actually modeled by neither Muller nor by the competition CCTP formulation although this is an important feature to be included. Also, the days' timeslots are of equal length while in real life they might be of different duration along specific pattern of days (MWF, TTh, …etc). At last, we can enhance this work by tuning the problem specific parameters (neighborhoods such as creating new kinds of moves, heuristics parameters, …etc) and on editing the problem specific constraints, objectives and requirements so that it can fit the university course timetabling problem model followed at, for example, the Lebanese American University.

REFERENCES

Abdule-Wahab, R., Monmarché, N., Slimane, M., Fahdil, M. A. & Saleh, H. H. (2006). A scatter search algorithm for the automatic clustering problem. In P. Perner (Series Ed.), *Lecture notes in computer science*: *Vol. 4065*. *Advances in data mining, applications in medicine, web mining, marketing, Image and signal mining* (pp. 350-364). Berlin, Heidelberg: Springer.

Abdullah, S. & Hamdan, A. R. (2008). A hybrid approach for university course timetabling. *International Journal of Computer Science and Network Security, IJCSNS*, *8*(8), 127-131.

Abdullah S., Shaker K., Shaker, H. (2011). Investigating a round-robin strategy over multi-algorithms in optimizing the quality of university course timetables. *International Journal of the Physical Sciences*, *ISI Journal*, *6*(6), 1452-1462.

Abuhamdah, A. & Ayob, M. (2009, October). *Multi-neighbourhood particle collision algorithm for solving course timetabling problems*. Paper presented at the 2nd Conference on Data Mining and Optimization, DMO'09, Selangor, Malaysia.

Alvarez, A., González-Velarde, J. L. & Alba, K. D. (2005). Grasp embedded scatter search for the multicommodity capacitated network design problem. *Journal of Heuristics*, *11*(3), 233-257.

Atan, T., & Secomandi, N. (1999). *A rollout-based application of the scatter search/path relinking template to the multi-vehicle routing problem with stochastic demands and restocking*. Technical Report, PROS Revenue Management Inc., Houston, TX.

Baldacci, R., Boschetti, M. A., Maniezzo, V. & Zamboni, M. (2005). Scatter search methods for the covering tour problem. In R. Sharda, S. Vob, C. Rego & B. Alidaee, (Eds.). *Computer science interfaces: Vol. 30. Metaheuristic optimization via memory and evolution, operations research* (pp. 59-91). US: Springer.

Blanco, R., Tuya, J. & Adenso-Díaz, B. (2009). Automated test data generation using a scatter search approach. *Information and Software Technology, 51*(4), 708-720.

Boughaci, D., Benhamou, B. & Drias, H. (2008). Scatter search and genetic algorithms for MAX-SAT Problems. *Journal of Mathematical Modelling and Algorithms, 7*(2), 101-124.

Burke, E. K., Jakub, M. Parkes, A. J. & Rudova, H. (2010). A super nodal formulation of vertex coloring with applications in course timetabling. *Annals of Operations Research, 179*(1), 105-130.

Cordón, O., Damas, S., Santamaría, J. & Martí, R. (2005). 3D inter-subject medical image registration by scatter search. In M. Blesa, C. Blum, A. roli & M. Sampels (Eds.), *Lecture notes in computer science: Vol. 3636*. *Hybrid metaheuristics* (pp. 90-103). Berlin, Heidelberg: Springer.

Chu, S. C., Chen, Y. T. & Ho, J. H. (2006). Timetable scheduling using particle swarm optimization. *Proceedings of the First International Conference on Innovative Computing, Information and Control, ICICIC'06*. Doi: 10.1109/ICICIC.2006.541.

Chu, S. C. & Fang, H. L. (1999, September). *Genetic algorithms vs. tabu search in timetable scheduling*. Paper presented at the 3rd International Conference on Knowledge-Based Intelligent Information Engineering Systems: Adelaide, SA.

