Exam Timetabling using Ant Colony Optimisation

Ankit Anand

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Abstract

There are several problems in nature that, on the onset, seem fairly simple to solve. But, They reveal their overall complexity when trying to find an optimal solution for the problem. Combinatorial problems are one such set of problems that are very difficult to solve. It involves finding an optimal arrangement of a finite set of objects that satisfies any specified conditions. Such problems, which don't have any trivial solutions, are commonly solved using heuristics. Heuristics are a type of approximation algorithms that find solutions to problems in a reasonable span of time.

Exam timetabling is one such problem, which is categorised under the category of NP-Hard complexity, and has been a common problem faced by many educational institutions. Different institutions have different requirements and face different constraints, which makes the problem even more complicated. Several heuristics have been devised and methods implemented to find an optimal solution of this problem which include local search, tabu search, evolutionary algorithms as well ass the Ant Colony Optimisation (ACO) algorithm, which is a population based meta-heuristic method.

MAX-MIN Ant System (MMAS), a variant of ant colony optimisation is implemented, and appropriate heuristic and pheromone models are defined. The obtained results prove that meta-heuristics such as Ant Colony Optimisation are highly effective at tackling exam timetabling and problems of its nature.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format.

Name: Ankit Anand Signature: Ankit Anand

Acknowledgements

With sincere regards, I would first like to thank my supervisor, Dr. Chris McCaig for his continual support and guidance throughout the period of the project. This endeavour would not have been possible without his knowledge and expertise. The meetings with him were always productive, filled with valuable advice and information useful for the implementation of this project.

I'd also like to express my heart felt gratitude to my significant other, Ms. Srishti Bhat for being the pillar of my strength and providing her love and support through the period of the project.

Contents

1 Introduction

The University Exam Timetabling problem is concerned with assigning venues and time slots to hold examinations for different courses, while satisfying the requirements of certain constraints, some of which are more critical than the others. It is an example of a combinatorial problem, where the primary goal is to find an optimal schedule, or rather a grouping of objects (in this case - students with course exams, time slots, and exam hall) such that the resulting groups satisfy the given constraints.

The standard definition of the exam timetabling problem was produced by **Burke, Kingston and de Werra**[5] (2004) which was given as:

"A timetabling problem is a problem with four parameters: T, a finite set of times; R, a finite set of resources; M, a finite set of meetings; and C, a finite set of constraints. The problem is to assign times and resources to the meetings so as to satisfy the constraints as far as possible."

Combinatorial problems like the Exam Timetabling problem are extremely hard and time consuming to solve manually. They are categorised under NP-hard class of problems[4]. Hence, there is a need to develop tools to generate the schedules automatically as well as optimally.

1.1 Origin

The problem of efficiently scheduling exams for students is a common problem that many educational institutions run into. For smaller universities, the time table may be easy enough to schedule by hand. But for large universities with thousands of students, offering multiple courses, and with limited exam halls, it becomes infeasible to create a schedule manually.

Consider a small school with two exam halls, with a seating capacity of three and five respectively. There are nine students attempting exams out of a total of six courses. The exams can be held in two slots per day, a morning slot and an afternoon slot. Taking these constraints into consideration, the school teacher would find it simple to create an exam schedule for the students by hand.

The resulting schedule may look something like this:

Day	Exam	Students	Time Slot	Room
1	Physics	A, B, C	Morning	AB103
1	Maths	D, E, F	Afternoon	AB103
2	Computer Sci	G, H, I	Morning	AB103
2	Chemistry	D	Afternoon	AB103
3	Electronics	A, B, G, H, I	Morning	AB105
3	Economics	C, E, F	Afternoon	AB103

Table 1: Typical Exam Schedule with less constraints

By Day 3, all students would have attempted exams for their respective courses with no conflicting schedules.

But what if there were instead three thousand students, attempting exams offered in twenty different courses, and the school had four exam halls with a maximum seating capacity ranging from forty to one hundred? It then becomes much harder for a person to manually calculate a conflict-free time schedule.

1.2 Constraints

An optimal algorithm for scheduling examinations needs to satisfy a certain number of constraints, as much as possible. The choice and design of these constraints directly influence the resulting design of the schedule. The constraints for the Exam Timetabling problem can be categorized into two types depending on their criticality - Hard constraints and Soft constraints[17].

