AN ANT COLONY OPTIMISATION ALGORITHM FOR TIMETABLING PROBLEM

BY

ADUBI STEPHEN AYODEJI

08CG07733

COMPUTER SCIENCE

AN MSc RESEARCH PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES, SCHOOL OF POSTGRADUATE STUDIES, COVENANT UNIVERSITY, OTA, OGUN STATE.

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE (MSc) DEGREE IN COMPUTER SCIENCE

JUNE, 2015

CERTIFICATION

I hereby certify that this project was written by Adubi Stephen Ayodeji and was supervised by me and submitted to the Department of Computer and Information Sciences, College of Science and Technology, Covenant University, Ota, Nigeria.

……..…… **……**

Dr. A.A. Adebiyi **Date**

(Supervisor)

……..…… **……..……**

Dr. A.A. Adebiyi **Date**

(Head of Department)

DEDICATION

The project is dedicated to God who is the author of my wisdom, knowledge and understanding and also for seeing me through the completion of the project erasing any form of apprehension of it not being completed along the way.

ACKNOWLEDGEMENTS

My complete adoration and praises go to the Lord Almighty for the successful completion of this project and also making me to successfully complete a part of my career as a post-graduate (MSc.) student in this great institution.

It has also been a privilege to be the son of Mr. and Mrs. Adubi who have supported me in many areas of my life especially the advice I have received from them and the sponsorship throughout my life as a student right from the elementary stage. I also use this medium to thank my supervisor Dr. A.A. Adebiyi for his support and the tutorship I gained under his supervision as well as Dr. Olawande Daramola for spending time with me for meaningful discussions with respect to the work.

I would also like to thank Dr. Oladipupo and Dr. Oyelade who have taken their time to review this work providing insightful comments along the way.

A word of appreciation is also directed towards my great friends in school for the series of academic work we have done together and also for their companionship.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The University Course Timetabling Problem (UCTP) is a combinatorial optimization problem which involves the placement of events into timeslots and assignment of venues to these events. Different institutions have their peculiar problems; therefore there is a need to get an adequate knowledge of the problem especially in the area of constraints before applying an efficient method that will get a feasible solution in a reasonable amount of time. Several methods have been applied to solve this problem; they include evolutionary algorithms, tabu search, local search and swarm optimization methods like the Ant Colony Optimisation (ACO) algorithm.

A variant of ACO called the MAX-MIN Ant System (MMAS) is implemented with two local search procedures (one main and one auxiliary) to tackle the UCTP using Covenant University problem instance. The local search design proposed was tailored to suit the problem tackled and was compared with other designs to emphasise the effect of neighbourhood combination pattern on the algorithm performance. From the experimental procedures, it was observed that the local search design proposed significantly bettered the existing one used for the comparison.

The results obtained by the implemented algorithm proved that metaheuristics are highly effective when tackling real-world cases of the UCTP and not just generated instances of the problem and can even be better if some tangible modifications are made to it to perfectly suit a problem domain.

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND INFORMATION

The University Course Timetabling Problem (UCTP) is a combinatorial optimization problem which involves the placement of events into timeslots and assignment of venues to these events while considering the participants: Lecturers and Students. It can also be classified as a constraint satisfaction problem whereby the primary goal is satisfying the amount of constraints as much as possible; researchers have tackled the problem from this perspective (Cambazard *et al.*, 2012; Wijaya & Manurung, 2009). The UCTP characteristically contains two types of constraints which are the hard and the soft constraints. The hard constraints are the conditions that must be satisfied before a constructed timetable can be regarded as feasible while all the soft constraints do not necessarily need to be satisfied although the satisfaction of these soft constraints is often used to measure the quality of a timetable (Lewis, 2008). An example of a hard constraint is: "Two events which are conflicting (both having to be taken by same set of students) should not be placed in the same timeslot"; this constraint is basic for almost every institution.

The soft constraints of a particular problem instance usually embody the timetabling policy of an institution and sometimes tailored towards satisfying the demands of some concerned parties (certain professors who have some preferences). An example of a soft constraint is "No student should have more than two classes in a row". Although regarded as a general problem, many universities have their peculiar policies which form additional constraints to the common ones mentioned above. For example, Covenant University timetabling policy enforces two new hard constraints:

(1) An event peculiar to students from 300-level to 500-level should not be placed in the first two timeslots (8-10 am) on Tuesday.

(2) An event peculiar to students from 100-level to 200-level should not be placed in the first two timeslots on Thursday.

Violation of any hard constraint renders a timetable useless; this has influenced the latest approaches used in solving timetabling problems. These approaches attempt to satisfy all hard constraints first before attempting to satisfy the soft constraints as much as possible without violating any hard constraint immediately a feasible solution is constructed. Different methods and modifications of these methods have been proposed in literature to tackle the UCTP, few are Evolutionary algorithms (Sigl *et al.,* 2003; Perzina, 2007; Wijaya & Manurung, 2009; Jat & Yang, 2011; Yang & Jat, 2011), Tabu Search (Lu & Hao, 2010) and Ant Colony Optimization algorithms (Socha *et al.,* 2003; Matijas *et al.,* 2010).

1.2 STATEMENT OF THE PROBLEM

The construction process of a timetable involves the assignment of timeslots to events in such a way that conflicting events do not get the same timeslots and also assigning rooms where the events will take place while considering "room clashes".

Conflicting events: in the context of this work, two events are conflicting if they are related by at least one of the two conditions: (1) they have the same set of students offering them (CSC 412 and CSC 418 are courses both taken by 400 level Computer Science students; therefore, they are conflicting and thereby fulfil this condition) (2) they are required to hold in the same room (Two programming courses CSC 211 and CSC 313 are conflicting because they both need the Computer Lab to hold).

The course timetabling problem is NP-Hard and therefore any little addition to the problem domain will likely escalate the difficulty of the problem. Due to the difficulty of the problem as well as the peculiarity that can be found in various institutions, it has become a very much studied problem.

Drafting a timetable can be a very difficult task especially when done manually; therefore efforts are being made by researchers to bridge the gap between theory and practice mostly by trying to automate the process in their various institutions. Covenant University has not moved to the point of automating their timetabling process and thus a successful automation of the process will go a long way in ensuring quick and stress-free preparation for the new semester. Previous researches published that have been surveyed do not emphasise some important issues of timetabling like dealing with the placement of events that require more than one timeslot which is very

critical to the successful construction of a feasible timetable of Covenant University. Therefore this new research will address this problem by presenting a way of overcoming the difficult process of timeslot selection.

Finally, it has been noted that the final draft released for the school sometimes contain clashes which will have to be resolved when the lectures have already started. The proposed automated timetabling system will avoid these clashes and save the administration the stress of having to move lectures from their timeslots due to clashes.

1.3 AIM AND OBJECTIVES OF THE STUDY

The aim of this work is to employ the Ant Colony Optimization (ACO) algorithm to solve the University Couse Timetabling Problem and also to determine the best local search design that will give the best result.

This aim would be realised through the following objectives:

1. To have a full grasp of the CU's problem instance and its constraints (both hard and soft).
2. To identify the variables required for solving the problem such as number of classrooms, hall capacity for all classrooms, etc. with their respective values.
3. To implement MAX-MIN Ant System (MMAS); an ACO algorithm variant to construct a feasible solution.
4. To evaluate the solutions generated by different variants of the implemented algorithm.

1.4 RESEARCH METHODOLOGY

To achieve the first two objectives, the conditions that make the institution's timetable a 'good' one is captured in form of the constraints to work with when attempting to construct an automated one. After the constraints are identified, they are then classified into hard and soft constraints based on the compulsion of satisfying some of the constraints.

MMAS is one of the most successful variant of the ACO algorithm (Adubi & Misra, 2014; Zecchin *et al.,* 2007; Zecchin *et al.,* 2006). It was proposed in (Stuetzle & Hoos,

2000) to improve the first version of ACO (an algorithm that mimics the behaviour of natural ants in constructing solutions to combinatorial optimisation problems), the Ant System (Dorigo *et al.,* 1996) and since then, it has been applied to several optimization problems.

The MMAS will be implemented to achieve the third objective and its implementation will be accompanied with an implementation of two local search procedures based on the proposed work of Rossi-Doria *et al.* (2002) to improve the solution of the MMAS algorithm. The details of the MMAS and local search implementations will be given in Chapter Three. Finally on this section, VB.net programming language will be used for the implementation of MMAS algorithm and Local Search. VB.net is the most natural language for communicating with the Microsoft Excel Spread Sheet and also more friendly when it comes to code debugging to deal with the limited time of project completion.

To evaluate the result of the MMAS implementation to the timetabling problem instance chosen, the amount of soft constraints violations (#scv) of the feasible solution returned by the MMAS+Local Search implementation will be measured, a lower #scv connotes a better timetable.

1.5 SIGNIFICANCE OF THE STUDY

Drafting a timetable in an academic institution is not an easy task, the mental and physical stress involved is quite significant, the timetable drafted may not be of a high quality and clashes can even be present in such timetables. In the light of this, the proposed automated timetable system will help reduce time and effort that are used when human labour is engaged and obviously clashes on the timetable are avoidable as long as the requirements are properly captured. This approach will also enhance promptness of timetable release for adequate preparation from both the students and lecturers.

1.6 SCOPE AND LIMITATION OF THE STUDY

The study is strictly limited to the task of assigning courses to classrooms and placing them in appropriate timeslots and rooms which is the main definition of a course timetabling problem. Covenant University will be used as a case study.

1.7 OUTLINE OF THE THESIS

The rest of the project follows with an extensive review of literature and comparison of the problem to graph colouring in the next chapter. The system methodology and the formulation of the model are the highlights of Chapter Three. The experimental results and the evaluation of the solution method are the contents of Chapter Four. The project is concluded in Chapter Five where the platform for future research work is highlighted.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

Existing methods like metaheuristics, operational research methods that have been used to tackle UCTP problem, existing automated timetabling systems, and comparison of the problem to the Graph Colouring Problem (GCP) are extensively presented in this chapter.

In the UCTP problem, we have a set of events/courses to be allocated along with the set of students and the set of lecturers that will be participants of these events, a set of resources: projectors, rooms, systems and the timeslots where events will be placed. The feature of an acceptable schedule is the positioning of events in timeslots where the following conditions must be met:

(1) No event-pair having common students should be placed in the same timeslot.

(2) No two or more events should be holding in the same room at the same time.

Optimising a timetable can be grouped into three classes according to Lewis (2008): One-stage optimisation, Two-stage optimisation and Optimisation by relaxation.

One-stage optimisation involves trying to satisfy both hard and soft constraints at the same time a solution is constructed. Two-stage optimisation involves satisfying the hard constraints from the outset and then soft constraints satisfaction is considered after a feasible solution is found. In the last class, some constraints are dropped from the beginning of a solution construction to relax the problem before they are introduced after an intermediate solution (when these dropped constraints were not considered) had been found.

2.1.1 Constraints

The two major types of constraints have been mentioned earlier, they are the hard and soft constraints; they mainly depict the degree of satisfaction of the conditions that must be fulfilled when constructing an ‘acceptable’ timetable. Apart from this classification, constraints can also be classified based on constraint nature taking into account the specific variables of UCTP affected by the constraints. Five main classes

have been identified in Corne *et al.*, (1995) and also in Lewis (2008), they are: Unary constraints, Binary constraints, Capacity constraints, Event Spread constraints and Agent constraints. They are briefly described below:

Unary constraints: These constraints involve only one event, such as "CSC 815 must take place between the hours of 8 and 10 on Monday mornings".

Binary Constraints: Predictably, these constraints involve two events, for example "CSC 812 must take place before CSC 815". It can also be chained, for example "CSC 819 must take place before CSC 812 which itself must take place before CSC 815".

Capacity constraints: These constraints involve room allocation, most times they deal with the question: "Is room A where CSC 813 is allocated large enough to contain the students offering the course?"

Event Spread constraints: In some institutions, the timetabling policy involves the "spreading out" of events in order to reduce the workload of students and lecturers; this constitutes what an event spread constraint is, an example is "No student should have more than two classes in a row."

Agent Constraints: These constraints are normally enforced by lecturers (professors) who cannot take classes in some certain timeslots of the day due to some certain reasons.

2.1.2 Other Variants of Timetabling Problem

Apart from the UCTP which is the main focus of this study, there are other main variants of timetabling problem as mentioned in the work of Schaerf (1999) namely: School timetabling and Examination timetabling. In school timetabling, the aim is to schedule classes of teachers for a week in a way that the subjects taken by a teacher do not get scheduled at the same time for two or more different classes; in other words there should be avoidance of a teacher meeting more than one class at the same time. This variant is peculiar to Secondary schools. In examination timetabling, the aim is to schedule exams avoiding courses having common set of students to be fixed at the same time and also spreading the exams for the students as much as possible. Unlike the UCTP where one event is expected to be held in a room per time, more than one event in the examination timetabling problem can be scheduled in the same room at the same time as long as the capacity constraint is not compromised. It should be noted

that the list of variants mentioned is not exhaustive; just the common variants are mentioned and to compare them with the UCTP.

2.2 REVIEW OF METAHEURISTICS

In this section, a review of a selected set of five metaheuristics is presented based on their popularity as well as their high level of application to the general Combinatorial Optimisation Problems (COPs) and UCTP. These five metaheuristics have often been compared with one another on different data sets of timetabling instances (small, medium and large). The metaheuristics to be reviewed are being studied by the metaheuristics network (Blum & Manfrin, 2015) and have also been compared among themselves on generated instances of the UCTP (Rossi-Doria, *et al.*, 2003), they are as follows: Evolutionary Algorithm (EA), Iterated Local Search (ILS), Ant Colony Optimisation (ACO), Tabu Search (TS) and Simulated Annealing (SA). They have enjoyed wide application to several combinatorial optimisation problems and have also been combined with other approaches (Chaudhuri & De, 2010; Abdullah *et al.*, 2009). The algorithms that are presented here with the short examples will be addressing **minimisation** problems.

2.2.1 Evolutionary Algorithms

Evolutionary algorithms are computational methods inheriting their behaviour from natural system features such as selection, recombination and mutation. The most popular among the evolutionary algorithms is the Genetic Algorithm (GA); other notable ones are Memetic Algorithms (MA), genetic programming and evolutionary programming. In this study, the GA will be discussed since it is the most basic form of evolutionary algorithms which encompasses all other evolutionary algorithms as far as the discussion of metaheuristics is concerned (Reeves, 2003). In addition to the discussion of GA, the MA will also be discussed briefly.

2.2.1.1 Genetic Algorithm

The genetic algorithm depicted in ALGORITHM 2.1 starts with the initialisation of a population of chromosomes which form the initial population and then their fitness values are calculated. The selection process starts by retaining a number of the population, members of the population with high fitness values are highly likely to be picked in this process making those ones with low fitness values to be likely dropped.

Crossover operation is performed on pairs of the selected members of the population to form new offspring. Mutation (genetic alteration) may be done on few of the offspring generated after the crossover operation, then the offspring replaces the worst members of the previous population to form the new population set alongside the best of the previous population. This process continues until some stopping criteria are met.

ALGORITHM 2.1: GENETIC ALGORITHM

Begin

initialise a random population of size N

calculate the fitness of chromosome C_i in the population $\forall i$

while termination criteria not met

select chromosomes among the current population as parents of the next generation based on their fitness values.

perform crossover among chosen pairs of parents in the last step using randomly chosen points to form new offspring (if condition is satisfied).

perform mutation (if condition is satisfied) with probability P_{mut} (a very small value)

replace existing population with the current one.

end while

end

Chromosomes of genetic algorithms' solutions are commonly represented using binary strings (the simplest form of representation) but complex representations such as real-valued numbers also exist when binary representation is not adequate for the nature of the problem being solved (Mitchell, 1999). Popular selection methods include the roulette wheel selection (RWS), rank selection and tournament selection, although the latter is the most recommended among the three (Reeves, 2003). The crossover operation is performed based on a probability value called the crossover rate $\mathbf{P_C}$; in the crossover procedure, pieces of information are exchanged between a pair of parents to produce two new offspring. Prominent crossover operators include One-point crossover (1X), Two-point crossover (2X) and Uniform crossover (UX). In 1X, an integer value **t** is selected at random as the crossover point, alleles before the crossover point in the first parent **P1** are copied into the first offspring **O1** and then the alleles after the crossover point in the second parent **P2** are copied into **O1** to complete its genetic makeup. The second offspring **O2** is created by reversing the roles of **P1** and

P2. The second type of crossover operator 2X is similar to 1X but in this case; two points $\mathbf{t_1}$ and $\mathbf{t_2}$ are selected as crossover points rather than a single crossover point that is used in the former approach. Lastly in the uniform crossover, a value called mixing ratio or Bernoulli parameter (Reeves, 2010) is used in controlling the transfer of alleles from a parent-pair to an offspring. A mixing ratio of 0.5 means that an equal number of alleles from the first parent and the second parent is used to form the genetic makeup of the offspring. A typical illustration is shown in Table 2.1.

Table 2.: A table illustrating the three crossover operators

Parents	One-point crossover (1X)	Two-point crossover (2X)	Uniform crossover (UX)
P1: 1 1 0 0 1 0 1 0 P2: 1 1 1 0 0 0 1 1	P1: 1 1 0 0 1 | 0 1 0 P2: 1 1 1 0 0 | 0 1 1	1 1 | 0 0 1 0 | 1 0 1 1 | 1 0 0 0 | 1 1	Mask1: 1 0 0 1 0 1 1 0 Mask2: 0 1 1 0 1 0 0 1
	O1: 1 1 0 0 1 0 1 1 O2: 1 1 1 0 0 0 1 0	O1: 1 1 1 0 0 0 1 0 O2: 1 1 0 0 1 0 1 1	O1: 1 1 1 0 0 0 1 1 O2: 1 1 0 0 1 0 1 0

Mask1 in the third column of Table 2.1 illustrates which allele from the two parents in that position is copied into the offspring; a bit of 1 in position 1 in Mask1 means that the allele of Parent 1 in that position is copied while a bit of 0 copies from the second parent P2. The bits in Mask1 are flipped to generate Mask2 which will be used to generate O2 doing the same thing that was done for Mask1. It should be noted that the mixing ratio in UX illustrated in the table is 0.5 since we have equal number of '1' bits and '0' bits.

Crossover in genetic algorithms is not that straightforward the way it seems to be in some circumstances. There is also a non-linear crossover. For example, considering two selected parents for crossover in a GA run for the Travelling Salesman Problem (TSP):

P1: 1 6 3 5 4 2

P2: 2 3 4 1 5 6

P1 is a solution which reads that the travelling salesman visits cities in the strict order: 1, 6, 3, 5, 4, and 2 and then back to 1, P2 can be interpreted the same way. In a naive GA implementation where for example one-point crossover is used and **t** is set to 3, we would have the following offspring:

O1: 1 6 4 1 5 6

O2: 2 3 3 5 4 2

These offspring produced represent infeasible solutions; O1 is a solution representation stating that the travelling salesman will have to visit both cities 1 and 6 twice violating the principal constraint of the TSP. The partially mapped crossover (PMX) operator (Goldberg & Lingle, 1985; Reeves, 2003; Reeves, 2010) deals with this problem. The PMX operator can be designed as an extension of 2X since 2X points are initially selected at random between 1 and l (the length of the string encoding a solution). In the example given above, if the crossover points selected are 2 and 5, then an interchange mapping $\leftrightarrow$ is defined such that we have the following:
$3 \leftrightarrow 4$, $5 \leftrightarrow 1$, and $4 \leftrightarrow 5$. Therefore these values in each parent are swapped rather than the exchange of alleles. The following offspring from the same example using PMX are as follows:

O1: 4 6 5 1 3 2

O2: 2 5 3 4 1 6

Another alternative PMX approach is to extend the uniform crossover operator where a binary mask like we have in UX is generated. If for example we have the mask: 1 0 1 0 0 1, this means that the alleles in the positions 'masked' by the bit 1 are copied from the first parent while the ones in the positions 'masked' by the bit 0 are taken from the other parent in the order in which they appear to fill the empty spaces. Still using the same example, we would have the following:

P1: 1 6 3 5 4 2 => 1 _ 3 _ _ 2

P2: 2 5 3 4 1 6 => 2 _ 3 _ _ 6

O1: 1 5 3 4 6 2

O2: 2 1 3 5 4 6

In constructing offspring O1 a mask 1 0 1 0 0 1 was used, therefore alleles in the positions where we have the bit 1 in P1 are copied while the other positions initially left blank are filled with alleles not copied from P1 in the order in which they appear in P2 to fill the blank spaces. O2 is constructed the same way using the same mask but P1

and P2 switch roles this time around (P2 becomes the first parent supplying the first set of alleles using the given mask).

If the mutation condition (to determine if the operation will be carried out) is true, then few alleles in the genes of some of the generated offspring are mutated (changed) to form an entirely new offspring. An offspring represented with the string 110010 becomes 110011 if the allele at the 6^{th} locus of the gene is mutated. The mutation process is more complex when dealing with real-value representations rather than binary bit strings.

2.2.1.2 Memetic Algorithm

Memetic algorithm is a class of evolutionary algorithm which can sometimes be regarded as a hybrid approach comprising of GA and another metaheuristics or a local search method. It uses problem-specific knowledge in the construction of solutions to a particular problem domain and because of their hybrid nature they are sometimes called 'Hybrid Evolutionary Algorithms' (Moscato & Cotta, 2003).

2.2.2 Iterated Local Search

Iterated Local Search (ILS) (Lourenco *et al.*, 2003) is another powerful metaheuristics which is part of the broad family of local search. The algorithm is a very simple but effective one which has shown promise in a lot of combinatorial optimisation problems that it has been applied to. The algorithm works by using a simple heuristic (greedy constructive heuristic or a random procedure) to construct an initial solution S_{init} and then applying a local search procedure to S_{init} to move it to a local optimal solution S_{best}. S_{best} is perturbed (tweaked) to move it away from local optimum using an embedded heuristic different from the one employed in the local search procedure deriving an intermediate solution S'. S' is subjected to the same local search routine used to derive S_{best} to arrive at another local optimum S'_{best} which is accepted if it is better than S_{best} or rejected otherwise based on an acceptance criterion. This process continues until a stopping criterion is met. The ILS algorithm has been very effective and faster (when it comes to the amount of local search runs made before arriving at local optimum) when compared to a Random Restart Local Search (RRLS) procedure (Lourenco *et al.*, 2010).