Clark, M., Henz, M. & Love, B. (2008). QuikFix: A repair-based timetable solver. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT'08*. Montreal, Canada. Doi: 10.1.1.150.7677.

Dandashi, A. & Al-Mouhamed, M. (2010). Graph coloring for class scheduling. *Proceedings of the 2010 ACS/IEEE International Conference on Computer Systems and Applications*, *AICCSA'10*. Doi: 10.1109/AICCSA.2010.5586963.

Debels, D., De Reyck, B., Leus, R. & Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research, 169*(2), 638-653.

De Cesco, F., Di Gaspero, L. & Schaerf, A. (2008). Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, and results. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT'08*. Montreal, Canada.

Di Gaspero, L., McCollum, B., & Schaerf, A. (2007). *The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (Track 3)*. Technical Report, QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0, Queen's University, United Kingdom: Belfast. Doi: 10.1.1.62.1880.

Drias, H. (2001). Genetic algorithm versus scatter search and solving hard MAX-W-SAT problems. In J. Mira & A. Prieto (Eds.), *Lecture notes in computer science: Vol. 2084. Connectionist models of neurons, learning processes, and artificial intelligence* (pp. 586-593). Berlin, Heidelberg: Springer.

Dueck, G. (1993). New optimization heuristics: The great deluge algorithm and the record to record travel. *Journal of Computational Physics, 104*, 86-92.

Egea, J. A., Rodríguez-Fernández, M., Banga, J. R. & Martí, R. (2007). Scatter search for chemical and bio-process optimization. *Journal of Global Optimization Archive, 37*(3), 481-503.

Ghaemi, S., Vakili, M. T. & Aghagolzadeh, A. (2007). Using a genetic algorithm optimizer tool to solve university timetable scheduling problem. *Proceedings of the 9th International Symposium in Signal Processing and its Applications, ISSPA 2007*. Doi: 10.1109/ISSPA.2007.4555397.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences, 8*, 156-166.

Glover, F. (1994). Genetic algorithms and scatter search: Unsuspected potentials. *Statistics and Computing, 4*, 131-140.

Glover, F. (1998). A template for scatter search and path relinking. In J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer & D. Snyers (Eds.), *Lecture notes in computer science: Vol. 1363. Artificial Evolution* (pp.13-54). Berlin, Heidelberg: Springer.

Glover, F., Laguna, M. & Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics, 29*(3), 653-684.

Glover, F., Laguna, M. & Martí, R. (2002). Scatter search. In A. Ghosh & S. Tsutsui (Eds.). *Theory and applications of evolutionary computation: Recent trends* (pp. 519–529). Berlin, Heidelberg: Springer.

Glover, F., Laguna, M. & Martí, R. (2003). Scatter search and path relinking: Advances and applications. In F. Glover & G. Kochenberger (Eds.). *Handbook of metaheuristics: Vol. 57* (pp. 1-35). Boston: Kluwer Academic Publishers.

Glover, F., Laguna, M. & Martí, R. (2004). New ideas and applications of scatter search and path relinking. In G. C. Onwubolu & B. V. Babu (Eds.), *Studies in fuzziness and soft computing: Vol. 141. New optimization technologies in engineering* (pp. 367-384). Berlin: Springer.

Haidar, G. (2009). *Parallel scatter search algorithm for exam timetabling* (Unpublished Master's Thesis). Faculty of Arts and Sciences, Department of Computer Science and Mathematics. Lebanese American University, Lebanon.

Hanafi, S. & Wilbaut, C. (2008). Scatter search for the 0–1 multidimensional knapsack problem. *Journal of Mathematical Modelling and Algorithms*, *7*(2), 143-159.

International timetabling competition 2007. (2008, October 1). Retrieved from: http://www.cs.qub.ac.uk/itc2007/

Irene, S. F. H., Deris, S., & Mohd Hashim, S. Z. (2009). A combination of PSO and local search in university course timetabling problem. *Proceedings of the 2009 International Conference on Computer Engineering and Technology*, 492-495. Doi: 10.1109/ICCET.2009.188.