1.2.1 Hard Constraints

Hard constraints can be described as constraints that can not be compromised on. It is compulsory for them to be satisfied by the optimisation function.

Some hard constraints for the examination scheduling problem are listed below:

- Conflicting exams should not be scheduled together, either in the same time-slot.

  A student can not realistically write two exams with overlapping time slots. All exams scheduled for a student must have distinct time slots with appropriate gaps in between.

- Total number of students assigned to an examination hall at a given time slot must not exceed the total seating capacity of that hall.

  Students should not be expected to attempt an exam without proper seating. Therefore, care should be taken to ensure that the number of students assigned to an exam hall is within the maximum seating capacity of the hall. If there are no halls which can satisfy this requirements, then multiple rooms can be assigned to accommodate the seating requirements for an exam.

- Students with special requirements should not be allocated in-accessible rooms.

  Exams with students requiring special arrangements, such as those requiring wheel-chair access, should be assigned adequate rooms which are accessible to students with such needs.

1.2.2 Soft Constraints

Soft constraints can be described as constraints that would be ideal to have, however they may be compromised upon if doing so results in a more efficient schedule. The exact soft constraints may vary with the organization seeking to schedule exams.

Some good examples of soft constraints are listed below:

- Students must have at least one overnight gap between consecutive exams.

  This constraint tries to guarantee that students will not be scheduled two exams on the same day. However, for efficiency reasons, this constraint may be compromised such that a student has two exams slotted for the same day, provided that the hard constraint of avoiding overlapping exams is satisfied.

- Exams must be spread out as evenly as possible.

  This soft constraint benefits the students as it ensures they have the maximum possible preparation time in between exams. However, it may not be possible to schedule evenly spaced exams for every student while also adhering to hard constraints. Hence, it is acceptable to compromise on equal spread.

- Exams should not be split across rooms

  The seating requirements for each exam should be satisfied by the room allocation. If there are no halls which can satisfy this requirement, then multiple rooms can be assigned to accommodate the seating requirements for an exam.

To measure the quality of the solution, Objective functions are used to measure the extent to which both the hard and soft constraints have been satisfied. A certain penalty will be associated with the formulation if the constraints are violated. The penalty for the violation of a hard constraint is usually weighed more, as minimisation of the violations associated with hard constraints is given priority for such scheduling problems[14].

2 Aim and Objective

There have been several approaches to finding an optimal solution for the examination timetabling problem. Few divide the problem into two stages, the construction stage that finds an initial solution which tries to meet all the required constraints (hard) and the improvement stage that optimises the solution found in the construction stage by minimizing desirable (soft) constraints.

The main objective of this project is to implement the Max-Min Ant Colony Optimization (ACO) algorithm to solve the Exam Timetabling Problem for different scenarios and also to identify the best parameter configuration or algorithm tweak which will produce the best results. This aim will be realised through the following objectives:

1. Gaining an in-depth understanding of the Exam Timetabling problem, the constraints associated with it, and procedures employed to tackle the same.
2. Identification of the parameter configurations required for developing an optimal model such as number of exam halls, number of students, number of time-slots etc
3. Implementation of an ACO algorithm variant, Max-Min Ant algorithm in order to construct an optimal solution.
4. Evaluation of the solutions generated by different parameter configurations of the implemented algorithm for different algorithm tweaks and variations in data-set.

3 Survey

This section introduces the various concepts and terminologies associated with combinatorial optimisation problems, their variants and how they are tackled. Finally, a brief overview of ant colony optimisation is presented along with sections providing a comparison of different ACO variants and others describing its application for the examination timetabling problem as designed in previously published research. A brief overview of the benchmark data-sets for the examination timetabling problem is also listed and described.

3.1 Combinatorial Optimisation Problems

Combinatorial problems are problems that involve finding a grouping of a finite set of objects such that the resulting groups satisfy the given criteria. Combinatorial problems can be classified into two types: Decision problems and Optimization problems.

Decision problems can be characterised as problems for which the solutions are obtained based on a set of logical conditions. Graph colouring problem is an example of decision problem where the goal is to assign colours to the graph vertices such that no two adjacent vertices have the same colour.