For brevity, the RRLS algorithm works by applying a local search procedure to an initial solution S_{init} until a local optimal solution is reached. The objective cost is then

'improved' by restarting the algorithm using another generated initial solution with the 'hope' of finding a better solution leading to multiple trials of the local search routine. The Iterated Local Search algorithm is shown in ALGORITHM 2.2.

ALGORITHM 2.2: ITERATED LOCAL SEARCH

procedure ILS( )
  S_{init} = GeneralInitialSolution( )
  S_{best} = LocalSearch(S_{init})
  while termination criterion not met **do**
    S' = Perturbation(S_{best}, history)
    S'_{best} = LocalSearch(S')
    S_{best} = AcceptanceCriterion(S_{best}, S'_{best}, history)
  end while
  return S_{best}
end

There are four main components of the ILS procedure; GeneralInitialSolution, LocalSearch, Perturbation and AcceptanceCriterion, discussion of these components follow:

General Initial Solution: The ILS procedure starts by constructing an initial solution which can be achieved either by a quick greedy heuristic or generated randomly. For example the initial solution of an ILS approach to TSP can be constructed using the nearest neighbour heuristic. It is after this stage that the main local search procedure begins to move the initial solution to a local optimal point.

Local Search: This is one of the fundamental elements of the ILS algorithm; a suitable local search procedure moves a solution S_{before} to another S_{after} in its neighbourhood **$N(S_{before})$**. The neighbourhood N(S) of a solution S is defined as the set of all possible solutions that can be derived by applying a certain move (swap, insertion, deletion) to S. Notable local search algorithms mostly employed for the TSP include 2-opt, 3-opt and the Lin-Kernighan heuristic (Lin & Kernighan, 1973), the latter known to be the most effective heuristic for the TSP (Stuetzle, 1998). Local search procedures for the UCTP for example can be the Kempe Chain neighbourhood move.

Perturbation: This stage of an ILS is what differentiates it from a random restart local search algorithm. The perturbation procedure makes a careful but significant change(s) to the current local optimal solution with the hope of moving it to a different basin of attraction. This perturbed solution (S′ in the algorithm outline) serves as the new starting solution for the local search procedure with the hope of generating a new local optimal solution that will be measured against the former local optimal solution to determine if it will be accepted as the new best solution. Still using the TSP, a perturbation procedure for the TSP can be a 4-opt move (especially if 2-opt or 3-opt is the primary local search procedure), also known as the double-bridge move (Lourenco *et al.,* 2010). In ILS, great care is taken before choosing a perturbation procedure; when the strength of perturbation (often measured as the number of changes made to a local optimal solution) is too strong, the ILS procedure is reduced to a random restart and if it is too weak, the local search procedure may undo the perturbation. Therefore, optimising a perturbation procedure to get a right balance in its strength is very crucial to the successful implementation of an ILS algorithm to a problem domain.

Acceptance Criterion: An acceptance criterion for an ILS algorithm may be designed to force the objective cost to decrease/increase (the latter for maximisation problems) or to disregard the objective cost of the most recently visited local optimum (Lourenco *et al.,* 2003). The first type of acceptance criterion accepts the newly found local optimal solution S'_{best} if its objective cost is better than that of the currently best solution S_{best} found. This type of acceptance criterion is defined for minimisation problems as:

$$Better(S_{best}, S'_{best}, history) = \begin{cases} S'_{best}, & C(S'_{best}) < C(S_{best}) \\ S_{best}, & otherwise \end{cases} \quad (2.1)$$

Where $C(.)$ is the objective cost of a particular solution

The second type called Random Walk always accept the newly found local optima regardless of the objective cost of the current best solution just to explore the search space; it is defined as:

$$RW(S_{best}, S'_{best}, history) = S'_{best} \quad (2.2)$$

The two very different acceptance criteria just discussed are not the only ones available, there is also an acceptance criterion based on Simulated Annealing which

will be called SA_Accept(.) for the purpose of distinguishing it from others. An ILS procedure with the SA_Accept(S_{best}, S'_{best}, history) criterion always accepts S'_{best} if its cost is better than that of S_{best}, otherwise it is accepted with the probability function given in (2.4) identical to the acceptability of a new solution found in an SA run.

$$SA_{Accept(S_{best},S'_{best},history)} = \begin{cases} S'_{best}, & C(S'_{best}) < C(S_{best}) \\ S'_{best} \; with \; 2.4, & otherwise \end{cases} \quad (2.3)$$

$$e^{-\frac{C(S'_{best})-C(S_{best})}{T}} \quad (2.4)$$

Better(.), RW(.) and SA_Accept(.) acceptance criteria have all been experimentally tested and compared on both small and medium instances of a UCTP; the SA_Accept(.) criterion emerging as the most efficient among the three (Rossi-Doria, *et al.*, 2003). The last statement indicates that SA_Accept(.) which attempts to strike a balance between intensification and diversification is a promising one for an ILS implementation for tackling the UCTP. On a final remark, the first two extremely different acceptance criteria favour extreme intensification and diversification respectively.

2.2.3 Ant Colony Optimisation Algorithm

Ant Colony Optimisation (ACO) algorithm is a population-based metaheuristics that mimics the behaviour of real ants to solve combinatorial optimisation problems. Ever since the emergence of the first variant; the Ant System (Dorigo *et al.*, 1996), improvements have been made which have given birth to other notable variants such as Ant Colony System (Dorigo & Gambardella, 1997), Rank-based Ant System (Bullnheimer *et al.*, 1997), and the MAX-MIN Ant System (Stuetzle & Hoos, 2000). Natural ants communicate through a substance called pheromone; this communication pattern is the main driving force behind the ACO algorithm; artificial ants also deposit 'pheromone' on the paths of the construction graph they tour with a proportion relative to how well the path is to the goal. The algorithm is presented in ALGORITHM 2.3.

ALGORITHM 2.3: ACO ALGORITHM

procedure ACO()
  initialise Ants
  initialise pheromone
  while stopping criteria not met **do**

each ant **k** constructs a solution
apply a local search procedure (optional)
update pheromone
end while
return best solution
end procedure

In constructing a solution to a given combinatorial optimisation problem (COP) using the ACO metaheuristics, the following procedures are followed:

Initialise Parameters: The required number of ants in the colony is initialised and also the pheromone values on the edges of the construction graph are initialised to typically a very small value in the range (0, 1]. Pheromones on graph edges in the MAX-MIN Ant System (MMAS) variant are initialised to the maximum pheromone value which must have been predetermined.

Solution Construction: In an iterative step, an ant **k** constructs a solution guided by the pheromone values and the heuristic information on the construction graph edges. Using the TSP as an example, an ant **k** chooses the next city to visit through a stochastic mechanism; the probability of **k** at location **i** choosing a location **j** is given as:

$$P_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{l \in N^k} \tau_{il}^{\alpha} * \eta_{il}^{\beta}}, & if\ j \in N^k \\ 0, & otherwise \end{cases} \qquad (2.5)$$

$Where$

$\tau_{ij} = pheromone\ on\ edge(i,j)$

$\eta_{ij} = heuristic\ information\ on\ edge(i,j)$

$N^k = set\ of\ cities\ unvisited\ by\ ant\ \boldsymbol{k}$

The parameters α and β control the weight of pheromone and heuristic information respectively; if α is set to zero then only the heuristic information will be used to guide the ant **k** and setting β to zero means only the pheromone is used to guide the ant. The setting α = 1 and β = 5 have been proposed as an optimised combination for the TSP after experimental tests (Dorigo *et al.*, 1996).

After the probability of choosing a city c_i to visit next in the tour construction $\forall i \; in \; N^k$ is calculated, the roulette-wheel selection procedure (similar to the one in GA) is used to determine the next city that the ant will eventually visit (Dorigo & Stuetzle, 2004). Equation (2.5) is used in the most basic form of ACO; the Ant System (AS). Ant Colony System (ACS) uses the pseudo-random proportional rule which selects the city with the highest desirability value calculated based on (2.5) if a condition is satisfied or uses the roulette-wheel selection procedure of the AS if otherwise. More on this rule can be gotten from Dorigo *et al.,* (2006) and Dorigo and Gambardella (1997).

Pheromone Update: In the early ACO algorithm, pheromone update is done after all ants have constructed a tour (given TSP as the COP). Pheromone update involves two processes; pheromone evaporation to prevent accumulation and pheromone deposition. The pheromone trail for the next iteration in Ant System is updated according to the formula:

$$\tau_{ij}(t+1) = (1-\rho) * \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \tag{2.6}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/L_k, & if \; ant \; \boldsymbol{k} \; traversed \; the \; edge(i,j) \\ 0, & otherwise \end{cases} \tag{2.7}$$

$where \; \rho(0,1) = the \; evaporation \; rate$
$m = number \; of \; ants \; in \; the \; colony$
$L_k = length \; of \; the \; tour \; constructed \; by \; ant \; \boldsymbol{k}$

The expression at the LHS of the addition sign on the RHS of (2.6) is the evaporation procedure and the expression on the RHS of the operator is the pheromone deposition which might evaluate to zero if no ant traversed the given edge (i, j) at the last iteration. At the end of all ants' construction processes, the best tour found during the run is returned as the solution. Pheromone update in ACS is a little bit different from that of AS; each ant performs local pheromone update after constructing a solution on the last edge traversed (Dorigo *et al.*, 2006). Then the global pheromone update on all edges of the construction graph immediately follows the end of solution constructions by all ants in an iterative procedure; performed by the 'iteration-best' ant (k_{ib}) or the 'global-best' ant (k_{gb}). The ant that constructed the best solution (a solution with the shortest distance) in the last iteration is called iteration-best ant called while the ant

that constructed the best solution since the run of the algorithm started is called the global-best ant.

Local pheromone update in ACS is achieved through the following:

$$\tau_{ij}(t+1) = (1-\theta) * \tau_{ij}(t) + \theta * \tau_{init} \tag{2.8}$$

where,

$\tau_{init} = initial\ pheromone$

$\theta = decay\ constant$

Global pheromone update is done according to the following equation using L_{best} which is the length of the tour constructed by k_{ib} or the length of the tour constructed by k_{gb} depending on the algorithm implementer's choice:

$$\tau_{ij}(t+1) = \begin{cases} (1-\rho) * \tau_{ij}(t) + \rho * \dfrac{1}{L_{best}}, & condition \\ \tau_{ij}(i) \quad , & otherwise \end{cases} \tag{2.9}$$

where,

$condition = "if\ the\ best\ ant\ traversed\ edge(i,j)"$

MMAS has a pheromone update uniquely different from AS update rule, only the 'best' ant updates pheromone and the update is only done once at the end of iteration unlike the update rule in ACS. In MMAS, the resultant pheromone $\tau_{ij}(t+1)$ is bound by the maximum pheromone τ_{max} and the minimum pheromone τ_{min}. The pheromone update rule in MMAS is done by the following equation using L_{best} which is the tour length of k_{ib} or the tour length of k_{gb}:

$$\tau_{ij}(t) = (1-\rho) * \tau_{ij}(t) + \frac{1}{L_{best}} \tag{2.10}$$

Pheromone update is completed via 2.11 to force the pheromone to be in the range $[\tau_{min}, \tau_{max}]$ as shown below:

$$\tau_{ij}(t+1) = \begin{cases} \tau_{max}, & if\ \tau_{ij}(t) > \tau_{max} \\ \tau_{min}, & if\ \tau_{ij}(t) < \tau_{min} \\ \tau_{ij}(t), & otherwise \end{cases} \tag{2.11}$$

Equation (2.11) controls the pheromone trail; stopping it from exceeding the bounds $\tau_{min}\ and\ \tau_{max}$. Ultimately this procedure was designed to avoid early stagnation of the search (Stuetzle & Hoos, 2000). The comparison of the mode of operations of the main variants of the ACO algorithm is presented in Table 2.2.

Table 2.: Comparison of the Modi Operandi of the main variants of the ACO algorithm

Algorithm	Solution Construction	Update Rule	Method of Update
Ant System	Select next solution component via (2.5)	Ant system update rule	All ants in the problem space update pheromone trails after every iteration.
Ant Colony System	Uses the pseudo-random proportional rule	Iteration-best or best-so-far	Each ant applies the local pheromone update rule on the last edge traversed and then the global pheromone update is done by the iteration-best or the best-so-far ant.
MAX-MIN Ant System	Select next solution component via (2.5)	Iteration-best or best-so-far or both	Only at the end of iteration are pheromone trails updated by the iteration-best or best-so-far ant.
Rank-based Ant System	Select next solution component via (2.5)	Elitism	Ants are sorted by the quality of solutions they generated in the last iteration; therefore the "best" ant with a rank of 1 takes its place on top of the list. The first ω ants in the rank are used

			to make weighted update of pheromone trails. The weight controls the level of update that can be done on the pheromone trails and it is inversely proportional to the rank of the ant.

2.2.4 Tabu Search

Tabu Search is a powerful method applied to combinatorial optimisation problems to help guide a local search method from being trapped in local optima (Glover, 1990). The method was first introduced in a paper by Glover (Glover, 1986; Glover & Laguna, 1997) where the term metaheuristics was also coined. The tabu search algorithm had been employed to many COPs due to its wide range of success and acceptance (Gendreau & Potvin, 2010). An interesting aspect in Tabu search that makes it so successful is its ability to drive a local search procedure from getting stuck in local optima by allowing non-improving moves. It is also capable of preventing a local search from cycling back to previously visited solutions through the use of memories called tabu lists. The algorithm is presented in ALGORITHM 2.4.

ALGORITHM 2.4: TABU SEARCH

procedure TS( )
  s ← initial solution
  s_{best} ← s
  while termination criterion not met **do**
    identify the neighbourhood set N(s)
    identify the tabu set T(s) ∈ N(s)
    identify the aspiration set A(s) ∈ N(s)
    determine the candidate set C(s) = N(s) ∪ A(s) \ T(s)
    s ← best candidate in C(s)
    if f(s) < f(s_{best}) then
      s_{best} ← s

```
    end if
  end while
  return s_best
end
```

An initial solution is generated which is used to initialise the best solution found so far. Then the tabu search algorithm properly starts at the execution of the **while** loop; the neighbourhood N(s) of the current solution s is determined, the set of solutions in N(s) which are forbidden are marked into a set called the tabu set T(s) and then the set of solutions in the tabu set which can be allowed due to the aspiration criterion; A(s) is also determined. Then the candidate list C(s) containing solutions that are eligible for consideration is scanned for the best candidate (solution with the best objective cost). It should be noted that the best picked may not be the best in the original neighbourhood set N(s) due to the removal of some solutions which are marked tabu; this is one of the clever ideas in tabu search; selecting non-improving solutions. The best candidate in C(s) called **s** is evaluated against the current best solution and replaces the current best if it is better than it. The process continues until a certain stopping criterion is met.

Important concepts in Tabu search include Search space, Neighbourhood Structure, Tabu Lists and Aspiration Criteria (Gendreau, 2003). The search space is a common concept in local search and metaheuristics; it includes all possible solutions (feasible and infeasible) that may be visited during a search procedure. The next three concepts will be discussed one after the other as they relate to Tabu Search.

Neighbourhood Structure: A neighbourhood structure defines the type of moves/adjustments made to a particular solution to generate different solutions. These new solutions generated from the current solution **s** are said to be in the neighbourhood of **s** (denoted by N(s)) or simply the neighbours of **s**. Still sticking with TSP, if for example in a 4-city problem, we have a current solution in a run to be ADCB; meaning the journey starts from city A, visiting other cities in the order specified before going back to A. The neighbourhood of ADCB are as follows: DACB, CDAB, BDCA, ACDB, ABCD and ADBC. The neighbourhood structure employed here is by a simple swapping of two city positions, for example the neighbour CDAB was simply achieved by swapping positions 1 and 3 of the current solution. An alternative

neighbourhood structure for the TSP might be the insertion of a city in a tour to a different position. For example moving city C from its position to position 2, we have the neighbour ACDB. In practical situations, if the number of candidate solutions is large, only a subset of N(s), called candidate list is considered (Gendreau, 2003; Dreo *et al.*, 2006; Gendreau & Potvin, 2010). This can be achieved by using a probabilistic approach to select possible solutions that will be in the candidate list thereby reducing the length of the tabu list (Gendreau & Potvin, 2010). The probabilistic approach can be detrimental to finding excellent solutions if not properly done, in fact selection of an effective procedure for generating the candidate list is what makes a sound implementation of a Tabu search (TS) procedure different from a naive one (Gendreau, 2003).

Tabu List: This is the hallmark of any TS algorithm. The tabu list contains solutions or moves that led to these solutions that have been marked 'forbidden' unless they fulfil the aspiration criterion. The length of a tabu list in most cases affects the tenure of an item in the tabu list. The criteria that make an item 'worthy' of being put into a tabu list must be very sound unless the search may be forced to stagnate without getting near an optimal solution. In practice, the tenure of each item in the tabu list is updated from time to time, typically whenever a new solution is found. An item in a tabu list gets its tenure decremented if a new item is pushed into the tabu list; this particular item may finally be free being "taboo" if its tenure finishes (finally reduced to zero).

Aspiration Criteria: Tabu lists are essential to tabu search but may be too strong; blocking the search from exploring promising regions of the search space. Aspiration criteria are used to overcome the problem associated with tabu lists; annulling the effect of a tabu status on a possible solution. The most common aspiration criterion used is the acceptance of a solution that its objective cost is better than that of the best solution found so far even if it is in the tabu list.

Example 2.1: The TS algorithm will be demonstrated in this example to solve an Asymmetric TSP of 7 cities, by permutation, we have 5040 possible solutions in the search space and only 7 are optimal. Given the distance matrix between each city in the set {1, 2, 3, 4, 5, 6, 7}:

0	8	17	6	1	14	3
5	0	3	19	11	12	1
6	1	0	7	3	6	2
4	18	4	0	11	5	8
1	15	8	12	0	9	3
14	11	6	8	4	0	2
14	10	20	13	8	17	0

The following tabu search settings are used:

Tabu tenure: 3

Neighbourhood structure: swapping of two city positions

Δcost = f(new) – f(current)

Iteration 0: Initial solution **s** = best solution so far $\mathbf{s_{best}}$ = (1 4 5 2 3 6 7) of cost 57, at this point the Tabu List **T** = Ø and the search starts since **s** is not the optimal solution.

Iteration 1: The value of Δcost for each swap (i, j) is calculated, the first ten best moves (in no particular order) are displayed in Table 2.3.

Table 2.: Iteration 1

Move	(1,4)	(1,5)	(4,7)	(3,5)	(5,6)	(5,7)	(2,6)	(2,3)	(4,5)	(2,4)
Δcost	-13	-14	-10	-10	-12	-19	-9	-3	-1	0

The best move (5,7) is made and then a new best solution is found since the move leads to a new objective cost (38) better than the one previously known (57). Finally the move (5,7) is marked tabu making the positions of cities 5 and 7 not permitted to be swapped with each other in three iterations. Current solution s = (1 4 7 2 3 6 5).

Iteration 2: In this iteration, 4 moves will improve the objective cost. The best of them; move (3,7) is made since it made a gain of 4 leading to a new best (1 4 3 2 7 6 5) with a cost of 34 and then move (3,7) with a tenure of 3 (a new entry) is placed into the tabu list, the move (5,7) in the tabu list gets its tenure decremented to 2. Possible moves are displayed in Table 2.4.

Table 2.: Iteration 2

Move	(1,4)	(1,7)	(1,6)	(4,7)	(2,4)	(4,6)	(4,5)	(3,7)	(6,7)	(3,6)
Δcost	4	3	7	10	-1	10	-3	-4	-2	8

Iteration 3: No move leads to a solution better than the current best, so the sideways move (6,7) is made which is put into the tabu list with tenure of 3. The possible moves are displayed in Table 2.5.

Table 2.: Iteration 3

Move	(1,7)	(1,6)	(4,6)	(4,5)	(3,7)	(3,6)	(2,7)	(2,6)	(6,7)	(5,6)
Δcost	10	4	13	6	4	13	5	6	0	9
Tabu?					Yes(3)					

At this point, s = (1 4 3 2 6 7 5), but s_{best} remains (1 4 3 2 7 6 5).

Iteration 4: This stage also does not feature any move that will improve the objective cost. Possible moves are displayed in Table 2.6.

Table 2.: Iteration 4

Move	(1,6)	(1,5)	(2,4)	(4,5)	(2,3)	(3,6)	(2,6)	(2,5)	(6,7)	(5,7)
Δcost	4	12	0	7	10	2	3	5	0	10
Tabu?									Yes(3)	Yes(1)

Two sideways move (2,4) and (6,7) present themselves, but the move (2,4) is selected since it is not tabu and s becomes (1 2 3 4 6 7 5), s_{best} remains the same. At this point the tenure of move (5,7) becomes 0 and by virtue of that, it is removed from the tabu list meaning the positions of cities 5 and 7 can now be swapped again!

Iteration 5: The two best options are move (2,4) and move (1,3), the former which is the better option is in the tabu list with a tenure of 3 (the latest move made). The possible moves are displayed in Table 2.7.

Table 2.: Iteration 5

Move	(1,3)	(1,6)	(1,7)	(2,4)	(2,5)	(3,6)	(3,7)	(4,6)	(4,5)	(5,7)
Δcost	1	11	9	0	4	9	3	8	8	10
Tabu?				Yes(3)			Yes(1)			

Move (1,3) is made leading to s = (3 2 1 4 6 7 5) with an objective cost of 35, there was no choice since it is the best non-tabu move available.