Isahakian, V. (2006). A *scatter search algorithm for exam timetabling* (Unpublished Master's Thesis). Faculty of Arts and Sciences, Department of Computer Science and Mathematics, Lebanese American University, Lebanon.

Kehyayan, C. (2008). *Scatter search for protein structure prediction* (Unpublished Master's Thesis). Faculty of Arts and Sciences, Department of Computer Science and Mathematics, Lebanese American University, Lebanon.

Kelly, J., Rangaswamy, B. & Xu, J. (1996). A scatter-search-based learning algorithm for neural network training. *Journal of Heuristics, 2*(2), 129-146.

Kennedy, J. & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of the IEEE International Conference on Neural Network*. 1942-1948. Doi: 10.1109/ICNN.1995.488968.

Kirkpatrick, S.. Gelatt, C. D. & Vecchi, M. P. (1983, May 13). Optimization by simulated annealing. *Science, 220*(4598), 671-680.

Laguna, M., Lourenço, H. & Martí, R. (2000). Assigning proctors to exams with scatter search. In M. Laguna & J. L. González-Velarde, (Eds.), *Interfaces in computer science and operations research: Vol. 12. Computing tools for modeling, optimization and simulation* (pp. 215–227). US: Kluwer Academic Publishers.

Laguna, M., & Martí, R. (2003). *Scatter search methodology and implementations in C*. Dordrecht: Kluwer Academic Publishers.

Liu, Y., Zhang, D. & Leung, S. C. H. (2009). A simulated annealing algorithm with a new neighborhood structure for the timetabling problem. *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation, GEC'09*, Shanghai, China, 381-386. Doi: 10.1145/1543834.1543885.

Maenhout, B. & Vanhoucke, M. (2010). A hybrid scatter search heuristic for personalized crew rostering in the airline industry. *European Journal of Operational Research, 206*(1), 155-167.

Mansour, N., Isahakian, V. & Ghalayini, I. (2011). Scatter search technique for exam timetabling. *Journal of Applied Intelligence, 34*(2), 299-310.

Mansour, N., Kehyayan, C. & Khachfe, H. (2009). Scatter search algorithm for protein structure prediction. *International Journal of Bioinformatics Research and Applications, 5*(5), 501-515.

Mansour, N. & Timani, M. (2007). Stochastic search algorithms for exam scheduling. *International Journal of Computational Intelligence Research, 3*(4), 353–361.

Martí, R., Laguna, M. & Campos, V. (2005). Scatter search vs. genetic algorithms: An experimental evaluation with permutation problems. In C. Rego & B. Alidaee (Eds.). *Metaheuristic optimization via adaptive memory and evolution: Tabu search and scatter search* (pp. 263–282). Dordrecht: Kluwer Academic Publishers.

Martí, R., Glover, F., & Laguna, M. (2006). Principles of scatter search. *European Journal of Operational Research, 169*, 359-372.

Massoodian, S. & Esteki, A. (2008). A hybrid genetic algorithm for curriculum based course timetabling. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT'08*. Montreal, Canada.

McMullan, P. (2007). An extended implementation of the great deluge algorithm for course timetabling. In Y. shi, G. Van Albada, J. Dongarra & P. Sloot (Eds.), *Lecture notes in computer science: Vol. 4487. Computational science, ICCS2007* (pp. 538-545), Heidelberg, Berlin: Springer.

Muller, T., Barták, R., & Rudová, H. (2004). Conflict-based statistics. In J. Gottlieb, D. Landa Silva, N. Musliu, & E. Soubeiga (Eds.), *EU/ME workshop on design and evaluation of advanced hybrid metaheuristics* (pp. 5). Nottingham, Uk: University of Nottingham.

Muller, T. (2005). *Constraint-based timetabling* (Unpublished doctoral dissertation). Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic.