There are two variants of decision problems based on the nature of the required solution:

1. Decision variant determines whether a solution to the given problem exists or not.
2. Search variant aims to find the solution, if one exists.

Both variants are closely related as the algorithms used to solve one variant can also be used to solve the other.

In real life, most combinatorial problems that one comes across are optimization problems rather than decision problems. As the name suggests, optimization problems concern themselves with finding the optimal solution - which may be either minimal or maximal.

Optimization problems can be considered as a generalisation of decision problems. For example, an optimisation variant of the graph colouring problem would be concerned with finding the solution using the minimum possible unique colours, rather than accepting any solution that satisfies the adjacent edge condition.

Optimisation problems also have two variants depending on the nature of the required solution:

1. Evaluation variant: For a given problem, determine the optimal objective function value
2. Search variant: Find a solution with optimal objective function value for a given problem.

Many combinatorial problems are NP problems. Graph colouring problem is an example of NP-complete problem. Timetable scheduling problems are generally NP-hard problems. They are usually tackled through heuristic methods, either meta-heuristics or hyper-heuristics. Nowadays, with the advent of machine learning a combination of the above techniques with ML algorithms are employed to produce optimal results.

3.2 Heuristic Methods

The term ‘heuristic’ comes from the Greek word ‘eurisko’ which means ‘to discover’. A heuristic method refers to finding the best possible solution to a problem as efficiently as possible. The resulting solution may not be perfect, but within the constraints of time, cost, and effectiveness, it may be the most practical solution.

Heuristics are problem-dependent techniques. They tend to follow the greedy approach - that is, they divide the problem into parts and try to get the optimal solution for each part (in other words, the local optimum). These local optimums do not necessarily give a global optimum - the best possible solution to the overall problem.

Meta-heuristics, on the other hand, are problem-independent techniques. They are not designed specific to the problem, although they do take into account parameters that would be needed to achieve a solution.

Meta heuristics have been used to get optimum solutions to various NP problems such as:

1. Knapsack Problem
2. Traveling Salesman Problem
3. Graph Colouring
4. Scheduling problems

3.2.1 Genetic Algorithm

The genetic algorithm is a method of solving both constrained and unconstrained optimization problems. The algorithm gets its name as it is based on natural selection, the primary driver behind biological evolution. The genetic algorithm works by selecting certain individuals from the current population to be parents who would be used to produce children for the next generation. This process occurs at each step. The population thus evolves towards an optimal solution over successive generations.

The genetic algorithm is used to solve various optimization problems that cannot be solved by standard optimization algorithms. This includes problems in which the objective function is discontinuous, not differentiable, stochastic, or highly nonlinear. The genetic algorithm can also address problems of mixed integer programming, where some components are restricted to be integer values only.

To create the next generation, three main rules are followed:

1. Selection rules decide how parents are selected. Parents are the individuals of a generation that will give rise to the population that forms the next generation. The selection process is generally stochastic and can depend on the scores of the individual.

2. Crossover rules combine two parents to form children for the next generation.

3. Mutation rules apply random changes to individual parents to form children.

3.2.2 Simmulated Annealing

Simulated annealing is a method for solving unconstrained and bound-constrained optimization problems. This method is based on the physical process in which a material is heated and its temperature slowly lowered to decrease defects, thereby minimising the system energy.

The algorithm generates a new point at each iteration. The extent of the search, determined by the distance between the current and new points, is based on a probability distribution whose scale is proportional to the temperature. All new points that lower the objective are accepted. However, with a certain probability, points that raise the objective also get accepted. The latter is because accepting such points ensures that the algorithm avoids being trapped in local minima and is able to look for more possible solutions globally. As the algorithm proceeds, an annealing schedule is selected to systematically decrease the temperature. The decrease in temperature allows for the algorithm to reduce the extent of its search and finally converge to a minimum.