Iteration 6: Accepting the non-improving move in the last iteration appears to be a master stroke, now we move from a dead-end to a promising path, the move (2,5) will improve the current solution's objective cost by 5 and the current best solution's objective cost by 4. Possible moves are displayed in Table 2.8.

Table 2.: Iteration 6

Move	(2,3)	(1,3)	(3,6)	(3,7)	(2,4)	(2,5)	(1,6)	(1,7)	(4,5)	(5,7)
Δcost	10	-1	10	3	14	-5	9	8	0	9
Tabu?		Yes(3)			Yes(2)					

The new solution after making the move = (3 5 1 4 6 7 2) with an objective cost of 30, this solution also overrides the current best known so far since it has a better objective cost.

Iteration 7: A new best (7 5 1 4 6 3 2) gotten by making the move (3,7) is found at this iteration. The new best is the global optimal solution since no improvement is made in subsequent iterations and the Tabu search ends. Possible moves are displayed in Table 2.9.

Table 2.: Iteration 7

Move	(1,3)	(3,6)	(3,7)	(2,3)	(1,5)	(4,5)	(2,5)	(1,6)	(4,6)	(2,4)
Δcost	8	9	-2	16	9	6	5	10	12	8
Tabu?	Yes(2)						Yes(3)			Yes(1)

The new best (7 5 1 4 6 3 2) with an objective cost of 28 is returned. In just 7 iterations, the tabu search procedure found an optimal solution to the TSP by making effective use of memory to prevent cyclic moves.

2.2.5 Simulated Annealing

Simulated Annealing (SA) is analogous to physical annealing in Physics; in fact that is where the inspiration of the algorithm lies (Aarts *et al.*, 2014). Like TS, it was designed to make a local search procedure escape local optima. The key difference between TS and SA is that while TS considers multiple candidates in a solution's neighbourhood, the SA algorithm considers only one. The candidate solution in SA is

outrightly accepted if its objective cost is better than that of the best solution found so far otherwise it is accepted based on a probability function given in (2.12).

$$e^{-\frac{\Delta(S',\ S)}{t_k}} \tag{2.12}$$

The Simulated Annealing algorithm is described in ALGORITHM 2.5.

ALGORITHM 2.5: SIMULATED ANNEALING

procedure SA( )
  initialise initial solution S ∈ Ω
  set initial temperature T
  t_k = T
  k = 0
  do
    set repetition counter c = 0
    do
      S′ ← a generated solution in N(S)
      Δ(S′, S) ← f(S′) – f(S)
      if Δ(S′, S) ≤ 0, **then** S ← S′
      if Δ(S′, S) > 0, **then** S ← S′ with (2.12)
      c ← c + 1
    until c = c_k
    k ← k + 1
    t_k ← α * t_k (cooling schedule)
  until a stopping criterion is met
end

Ω is the search space, k is the temperature-change counter, and α is the cooling rate of the annealing procedure. The algorithm presented above is patterned after the one in Nikolaev and Jacobson (2010).

An initial solution is generated (either randomly or through a greedy heuristic) and the initial temperature T is set. Before entering the outer loop, t_k (temperature at iteration k) is set to T and the algorithm run begins. The repetition counter c is set to zero before the execution of the inner loop. Inside the inner loop, the following processes take

place: (1) A solution S′ in N(S) is generated (2) the delta-value of S′ is evaluated (3) S′ is accepted as the new solution if its delta-value is less than or equal to zero (better or equivalent to the previously known best solution) (4) S′ is accepted based on (2.12) if its delta-value is greater than zero (worse than the currently best known solution) (5) the repetition counter is incremented.

If the predefined total number of iterations for the inner loop is met then the loop execution stops or continues otherwise. The stopping criterion for the inner loop can also be when a certain quality in a solution has been found (Nikolaev & Jacobson, 2010). The temperature change counter is incremented prompting an update of the temperature (cooling schedule t_k). At the end, we have $c_0 + c_1 + c_2 + \ldots + c_k$ total iterations executed. If $c_k = 1$ for all k, then the temperature changes at each iteration. The temperature is updated using a cooling rate α where $0 < \alpha < 1$. The cooling rate must be somehow close to 1 to make sure the temperature decreases at a very low rate to avoid early convergence of the search.

2.2.6 Applications of Metaheuristics

In this section a list of some recent applications of these metaheuristics discussed earlier are presented with their references given in Table 2.10.

Table 2.: Recent Applications of Metaheuristics

PROBLEM	ALGORITHM(S)	REFERENCE
Open Shop Scheduling	ACO: Beam-ACO	(Blum, 2005)
Knapsack Problem	ACO: Binary Ant System (BAS)	(Kong *et al.*, 2008)
Decision Tree Induction	ACO: Ant-Tree-Miner (ATM)	(Otero *et al.*, 2012)
TSP	ACO: Individual Variation & Routing Strategies (IVRS), ACS-TSPTW	(Jun-Man & Yi, 2012), (Cheng & Mao, 2007)
	EA: Generalised Chromosome Genetic Algorithm (GCGA)	(Yang *et al.*, 2008)
Vehicle Routing Problem (VRP)	ACO: ACS, Multi-ant Colony System (MACS)	(Montemanni *et al.*, 2005), (Gajpal &

		Abad, 2009)
	ILS	(Cuervo *et al.*, 2014), (Silva *et al.*, 2015), (Morais *et al.*, 2014)
	TS	(Jia *et al.*, 2013)
	Integer Programming & SA	(Wang *et al.*, 2015)
Protein Folding	ACO: ACO-HPPFP-3	(Shmygelska & Hoos, 2005)
Graph Colouring	ACO: Ant System, ANTCOL(LS)	(Bui *et al.*, 2008), (Dowsland & Thompson, 2008)
	ILS	(Caramia & Paolo, 2008)
	TS: Multistart Iterated Tabu Search (MITS)	(Lai & Lu, 2013)
Quadratic Assignment Problem (QAP)	EA: GA	(Tsutsui & Fujimoto, 2009)
	TS vs. SA	(Hussin & Stuetzle, 2014)
	ILS	(Stuetzle, 2006)
Permutation Flowshop Problem (PFSP)	ILS	(Dong *et al.*, 2009)
Job Shop Scheduling Problem (JSP)	TS & Path Relinking	(Peng *et al.*, 2015)
Cyclic Bandwidth Problem (CBP)	TS	(Rodriguez-Tello *et al.*, 2015)
Hybrid Flowshop Scheduling	TS	(Bozejko *et al.*, 2013)
Clustering	EA: Grouping Genetic Algorithm (GGA)	(Hong *et al.*, 2015)

2.3 COMPARISON OF UCTP TO GRAPH COLOURING

Graph Colouring Problem (GCP) is defined as the following:

Given a simple and undirected graph G (V, E) containing a set V of vertices and a set E of edges between the vertices. Then an optimal solution to the problem is to colour the vertices with **k** colours such that adjacent vertices do not get coloured by the same colour and the value of k is minimum. A graph is said to be k-chromatic if the least number of colours that can be used to colour it is k. A simple timetabling problem (with only event-clash constraints put into consideration) can be easily converted to a GCP. To make a successful conversion, the events are seen as vertices and edges are created between a pair of conflicting events. Each colour used in the resulting GCP solution represents each timeslot; therefore a vertex **v** coloured with a colour **b** is equivalent to an event **v** placed in a timeslot **b**.

Example 2.2: There are six committees in Computer Science Department who are to meet during the first week of classes in a new semester and a schedule must be made for their meetings. The members of each committee are listed below:

Undergraduate Education (U): Dr. Oluwagbemi, Dr. Olajide, Dr. Oyelade, Dr. Azeta

Graduate Education (G): Dr. Daramola, Dr. Adebiyi, Dr. Azeta, Prof. Misra

Colloquium (C): Dr. Oladipupo, Dr. Marion, Dr. Okuboyejo

Library (L): Dr. Omogbadegun, Dr. Oluwagbemi, Dr. Oladipupo

Staffing (S): Dr. Daramola, Dr. Marion, Dr. Adebiyi, Dr. Olajide

Promotion (P): Dr. Adebiyi, Dr. Omogbadegun, Prof. Adebiyi

It should be noted by the reader that the generated committee is simply for the purpose of this work and not seen as the probable one if such committees exist.

Back to Example 2.2, if there are only three timeslots available for these committees to meet, then a schedule is to be made using these available timeslots such that no two committees with at least a common member share the same timeslot.

The scheduling process starts by identifying conflicting events (committees with at least one member in common) and the committees Colloquium and Library are such events. A graph is drawn to represent the problem fixing the right edges between the conflicting event-pairs and then a solution is provided (appropriate colouring of the

graph such that no two conflicting vertices share the same colour). The solution process of this problem example is illustrated in Figure 2.1(a-c).

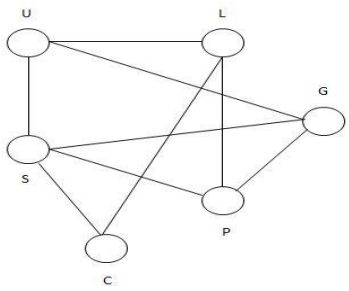

Figure 2.1(a): The scheduling problem converted into its GCP

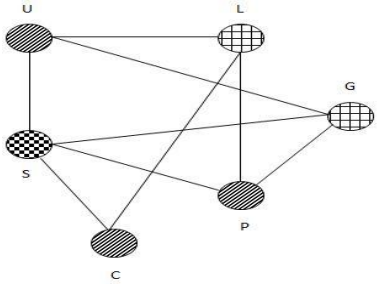

Figure 2.1(b): The solved GCP

1	2	3
U	L	S
P	G	
C		

Figure 2.1(c): The real schedule converted from the solved GCP

2.4 REVIEW OF METHODS USED TO TACKLE UCTP

In this section a list of research methods like the use of metaheuristics especially the ones discussed in Section 2.2 and other methods that have also been used like Case-Based Reasoning (CBR), Integer Linear Programming (ILP) and Constraint Programming (CP) are discussed on how they have been used to solve different instances of the UCTP.

2.4.1 Simulated Annealing

Basir *et al.,* (2013) used a simulated annealing algorithm to tackle the UCTP instance of a Malaysian university, considering parameters such as number of subjects, number of timeslots, number of class rooms, number of teachers, number of students and number of workloads. Unlike some other approaches where lecturers' course-clash is not given a consideration, they considered this scenario in their hard constraints formulation. Their approach spans through five stages:

(1) Data Collection: They achieved this by conducting interviews with administration staffs, teachers and students and analysed data from previous semesters over a two-year period.
(2) Formulation: They used the existing fitness function to make updates on the hard and soft constraints of their problem instance.
(3) Model: Modelling of the fitness function with the SA method
(4) Testing: the algorithm was tested with their proposed fitness function.

(5) Implementation: The SA method guided by their fitness function was implemented to solve their problem instance.

2.4.2 Evolutionary Algorithms

Sigl *et al.,* (2003) implemented a genetic algorithm procedure for the UCTP, the algorithm starts from an infeasible solution to a feasible one. In their approach, the fitness of an $\mathbf{i^{th}}$ individual in the population denoted as $\mathbf{pop_i}$ is calculated as the following:

$$fitness(pop_i) = ncf * K + Quality \tag{2.13}$$

Where **ncf** represents the number of conflicts in the individual timetable and **K** represents the weight of penalising this constraint. The variable **Quality** which simply means the quality of the timetable is determined by 'early schedule' of classes; a timetable with a lot of classes scheduled in the early hours of the day like in the morning will have a very high quality rating. The main drawback of the quality measure in this method is that the amount of free periods for the students will likely be reduced rendering it not applicable for a problem domain where free periods are favoured.

Rossi-Doria and Paechter (2004) implemented a memetic algorithm on benchmark instances of the International Timetabling Competition comparing their results with the first four best results recorded at the competition.

The Grouping Genetic Algorithm (GGA) which is a specialised form of GA for grouping problems has been applied to the UCTP (Lewis & Paechter, 2005). The timetable solution representation is done through a two-dimensional matrix $T_b(N(r) \times N(t))$ of N(r) rows and N(t) columns where N(r) is the number of rooms and N(t) is the number of timeslot. Therefore with this representation, an entry $T_b(r, t)$ represents an event to be scheduled in room **r** and timeslot **t**; a blank entry means no event is placed in r and t. Their method restricts the search procedure to 45 timeslots, although they initially attempted scheduling events in **t** timeslots where $t < 45$ and then increment the timeslot as the solution process proceeds. Their method includes the regular features of a conventional genetic algorithm: recombination and mutation.

Quarooni and Akbarzadeh-T (2013) also proposed a memetic algorithm to achieve the following: To test it with other existing algorithms and to study the significance of

evolutionary operators on some benchmark instances of the course timetabling problem. Their solution to UCTP is represented by a two-dimensional matrix of r x t dimension where r is the number of rooms and t is the number of timeslots.

Their memetic algorithm combines the GA and the local search procedure using a population of size **p**, an initial population is created, the algorithm shuffles the events and timeslots of each timetable in the population to achieve diversity. These timetables in the initial population may contain a set of unscheduled events that are to be scheduled later in the course of the algorithm run. Three stages follow the initialisation of the population, they are: recombination, mutation and local search. The recombination operation has four steps: selection, injection, duplicate removal and reinsertion. The selection sub-stage randomly picked entries in the two timetables selected for 'mating'. At the injection sub-stage, the entries selected in parent are replaced by the corresponding entries in the second parent. The third sub-stage as the name implies involves removing duplicate events that might have been copied from the second combining parent. Reinsertion which is the final sub-stage of the recombination operator attempts to fix in unscheduled events into the timetable by selecting a feasible slot that each unscheduled event can be placed; and in case of multiple slots available, a slot is chosen at random. The recombination operator gets completed by applying the same four stages all over again; but this time reversing the roles of the combining parents to produce the second offspring. The last two stages; mutation and local search (which has three stages of its own: slot selection, shoving and reinsertion) immediately follow after recombination. These three stages are repeated until a feasible timetable is constructed.

2.4.3 Tabu Search

The Tabu Search metaheuristics has been employed to solve UCTP (Aladag *et al.*, 2009), although the major feature of the research presented in the paper was to explore and compare the effectiveness of four neighbourhood moves. The neighbourhood moves compared are simple (S^N), swap (W^N), mixed1 (M_1^N) and mixed2 (M_2^N). S^N is characterised by moving an event to a different timeslot, the W^N swaps the timeslots of two randomly chosen events, M_1^N and M_2^N are combinations of the first two albeit in different ways. From their study, S^N and M_1^N which gave similar results were concluded to be the best neighbourhood moves for the Tabu Search approach for course timetabling.

Lü and Hao (2010) implemented an Adaptive Tabu Search (ATS) for curriculum-based course timetabling (CB-CTT); a variant of UCTP. Their proposed algorithm spans through three phases; initialisation, intensification and diversification. In initialisation, a fast greedy heuristic is used to construct an initial feasible solution. Intensification is achieved by Tabu Search and then the perturbation feature of ILS is used to diversify the search when the TS algorithm cannot find a better solution anymore. From their report, ATS achieved better results than when only ILS or TS was used.

The TS metaheuristics was also successfully applied to the UCTP instance of University of Dar es Salaam, Tanzania (Mushi, 2006).

2.4.4 Ant Colony Optimisation

Socha *et al.,* (2002) became the first set of researchers to implement MMAS for UCTP. Following strictly the features of the MMAS algorithm, all pheromone values associated with an edge connecting each event to a timeslot on the construction graph are initially set to the maximum pheromone value. Some computations were carried out to determine how 'hard' each event is[1] and then the evaluations are used to sort the events in descending order of 'hardness'. In the construction phase, each ant completes an assignment of timeslots to events and an algorithm is used to make room assignments for these event-timeslot pairs. The best assignment C_b (the best solution constructed after comparing all solutions of the ants used) is selected for further improvement by a local search algorithm (Rossi-Doria *et al.,* 2002). If C_b is better than the best solution found so far, then it replaces it and then the pheromone update takes place using the best assignment found so far. The algorithm run continues until a time limit is reached. In conclusion, it was reported from experimental tests that the use of heuristic information did not improve the quality of timetables constructed by the MMAS+LocalSearch procedure but it did improve the quality of the solution when no local search is used. The MMAS algorithm was tested against a random restart local search (RRLS) procedure to prove that the ants were actually involved in generating quality solutions (Socha *et al.,* 2002; Socha, 2003) and the report showed that the MMAS approach outperformed the RRLS method.

[1] An event's hardness is defined by the difficulty of finding a feasible timeslot for it

Socha *et al.,* (2003) compared the performances of two of the most successful variants of ACO; ACS and MMAS on some instances of the UCTP. The MMAS was specifically designed to replace ACS as the ACO procedure used in the Metaheuristics Network (Blum & Manfrin, 2015) comparison of metaheuristics on UCTP instances (Rossi-Doria, *et al.,* 2003). MMAS was more successful than ACS when their results were compared.

ACO algorithm was used as the solution method for a Laboratory Exercises Timetabling Problem (LETP) by Matijas *et al.,* (2010), implementing the MMAS variant. The MMAS method implemented outperformed the Greedy Randomised Adaptive Search Procedure (GRASP) procedure which was compared with it in terms of performance.

Thepphakorn *et al.,* (2014) implemented two of the newest variants of ACO; Best-Worst Ant System (BWAS); based on AS and Best-Worst Ant Colony System (BWACS); based on ACS for the UCTP. They compared the performances of the two algorithms with other established ACO variants which included AS, ACS, MMAS, Elitist ant System (EAS) and $AS_{RANK.}$ From their results, BWACS recorded the overall best result among the seven compared variants although AS_{RANK}, MMAS and ACS recorded the best results for the small instances used for comparison.

2.4.5 Comparison of Metaheuristics

EA, TS, ACO, ILS and SA were all compared in Rossi-Doria *et al.,* (2003) as part of the research effort of the Metaheuristics Network. To make a fair comparison among all the algorithms implemented, the same solution representation and local search procedure proposed in (Rossi-Doria *et al.,* 2002) were used for all metaheuristics approaches compared. They used a neighbourhood move (N1 ∪ N2) which is a combination of two neighbourhood moves N1 and N2. N1 moves a single event to a different timeslot while N2 swaps the timeslots of two events. The variant of EA implemented was the Memetic Algorithm while the variant of ACO algorithm implemented was the Ant Colony System (ACS). For the ILS implementation, three perturbation operators were considered:

P1: place an event in a randomly chosen timeslot different from its present timeslot

P2: swaps the timeslots of two events chosen randomly

P3: randomly choose between P1 and P2 for a 3-cycle of timeslots for three different events.

Additionally, the three acceptance criteria discussed in section 2.2.2 were tested in the ILS procedure and the best configuration found from the experiment (by its level of performance) was the SA_Accept(.) acceptance criterion for each of the instance tested.

In conclusion, ILS recorded the best performance while SA did not manage to construct any feasible solution in all the trials done on the two large instances of the problem considered.

2.4.6 Hybrid Approach

Hybrid approaches in UCTP involves the combination of two or more methods by selecting the features that make each individual method stand out for the hybridised approach. A combination of the Great Deluge Algorithm (GDA) and Tabu Search was proposed and implemented for the UCTP in Abdullah *et al.,* (2009). The GDA for a brief introduction was introduced alongside the Record-Record Travel (RRT) algorithm in the same paper based on the success of a similar approach called Threshold Accepting (TA) (similar to simulated annealing) proposed earlier (Dueck, 1993). The construction of an initial feasible solution was made by the least saturation degree (LSG) heuristic algorithm and two neighbourhood moves (applied if the former do not construct an initial feasible solution). The optimisation stage of their method encapsulates the two algorithms (GDA and TS), during this stage, hard constraints are not violated in order to maintain feasibility. The algorithm is described in ALGORITHM 2.6.

ALGORITHM 2.6: HYBRID ALGORITHM (GDA & TS)

$S_{init} \leftarrow$ initial solution by the constructive heuristic

calculate $f(S_{init})$ % fitness of S_{init}

$S_{best} \leftarrow S_{init}$

while not termination condition **do**

  $GD_{best} \leftarrow GreatDeluge(S_{best})$

  $TS_{best} \leftarrow TabuSearch(S_{best})$

  $S^* \leftarrow better(GD_{best}, TS_{best})$

if $f(S^*) < f(S_{best})$

$S_{best} \leftarrow S^*$

end if

end while

The first two lines of the algorithm set up the initial solution and also calculate its fitness making up the first stage of the algorithm. The best solution so far is initialised by the initial solution at this stage before the improvement stage (while loop) kicks off. The improvement stage is iterated until certain termination criteria are met. The hybrid-pair of algorithms; GDA and TS are run independently on the best solution found so far and their best results are compared, the better of the two is used to set the intermediate best solution (S*). If the fitness of S* is better than that of S_{best} then it is accepted as the new best solution otherwise it is rejected.

Yassin *et al.,* (2013) hybridised the Tabu Search and Non-Linear GDA (an extension of GDA) to tackle UCTP and compared the result of their approach with that of 28 other state-of-the-art methods. Their method incorporates one of the key features of Tabu Search; the tabu list with a tabu tenure of 6 which was the best after considering other tabu tenures in the set {2, 4, 5, 7, 8} according to the researchers. Three tabu lists were used from their report, each one specified for each neighbourhood move used in the research; therefore if an event **e** is involved in a neighbourhood move N_k that reduces the objective cost, it is placed in the tabu list T_k associated with $N_{k.}$ The tabu list only guides the Non-Linear GDA from being trapped in local optima and at the same time influencing it in choosing diverse components apart from the ones in the tabu list to improve solution quality. From their result presentation, their approach ranked second using five small instances, five medium instances and a large instance of the UCTP to make the judgement.