Muller, T., Barták, R., & Rudová, H. (2005). Minimal perturbation problem in course timetabling. In E. Burke & M. Trick (Eds.), *Lecture notes in computer science: Vol. 3616. Practice and theory of automated timetabling* (pp. 126–146). Berlin: Springer.

Muller, T., Murray, K. & Schluttenhofer, S. (2007). University course timetabling and student sectioning system (system demonstration). *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS'07.* Rhode Island, USA.

Muller, T. (2008). ITC2007 solver description: A hybrid approach. *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT'08*. Montreal, Canada.

Muller, T. (2009). ITC2007 solver description: A hybrid approach. *Journal of Annals of Operations Research, 172*(1), 429-446.

Murray, K., Muller, T., & Rudová, H. (2007). Modeling and solution of a complex university course timetabling problem. In E. Burke & H. Rudová (Eds.), *Lecture notes in computer science: Vol. 3867. Practice and theory of automated timetabling* (pp. 189–209). Berlin: Springer.

Nandhini, M. & Kanmani, S. (2009*)*. Implementation of class timetabling using multi-agents. *Proceedings of the 2009 International Conference on Intelligent Agent & Multi-Agent Systems, IAMA 2009*. Doi: 10.1109/IAMA.2009.5228065.

Nepomuceno, J. A., Troncoso, A. & Aguilar-Ruiz, J. S. (2010). Correlation-based scatter search for discovering biclusters from gene expression data. In C. Pizzuti, M. D. Ritchie, M. Giacobini (Eds.), *Lecture notes in computer science: Vol. 6023*. *Evolutionary computation, machine learning and data mining in bioinformatics, 8th european conference, EvoBIO* (pp. 122-133). Berlin, Heidelberg: Springer.

Ozcan, E. & Alkan, A. (2002). Timetabling using a steady state genetic algorithm. *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling, PATAT'02*. 104-107.

Pantrigo, J. J., Duarte, A., Sánchez, Á. & Cabido, R. (2005). Scatter search particle filter to solve the dynamic travelling salesman problem. In G. Raidl & J. Gottlieb (Eds.), *Lecture notes in computer science: Vol. 3448*. *Evolutionary computation in combinatorial optimization* (pp. 177-189). Berlin, Heidelberg: Springer.

Pinol, H. & Beasley, J. E. (2006). Scatter search and bionomic algorithms for the aircraft landing problem. *European Journal of Operational Research, 171*(2), 439-462.

Qu, R. & Burke, E.K. (2009). Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society, 60*, 1273-1285.

Ranjbar, M. & Kianfar, F. (2009). A hybrid scatter search for the RCPSP. *Transaction E: Industrial Engineering, 16*(1), 11-18.

Rego, C. & Leao, P. (2000). *A scatter search tutorial for graph based permutation problems*. Technical Report, Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi: USA.

Russell, R. A. & Chiang, W. C. (2006). Scatter search for the vehicle routing problem with time windows. *European Journal of Operational Research, 169*(2), 606-622.

Sacco, W. F. & de Oliveira, B. (2005). A new stochastic optimization algorithm based on particle collisions. *Transactions of the American Nuclear Society, 92*, 65–69.

Saravanan, M., Haq, A. N., Vivekraj, A. R. & Prasad, T. (2008). Performance evaluation of the scatter search method for permutation flowshop sequencing problems. *The International Journal of Advanced Manufacturing Technology, 37*(11-12), 1200-1208.

Shaker, K. & Abdullah, S. (2009). Incorporating great deluge approach with kempe chain neighbourhood structure for curriculum-based course timetabling problems. *Proceedings of the 2nd Conference on Data Mining and Optimization*, Selangor, Malaysia,149–153. Doi: 10.1109/DMO.2009.5341894.

Socha, K., Sampels, M. & Manfrin, M. (2003). Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In G. Raidl et al. (Eds.), *Lecture notes in computer science: Vol. 2611. Applications of evolutionary computing* (pp. 334–345). Berlin: Springer.