3.3 Ant Colony Optimisation

Ants exhibit complex social behaviors that have been an interesting area of scientific study and research, and a fascination for the humans since a long time. Some of the notable behaviours include their ability to arrange themselves and elect leaders democratically[19], use chemical signatures to map different sections of the nest[15]. One of the most noticeable behaviors among these is the formation of ant streets. This behavioral patterns exhibited by ants is the ability of certain ant species to find shortest paths by utilising an odorous chemical substance known as Pheromones. This behavioural trait of the ants was exploited by the computer scientists[12] to develop algorithms for the solution of optimization problems. Ant colony optimisation algorithms are one of the most successful and widely recognized algorithmic techniques based on ant behaviors and have been applied to many combinatorial optimisation problems.

Ant colony optimisation can be defined as a population-based probabilistic search technique utilised to find solutions of combinatorial optimisation problems. In the algorithm, the simulated ants attempt to locate optimal solutions by traversing through a parameter space representing all the possible solutions. The simulated ants, just like the real ants that lay down pheromones to guide other ants, similarly record their current positions and the quality of their solutions. The simulated ants, in the next iteration, can utilise the previous information to locate better solutions.

A typical ant colony optimisation algorithm **??** is presented below:

Algorithm 1 Ant colony optimisation meta-heursitic

```
Set parameters and initialise the pheromone trail
while until exit condition is not met do
    solution graph produced by the ants
    update pheromones
end while
return best solution
```

The selection of an edge is given by the following formula:

$$p_{xy}^{k} = \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum_{z \in allowed_x}(\tau_{xz}^{\alpha})(\eta_{xz}^{\beta})} \tag{1}$$

where,
τ_{xy} is the amount of pheromone deposited for transition from state x to y,
α is the parameter to control the influence of τ_{xy},
η_{xy} is the desirability of state transition xy,
β is a parameter to control the influence of η_{xy},
τ_{xz} and η_{xz} represent the trail level and attractiveness for the other possible state transitions.

Depending on the type of moves, whether good or bad, the trails are updated when an iteration is finished and ants have completed their solutions. For the Max-Min Ant System, invented by Thomas Stützle in 2000 [21], The pheromone updating rule is given by

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_{k}^{m} \Delta\tau_{xy}^{k} \tag{2}$$

where,
τ_{xy} is the amount of pheromone deposited for a state transition xy,
ρ is the pheromone evaporation coefficient,
m is the number of ants and
$\Delta\tau_{xy}^{k}$ is the amount of pheromone deposited by the kth ant

3.3.1 Comparison of ACO variants

The table below presents a comparison of the various ant algorithms, their edge selection strategy and pheromone updation strategy.

Algorithm	Edge Selection Strategy	Update Strategy
Max-Min Ant System	Selected via Equation(1)	Global best ant/solution at the end of the iteration updates the values of the pheromones[21]
Ant System	Selected via Equation(1)	Every ant updates pheromones at the end of each successive iteration[11]
Ant Colony System	Pseudo-random-proportional rule, which incentivises transitions with short edges with greater pheromone deposit [10]	Local pheromone updates are performed by the ants on the last edge. Global Pheromone update is performed by the iteration best ant.[10]
Rank Based Ant System	Selected via Equation(1)	The pheromone trails are updated based on the weighted rank of every ant. The rank is decided based on the quality of generated solutions by each ant[3].

Table 2: Comparison of few Ant algorithm variants

3.4 Exam Timetabling as Graph Coloring Problem

The literature review presented by **Burke** et al. [18] provides and in-depth overview of the prior research carried out to tackle Exam timetabling. Various formulations of the timetabling problems are discussed, one of which is graph coloring method. It has been particularly useful in developing good solutions when the principles of Graph Coloring problem are utilised to develop ant algorithms in order tackle the examination timetabling problem.

De Werra, in 1985 [9], compared course timetabling with the examination timetabling problem, identifying the differences between them while emphasising on the notable similarities between the two problems. The author also explored the various methods for time-tabling based on graph coloring methods. Graph Colouring is a type of NP problem where the goal is to assign colours to the nodes of a graph such that it follows certain restrictions. Commonly it involves the following restrictions:

1. No two adjacent nodes must have the same colour
2. No more than k colours may be used to colour the graph (where k is a positive integer and k > 2).
3. Find a solution where all nodes are coloured such that the least number of unique colours are used.