Three different methods; Tabu Search, Variable Neighbourhood Descent (VND); similar to ILS and Simulated Annealing were combined in Chiarandini *et al.,* (2006) to form a hybridised method for the UCTP. They made use of four neighbourhood moves $N_1, \ldots, N_4$ for their approach. Their method implemented 60 construction heuristics for building 60 different initial assignments and then each assignment is improved by a local search procedure specifically designed to find a feasible solution (satisfy all hard constraints). The best derived feasible solution from the 60 construction heuristics is

then selected to be improved further by minimising its soft constraints violations. This method was reported to comfortably outperform the best method submitted to an International Timetabling Competition (ITC); which 24 algorithms made entries. ALGORITHM 2.7 describes their method:

ALGORITHM 2.7: HYBRID METHOD IN CHIARANDINI *et al.*, (2006)

Input: $I_k \in I$ % an instance of the problem

Pre_processing(I_k)

begin

T → Ø

t ← Build_Initial_Solution(h)

t ← Find_Feasible_Solution(t)

Q(t) ← Assess(t)

T → T ∪ {t}

repeat ∀ h ∈ H

t* ← Select_Best(T)

t* ← Optimise(t*)

Output: t* % best timetable constructed

T: set of assignments (timetables) initially constructed from the construction heuristics

Q(t): returns the quality of a given timetable **t**

H: a set of construction heuristics

t*: the best in T

Each construction heuristic h_i (i = 1, 2, …, 60) is used to generate an initial timetable t_i which will be made feasible by Find_Feasible_Solution(t); a local search procedure that initially uses the first two neighbourhood moves (N_1 and N_2) to improve the quality of a timetable. The Tabu Search procedure is triggered if the combination of N_1 and N_2 fails to produce a feasible solution after five iterations. The tabu search procedure uses a local search that only makes moves based on N_1. When all t in T are made feasible, their qualities are evaluated and then the best one (t*) is selected for optimisation by the procedure Optimise(t*). In optimising t*, the VND algorithm which combines all the neighbourhood moves is used to ease the soft constraints violations without violating any hard constraints. The VND runs until no improvement

in the solution is found and then the control is passed to the SA procedure. The possibly further improved t* by Optimise(t*) is returned as the solution of the hybridised method.

Tuga *et al.*, (2007) applied a hybrid simulated annealing with kempe chain local search move for the UCTP. Their hybrid approach called Hybrid Simulated Annealing (HSA) uses a combination of LSD and Least Degree (LD) heuristics to generate an initial feasible solution. The HSA method is activated after the completion of the heuristics used to generate the initial solution. In HSA, SA is the first procedure which attempts to reduce the number of soft constraint violations (#scv). The kempe chain move is then triggered if the SA procedure does not find an improving solution after a pre-defined number of steps. Finally, the initial temperature which is an integral part of any SA algorithm was set to a value such that the probability of accepting a worse solution is adequately high. The temperature is updated as follows:

$$T(i+1) = \frac{1}{\left(1/_{T(i)}\right) + \beta} \tag{2.14}$$

$where\ 0.0005 \leq \beta \leq 0.001$

The number of trials for each temperature is set to a * |V| where V is the set of vertices in the construction graph of the problem. The value of **a** is initially set to 10 and then linearly increased after every maximum number of iterations for each temperature.

A hybrid method can also be composed of Constraint Programming (an operational research method) and Simulated Annealing, this pair has been used to tackle examination timetabling (Duong & Lam, 2004). Since the focus of this work is on the UCTP, details about this will not be captured here.

2.4.7 Case-Based Reasoning

In case-based reasoning (CBR), old experiences are often employed to analyse and solve new problems or used to evaluate the solutions proffered to new problems. A strong example how case-based reasoning is done is illustrated in (Kolodner, 1992). It can also mean reworking of old solutions to meet new situations and using the old cases to explain or analyse new solutions (Kolodner, 1992).

Edmund Burke is one of the leading names when it comes to CBR and course timetabling, in his work along with three other researchers (Burke *et al.,* 2001), he experimented on the CBR approach and claimed that high quality timetables can be retrieved from similarly existing timetables with little effort (Burke *et al.,* 2001). The fitness function for a timetable generated for the new case is given as:

$$fitness = (\#hcv + unsch) * 100 + \#scv * 5 \quad (2.15)$$

where,
$\#hcv = number\ of\ hard\ constraints\ violations$
$unsch = number\ of\ unscheduled\ events$
$\#scv = number\ of\ soft\ constraints\ violations$

To generate a timetable for the new case, the following processes are made:

(a) Similarity Measure: They evaluated the similarity between a new case C_{new} and a case C_{old} in the case base. This similarity measure takes into account the cost of substitutions, insertions and deletions of vertices and edges to and from the new case.
(b) Branch and Bound technique: The retrieval process implemented in this work need to search through the decision tree to find all cases in the case base which are similar to the new case. To reduce computation time, the branch and bound procedure is used to trim the size of the tree during retrieval.
(c) Reuse and Adaptation: Matched courses are substituted while the unmatched courses in the retrieved case are deleted. Courses that violate the constraints in the newly constructed timetable are placed in an unscheduled list; sorted in descending order of placement difficulty. The same is done for courses in the new case that are not yet scheduled.

Finally, the unscheduled courses are tried to be placed using a graph heuristic method; then the timetable with the least constraints violations is picked as the solution for the new case.

Another CBR method mainly governed by heuristics has been used to tackle the UCTP in Burke *et al.,* (2006). Other papers published where CBR method has been used for the UCTP and Examination Timetabling include (Burke *et al.,* 2006) and (Petrovic *et al.,* 2007) respectively.

2.5 REVIEW OF EXISTING TIMETABLING SYSTEMS

This section is similar to the last but differs from it since some of the research papers mentioned in the last section applied their algorithms to generated instances rather than real-world instances of the UCTP. The papers reviewed here used their methods to tackle the UCTP instance of their institutions and reported that the working system is being used in their institutions for timetable construction on a regular basis.

2.5.1 UniTime

UniTime reported in Mueller *et al.,* (2010) is an interactive timetabling system developed and applied successfully at Purdue University. It is a web-based system using the Enterprise Edition of Java (J2EE) and an XML interface for communicating with other systems used by a University. The system does not only provide schedules for UCTP but also for examination timetabling problem and student sectioning problems. These three problems are modelled as Constraint Satisfaction Problems (CSPs) and the major goal of the system is to make changes to a timetable when inevitable requests are made.

The system uses colours to mark timeslots and rooms for the following classes: Prohibited, Preferred, Strongly Preferred, Discouraged and Strongly Discouraged. The following are the possibilities during a user's interaction with UniTime:

(1) Commit a change made to the timetable
(2) Abandon a change made to the timetable
(3) Approve a suggestion made by the system
(4) Select a placement for an event after ignoring possible suggestions made by the system
(5) Remove a selected assignment
(6) Select another event to replace one of a conflicting pair of events

The system documentation and great details of UniTime can be found on the website: http://www.unitime.org/.

2.5.2 MAS_UP-UCT

MAS_UP-UCT stands for "Multi-Agent System for University Course Timetable Scheduling". This is a multi-agent based system for solving the UCTP (Oprea, 2007).

The architecture of the system developed which is depicted in Figure 2.2 assumes there are five faculties (F_1, …, F_5) in the University with each one having its own scheduler multi-agent system (MAS-F_i) for scheduling courses in the faculty. The main scheduler agent (MScheduler Agent) is used to allocate rooms. Each faculty scheduler agent is designed to communicate with others to avoid critical situations that may arise when scheduling courses that Professors teach because Professors most times teach in more than one faculty.

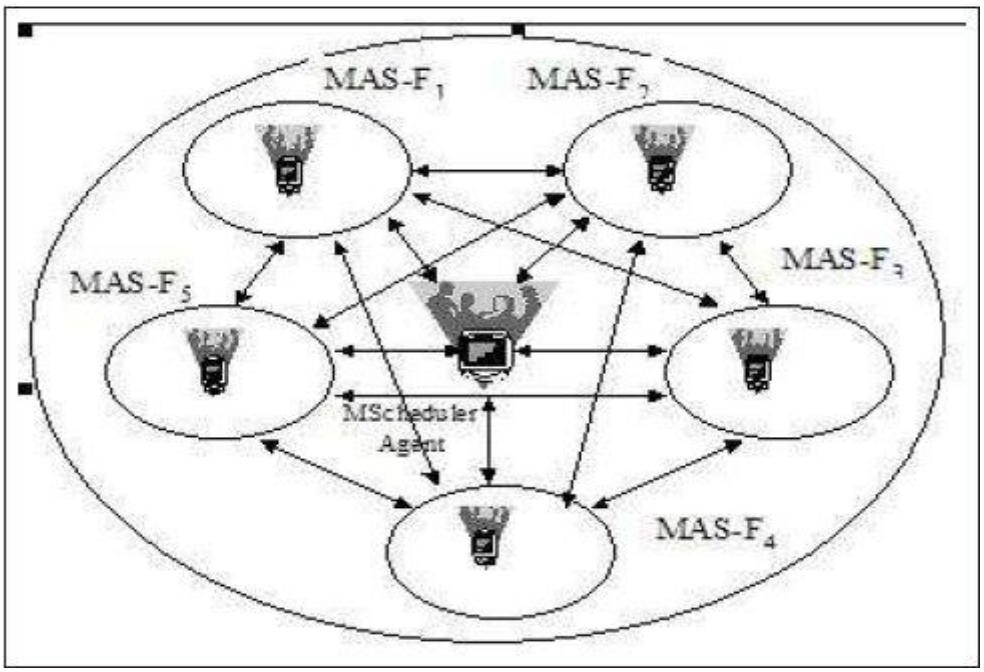

Figure 2.2: MAS_UP-UCT architecture (Source: Oprea, 2007)

2.5.3 Automated System for University Timetabling

The system (Murray & Mueller, 2006) developed here is very similar to UniTime in interface and architecture. The constraints-based solver utilizes an iteration-forward search algorithm. The authors of this paper attempted to address critical aspects of UCTP which are not often found in literature; these include the issue of satisfying all departments to avoid partiality towards one of them, ability to check and resolve inconsistencies in input data and the ease at which an existing feasible timetable can be modified, the creation and management of constraints, and the ability to deal with vagueness in the problem formulation.

Older existing systems developed for the UCTP include one developed for the University of Waterloo (Carter, 2001) and another based on Constraint Logic Programming developed for a University in Berlin (Goltz & Matzke, 1999).

2.6 SUMMARY

A survey of five metaheuristics has been presented in terms of their underlying principles, a non-exhaustive list of applications of these metaheuristics to COPs and their applications to the UCTP. A similarity between the Graph Colouring Problem and the University Course Timetabling Problem has also been presented with a practical example given. In terms of methods that have been applied to UCTP, an effort has been made to review these methods (especially the metaheuristics). The review of the methods does not end there, methods like Integer Programming (Boland *et al.,* 2008), Constraint Logic Programming (Rudova & Murray, 2003) are other approaches based on Operational Research that have been used to tackle the problem. Finally, a population-based metaheuristic algorithm called Harmony Search algorithm has also been employed for UCTP with fairly good comparative results when compared with other existing metaheuritics (Al-Betar *et al.,* 2012). The MMAS algorithm of Socha *et al.,* (2002) will be adopted for CU's instance to bridge the gap between generated instances of the problem and real-world instances of the problem by dealing with events requiring more than one timeslot and also investigating the influence a local search design has on the quality of the solution generated by the MMAS algorithm.

CHAPTER THREE

SYSTEM DESIGN, MODEL FORMULATION AND METHODOLOGY

3.1 INTRODUCTION

The System (UCTP) developed starts its timetable construction by asking a simple question from the user about which semester the timetable is to be constructed for; an invalid response will make the system load instances for the first semester (which was set as the default setting). The result is displayed on an excel spread sheet, the choice of the excel spread sheet was made because it is the most natural platform to display a timetable. The system implements eight (8) modules (classes and structures) which were carefully integrated for easy communication and debugging. The classes implemented in the system include **MMAS_UCTP**, **Problem**, **Solution**, **ANT** and **util**. The structures include **struct_room**, **struct_program**, **and struct_event**. The details and functions of these modules are discussed later during the presentation of the way these modules interact with one another. The module **util** carries out helping functions to reduce code duplication in several modules that make use of the class.

3.2 SYSTEM ARCHITECTURE

This section presents the system architecture (Figure 3.1) showing the interactions between the main components of the system.

UCTP constructs a feasible timetable using five ants (from the architecture depicted above) over several iterations until a stopping criterion is met. The local search module runs the local search procedure on all the intermediate solutions constructed by the ants. The text file contains data that will be processed by the **Problem** module. The best solution constructed is encoded into an excel spread sheet as the final solution of the system.

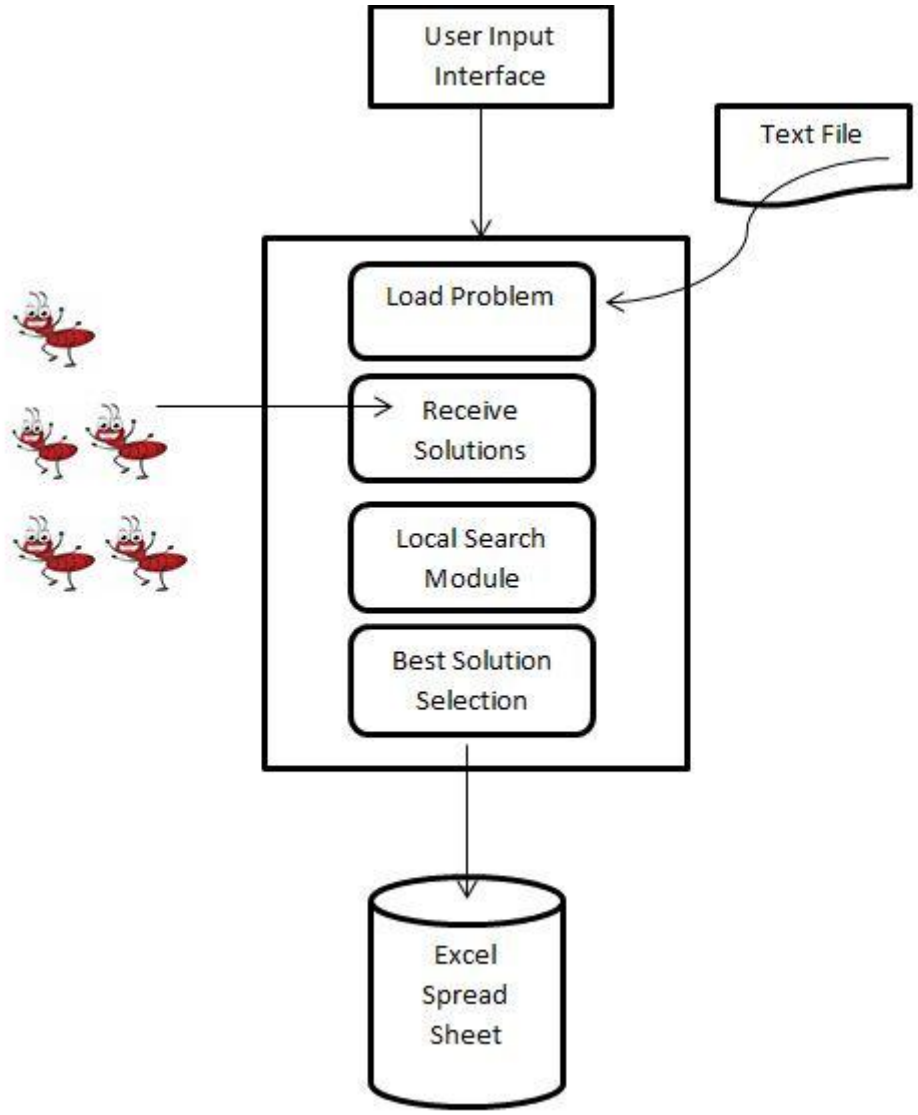

Figure 3.1: System Architecture of UCTP

3.3 FILE STRUCTURE

Text files are the main resource point for this system, they are crucial to the successful loading of the problem data (events, programmes, rooms, and the likes) into the system via the **Problem** module. The files used for this system are divided into five categories: Rooms, programInfo, programFiles and 17 program-events files and 17 program-rooms files.

In the first category, there is only one file which contains the classrooms information; name and capacity. A line of text in Rooms file is in this format: Room-Name:[Capacity], the **Capacity** value of a room is an optional value; it is not required for labs (Physics Lab, Chemistry Lab, Computer Lab, etc.) because the required

number of students needed in a program are admitted and the labs are generally suitable to contain the number of students that will have their lectures there. If the lab is not going to contain the required number of students for an event like we have in some cases, then it is not considered or the students are divided into sections. These student sections are then scheduled into different sections of the event like we have in CHM119 and PHY119; in this case, the capacity constraint will be respected through sectioning.

The file programInfo contains the programme names and the number of students in each programme, Computer Science 100 level students (coded as CSC100) is a different programme to Computer Science 200 level students (CSC200). The format of the lines of text in the file is: ProgramName:NumberOfStudents.

The file programFiles contains the files to be checked for each programme when the events offered by students in those programmes are to be captured by the system. For example, the programme ARC300; representing 300 level students of Architecture has the file name ARC.txt stored in programFiles.

The files in the fourth category stores the events of students in 3 – 5 programmes are involved in, the file names are stored in the last file discussed (programFiles). Students in CSC100, CSC200, CSC300 and CSC400 all have their events-list stored in the same file (CSC.txt) but separated by a delimiter (*); for example, the last event for the programme CSC100 and the first event for the programme CSC200 are separated by a string of the character '*'. In a nutshell, students in similar programmes (studying the same course but in different levels) get their events stored in the same file but their lists are separated by a delimiter. Programmes that have options like Chemistry and Physics are treated slightly differently to cater for their peculiarities. The delimiter is used by the system to determine which specific programme events are being read for; during the reading of events into the system, the number of delimiter (**del-counter**) is updated as soon as a delimiter is read therefore, if the counter is 2, the system 'knows' that it is reading events for 300 level students of that programme.

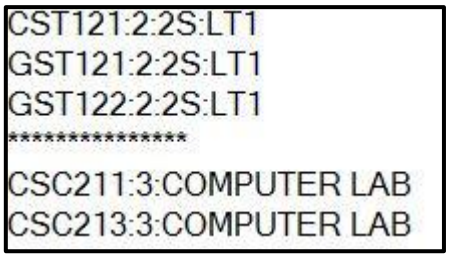

Figure 3.2: A snapshot of a section of a file with a delimiter

Figure 3.2 depicts the use of delimiter in the file category program-event; specifically showing the separation between the event lists of CSC100 and CSC200 programmes. The format for storing texts in this file category is: EventName:Duration:[Semester]:[SuitableRoom]. A line of text in this category represents the details of a particular event, for example, "CSC211:3:COMPUTERLAB" is interpreted as "CSC211" as the event, "3" as the duration (number of hours the event needs in a week) and "COMPUTERLAB" as the room where this event must hold. The "Semester" part of a line may be missing, if missing it means the event is for the first semester or for the second semester if otherwise. The term "Duration" might be seen as a misnomer but it was used rather than "Unit" because it actually gives the exact depiction of things; for example the event PHY119 of 1 unit has a duration of 6 (3 hours each on two different days of a week).

The files in the last category are similar to those ones in the fourth category but store the probable rooms where events concerning these programmes can be scheduled rather than events details. The names of these files are derived from the names of the files in the last category; a program-events file called "CSC.txt" has its corresponding program-rooms file named as "CSC-ROOM.txt".

3.4 CLASSES AND THEIR INTERACTIONS

In this section, the details of the modules introduced in Section 3.1 are given along with the interactions between them. The structures in the system are also grouped as classes in this section since they are a kind of simplified classes (Savitch & Mock, 2009), a class without method can be directly converted into a structure. **MMAS_UCTP**, **Problem**, **Solution**, **ANT**, **util, struct_room**, **struct_program**, **and struct_event** are the classes implemented and they can be regarded as the different components of the system. Table 3.1 gives the details of these classes.

Table 3.: Classes and their details

Class	Function
ANT	Encodes the virtual ants used in constructing the timetable
Problem	Responsible for loading events, rooms, programmes and every other variables representing the problem to be tackled from various text files
Solution	Encodes a solution to the UCTP; can only be accessed via objects of ANT
MMAS_UCTP	Implements the MAX-MIN Ant System for the problem using instances of ANT
struct_event	Implements the structure for encoding the events
struct_program	Implements the structure for encoding the programmes
struct_room	Implements the structure for encoding the rooms
Util	Implements helper functions for the main classes

3.4.1 Class Diagram

The Class diagram shows the main components of any object-oriented system. Class diagrams depict a static view of the model, or part of the model, describing the model's attributes and behaviour rather than the details of how operations are carried out. Important concepts such as generalisation, aggregation and association describe the inheritance, composition and interactions between classes respectively (Sparx Systems).

Figure 3.3 is a class diagram describing the components of the system and the interactions between them.