Talbi, G. (2009). *Metaheuristics: From design to implementation*. Canada: A John Wiley & Sons*, INC*.

Yamashita, D.S., Armentano, V.A. & Laguna, M. (2004). Scatter search for project scheduling with resource availability cost. *European Journal of Operational Research, 169*(2), 623–637.

Zalmiyah, Z. (2001). *Case-based reasoning approach for reactive timetabling* (Unpublished Master's Thesis). Faculty of Computer Science and Information System, University of Technology, Malaysia.

APPENDIX I

BACKGROUND ON SCATTER SEARCH

In this appendix, we will give some introductory background and history information about scatter search (SS), describe its template, main features and basic design, as well as compare it to other evolutionary approaches like Genetic Algorithms (GAs). In our approach, we decided to use Scatter Search in the second phase during the solving of curriculum-based course timetabling problem.

I.1 Historical Background

Scatter search (SS) is an interesting evolutionary search strategy. It is a population-based meta-heuristic algorithm that was successfully used to solve different hard optimization problems. Its original formulations goes back to 1970s when its basic principles and concepts were first proposed based on the concepts of combining existing solutions or decision rules and problem constraints, which date back to 1960s, to produce new solutions better than the original ones (Laguna and Marti, 2003; Glover, Laguna and Marti, 2000, 2002, 2003). Scatter Search evolved over the years to become what it is right now.

The first officially published description of SS was proposed by Glover in 1977 as a heuristic for integer programming and he initially described it as a method that uses a succession of coordinated initializations to create new solutions; which means that, according to that initial proposal, solutions are non-randomly constructed to take into consideration the properties in different parts of the solution space where *SS* adjusts its search and exploration systematically with respect to a set of reference solutions (Laguna and Marti, 2003; Glover, Laguna and Marti, 2004). Until 1990s, *SS* procedure was coupled with Tabu Search (*TS*) and was applied on continuous (non-linear) optimization, graph-partitioning problem and binary permutations problem where several new voting mechanisms as well as new adaptive methods to dynamically change the linked weights were introduced. In 1998, Glover finally published his famous SS template that became the major and main reference for most of *SS* implementations up till now (Laguna and Marti, 2003; Isahakian, 2006).

I.2 Scatter Search Template and Basic Design

In 1998, Glover introduced the first template and outline of SS approach. SS can be generally be sketched as follows (Glover *et al.*, 2002; Laguna and Marti, 2003; Kehyayan, 2008):

1. At the first step, SS generates an initial set of solutions by employing specific strategies like the use of frequency-based memory that can produce solutions spanning the whole search space and using parameters from the pool of solutions itself in an equally balanced manner in order to guarantee certain level of diversity and randomness. The solutions produced at this step, and at any other step in SS, may not be feasible allowing the algorithm to explore more choices in the search space. This is what is so called diversification generation method of SS.

2. However, the use of some heuristic methods is very important in improving the new solutions generated in order to get better solutions in terms of quality as well as diversity. Therefore, an improvement method is needed in order to improve every generated solution (either from Step 1 or from the following step 5). The implementation for this method is problem-specific, and the solution generated after applying it might be feasible or not (though most probably it produces feasible solutions) depending on the problem context and requirements.

3. SS works on a quite small set of "good" solutions that is named the reference set*, RefSet*. *SS* builds *RefSet* by selecting a small set of "best" solutions from the initially constructed diverse population set. Best solutions in this context does not mean solutions with only good objective function values, so called high quality solutions, but also solutions that can introduce a certain diversity level to the overall set of reference solutions in order to exploit the search space in a better way. During SS, *RefSet* cycle is divided into two major stages where in the first one, *RefSet* of size *b* will be initially constructed, while on later stages, it will be updated with new solutions all using reference set update method (RSUM). In the first initial stage (iteration) that comes after the construction of the initial set of solutions, b_1 solutions of high quality will be chosen from initial population, and it will be added to *RefSet*. Then, the remaining b_2 solutions

will be also selected from the first initial solutions' population, such that, they should be diverse from the high quality b_1 previously selected solutions. Therefore, the resulting reference set, *RefSet*, of size b = b_1 + b_2, which is, in most of the cases, is 10-20% of the initial population size. In the following stages (or iterations), *RefSet* will be updated again to include best solutions generated from the combination method discussed in Step 5.