Consider a graph with seven vertices as shown below:

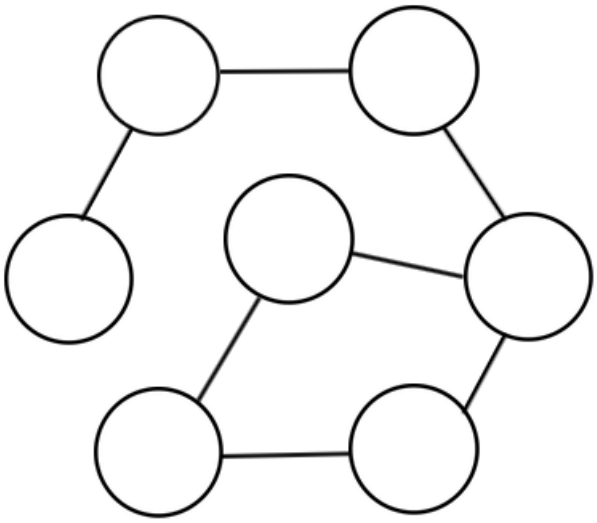

Figure 1: Empty Graph

The following are two of the many possible solutions to the problem.
Solution 1:

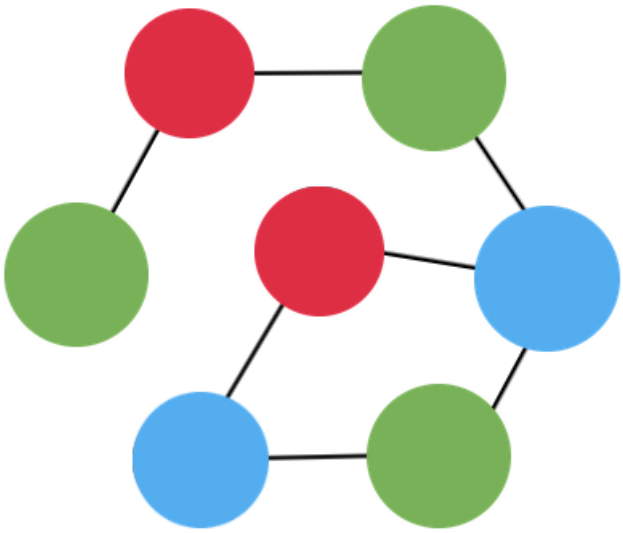

Figure 2: Sub-Optimal Solution

Solution 2:

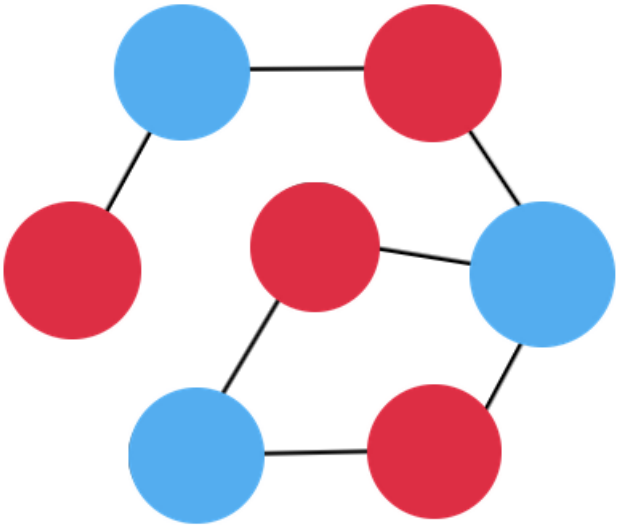

Figure 3: Optimally Assigned Graph

Clearly, solution 2 is better as it uses less unique colours than solution 1.
Exam timetabling problem can be rephrased as a graph colouring problem. In this case:

1. Each node would represent an exam.
2. Edges connecting two nodes indicate a conflict between the respective exams
3. Edges have a cost that corresponds to the common student ratio, which tells the degree of conflict caused by students having exams in common adjacent to each other.
4. Each colour represents a time slot. Two adjacent nodes cannot have the same colour, aka two exams cannot be scheduled in the same time slot.

3.5 Benchmark Data-Sets

Since the timetable scheduling problem is of great relevance to many institutions, there has been a significant amount of research interest on the problem. Thus, different benchmarks have been established to make scientific studies and comparisons of results more meaningful.

The following section will talk about the exam scheduling benchmark data sets used by different universities.