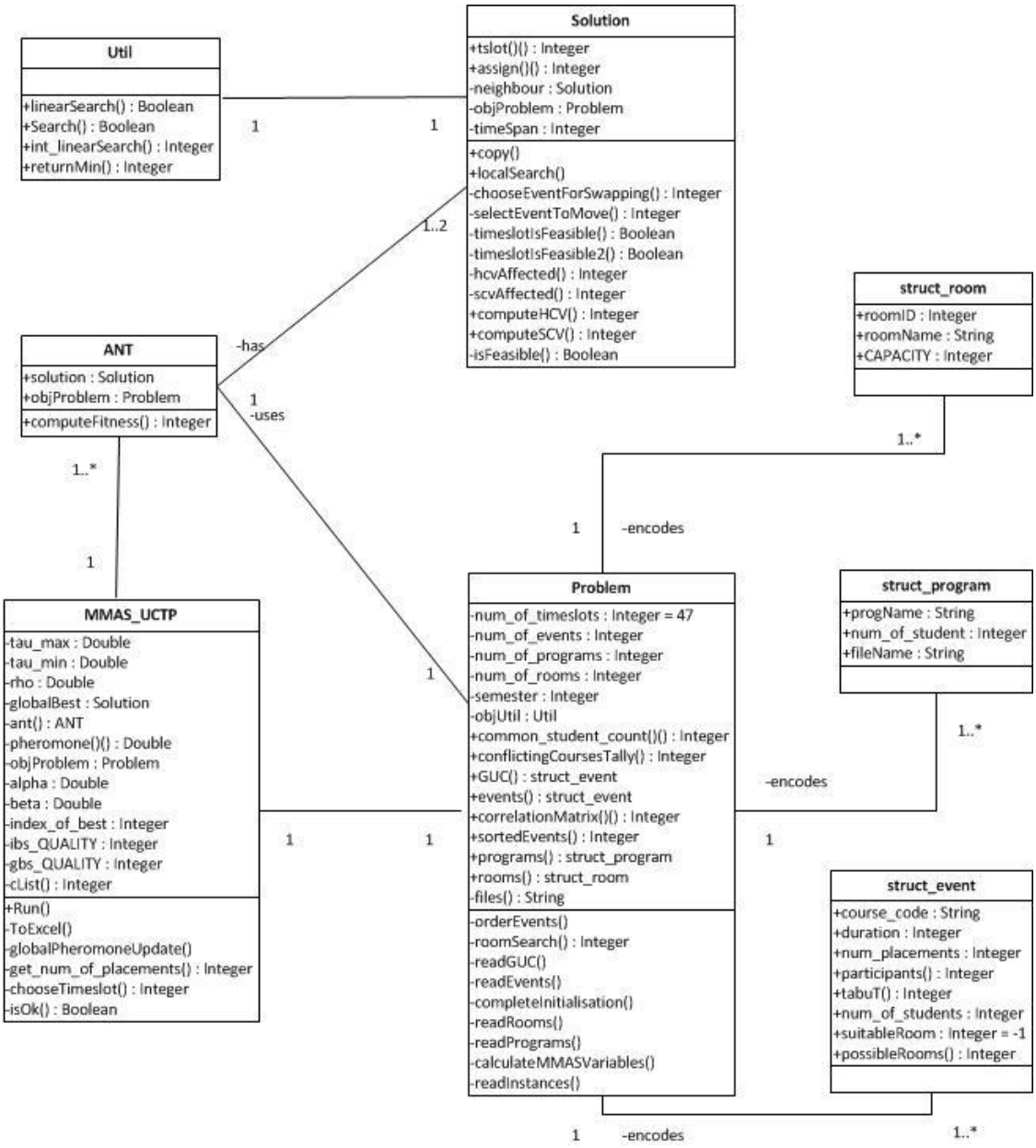

Figure 3.3: System's Class Diagram

3.5 MODEL FORMULATION

The problem can be formulated as a four-tuple (P, R, E, T) with the following definitions:

- A set P of **n** programmes (a programme represents a group of students) = $\{p_1, p_2, \ldots, p_n\}$
- A set R of **c** classrooms (laboratories included) = $\{r_1, r_2, \ldots, r_c\}$
- A set E of **d** events (classes to be scheduled) = $\{e_1, e_2, \ldots, e_d\}$

- A set T of **m** timeslots = $\{t_1, t_2, ..., t_m\}$ representing the periods available for these events to take place; m = 47. For example, t_1 is the period "8:00 – 9:00 am on Monday"

Two subsets of P; P_{jun} and P_{sen} exist, the first consists of 100-200 level students and the second consists of 300-500 level students.

For the purpose of simplicity, $T = T_{mon} \cup T_{tues} \cup T_{wed} \cup T_{thur} \cup T_{fri}$

$$T_{mon} = \{t_1, t_2, \dots, t_{10}\}$$
$$T_{tues} = \{t_{11}, t_{12}, \dots, t_{20}\}$$
$$T_{wed} = \{t_{21}, t_{22}, \dots, t_{30}\}$$
$$T_{thur} = \{t_{31}, t_{32}, \dots, t_{40}\}$$
$$T_{fri} = \{t_{41}, t_{42}, \dots, t_{47}\}$$

The first and last elements in the first four sets represent the periods "8:00 – 9:00 am" and "5:00 – 6:00 pm" respectively while the last element in the last set (timeslots for Friday) represent the period "2:00 – 3:00 pm".

Three sets of 'forbidden' timeslots exist, F_{jun}, F_{sen1} and F_{sen2}, they are defined below:

$$F_{jun} = \{t_{31}, t_{32}\}$$
$$F_{sen1} = \{t_{11}, t_{12}\}$$
$$F_{sen2} = \{t_{28}, t_{29}\}$$

The F_{jun} concerns students in P_{jun} while the other two concern the students in P_{sen}. The three sets enforce the timeslots chosen during solution construction to be subjected under the following three constraints:

- No event concerning students in 100-200 level should be placed between 8:00 am and 10:00 am on Thursday because of Thursday's Chapel Service.
- No event concerning students in 300-500 level should be placed between 8:00 am and 10:00 am on Tuesday because of Tuesday's Chapel Service.
- No event concerning students in 300-500 level should be placed between 3:00 pm and 5:00 pm on Wednesday because of EDS Practical.

Therefore, students in P_{jun} are entitled to 45 timeslots (standard maximum number from literature) since $|T - F_{jun}| = 45$ and students in P_{sen} are entitled to 43 timeslots since $|T - (F_{sen1} + F_{sen2})| = 43$ making events for such students harder to place.

In the CU's version of the problem, four additional constraints exist in addition to the three most common hard constraints of the UCTP (HC1-HC3). The hard constraints (seven of them) are presented below:

HC1: No conflicting events should be placed in the same timeslot

HC2: No room should be assigned to more than one event at the same time

HC3: The room assigned to an event should be large enough to handle it

HC4: Events concerning students in 100-200 level should not be placed in the first two timeslots of Thursday

HC5: Events concerning students in 300-500 level should not be placed in the first two timeslots of Tuesday

HC6: Events concerning students in 300-500 level should not be placed between 3:00-5:00 pm on Wednesday

HC7: An event requiring more than two timeslots divided into two sections should not have both sections scheduled on the same day.

The formal definitions of HC1 to HC7 are as follows:

Firstly, the following variable definitions are established:

$$x(p_n, e_d) = \begin{cases} 1, & if\ program\ p_n\ attends\ event\ e_d \\ 0, & otherwise \end{cases} \quad (3.1)$$

$$y(r_c, e_d) = \begin{cases} 1, & if\ room\ r_c\ is\ large\ enough\ for\ event\ e_d \\ 0, & otherwise \end{cases} \quad (3.2)$$

$$z(e_d, t_m, r_c) = \begin{cases} 1, & if\ event\ e_d\ is\ placed\ in\ timeslot\ t_m\ and\ room\ r_c \\ 0, & otherwise \end{cases} \quad (3.3)$$

The following hard constraints definitions follow:

$$HC1: \ \forall p \in P \ \forall t \in T \ \sum_{d=1}^{|E|} \sum_{c=1}^{|R|} x(p, e_d) * z(e_d, t, r_c) \leq 1 \quad (3.4)$$

$$HC2: \ \forall t \in T \ \forall r \in R \ \sum_{d=1}^{|E|} z(e_d, t, r) \leq 1 \quad (3.5)$$

$$HC3: \ \forall e \in E \ \sum_{m=1}^{|T|} \sum_{c=1}^{|R|} z(e, t_m, r_c) * y(r_c, e) = 1 \quad (3.6)$$

$$HC4: \ \forall p \in P_{jun} \ \forall t \in F_{jun} \ \sum_{d=1}^{|E|} \sum_{c=1}^{|R|} x(p, e_d) * z(e_d, t, r_c) = 0 \quad (3.7)$$

$$HC5: \ \forall p \in P_{sen} \ \forall t \in F_{sen1} \ \sum_{d=1}^{|E|} \sum_{c=1}^{|R|} x(p, e_d) * z(e_d, t, r_c) = 0 \quad (3.8)$$

$$HC6: \ \forall p \in P_{sen} \ \forall t \in F_{sen2} \ \sum_{d=1}^{|E|} \sum_{c=1}^{|R|} x(p, e_d) * z(e_d, t, r_c) = 0 \quad (3.9)$$

$$HC7: \ \forall dy \in D \ \ \forall p \in P \ \ \forall e \in E \ \sum_{i=1}^{A} \sum_{c=1}^{|R|} x(p, e) * z(e, t_i, r_c) \leq dur_1 \quad (3.10)$$

$$B = \begin{cases} 10dy, & if \ dy \leq 4 \\ 10(dy - 1) + 7, & otherwise \end{cases} \quad (3.11)$$

$D = \{1,2,3,4,5\} \equiv \{Monday, Tuesday, Wednesday, Thursday, Friday\}$

Two variables dur_1 and dur_2 represent the number of timeslots needed for the first and second sections of an event; they are computed via 3.12 and 3.13:

$$dur_1 = \begin{cases} dur(e), & if \ dur(e) \leq 2 \\ \dfrac{dur(e)}{2}, & if \ dur(e) > 2 \end{cases} \quad (3.12)$$

$$dur_2 = dur(e) - dur_1 \quad (3.13)$$

*where $dur(e)$ represents total hours required by the event **e** in a week*

The following soft constraints in the problem:

SC1: A student has a class in the last timeslot of the day (except Friday)

SC2: A student has more than two classes in a row

SC3: A student has exactly one class in a day

Formally, we have the following definitions for SC1 to SC3:

$$SC1 = \sum_{d=1}^{4} \sum_{c=1}^{|R|} \sum_{d=1}^{|E|} \sum_{n=1}^{|P|} z(e_d, t_{10d}, r_c) * x(p_n, e_d) \tag{3.14}$$

$$SC2 = \sum_{d=1}^{5} \sum_{n=1}^{|P|} \sum_{k=LB}^{A} con() \tag{3.15}$$

$$LB = 10(d-1) + 1 \tag{3.16}$$

$$A = \begin{cases} 10(d-1) + 8, & if\ d \leq 4 \\ 10(d-1) + 5, & otherwise \end{cases} \tag{3.17}$$

$$con() = \begin{cases} 1, & if \sum_{i=0}^{2} \sum_{c=1}^{|R|} \sum_{d=1}^{|E|} z(e_d, t_{k+i}, r_c) * x(p_n, e_d) = 3 \\ 0, & otherwise \end{cases} \tag{3.18}$$

$$SC3 = \sum_{d=1}^{5} \sum_{n=1}^{|P|} oneclass() \tag{3.19}$$

$$oneclass() = \begin{cases} 1, & if \sum_{k=LB}^{B} \sum_{c=1}^{|R|} \sum_{d=1}^{|E|} z(e_d, t_k, r_c) * x(p_n, e_d) = 1 \\ 0, & otherwise \end{cases} \tag{3.20}$$

The goal is to minimise the amount of soft constraints violations (#scv).

Where #scv = SC1 + SC2 + SC3.

The total number of soft constraints violation (#scv) is determined as the sum of the following components:

- Count the number of programmes participating in an event placed in the last timeslot of the day (except Friday) to compute SC1.
- Count the number of occurrence each programme has more than two consecutive classes (3 consecutive classes score 1, 4 consecutive classes score 2, 5 consecutive classes score 3, etc.) in a day to compute SC2.

- Count the total number of occurrence each programme has only one class in a day to compute SC3.

3.6 METHODOLOGY

In this section, the solution representation, the two main algorithms used in constructing a feasible solution to the problem (MMAS and Local Search) and briefly how rooms are assigned to events are presented. The data (courses and their details) used for the work and the experiments were gotten from the CST Undergraduate Academic Handbook of 2014-2017 which can be gotten from the link: (http://eprints.covenantuniversity.edu.ng/id/eprint/3269).

3.6.1 Solution Representation

The representation of a given solution is handled by the component "Solution" introduced in Section 3.4; during the process of constructing a solution by a given ant **k**, a vector V^T of timeslots (which are variable-length vectors themselves) is used to store the events scheduled by an ant k. V^T is a vector of size |T| x |E| where T and E are the sets of timeslots and events respectively, therefore an entry $V^T(i, j)$ represents an event j placed in a timeslot i. During the assignment of rooms to the iteration-best solution, the solution representation switches to a 2-dimensional matrix M of size |R| x |T| where R represents the set of rooms, an entry M(i, j) represents an event placed in room i and timeslot j.

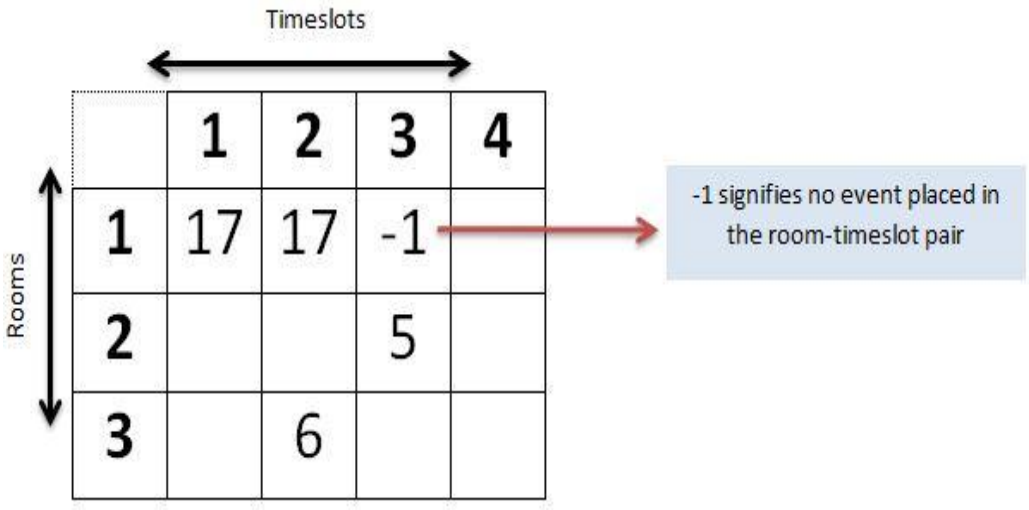

Figure 3.4: Pictorial representation of matrix M

Figure 3.4 presents the matrix **M**, assuming that there are three rooms and four timeslots in the problem. For example, Event 17 which occupies two timeslots (probably a 2-unit course) is assigned Room 1 while Event 6 placed in Timeslot 2 is assigned Room 3. Each of the blank spaces contains a -1 or appositive integer.

3.6.2 The MAX-MIN Ant System for the UCTP

A slight modification to the MAX-MIN Ant System implementation for the UCTP which can be found in (Socha *et al.*, 2002) is presented in ALGORITHM 3.1. A colony of **m** ants is used in the timetable construction with the local search procedure LS1() run on each solution constructed by the ants.

ALGORITHM 3.1: MAX-MIN Ant System for the UCTP

load problem instance

$\tau_{max} \leftarrow \frac{1}{\rho}$

$\tau(e, t) \leftarrow \tau_{max}\ \forall(e,t) \in E \times T$

compute c(e, e′) $\forall(e,e') \in E \times E$

compute d(e) $\forall e \in E$

$E_{sorted} \leftarrow$ sort(E)

while stopping criterion not met **do**

  for k = 1 to m **do**

    % ant **k** constructs a timetable

    A ← Ø

    for i = 1 to |E| **do**

      choose a timeslot **t** using the influence of pheromone and heuristic information for event $\mathbf{e_i}$ in E_{sorted}

      A ∪ {(e_i, t)}

    end for

    A ← improved solution using LS1() % apply main local search procedure

    $A_{ib} \leftarrow$ better(A, A_{ib})

  end for

  $A_{ib} \leftarrow$ assign rooms to events in A_{ib} while applying LS2()

  $A_{gb} \leftarrow$ better(A_{ib}, A_{gb})

  perform global pheromone update using A_{gb}, τ_{min} and τ_{max}

end while

Firstly, the problem instance is loaded and then parameter settings are done. Parameters set are evaporation rate (ρ), maximum pheromone (τ_{max}) and minimum pheromone (τ_{min}). The pheromone matrix τ(E X T) is initialised and all the values are set to the maximum pheromone τ_{max}. The pheromone trail between event **e** and a timeslot **t** represented as τ(e, t) is the relative likeness/possibility of timeslot **t** being chosen for the event **e**. Then some computations (pre-processing) are done before the events are sorted which will lead to putting to the top of the list the most difficult event to schedule based on the results of the computation. At the end of the solution construction by all ants, the best assignment made among the ants is improved further by a specialised local search procedure LS2() which is embedded in the procedure that assigns rooms to events placed in the best assignment. The better solution between the best assignment done in the last iteration and the best assignment found so far (A_{gb}) (also called global-best) is used to update A_{gb} which is used to perform the global pheromone update. When the termination criterion is met, the feasible timetable constructed is published to an excel spread sheet.

During pre-processing, (1) the number of students offering both an event **e** and another **e′** given as c(e, e′) and (2) also the number of events conflicting with an event **e** given as d(e) are calculated.

Formally,

$$d(e) = |e' \in E \setminus \{e\} \mid c(e, e') \neq 0|$$

A total order $\prec$ of events is defined by the following ordering rules:

$$\begin{aligned} e \prec e' :\Leftrightarrow\ & d(e) > d(e') \vee \\ & d(e) = d(e') \wedge N_p(e) > N_p(e') \vee \\ & d(e) = d(e') \wedge N_p(e) = N_p(e') \wedge dur(e) > dur(e') \vee \\ & d(e) = d(e') \wedge N_p(e) = N_p(e') \wedge dur(e) = dur(e') \wedge alphaOrder(e, e') \end{aligned}$$

The definition above places an event **e** over another event **e′** if it has more conflicting events than the other event. If d(e) = d(e′); it uses the number of programmes offering the events, N_p(e) and N_p(e′) to order them. If both d(.) and N_p(.) cannot separate the two events, then their durations are used (an event requiring four hours per week will be more difficult to place than an event requiring only one hour in a week). The tie-

breaking rule alphaOrder(e, e′) which determines the event to come first in alphabetical order is applied if they are still tied after applying the first three ordering rules.

The following are the differences between the MMAS algorithm of Socha *et al.,* (2002) (referred to Socha in the comparison) and the one of this study:

1. Only the variable d(e) was used to sort events in Socha while $N_p(e)$ and dur(e) are the two new variables introduced in this study because only the former is not sufficient to properly sort the events that constitute the problem domain tackled.
2. A more robust approach was used in selecting timeslots for events; to cater for events sectioning and also events that need more than one timeslot (these two situations were absent in Socha).
3. The local search procedure in this study was different from that of Socha in terms of the neighbourhood moves combination; comparison between them is done in Chapter Four.
4. The room assignment procedures in the two implementations are different.
5. In Socha, only the best-ant's initial solution is improved by the local search procedure. In this study, the MMAS implementation where all ants' solutions are improved by the local search procedure is experimented with.

3.6.2.1 Actual Coding of events ordering

To make a complete ordering of events in E, a vector named sup_value of length |E| is used to store the 'superiority value' given to each event **e** in E. Therefore we have the set SUP = {sup_value_1, $supvalue_2$, …, $sup\_value_{|E|}$} where sup_value_i is the superiority value for an event e_i. The partial event ordering algorithm is presented in ALGORITHM 3.2.

ALGORITHM 3.2: PARTIAL EVENT ORDERING

for i = 1 to |E| - 1 **do**
  for j = 1 to |E| **do**
    % apply the first ordering rule
    if $d(e_i) > d(e_j)$ **then**
      $sup\_value_i \leftarrow sup\_value_i + 1$
      continue

```
        else if d(e_i) < d(e_j) then
            sup_value_j ← sup_value_j + 1
            continue
        end if
        % apply the second ordering rule
        if N_p(e_i) > N_p(e_j) then
            sup_value_i ← sup_value_i + 1
            continue
        else if N_p(e_i) < N_p(e_j) then
            sup_value_j ← sup_value_j + 1
            continue
        end if
        % apply the third ordering rule
        if dur(e_i) > dur(e_j) then
            sup_value_i ← sup_value_i + 1
            continue
        else if dur(e_i) < dur(e_j) then
            sup_value_j ← sup_value_j + 1
            continue
        end if
        % apply the tie-breaking rule
        if alphaOrder(e, e′) = 1 then % returns 1 if e is before e′ in alphabetical order
            sup_value_i ← sup_value_i + 1
            continue
        else
            sup_value_j ← sup_value_j + 1
            continue
        end if
    end for
end for
```

The events in E are now sorted based on the values we have in the set **SUP**. For example, if sup_value_i for an event e_i is N and N is the highest value in SUP then the event e_i becomes the first element in the sorted set of events since it will be the most

difficult event to be placed. Therefore, we will have a complete ordering of events: $e_1 \prec e_2 \prec \cdots \prec e_{|E|}$.

3.6.2.2 Choosing a timeslot for an event

This process depicted in ALGORITHM 3.3 is one of the most critical stages of solution construction; it has to be done right otherwise solutions of poor quality or even infeasible solutions will be constructed most of the time. No literature currently surveyed addressed the complexity involved in choosing a timeslot in a UCTP that comprises of events that require more than a timeslot in a week; this (situation) is addressed in this section of the study.

In typical Covenant University timetabling, a three unit course is sectioned into two; the first section is placed into two timeslots and the second placed into a timeslot of another day. Courses like **PHY129** and **CHM129** which both require 6 timeslots in a week (the highest), are also divided into two sections: The first and second both requiring three timeslots each which are going to be scheduled on different days. The highest number of placements (sections) required by any event is 2, therefore an event requiring two placements 'calls' the function chooseTimeslot() twice. The function chooseTimeslot() as described in Table 3.1 returns a feasible timeslot **t** where an event **e** can be placed in. The value **t** returned by the first call to the function is passed when making a second call to it to avoid the second section to be scheduled on the same day as the first. The number of placements **pl** needed for an event is computed via (3.21):

$$pl = \begin{cases} 1, & if\ dur(e) \leq 2 \\ 2, & if\ dur(e) > 2 \end{cases} \tag{3.21}$$

The timeslot selection process for an event with duration more than 2 is done via the following pseudocode which features only the key parameters passed into the function:

for i = 1 to pl

chooseTimeslot(e, dur_i, [t_Chosen])

end for

The parameter t_Chosen is an optional parameter which takes the default value -1 if the caller of the function does not pass it.