4. SS uses different operators to manipulate the reference set. In addition to the improvement operator represented in step 2, SS applies the combination method to combine two or more different solutions together to lead to the construction of another new solution. The selection of solutions to be combined is not a random process. Subsets of solutions to be combined have to be generated using the subset generation method that will decide which solutions to be combined. In every iteration, different subsets of different types (depending on the method design) can be generated. Then, their solutions will be combined on condition of not including the same subset in later iterations.

5. At last, a solution combination method is applied to systematically combine solutions in the set of subsets produced by the subset generation method (SGM) in order to produce new trial solutions with which *SS* will try to update *RefSet* using RSUM we previously talked about in step 3. The implementation for this method is problem-specific. Some problems uses linear, score-based or voting-based CM*s* depending on the problem context like unconstrained nonlinear optimization problems, linear ordering problem (LOP), and 0-1 knapsack problem respectively (Laguna and Marti, 2003).

There are three major notable features in Scatter Search template (Laguna and Marti, 2003; Glover *et al.*, 2002, 2003; Kehyayan, 2008):

- *SS* is designed to apply the improvement method not only to try to improve the value of the soft constraints penalty of the solution, but the standard (IMP) can work on infeasible solutions and tries to remove the hard constraints violations from them to return them to their feasibility state in order to check their introduction to *RefSet*.

- Subset combination method will assure combinations of solutions "within clusters" and "across clusters" by combining, for example, two high quality solutions with a diverse solution resulting in a new different solution that could be from a completely different cluster. This is one of the most important key features of Scatter Search, which helps in searching the domain space of solutions exhaustively.
- The implementation of the combination method is problem-specific. On the other hand, there is a common feature between all *CMs* applied for different problems is that they extrapolate regions and parameters that are not taken into consideration in the existing set of solutions or in the pool of possible solutions. Not only this, but the combination process is a structured and systematic process rather than a random or probabilistic one.

Therefore, in general, Scatter Search approach combines features and methods from different meta-heuristic approaches and methods. Scatter Search is a very flexible approach that can be customized easily to satisfy any problem's needs such that each of the Scatter Search operators can be adapted in different ways and in different degrees of complexity. However, all the advanced implementations of *SS* are built on the basis of the basic implementation of Scatter Search as shown in *Figure I.1*.