3.5.1 University of Toronto

Introduced in 1996 by Carter, Laporte, and Lee[8], it comprises of thirteen real world exam scheduling problems from several schools and universities from Canada, USA, Great Britain, and Saudi Arabia.

A Conflict Matrix C was defined to indicate the density of conflicting exams in each instance. Each element c_{ij} was defined as 1 if exam i conflicts with exam j (aka, they have common students), and 0 otherwise. The conflict density gives the ratio between the number of elements having the value of '1' and the total number of elements in the conflict matrix.

Two objectives were defined:

1. Minimise the number of time slots needed for the problem (graph colouring). The aim is to find time slots of the shortest length possible.
2. Minimise the average cost per student

The cost of timetables is calculated using an evaluation function. Any conflicting exams are to be spaced out within a limited number of time slots. Real world constraints, such as maximum room capacity for a given time slot, avoiding multiple exams with overlapping time slots, etc. were also introduced by the authors.

3.5.2 University of Nottingham

Introduced in 1996 by Burke, Newall, and Weare[6], it was used by researchers to test different approaches. The main objective of this dataset was to minimise the number of students who would sit for consecutive exams held on the same day.

3.5.3 University of Melbourne

Introduced at the PATAT conference in 2002 by Merlot et al[16], the University of Melbourne benchmark consists of two data-sets. Two time slots were provided per day for each of the five working days of the week. The capacity of each session varied. Time slot availability for some exams were restricted. These constraints meant that in one problem instance, no feasible solutions were possible. Hence an alternative data set was created that would allow feasible solutions.

3.6 Existing Exam Timetabling Systems

In this section we review the software systems currently used for solving the timetabling problems in real life, specifically examination timetabling. Two systems are reviewed, Optime and CMIS.

3.6.1 Optime

Optime[7] is an examination product that was released in 2006 by EventMAP Limited. The software addresses a pressing need in the education sector for automated scheduling systems. One of the major focuses of the company behind Optime is to bridge the gap between latest research successes and the actual demands of the industry, that is, between the time scheduling research and the actual requirements needed to schedule exams for universities and the like.

To achieve this, it is important to gather the important aspects of the institutional requirements and provide them to researchers in the field, so that they can work with these requirements to update their algorithmic techniques within the software. The resulting solutions would thus be workable and of high quality.The aim of improving Optime is to make the system as intelligent and intuitive as possible. It must provide maximum information to the institute administrator that would allow them to make informed strategic decisions.

3.6.2 Central Management Information System(CMIS)

Central Management Information System, or CMIS for short, is a university timetabling system. It also offers data relating to undergraduate and postgraduate timetabling as well as reports based on timetabling activities and performance.

CMIS provides several useful scheduling based features. It contains recordings of all learning and teaching activities with their schedules. It provides a record of all available spaces for booking. Rooms can be booked for various activities such as examinations, conferences, or other events. It allows for enrolling of students into programmes or other categories with specific custom criteria as needed. Fitting students into module sub-groups takes place offline in a separate CMIS application called TTCMIS, before being integrated back into CMIS. It allows for students and staff to access personal timetables from the portal. These are some of the features offered by CMIS.

4 Experiments

This section details the various elements used for the purpose of this project such as data-sets, model, parameters and algorithm definitions.

4.1 Data

To carry out the experimentation for this thesis, University of Nottingham data-set[2] was utilised with an addition of exam room and December diet time-slot data from University of Glasgow. The data was ingested in the csv (comma separated values) format and examples of structure of each data-set are described in the sections below.

4.1.1 University of Nottingham Data-Set

The University of Nottingham data-set utilised for the purpose of this project was categorised into three different data files containing students, exam and enrolment related information. Enrolments data-set provides a mapping of the students and their respective exams. The tables below briefly summarise the structure of each of the aforementioned data-sets.