The function chooseTimeslot() takes the following parameters:

e := the event under consideration

index_of_ant := the index of the virtual ant currently building a solution

duration := the number of timeslots needed for **e** at the current time **tm** (it may be the total number of timeslots needed for **e** in the week if the number of placements for the event is 1).

t_Chosen := this is an optional parameter, a value of x means that a timeslot x had been chosen for that same event in time **tm** – 1 and a value of -1 means no timeslot have been chosen for the event before.

ALGORITHM 3.3: CHOOSING TIMESLOT

sum_desirability ← 0.00
cList ← Ø % candidate list, empty from the outset
cList ← cList ∪ T % fill candidate list with timeslots
cList ← cList \ F % remove all forbidden timeslots
if t_Chosen > -1 % if t_Chosen was passed by the caller
  cList ← cList \ B
end if
for i = 1 to |cList| **do**
  $\eta\_et(i) \leftarrow \frac{1.0}{1.0+V\_et(i)}$
end for
for i = 1 to |cList| **do**
  $desirability_i \leftarrow \tau(e, t_i)^{\alpha} * \eta\_et(i)^{\beta}$
  $sum\_desirability \leftarrow sum\_desirability + desirability_i$
end for
tslot ← RWS(cList)
while not isOk(tslot, duration)
  tslot ← RWS(cList)
end while
return tslot

The definitions of the variables in ALGORITHM 3.3:

T := all timeslots

cList := candidate list; all timeslots initially make entry

F := a list of 'forbidden' timeslots for event **e**; it is defined as:

$$F = \begin{cases} F_{jun}, & if\ the\ event\ concerns\ only\ students\ in\ P_{jun} \\ F_{sen1} \cup F_{sen2}, & if\ the\ event\ concerns\ only\ students\ in\ P_{sen} \\ F_{jun} \cup F_{sen1} \cup F_{sen2}, & clause \end{cases}$$

clause := "if the event concerns students both in sets P_{jun} and P_{sen}"

B := a list of timeslots that event **e** cannot be placed because it had already been placed in a timeslot which belongs to a day where B belongs.

$$LB = 10(d-1) + 1$$

$$UB = \begin{cases} LB + 6, & if\ d > 4 \\ LB + 9, & otherwise \end{cases}$$

$$B = \{LB, LB+1, LB+2, \dots, UB\}$$

Where d = {1, 2, 3, 4, 5} representing the days of the week {Monday, Tuesday, Wednesday, Thursday, Friday}.

η_et(i) := heuristic information between the event **e** under consideration and a timeslot $\mathbf{t_i}$

V_et(i) counts the number of additional violations that will be made if timeslot $\mathbf{t_i}$ is chosen for event **e**.

$desirability_i$:= a value representing how good a timeslot $\mathbf{t_i}$ is for the event **e** under consideration

$\tau(e, t_i)$:= the pheromone trail between event **e** and timeslot $\mathbf{t_i}$

sum_desirability := a parameter used in the roulette-wheel selection procedure

α := controls the influence of $\tau(e, t_i)$

β := controls the influence of η_et(i)

tslot := the eventual timeslot chosen via RWS(cList)

RWS(cList) := returns a timeslot via a roulette-wheel procedure (RWS)

isOk(tslot, duration) := returns a Boolean value indicating if tslot is feasible for the event

$$isOK(tslot, duration) = \begin{cases} false, & NotFeasible \\ true, & otherwise \end{cases}$$

NotFeasible := "if tslot will make the event **e** spill over to the next day or to a forbidden timeslot or if by virtue of the duration of the event **e**, tslot is infeasible for **e**."

The NotFeasible clause states that the timeslot chosen by the RWS is infeasible (1) if it will make the event **e** spill over to another day or to a forbidden timeslot (bad placement)[1] or (2) if the timeslot tslot′ that follows it in cList is distant from it by a factor of more than 1[2] by virtue of the duration of **e**.

[1]If for example tslot = 10 (the timeslot t_{10}) is returned as the chosen timeslot for an event with a duration of 2 (needing two timeslots), isOK() is evaluated to false because the event will be made to occupy timeslots t_{10} and t_{11} which is not practical since both timeslots are on different days. [An event **e** with duration of 2 cannot occupy the timeslots 5:00-6:00pm on Monday (t_{10}) and 8:00-9:00 am on Tuesday (t_{11})].

[2]Another example is when for example we have cList = {…, 10, 12, …} and tslot = 10 via the RWS procedure for an event with duration of 2. The value of tslot will be an infeasible value [event sections fill consecutive timeslots and not timeslots with gaps in-between].

3.6.2.3 Pheromone Update

The best solution since the beginning of the solution construction process A_{gb} is used to update the pheromone trails. The update rule is as follows:

$$\tau(e,t) = \begin{cases} (1-\rho) * \tau(e,t) + 1, & if\ A_{gb}\ placed\ event\ e\ in\ timeslot\ t \\ (1-\rho) * \tau(e,t) + 0, & otherwise \end{cases} \quad (3.22)$$

$$where\ \rho \in (0,1) is\ the\ evaporation\ rate$$

The pheromone trails update is completed via 3.23 to force them to be in the range $[\tau_{min}, \tau_{max}]$

$$\tau(e,t) = \begin{cases} \tau_{min}, & if\ \tau_{min} > \tau(e,t) \\ \tau_{max}, & if\ \tau_{max} < \tau(e,t) \\ \tau(e,t), & otherwise \end{cases} \quad (3.23)$$

3.6.3 Local Search Routines

Two local search routines were implemented in ALGORITHM 3.1, the first one tagged LS1() is the main local search routine; it was proposed by Rossi-Doria et al; (2002). The original implementation of LS1() comprises of three neighbourhood moves:

N1: move an event from a timeslot to another timeslot

N2: swap the timeslots of two events

N3: tries to perform a permutation of the timeslots of three events using N1 or N2 (the choice is made in a random manner)

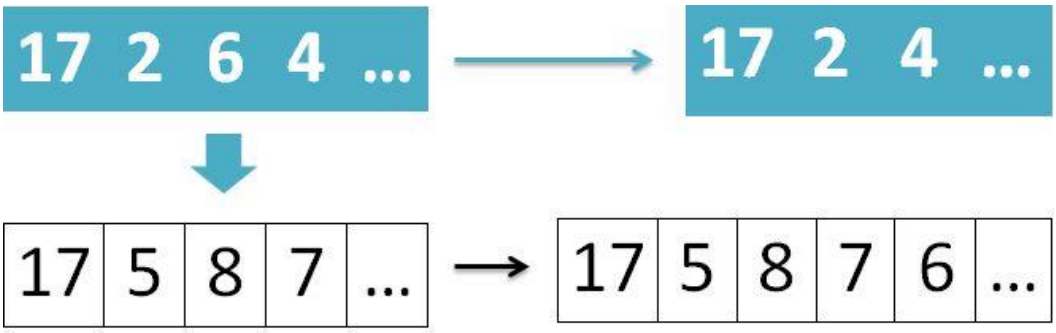

Figure 3.5: Depiction of N1: Event 6 moved from its timeslot to another

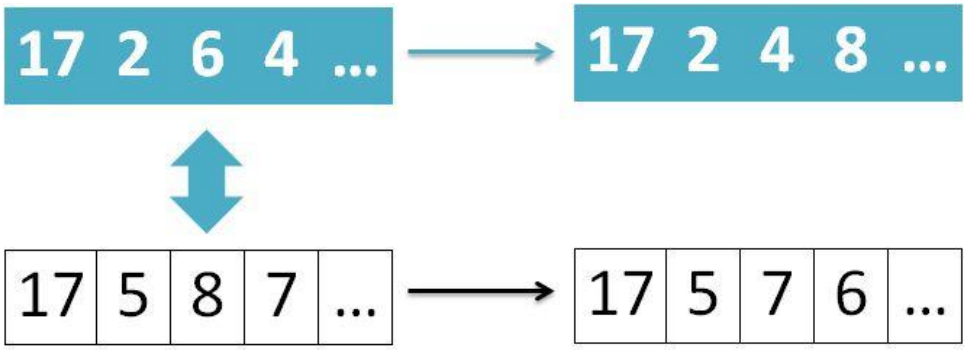

Figure 3.6: Depiction of N2: Events 6 and 8 have their timeslots swapped

The implementation of LS1() in the project work only comprises of the first two neighbourhood moves because of the longer time the local search procedure will take to completely run when the three moves are considered. N2 (see Figure 3.6) is only triggered whenever N1 (see Figure 3.5) fails to achieve feasibility after attempting to

move all events causing hard constraints violations from their respective timeslots (a major difference between past implementations which apply the two moves concurrently for each conflicting event and the one of this project).

If feasibility is achieved by the above local search procedure after considering all conflicting events in each timeslot, the local search starts again from the first timeslot again to ease the amount of soft constraints violations (#scv) without introducing any hard constraints violation (#hcv) using the same first two neighbourhood moves used initially, otherwise it is forced to terminate.

The second local search routine LS2(); which is derived from LS1() is applied only after a feasible solution has been found, in fact it is applied to the iteration-best solution A_{ib} to remove events from tightly packed timeslots and place them in 'idle' ones so as to make it easy to assign rooms to events in A_{ib}. If any event exists that cannot be assigned a room due to the fact that all suitable rooms of that event have been assigned to other events in the same timeslot then it is moved to another timeslot by LS2() and the room assignment is restarted. LS2() implements only the neighbourhood move N1 in LS1() and its failure to find a new feasible timeslot for an event which cannot be assigned a room in its current timeslot triggers the end of room assignment and the solution is considered infeasible.

3.6.4 Room Assignment

The procedure assignRooms() of the solution component handles the assignment of rooms to various events in each timeslot. The room assignment is done on a daily basis; firstly, all the timeslots for Monday are considered then the ones for Tuesday until assignments are made for all timeslots of each day. To avoid too much technicality, the steps involved in this process is briefly highlighted; the following are carried out for the assignment of rooms to events in timeslots of each day:

1. All rooms are declared idle by setting 'usage status' for each one to zero
2. For each timeslot on the day, do the following:
    a) Select the next event yet to be assigned a room based on two criteria
    b) (i) if suitable rooms (rooms with capacities that can hold the number of students to participate in the event) exist for the event, assign the event the room with the least capacity among the suitable rooms (ii) otherwise try to move the event to another timeslot and restart the process, if no

feasible timeslot can be found for the event then stop the search and declare the solution to be infeasible. (iii) Proceed to 2(c) if condition b(i) is true.

c) Update the usage status of the room assigned to the event in b(i) by the number of timeslots the event placed in it will occupy preventing it from being assigned to another event while being busy.

d) If events remain in the timeslot not yet assigned a room then go back to 2(a).

3. Move to another day

The "usage status" variable introduced is just an integer value representing the number of periods a room will be unavailable for another event while it is serving an event. Selection of events for room assignment is initially done based on the **first criterion**: "all events having a pre-determined room assigned to them are selected first for room assignment" before the consideration of events based on the **second criterion**: "selection of events by the highest number of participants".

CHAPTER FOUR

IMPLEMENTATION RESULTS AND DISCUSSION

4.1 PROBLEM INSTANCES

Before result presentation, a comparison is made between the problem tackled (Table 4.2) and another found in literature (Table 4.1).

Table 4.: Parameter values for generating UCTP instances in Socha *et al.*, (2002)

Parameter	Small	Medium	Large
Number of Events	100	400	400
Number of Rooms	5	10	10
Number of Students	80	200	400
Maximum Events Per Student	20	20	20
Maximum Students Per Event	20	50	100

Table 4.: Parameter values for the instances solved

Parameter	Instance1	Instance2
Number of Courses	327	229
Number of Events	719	505
Number of Rooms	23	23
Number of Programmes	57	45
Number of Students	2194	1771
Maximum Possible Rooms Per Event	12	12
Maximum Events Per Student	13	13
Maximum Students Per Event	642	642

Table 4.2 displays two classes of the problem solved, Instance1 is for the first semester and Instance2 is for the second semester of the College of Science and Technology (CST) programmes respectively. Instance1 is larger than Instance2 and therefore more difficult, this is due to the presence of more events for courses that undergo the SIWES programme in the second semester. The number of courses is actually the tally of events irrespective of the number of periods they need for a week while in the computation of the number of events, courses that need more than one period in a

week are counted based on the number of periods needed for them. For example, if **CSC321** is a 3-unit course, then it will be counted as a course and as three events rather than one because it will occupy three timeslots.

4.2 EVALUATION

In this section, an investigation is made into three local search designs[2] on their performance in optimising a timetable. The implementation of the local search procedure (Rossi-Doria et al., 2002) in Socha et al., (2002) applies all neighbourhood moves for a conflicting event before moving to the next one. For example, the local search procedure tries to move an event into another timeslot using N1 before considering neighbourhood move N2 if N1 fails to find a feasible timeslot for the event. The local search design adopted in this research does not consider N2 until all conflicting events have been attempted to be moved to other timeslots via N1. The first two designs discussed will be called the first design and second design respectively. Another local search design was considered to emphasise the effect of a local search design in optimising a timetable. This third design attempts to move an event to a prospective timeslot $\mathbf{t_p}$ via N1. If N1 fails, it immediately moves through the neighbourhood N2 to see if it can swap the timeslot of an event in $\mathbf{t_p}$ with that of the current event to be moved. The MMAS implementation of this research applies local search procedure on all the solutions generated by the colony, therefore we consider another case whereby only the best solution constructed by ants (with the least amount of #hcv) in the colony is improved by the local search procedure like we have in the MMAS implementation in Socha et al; (2002). There are six different variants of the MMAS implementation, they have the following details:

$\boldsymbol{N_{sep}}$: All ants' solutions are improved by the local search procedure that uses the second design.

$\boldsymbol{N_{sep}^{B}}$: Only the iteration-best solution is improved by the local search procedure that uses the second design.

N_{con}: All ants' solutions are improved by the local search procedure that uses the first design.

[2] a local search design is defined as the way the neighbourhood moves are combined

N_{con}^{B}: Only the iteration-best solution is improved by the local search procedure that uses the first design.

N_{imm}: All ants' solutions are improved by the local search procedure that uses the third design.

N_{imm}^{B}: Only the iteration-best solution is improved by the local search procedure that uses the third design.

The second local search design is proposed in this research.

The MMAS algorithm was implemented in VB.net on a machine with a 2.2 GHz clock speed and a 3 GB RAM. Ten independent runs were made for each of the six variants setting a time limit of 60 minutes for each run. The experimental results showing the amount of soft constraints violations (#scv) made by each variant are presented in Table 4.3 and Table 4.4.

Table 4.3: Performances of the six implementations on Instance1

S/N	N_{imm}^{B}	N_{imm}	N_{con}^{B}	N_{con}	$\boldsymbol{N_{sep}^{B}}$	$\boldsymbol{N_{sep}}$
1	158	206	119	113	84	123
2	176	166	140	123	108	124
3	184	153	111	126	66	103
4	188	167	142	152	102	136
5	166	165	123	129	134	89
6	194	182	126	118	117	120
7	216	148	139	125	89	69
8	206	130	98	140	90	131
9	160	155	109	131	78	123
10	146	170	144	143	87	91
Min.	146	130	98	113	66*	69
Max.	216	206	144	143	134*	136
Avg.	179.4	164.2	125.1	130	95.5*	110.9
Avg. Iter.**	709.5	133	643.7	95.6	519.2	63.2

*the best result among the six variants
**average number of iterations made by the algorithm over ten trials

Table 4.4: Performances of the six implementations on Instance2

S/N	N_{imm}^{B}	N_{imm}	N_{con}^{B}	N_{con}	$\boldsymbol{N_{sep}^{B}}$	$\boldsymbol{N_{sep}}$
1	119	95	72	41	60	49
2	118	89	79	70	49	58
3	111	102	65	56	61	57
4	97	96	78	46	38	52
5	98	85	63	43	76	49
6	82	116	52	30	47	63
7	117	102	66	69	48	40
8	114	101	65	67	45	48
9	122	114	75	67	54	48
10	124	95	59	54	41	42
Min.	82	85	52	30*	38	40
Max.	124	114	79	70	76	63*
Avg.	110.2	99.5	67.4	54.3	51.9	50.6*
Avg. Iter.**	1046.4	141.3	1044.7	186	920.4	93.1

*the best result among the six variants
**average number of iterations made by the algorithm over ten trials

From the results displayed in Table 4.3 and Table 4.4, it can be seen that using the second local search design proposed in this research gives better results than the other two considered; that can be confirmed from the results obtained by N_{sep} and N_{sep}^{B} on both instances. In terms of the number of iterations achieved by the algorithms, the variants that run the local search procedure on only the iteration-best result (N_{sep}^{B}, N_{con}^{B}, and N_{imm}^{B}) as expected run faster than their counterparts (N_{sep}, N_{con} and N_{imm}) and the impact on the algorithm performance is not conclusive since it only had positive effects on Instance1 (harder instance).

Direct comparisons are made between four of the implementation variants (N_{sep} vs. N_{con}) in Figure 4.1 to Figure 4.3 and (N_{sep}^{B} vs. N_{con}^{B}) in Figure 4.4 to Figure 4.6 using the convergence of global-best solutions for each one of them recording the total violations recorded per iteration using the same time limit. A value of 100 units was used to penalise each hard constraint violation. The best three results of each variant were selected from Table 4.3 and Table 4.4.

For Figure 4.1 to Figure 4.3: A = N_{sep} on Instance1, B = N_{sep} on Instance2, C = N_{con} on Instance1 and D = N_{con} on Instance2.

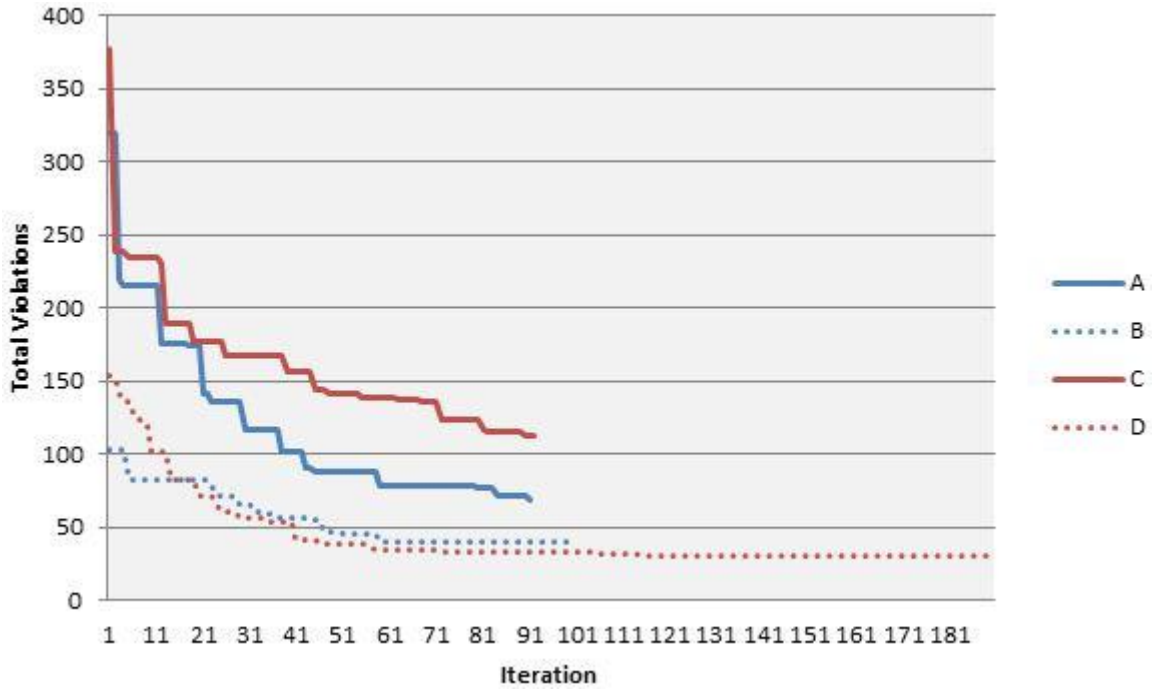

Figure 4.: Convergence of the best global-best solutions of N_{sep} and N_{con}

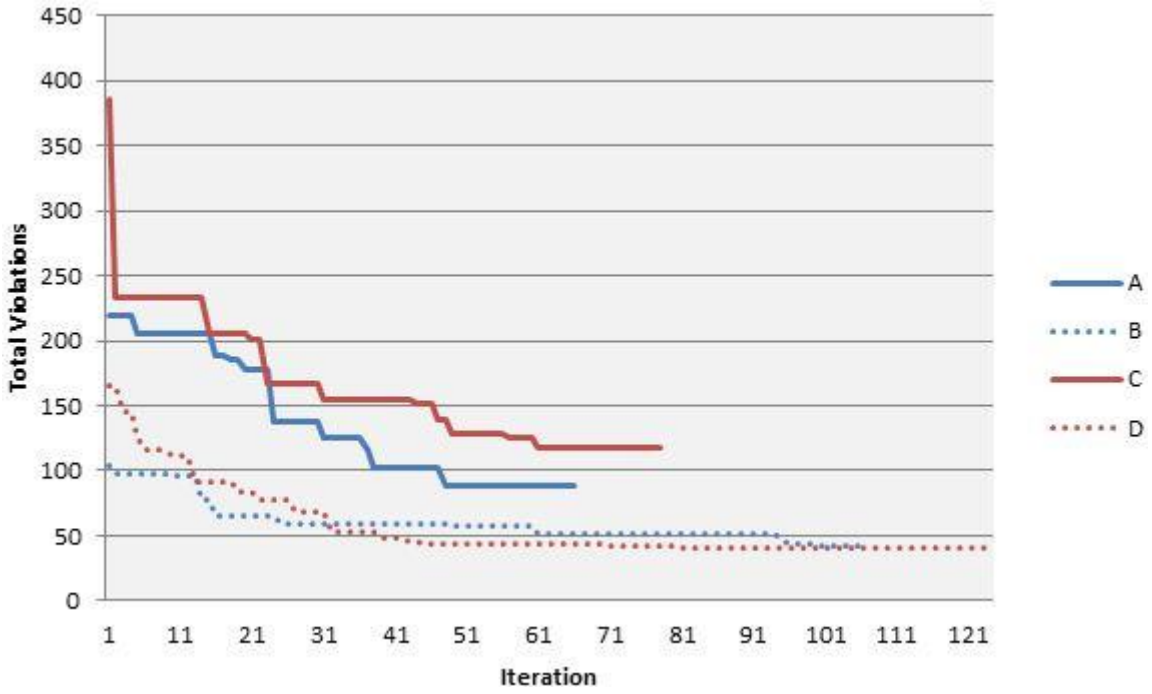

Figure 4.: Convergence of the second-best global-best solutions of N_{sep} and N_{con}