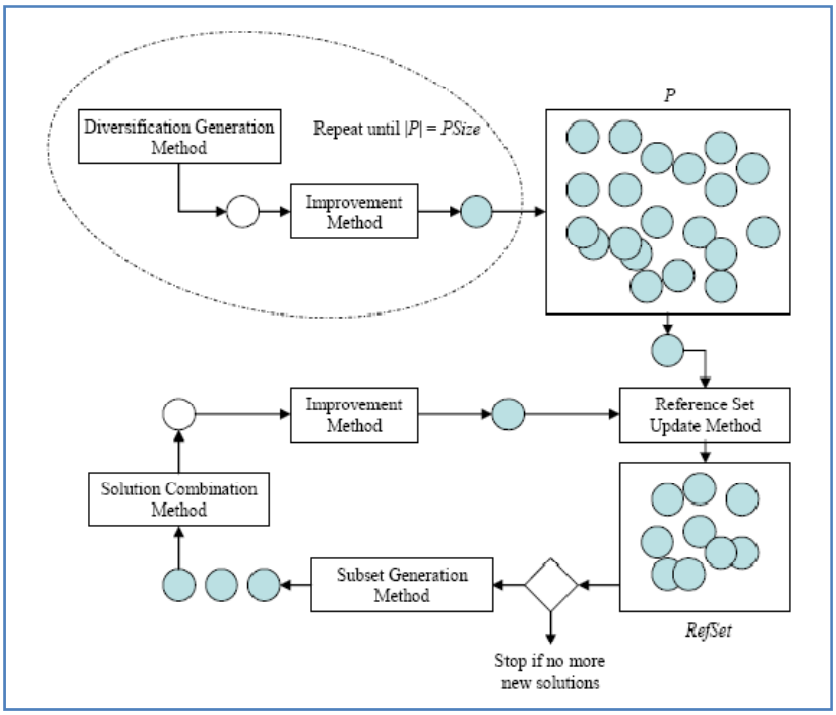

Figure I.1: Schematic Diagram of the Basic SS Algorithm Design

Figure 38 is taken from Laguna and Marti's book (2003), and it mainly demonstrates how the five basic methods (operators) of scatter search interact among each other, and it clearly sheds the light on the essential role of the reference set.

In relation to what we discussed previously about Scatter Search template and as shown in *Figures 38 and 39*, SS approach starts by creating the initial solutions' set *P* by applying the DGM. Then, the IMP will be applied on each solution in *P* to produce an improved one. Solutions are depicted as circles in *Figure 38*, while the dark circles stands for the improved solutions.

1. Start with P= Ø. Use the *Diversification Generation Method* to construct a solution and apply the *Improvement Method* to it. Let *x* be the resulting improved solution. If $x \notin P$ then $P = P \cup x$, otherwise, discard *x*. Repeat this step until $|P|=PSize$.
2. Use the *Reference Set Update Method* to build RefSet = $\{x^1, \ldots, x^b\}$ with the "*best*" *b* solutions from *P* and sort them according to their objective function values such that x^1 is the best solution and x^b is the worst.

   Set *NewSolutions* = TRUE.

while (*NewSolutions*) **do**

3. Generate *NewSubsets* with the *Subset Generation Method* from solutions in RefSet. Make *NewSolutions* = FALSE.

while (*NewSubsets* ≠ Ø) **do**

4. Select the next subset *s* in *NewSubsets*.
5. Apply *Solution Combination Method* to *s* to obtain one or more new trial solutions *x*. Apply the *Improvement Method* to the trial solutions.
6. Apply the *Reference Set Update Method*.

if (*RefSet* has changed) **then**

7. Make *NewSolutions* = TRUE.

end if

8. Delete *s* from *NewSubsets*.

end while

end while

Figure I.2: Scatter Search Algorithm Basic Outline

Then, SS applies the RSUM in order to begin with creating the initial reference set, *RefSet*, from population *P*. The size of *RefSet* is conventionally 10 to 20% the size of the population size *PSize*. The size of reference set will be constant throughout all later iterations. After that, SS applies SGM in order to generate subsets of solutions out of the reference set of solutions. The solutions of each subset are combined later using the CM to produce new solutions. Then the IMP is applied on those new solutions to produce new trial candidate solutions that will be examined by the RSUM to be added to *RefSet* in case they proved to be "better" than the

solutions in the reference set in terms of OF value or diversity. The standard termination condition for SS algorithm says that once there will not be any more new trial solutions that can be possibly added to *RefSet*, then the algorithm stops or terminates.

I.3 Scatter Search Verses Genetic Algorithm

In contrast to other existing evolutionary approaches, SS was first found with the basis of having systematic methods' design that helps in generating new solutions that can show more enhancements above those resulting from basic randomization. SS uses diversification and intensification techniques while it is being customized in our work, for the first time, in order to be the second improving phase that targeted to solve CCTP.