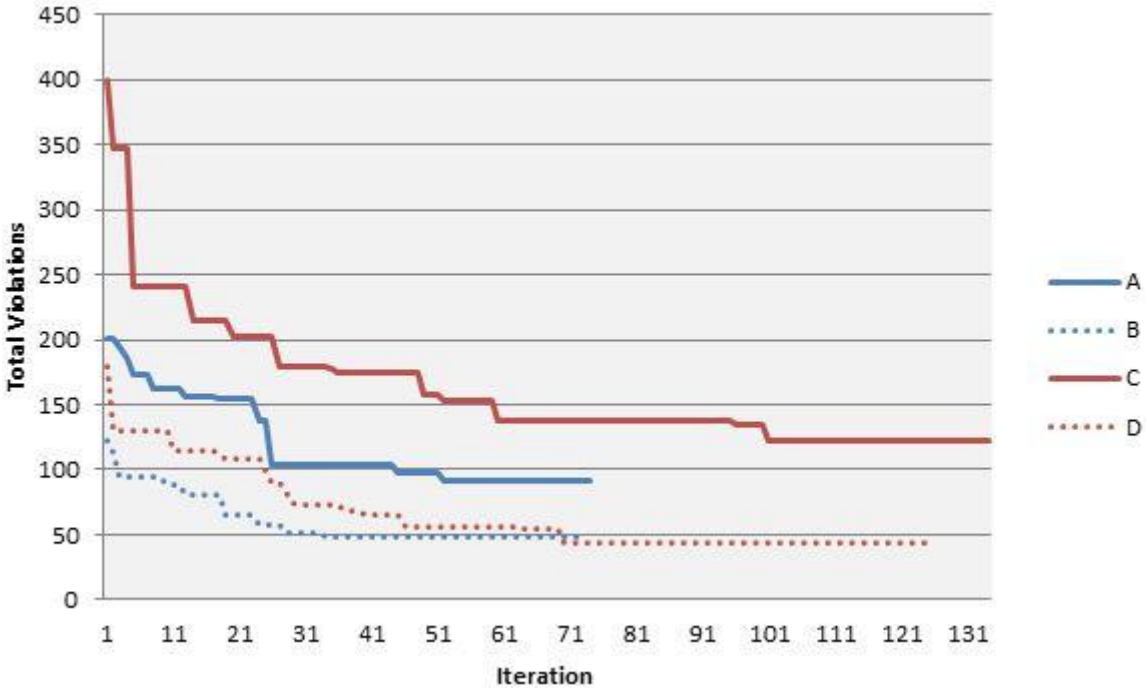

Figure 4.: Convergence of the third-best global-best solutions of N_{sep} and N_{con}

For Figure 4.4 to Figure 4.6: A = N_{sep}^{B} on Instance1, B = N_{sep}^{B} on Instance2, C = N_{con}^{B} on Instance1 and D = N_{con}^{B} on Instance2.

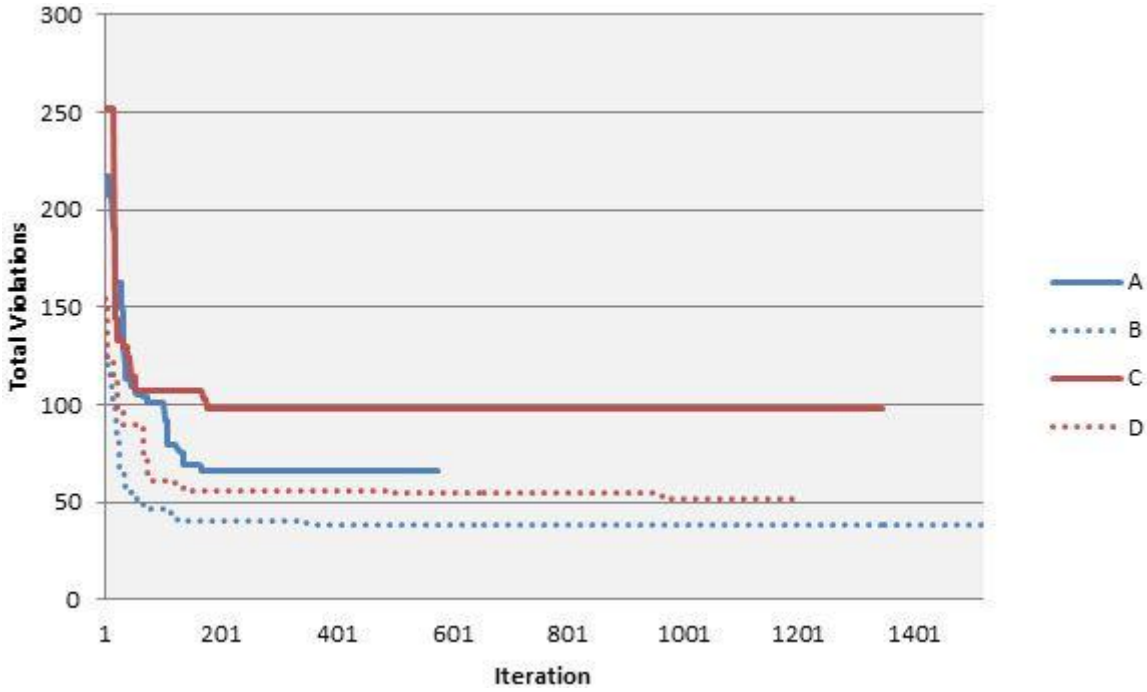

Figure 4.: Convergence of the best global-best solutions of N_{sep}^{B} and N_{con}^{B}

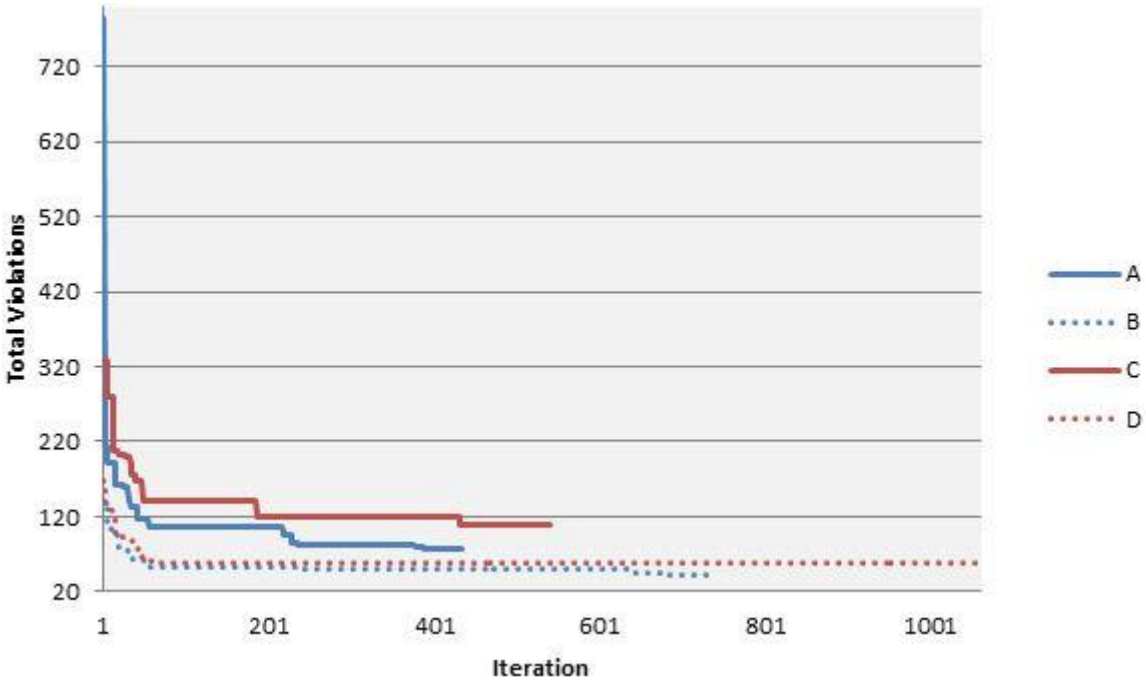

Figure 4.: Convergence of the second-best global-best solutions of N_{sep}^{B} and N_{con}^{B}

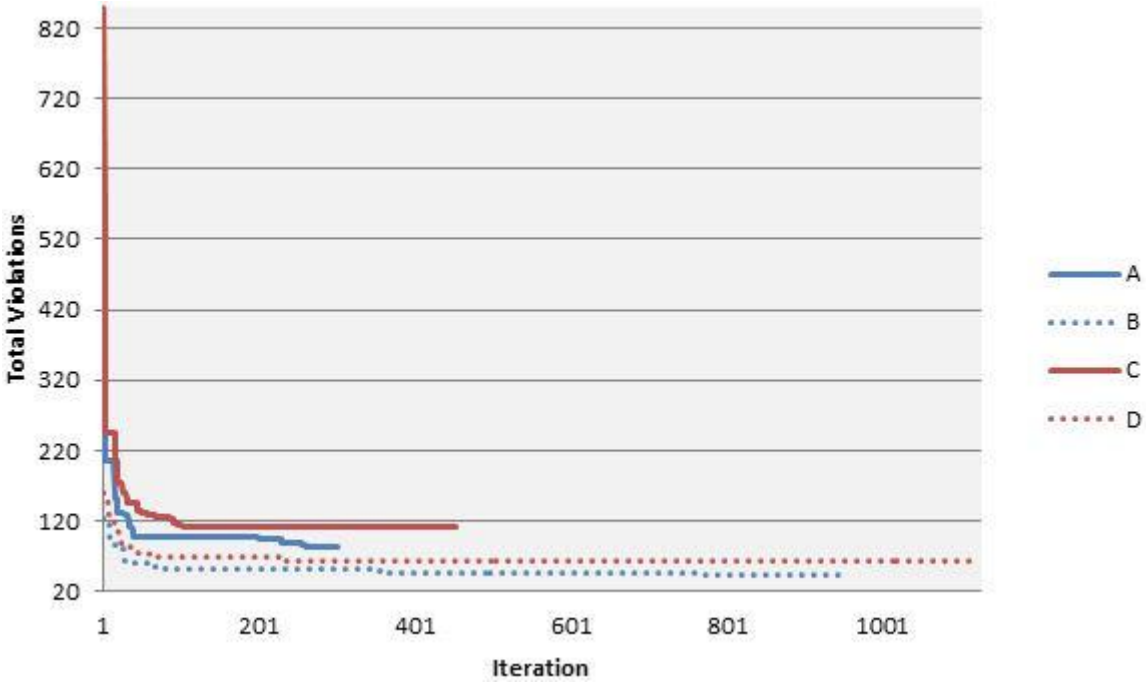

Figure 4.: Convergence of the third-best global-best solutions of N_{sep}^{B} and N_{con}^{B}

From Figure 4.1 to Figure 4.6, it can be concluded that the algorithms generally run faster on the smaller instance (Instance2) but for few exceptions. On Instance1, $\boldsymbol{N_{sep}}$ clearly outperformed $\boldsymbol{N_{con}}$ and $\boldsymbol{N_{sep}^{B}}$ also clearly outperformed $\boldsymbol{N_{con}^{B}}$. There is a close competition between $\boldsymbol{N_{sep}}$ and $\boldsymbol{N_{con}}$ on Instance2 (can be confirmed from the average #scv in Table 4.4) due to more number of solutions optimised by the former

(optimising a feasible timetable takes a long time). $\boldsymbol{N_{sep}^{B}}$ outperformed $\boldsymbol{N_{con}^{B}}$ on Instance2 defying the status quo between $\boldsymbol{N_{sep}}$ and $\boldsymbol{N_{con}}$ since the local search procedure is run only the best-ant's solution. In conclusion, though $\boldsymbol{N_{sep}}$ and $\boldsymbol{N_{sep}^{B}}$ generally complete less iterations than their counterparts, they do optimise timetables better.

The same parameter configurations were used for the experimental procedure in this section to avoid bias; they are presented in Table 4.5.

Table 4.5: Parameter Configurations used in the experiment

Parameter	Value
Number of ants (m)	6
Evaporation rate (ρ)	0.2
Maximum Pheromone ($\tau_{max} = \frac{1}{\rho}$)	5.0
Minimum Pheromone (τ_{min})	0.0019
Influence of Pheromone (α)	1.0
Influence of heuristic information (β)	2.0

CHAPTER FIVE

SUMMARY, FUTURE STUDIES AND CONCLUSION

5.1 SUMMARY

The UCTP is a hard combinatorial optimisation problem which takes different form from one institution to another. Although the problem of one institution peculiar to itself may not be present in other institutions, these problems are unified by containing conditions that must be fulfilled when planning/constructing/drafting a timetable (hard constraints) and the conditions that might not be fulfilled (soft constraints).

This study had addressed two instances of Covenant University's case which are centred on College of Science and Technology (CST). The underlying variables that make up the problem; soft and hard constraints were extensively dealt with in the course of the research work. A decentralised system can be established since events that are peculiar to each college need rooms different from the ones needed by another and therefore there will be a very little concern about the room-clash constraint. Courses that are taken by students between colleges; for example BFN311 (a course peculiar to students in College of Development Studies) taken by 300 level MIS students will be scheduled first alongside University Wide Courses and NUC courses like the TMCs, EDSs and CSTs if a combined schedule is to be made.

The developed system though being run via a command line interface (CLI) has been painstakingly built in order to make it effective in solving course timetabling problems. To subsequent research students, this work will be useful in understanding some underlying principles concerning UCTP in relation to general instances and that of the university. With some modifications this system can be used as the official timetable solver or as a decision-making tool in timetable construction since real data were used (http://eprints.covenantuniversity.edu.ng/id/eprint/3269).

5.2 FUTURE STUDIES

Since only instances from CST were tackled in this research work, there is still room for further work in the area of extending the work to other colleges and making adequate integration to build a complete system.

It was reported in an earlier application of ACO (Socha *et al.*, 2003), that the influence of heuristic information does not improve solution quality when local search is implemented to improve ants' solutions. The influence of heuristic information was used in the solution construction of this work; therefore further work may experiment without the use of heuristic information to confirm if the claim is valid in this particular class of problem solved.

In the course of the work, the parameter setting used was: $\alpha = 1$ and $\beta = 2$; which are the influences of pheromone and heuristic information respectively. Further work may involve setting different combinations of α and β to investigate the results that will be achieved by these combinations. Ultimately, another method entirely can be applied to these problem instances to compare its performance against that of MMAS applied in this study.

Kempe Chain Neighbourhood Move: The kempe chain neighbourhood move is a flexible neighbourhood move which is not limited unlike the two neighbourhood moves used in the implementation but more complex; as many as five or more events can have their timeslots swapped at a go in a kempe chain depending on how long the chain is.

The idea of kempe chain neighbourhood move can also be applied in a future research work since it is a very effective way of satisfying soft constraints and its effectiveness can also be compared against the two neighbourhood moves (N1 and N2) used in the project. Finally, the kempe chain neighbourhood can even be combined with N1 and N2 to make further improvements on a solution's quality.

5.3 CONCLUSION

University Course Timetabling Problem (UCTP) which is a combinatorial optimisation problem has been tackled using a population-based metaheuristics in the mode of Ant Colony Optimisation (ACO) algorithm alongside two local search procedures. Two instances of the CU problem class were considered with good results achieved by the algorithm on both of them. An investigation was also carried out on how a local search set-up affect the performance of the algorithm and from the results obtained the local search design proposed in this research outperformed its competitors. Possibilities for future work have been highlighted; a platform for better results on the UCTP instance

addressed. Finally, this study has proven that metaheuristics if properly tuned to a particular problem domain can achieve high-quality results on real-world instances of the UCTP.

REFERENCES

Aarts, E., Korst, J., & Michiels, W. (2014). Simulated Annealing. In E. K. Burke, & G. Kendall (Eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (2nd ed., pp. 265-285). New York, US: Springer.

Abdullah, S., Shaker, K., McCollum, B., & McMullan, P. (2009). Construction of course timetables based on great deluge and tabu search. *Proceedings of MIC 2009: VIII Metaheuristic International Conference*, (pp. 13-16).

Adubi, S. A., & Misra, S. (2014). A comparative study on the ant colony optimization algorithms. *11th International Conference on Electronics, Computer and Computation (ICECCO), 2014* (pp. 1-4). Abuja: IEEE.

Aladag, C. H., Hocaoglu, G., & Basaran, M. A. (2009). The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem. *Expert Systems with Applications, 36*(10), 12349–12356.

Al-Betar, M. A., Khader, A. T., & Gani, T. A. (2012). A harmony search algorithm for university course timetabling. *Annals of Operations Research, 194*(1), 3-31.

Basir, N., Ismail, W., & Norwawi, N. M. (2013). A Simulated Annealing for Tahmidi Course Timetabling. *The 4th International Conference on Electrical Engineering and Informatics (ICEEI 2013). 11*, pp. 437–445. Procedia Technology.

Blum, C. (2005). Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research, 32*(6), 1565-1591.

Blum, C., & Manfrin, M. (2015). *Metaheuristics Network*. Retrieved May 2, 2015, from Metaheuristics Network: http://www.metaheuristics.net/

Boland, N., Hughes, B. D., Merlot, L. T., & Stuckey, P. J. (2008). New integer linear programming approaches for course timetabling. *Computers & Operations Research, 35*(7), 2209-233.

Bozejko, W., Pempera, J., & Smutnicki, C. (2013). Parallel tabu search algorithm for the hybrid flow shop problem. *Computers & Industrial Engineering, 65*(3), 466-474.

Bui, T. N., Nguyen, T. H., Patel, C. M., & Phan, K.-A. T. (2008). An ant-based algorithm for coloring graphs. *Discrete Applied Mathematics, 156,* 190-200.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1997). A New Rank Based Version of the Ant System - A Computational Study.

Burke, E. K., McCarthy, B. L., Petrovic, S., & Qu, R. (2006). Multiple-Retrieval Case-Based Reasoning for Course Timetabling Problems. *The Journal of the Operational Research Society, 57*(2), 148-162.

Burke, E. K., McCarthy, B., Petrovic, S., & Qu, R. (2001). Case-based Reasoning in Course Timetabling: An Attribute Graph Approach. *Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR-2001)* (pp. 90-104). Vancouver: Springer Berlin Heidelberg.

Burke, E. K., Petrovic, S., & Qu, R. (2006). Case-based heuristic selection for timetabling problems. *Journal of Scheduling, 9*(2), 115-132.

Cambazard, H., Hebrard, E., O'Sullivan, B., & Papadopoulos, A. (2012). Local Search and Constraint Programming for the Post Enrolment-based Course Timetabling Problem. *Annals of Operations Research, 194*(1), 111-135.

Caramia, M., & Paolo, D. (2008). Coloring graphs by iterated local search traversing feasible and infeasible solutions. *Discrete Applied Mathematics, 156*(2), 201-217.

Carter, M. W. (2001). A comprehensive course timetabling and student scheduling system at the University of Waterloo. *PATAT III* (pp. 64-82). Springer Berlin Heidelberg.

Chaudhuri, A., & De, K. (2010). Fuzzy Genetic Heuristic for University Course Timetable Problem. *International Journal of Advanced Soft Computing Application, 2*(1).

Cheng, C.-B., & Mao, C.-P. (2007). A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling, 46*, 1225–1235.

Chiarandini, M., Birattari, M., Socha, K., & Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. *Journal of Scheduling, 9*(5), 403-432.

Corne, D., Ross, P., & Fang, H. (1995). Evolving Timetables. In L. D. Chambers (Ed.), *The practical handbook of genetic algorithms* (Vol. 1, pp. 219-276). CRC Press.

Cuervo, D. P., Goos, P., Sorensen, K., & Arraiz, E. (2014). An iterated local search algorithm for the vehicle routing problem with backhauls. *European Journal of Operational Research, 237*(2), 454-464.

Dong, X., Huang, H., & Chen, P. (2009). An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers & Operations Research, 36*(5), 1664-1669.

Dorigo, M., & Gambardella, L. M. (1997, April). Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, 1*(1), 53-66.

Dorigo, M., & Stuetzle, T. (2004). *Ant Colony Optimization.* London, England: The MIT Press.

Dorigo, M., Birattari, M., & Stuetzle, T. (2006). Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique. *IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE,* 28-39.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on System, Man and Cybernetics, 26*(1), 29-41.

Dowsland, K. A., & Thompson, J. M. (2008). An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics, 156*(3), 313-324.

Dueck, G. (1993). New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics, 104*(1), 86-92.

Duong, T.-A., & Lam, K.-H. (2004). Combining Constraint Programming and Simulated Annealing on University Exam Timetabling. *RIVF*, (pp. 205-210). Hanoi.

Gajpal, Y., & Abad, P. L. (2009). Multi-ant colony system (MACS) for a vehicle routing problem with backhauls. *European Journal of Operation Research, 196*(1), 102-117.

Gendreau, M. (2003). An Introduction to Tabu Search. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 37-54). New York, US: Kluwer Academic Publishers.

Gendreau, M., & Potvin, J.-Y. (2010). Tabu Search. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (2nd ed., Vol. 146, pp. 41-59). New York, US: Springer.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research, 13*(5), 533-549.

Glover, F. (1990). Tabu search: A tutorial. *Interfaces, 20*(4), 74-94.

Glover, F., & Laguna, M. (1997). *Tabu Search.* New York: Springer.

Goldberg, D. E., & Lingle, R. (1985). Alleles, Loci, and the travelling salesman problem. *Proceedings of the first international conference on genetic algorithms and their applications* (pp. 154-159). Lawrence Erlbaum Associates.

Goltz, H.-J., & Matzke, D. (1999). Combined interactive and automatic timetabling. *PACLP99.*

Hong, T.-P., Chen, C.-H., & Lin, F.-S. (2015). Using Group Genetic Algorithm to Improve Performance of Attribute Clustering. *Applied Soft Computing, 29,* 371-378.

Hussin, M. S., & Stuetzle, T. (2014). Tabu search vs. simulated annealing as a function of the size of quadratic assignment problem instances. *Computers & Operations Research, 43,* 286-291.

Jat, S. N., & Yang, S. (2011). A Hybrid Genetic Algorithm and Tabu Search Approach for Post Enrolment Course Timetabling. *Journal of Scheduling, 14*(6), 617-637.

Jia, H., Li, Y., Dong, B., & Ya, H. (2013). An Improved Tabu Search Approach to Vehicle Routing Problem. *Procedia - Social and Behavioral Sciences, 96,* 1208-1217.

Jun-Man, K., & Yi, Z. (2012). Application of an Improved Ant Colony Optimization on Generalized Traveling Salesman Problem. *Energy Procedia, 17*, 319-325.

Kolodner, J. L. (1992). An Introduction to Case-Based Reasoning*. *Artificial Intelligence Review, 6*(1), 3-34.

Kong, M., Tian, P., & Kao, Y. (2008). A new ant colony optimization algorithm for the multidimensional Knapsack problem. *Computers & Operations Research, 35*(8), 2672-2683.

Lai, X., & Lu, Z. (2013). Multistart iterated tabu search for bandwidth coloring problem. *Computers & Operations Research, 40*(5), 1401-1409.

Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. *OR spectrum, 30*(1), 167-190.

Lewis, R., & Paechter, B. (2005). Application of the grouping genetic algorithm to university course timetabling. In G. R. Raidl, & J. Gottlieb (Ed.), *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)* (pp. 144-153). Lausanne: Springer-Verlag.

Lin, S., & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Travelling Salesman Problem. *Operations Research, 21*(2), 498-516.

Lourenco, H. R., Martin, O. C., & Stuetzle, T. (2003). Iterated Local Search. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (1st ed., pp. 321-353). New York: Kluwer Academic Publishers.

Lourenco, H. R., Martin, O. C., & Stuetzle, T. (2010). Iterated Local Search: Framework and Applications. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (Vol. 146, pp. 363-397). New York, US: Springer.

Lu, Z., & Hao, J.-K. (2010). Adaptive Tabu Search for Course Timetabling. *European Journal of Operational Research, 200*(1), 235-244.

Matijas, V. D., Molnar, G., Cupic, M., Jakobovic, D., & Basic, B. D. (2010). University Course Timetabling Using ACO: A Case Study on Laboratory Exercises. *Knowledge-Based and Intelligent Information and Engineering Systems* (pp. 100-110). Springer Berlin Heidelberg.

Mitchell, M. (1999). *An Introduction to Genetic Algorithms.* Cambridge: The MIT Press.

Montemanni, R., Gambardella, L. M., Rizzoli, A. E., & Donati, A. V. (2005). Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization, 10*(4), 327-343.

Morais, V. W., Mateus, G. R., & Noronha, T. F. (2014). Iterated local search heuristics for the Vehicle Routing Problem with Cross-Docking. *Expert Systems with Applications, 41*(16), 7495-7506.

Moscato, P., & Cotta, C. (2003). A Gentle Introduction to Memetic Algorithms. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 105-144). New York: Kluwer Academic Publishers.

Mueller, T., Murray, K., & Rudova, H. (2010). System Demonstration of Interactive Course Timetabling. *PATAT 2010*, (pp. 573-577).

Murray, K., & Mueller, T. (2006). Automated System for University Timetabling. *PATAT VI*, *6*, pp. 536-541.

Mushi, A. R. (2006). TABU SEARCH HEURISTIC FOR UNIVERSITY COURSE TIMETABLING PROBLEM. *African Journal of Science and Technology (AJST), 7*(1), 34-40.

Nikolaev, A. G., & Jacobson, S. H. (2010). Simulated Annealing. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (2nd ed., Vol. 146, pp. 1-39). New York, US: Springer.

Oprea, M. (2007). A multi-agent system for university course timetable scheduling. *International Journal of Computers, Communications & Control, 2*(1), 94-102.

Otero, F. E., Freitas, A. A., & Johnson, C. G. (2012). Inducing decision trees with an ant colony optimization algorithm. *Applied Soft Computing, 12*(11), 3615-3626.

Peng, B., Lu, Z., & Cheng, T. (2015). A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, *53*, 154-164.

Perzina, R. (2007). Solving the University Timetabling Problem with Optimized Enrollment of Students by a Self-adaptive Genetic Algorithm. *Practice and Theory of Automated Timetabling VI* (pp. 248-263). Brno: Springer Berlin Heidelberg.

Petrovic, S., Yang, Y., & Dror, M. (2007). Case-based selection of initialisation heuristics for metaheuristic examination timetabling. *Expert Systems with Applications, 33*(3), 772-785.

Quarooni, D., & Akbarzadeh-T, M.-R. (2013). Course timetabling using evolutionary operators. *Applied Soft Computing, 13*(5), 2504-2514.

Reeves, C. (2003). Genetic Algorithms. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (1st ed., pp. 55-82). New York: Kluwer Academic Publishers.

Reeves, C. (2010). Genetic Algorithms. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (Vol. 146, pp. 109-139). New York, US: Springer.

Rodriguez-Tello, E., Romero-Monsivais, H., & Ramirez-Torres, G. (2015). Tabu search for the cyclic bandwidth problem. *Computers & Operations Research, 57*, 17-32.

Rossi-Doria, O., & Paechter, B. (2004). A memetic algorithm for University Course Timetabling. *Combinatorial optimisation, 56*.

Rossi-Doria, O., Blum, C., Knowles, J., Sampels, M., Socha, K., & Paechter, B. (2002). A local search for the timetabling problem. *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*, (pp. 124-127).

Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L. M., et al. (2003). A comparison of the performance of different metaheuristics on the timetabling problem. *Practice and Theory of Automated Timetabling IV* (pp. 329-351). Springer Berlin Heidelberg.

Rudova, H., & Murray, K. (2003). University course timetabling with soft constraints. *PATAT IV* (pp. 310-328). Springer Berlin Heidelberg.

Savitch, W. J., & Mock, K. (2009). *Problem Solving with C++* (7th ed.). Boston, US: Pearson Addison Wesley.

Schaerf, A. (1999). A Survey of Automated Timetabling. *Artificial Intelligence Review, 13*(2), 87-127.

Shmygelska, A., & Hoos, H. H. (2005). An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. *BMC Bioinformatics, 6*(30), 1-22.

Sigl, B., Golub, M., & Mornar, V. (2003). Solving Timetable Scheduling Problem by Using Genetic Algorithms. *Proc. of the 25th int. conf. on information technology interfaces*, (pp. 519-524).

Silva, M. M., Subramanian, A., & Ochi, L. S. (2015). An iterated local search heuristic for the split delivery vehicle routing problem. *Computers & Operations Research, 53*, 234-249.

Socha, K. (2003). *Metaheuristics for the Timetabling Problem.* Technical Report No.: TR/IRIDIA/2003-29, Universite Libre de Bruxelles, IRIDIA, Brussels.

Socha, K., Knowles, J., & Sampels, M. (2002). A MAX-MIN Ant System for the University Course Timetabling Problem. In M. Dorigo, G. Di Caro, & M. Sampels (Eds.), *Ant Algorithms* (pp. 1-13). Springer Berlin Heidelberg.

Socha, K., Sampels, M., & Manfrin, M. (2003). Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. *Applications of evolutionary computing* (pp. 334-345). Springer Berlin Heidelberg.

Sparx Systems Pty Ltd. (n.d.). *Enterprise Architect - Logical Model*. Retrieved May 11, 2015, from Sparx Systems: http://www.sparxsystems.com/resources/tutorial/logical_model.html

Stuetzle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research, 174*(3), 1519-1539.

Stuetzle, T. G. (1998). *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements and New Applications.* PhD Thesis, Technische Universitaet Darmstadt.

Stuetzle, T., & Hoos, H. H. (2000). MAX-MIN Ant System. *Future Generation Computer Systems, 16*(8), 889-914.

Thepphakorn, T., Pongcharoen, P., & Hicks, C. (2014). An ant colony based timetabling tool. *International Journal of Production Economics, 149*, 131-144.

Tsutsui, S., & Fujimoto, N. (2009). Solving Quadratic Assignment Problems by Genetic Algorithms with GPU Computation: A Case Study. *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers* (pp. 2523-2530). ACM.

Tuga, M., Berretta, R., & Mendes, A. (2007). A hybrid simulated annealing with kempe chain neighborhood for the university timetabling problem. *6th International Conference on IEEE/ACIS* (pp. 400-405). IEEE.

Wang, C., Mu, D., Zhao, F., & Sutherland, J. W. (2015). A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows. *Computers & Industrial Engineering, 83*, 111-122.

Wijaya, T., & Manurung, R. (2009). Solving University Timetabling As a Constraint Satisfaction Problem with Genetic Algorithm. *Proceedings of the international conference on advanced computer science and information systems.* Depok.

Yang, J., Wu, C., Lee, H. P., & Liang, Y. (2008). Solving traveling salesman problems using generalized chromosome genetic algorithm. *Progress in Natural Science, 18*(7), 887-892.

Yang, S., & Jat, S. N. (2011). Genetic Algorithms With Guided and Local Search Strategies for University Course Timetabling. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS, 41*(1), 93-106.

Yassin, R. M., Nazri, M. Z., & Abdullah, S. (2013). Hybrid Approach: Tabu-Based Non-Linear Great Deluge for the Course Timetabling Problem. *Research Journal of Applied Sciences, 8*(2), 131-138.

Zecchin, A. C., Maier, H. R., Simpson, A. R., Leonard, M., & Nixon, J. B. (2007). Ant Colony Optimisation Applied to Water Distribution System Design: A Comparative Study of Five Algorithms. *Journal of Water Resources Planning and Management, 133*(1), 87-92.

Zecchin, A. C., Simpson, A. R., Maier, H. R., Leonard, M., Roberts, A. J., & Berrisford, M. J. (2006). Application of two ant colony optimisation algorithms to water distribution system optimisation. *Mathematical and Computer Modelling, 44*(5), 451–468.

APPENDIX

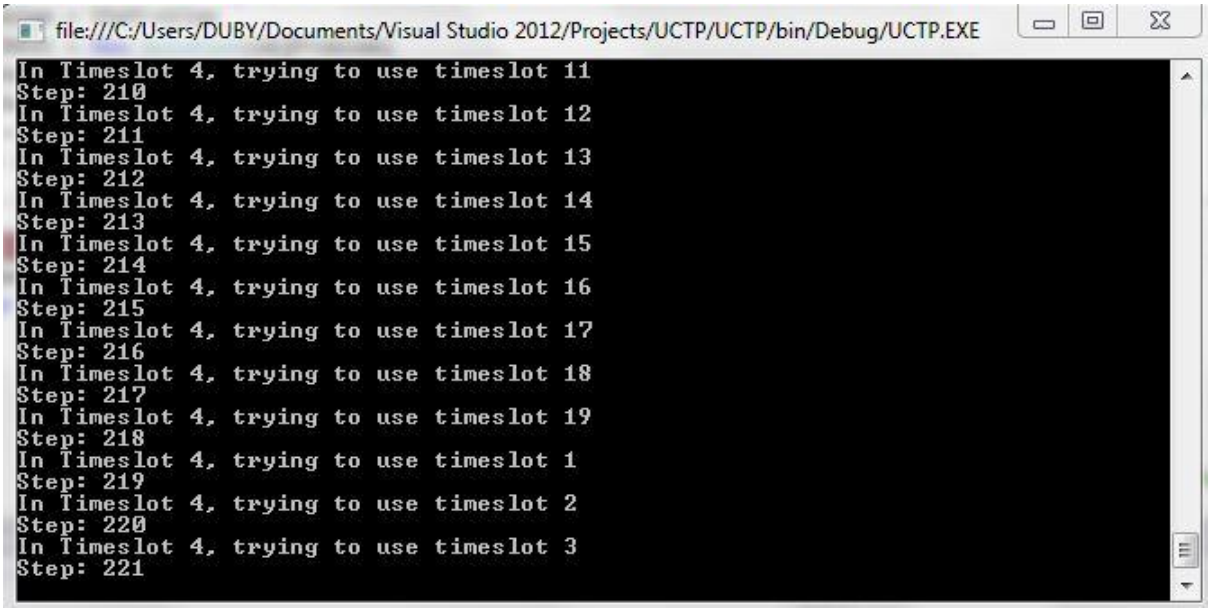

Figure I: Snapshot of the main local search run

	8am-9am	9am-10am	10am-11am	11am-12noon	12noon-1pm	1pm-2pm	2pm-3pm	3pm-4pm	4pm-5pm	5pm-6pm
STUDIO100	ARC122	ARC122	ARC121							
STUDIO200		ARC225	ARC225	ARC229	ARC229		ARC223	ARC223		
STUDIO300		ARC327	ARC327				ARC324	ARC324	ARC322	ARC322
STUDIO400	ARC423	ARC423	ARC422	ARC422	ARC427	ARC427				
BCH LAB	BCH423	BCH423	BCH223	BCH223	BCH441	BCH441	BCH421	BCH421	BCH224	BCH224
HALL107	CHM462	CHM462	BLD325	BLD325	MIS221	MIS221	BLD527	BLD527	CSC423	
HALL108	MAT425	MAT425	BLY225	BLY225	MAT123	MAT123	BLY222	BLY222	BLY122	BLY122
HALL201	CHM426	CHM426	ESM227	ESM227		MAT225	MAT225	ESM522	ESM522	
HALL202	BLD523	BLD523	BLD228	BLD228	ESM528	ESM528	ESM125	ESM125	SES221	SES221
HALL203	ESM524	ESM524	MAT423	MAT423	CSC445	CSC445	ESM328	ESM328	MIS425	MIS425
HALL204	ESM123	ESM123	MAT221	MAT221	PSY326	PSY326	MIS422	MIS422	ESM322	ESM322
HALL306	CSC241	CSC443	CSC443	PHY222	PHY222	MAT421	MAT421	MAT426	MAT426	
HALL307	MCB121	MCB121	BLD526	BLD526	CHM222	CHM222	CSC447	CSC447	BLD324	BLD324
HALL308	BLY226	BLY226	ESM327	ESM327	BCH121	BCH121	BLD326	BLD326	BLD524	BLD524
HALL313	BLD224	BLD224	ESM122	ESM122			ESM221	ESM221	MAT222	
MCB LAB	MCB421	MCB422			MCB122				MCB423	MCB424
BLY LAB	BLY426	BLY426	BLY425	BLY425			BLY422	BLY422	BLY428	BLY428
CHM RES LAB	CHM420		CHM421	CHM484	CHM484		CHM446	CHM446	CHM460	CHM460
CHM LAB	CHM224	CHM224	CHM480	CHM480	CHM448	CHM448	CHM229	CHM225	CHM225	
COMPUTER LAB	CIS227	CIS227	CSC225	CSC225		CIS225		CSC221		
HALL213	PHY426	PHY426	PHY448	PHY448	PHY424	PHY424	PHY428	PHY442	PHY442	
HALL302	PHY228	PHY224	PHY226	PHY226				PHY227	PHY227	PHY221
LT1	PHY122	PHY122	CHM122	CHM122			MAT122	MAT122	MAT121	MAT121

Figure II: Snapshot of Monday Schedule

	8am-9am	9am-10am	10am-11am	11am-12noon	12noon-1pm	1pm-2pm	2pm-3pm	3pm-4pm	4pm-5pm	5pm-6pm
STUDIO100		ARC124	ARC124							
STUDIO200	ARC226	ARC226	ARC224	ARC224						
STUDIO300			ARC323	ARC323				ARC325	ARC325	
STUDIO400										
BCH LAB		BCH221	BCH221	BCH424	BCH424					BCH225
HALL107			MAT421	MAT423						
HALL108	ESM222	ESM222	MAT224	MAT224	BLD522	BLD522	CSC441	CSC441	CSC446	CSC446
HALL201			ESM225	ESM225						
HALL202			BLD525	BLD525			ESM226	ESM226	MAT426	
HALL203			ESM529	ESM529				MAT425		
HALL204			MIS426	MIS426				BLD328	BLD328	
HALL306			CSC241	CSC241	BCH224		CIS226	CIS226		
HALL307			ESM323	ESM323						
HALL308			BLD327	BLD327						
HALL313			CHM480	ESM126	ESM126					
MCB LAB			MCB424	MCB424		MCB421	MCB421	MCB423	MCB423	
BLY LAB			BLY424	BLY424				BLY221	BLY221	
CHM RES LAB			CHM425	CHM425				CHM487	CHM487	
CHM LAB	CHM228	CHM222	CHM444	CHM444	CHM129	CHM129	CHM129			
COMPUTER LAB	CSC221	CSC221			CSC424					
HALL213			PHY422	PHY422			PHY421	PHY421	PHY441	PHY441
HALL302							PHY223	PHY223		
LT1	PHY121	PHY121	MAT121		EDS321	TMC221	TMC321	GST121	GST121	

Figure III: Snapshot of Tuesday Schedule

	8am-9am	9am-10am	10am-11am	11am-12noon	12noon-1pm	1pm-2pm	2pm-3pm	3pm-4pm	4pm-5pm	5pm-6pm
STUDIO100		ARC123	ARC123							
STUDIO200										ARC226
STUDIO300										
STUDIO400	ARC425	ARC425				ARC426	ARC426			
BCH LAB	BCH425	BCH425								
HALL107										
HALL108	MAT224		MAT222	MAT222	PHY225	PHY225			MAT225	BLD122
HALL201										
HALL202	CSC444	CSC444	BLD121	BLD121	CSC423	CSC423				
HALL203	ESM523	ESM523	CSC223	CSC223	ESM121	ESM121				
HALL204	ESM325	ESM325	ESM326	ESM326	BLD222	BLD222				
HALL306	BLD329	BLD329				MIS421	MIS421			
HALL307										
HALL308										
HALL313										
MCB LAB	MCB422	MCB422		MCB122	MCB122					
BLY LAB				BLY223	BLY223				BLY121	BLY225
CHM RES LAB	CHM423	CHM423		CHM446		CHM420	CHM420			
CHM LAB	CHM221	CHM221								
COMPUTER LAB							CSC121	CSC121	CSC225	
HALL213					PHY428	PHY423	PHY423			
HALL302				PHY129	PHY129	PHY129				
LT1	CHM121	CHM121	TMC421	TMC521	EDS521		GST222	GST222		

Figure IV: Snapshot of Wednesday Schedule

	8am-9am	9am-10am	10am-11am	11am-12noon	12noon-1pm	1pm-2pm	2pm-3pm	3pm-4pm	4pm-5pm	5pm-6pm
STUDIO100					ARC121	ARC121				
STUDIO200									ARC221	ARC221
STUDIO300			ARC326			ARC321	ARC321			ARC329
STUDIO400					ARC422	ARC421	ARC421			ARC426
BCH LAB	BCH428	BCH428								BCH222
HALL107			MIS221							
HALL108					BLD321	CHM441	CHM441	MIS423	MIS423	
HALL201								CHM461	CHM461	
HALL202						CHM481	CHM481	CHM223	CHM223	
HALL203						BLD221	ESM228	ESM228		
HALL204						BLD521	ESM324	ESM324		
HALL306						ESM521	MAT422	CHM422	CHM422	
HALL307								CSC442	CSC442	
HALL308								BLD223	BLD223	
HALL313								MAT424	MAT424	
MCB LAB					MCB222	MCB222				
BLY LAB						BLY423	BLY427			
CHM RES LAB					CHM488	CHM488	CHM464	CHM464		
CHM LAB				CHM129	CHM129	CHM129	CHM440	CHM440		
COMPUTER LAB	CSC424	CSC424			CIS228	CIS228				
HALL213						PHY421	PHY425	PHY425	PHY427	PHY427
HALL302						PHY229				
LT1			EDS121	EDS221				GST122	GST122	TMC121

Figure V: Snapshot of Thursday Schedule

	8am-9am	9am-10am	10am-11am	11am-12noon	12noon-1pm	1pm-2pm	2pm-3pm
STUDIO100							
STUDIO200			ARC221	ARC221			
STUDIO300	ARC326	ARC326	ARC321	ARC321			
STUDIO400	ARC429	ARC421	ARC421			ARC425	
BCH LAB		BCH422	BCH422				
HALL107			BLD522	CSC223			
HALL108	BLD323	BLD323	BLD122	BLD122		MIS423	
HALL201			CHM223	BLY223			
HALL202	BLD521	BLD521	ESM321	ESM321			
HALL203	ESM521	ESM521	BLD321	BLD321			
HALL204	MAT424	BCH222	BCH222	PHY225			
HALL306		MIS421	MAT422	MAT422			
HALL307			ESM526	ESM526			
HALL308			CIS421	CIS421			
HALL313			MAT221	BLD222			
MCB LAB	MCB222			MCB121			
BLY LAB	BLY423	BLY423	BLY427	BLY427	BLY121	BLY121	
CHM RES LAB	CHM424	CHM424	CHM421	CHM421			
CHM LAB	CHM226		CHM442	CHM442			
COMPUTER LAB							
HALL213		PHY428		PHY427	PHY423		
HALL302			PHY129	PHY129	PHY129		
LT1	CST121	CST121			GST221	GST221	EDS421

Figure VI: Snapshot of Friday Schedule