Evolutionary algorithms (EAs) have also been applied successfully on hard problems from diverse domains like optimization, automatic programming, machine learning, …etc. Genetic algorithm (GA) is another evolutionary population-based search algorithm. Since its first proposal, different implementations and designs GA were introduced. While considering the basic GA structure and design, we can clearly notice some major differences between its features and the major features of SS as shown in the following list (Laguna and Marti, 2003; Marti, Laguna and Campos, 2005):

- A typical GA population size is about 100 solutions, while that of SS is 10 or 20. Therefore, *SS* operates on a reference set, generated from the initial population, with size 10 to 20% maximum of the GA's population size.
- While building the initial population, controlled randomization technique is applied by SS in order to introduce diversity in the population. On the other hand, GA uses pure randomization mechanism to construct its initial population.
- SS constructs a subset of reference solutions on which SS applies its improvement and combination methods, while GA applies its operators on a large population set where it probabilistically chooses chromosomes from the existing population to apply crossover and mutation over.
- SS update the reference set, *RefSet*, by using deterministic rules developed in the RSUM, while the GA's population is constructed and continuously

updated using probabilistic rules developed on the concept of the "survival of the fittest" behavior.

- Scatter Search incorporates the use of local search heuristics as part of its improvement and combination methods to help in investigating the solutions' search space (or neighborhoods) in an advance way, while local search was added to GAs to produce different types of GA operators in order to enhance more the quality of its solutions.
- SS uses the CM to combine two or more solutions together so that we can benefit from the good properties of the elements in each solution, while, for *GA,* the crossover operator can only combine two solutions together.

Not only this, but also CM of SS can implement versions of crossover operators of GAs. However, the main argument here will be that the crossover operator might lose the path and the link to some promising elements in the solutions which might lead to losing good and high quality solutions.

I.4 Scatter Search Application on Different Problems

After the publication of Glover's SS Template in 1998, SS gained the interest of many researchers and practitioners who used *SS* to solve many problems in different domains up till today. *Table I.1* shows a sample of those applications and domains of research on which SS has been applied.

Table I.1: SS Application on Different Types of NP Problems

Domain of Research	Problems	Authors
Scheduling	Exam Timetabling	Mansour et al. (2009);
	Assigning Proctors to Exams	Laguna, Lourenço and Martí (2000).
	Project Scheduling	Debels, De Reyck, Leus and Vanhoucke (2006); Yamashita, Armentano and Laguna (2004); Ranjbar and Kianfar (2009).
	Airline Crew Rostering and Aircraft Landing Scheduling Problem	Pinol and Beasley (2006); Maenhout and Vanhoucke (2010).
Vehicle Routing	Vehicle Routing	Atan and Secomandi (1999); Russell and Chiang (2006).
Graph Permutation	Graph Permutation Problem	Rego and Leao (2000);

NN	Neural Network Training	Kelly, Rangaswamy and Xu (1996);
TSP	Dynamic Travelling Salesman Problem	Pantrigo, Duarte, Sánchez and Cabido (2005);
	Covering Tour Problem	Baldacci, Boschetti, Maniezzo and Zamboni (2005);
Bioinformatics	Protein Structure Prediction	Mansour, Kehyayan and Khachfe (2009);
	Gene Expression Data / Biclustering	Nepomuceno, Troncoso and Aguilar-Ruiz (2010).
Chemical Engineering	Chemical and Bio-Process Optimization problem	Egea, Rodríguez-Fernández, Banga and Martí(2007).
MAX-SAT	MAX-SAT and MAX-W-SAT Problems	Boughaci, Benhamou and Drias (2008).
Flow-shop	Flow-shop problem	Saravanan, Haq, Vivekraj and Prasad (2008).
Image Registration	Medical Image Registration	Cordón, Damas, Santamaría and Martí (2005);
Data Mining	Automatic Clustering Problem	Abdule-Wahab, Monmarché, Slimane, Fahdil and Saleh (2006).
Knapsack	0–1 Multidimensional Knapsack	Hanafi andWilbaut (2008).
Software Engineering	Software Testing and Automatic Test Case Generation	Blanco, Tuya and Adenso-Díaz (2009).
Network Design	Multicommodity Capacitated Network Design Problem	Alvarez, González-Velarde and Alba (2005).