Structure of Enrolments (Left) and Students(Right) data-sets

student_code	exam_code
A890186790	R13001E1
A892884191	Q4B102E1

student_code	course_code	accessibility_requirement
A890186790	R100	FALSE
A891097581	R200	TRUE

exam_code	name	duration	department_code
AA2016E1	OPERA STUDIES, I	01:30	GM
AA3008E1	HOLLYWOOD & THE EUROPEAN CINEMA	03:00	AI

Table 3: Structure of Exam data-set

4.1.2 Time Slots

At University of Glasgow, the December diet exams for the year 2021-22 were carried out from 9th to 17th December, with three slots available per day: 9:30, 13:00 and 16:30 respectively. The data-set was manually created and the basic structure of the data file is presented in the table below.

index	date	day	type
1	08/12/2021	Wednesday	Morning
2	08/12/2021	Wednesday	Afternoon

Table 4: Structure of Time-slot data-set

4.1.3 Exam Rooms

The room data available for the December diet exams consisted of 34 rooms, with specific capacity for each of the aforementioned time slots. The total seating capacity available for any of the slots was found to be 2653. This gave rise to another hard constraint that could be added to the model definition, which is that the total number of students assigned to any slot should not exceed above 2653. Although, this number was contingent upon the availability of the rooms per slot. The structure of the data is presented in the table below. Note that 'accessible' column represents its accessibility status of the room.

index	room_code	capacity	accessible
1	59 Oakfield Ave Rm 302 (no lift)	10	FALSE
2	Gilbert Scott Conf Suite Rm 251	15	TRUE

Table 5: Structure of Room data-set

4.1.4 Constraints

The following constraints, in addition to those specified in section 1.2, were given to meet the minimum set objectives for the University of Glasgow exam timetabling scenario.

1. A maximum of two exams should be scheduled in one day for a student
2. No more than 3 consecutive exams can be scheduled for one student over 3 days
3. The duration of evening exams should not be greater than 90 minutes
4. No evening slots should be scheduled for Saturday

4.2 Software Architecture

The following section presents the overall software architecture as given in Figure 4 of the system, showing its main components and the interactions between them. Python programming language and its libraries like Pandas and Numpy were used to process data as well as perform complex calculations. The data files in csv format, contain data which is pre-processed to intitialise various objects for the model. After the solution construction by the ant model, the resulting schedule is generated in the csv format.

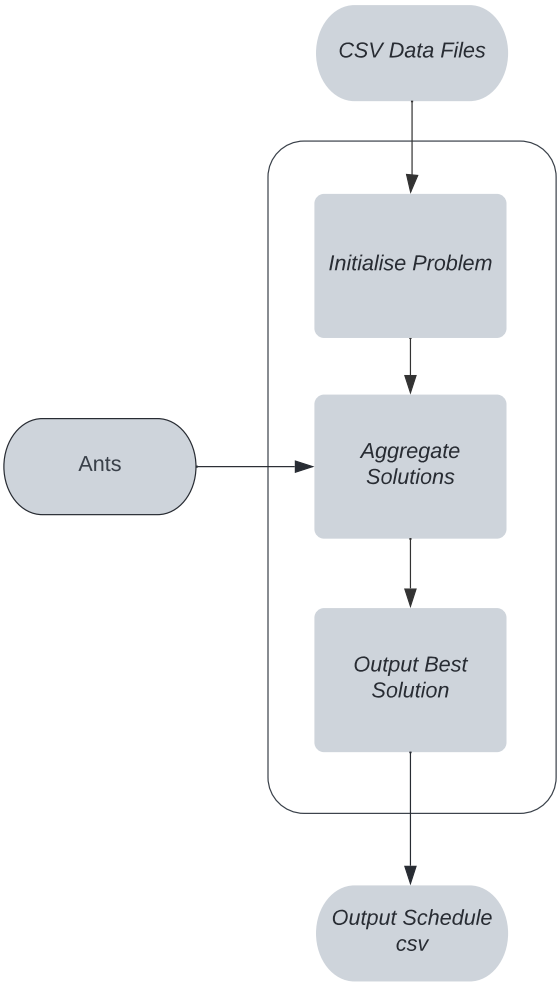

Figure 4: High level Software Architecture Overview

4.2.1 UML Class Diagram

The UML diagram shows the various classes used for the implementation of the project work and the relationships between them. List of objects for each class is initialised by the ingested csv data-files, which after pre-processing are further passed to the model to initiate solution construction. Considering the project work was carried out in a python notebook environment, the delineation of functions within the classes was deemed unworthy due to time constraints. Although, a more structured software application can be developed by using the code developed in this project.

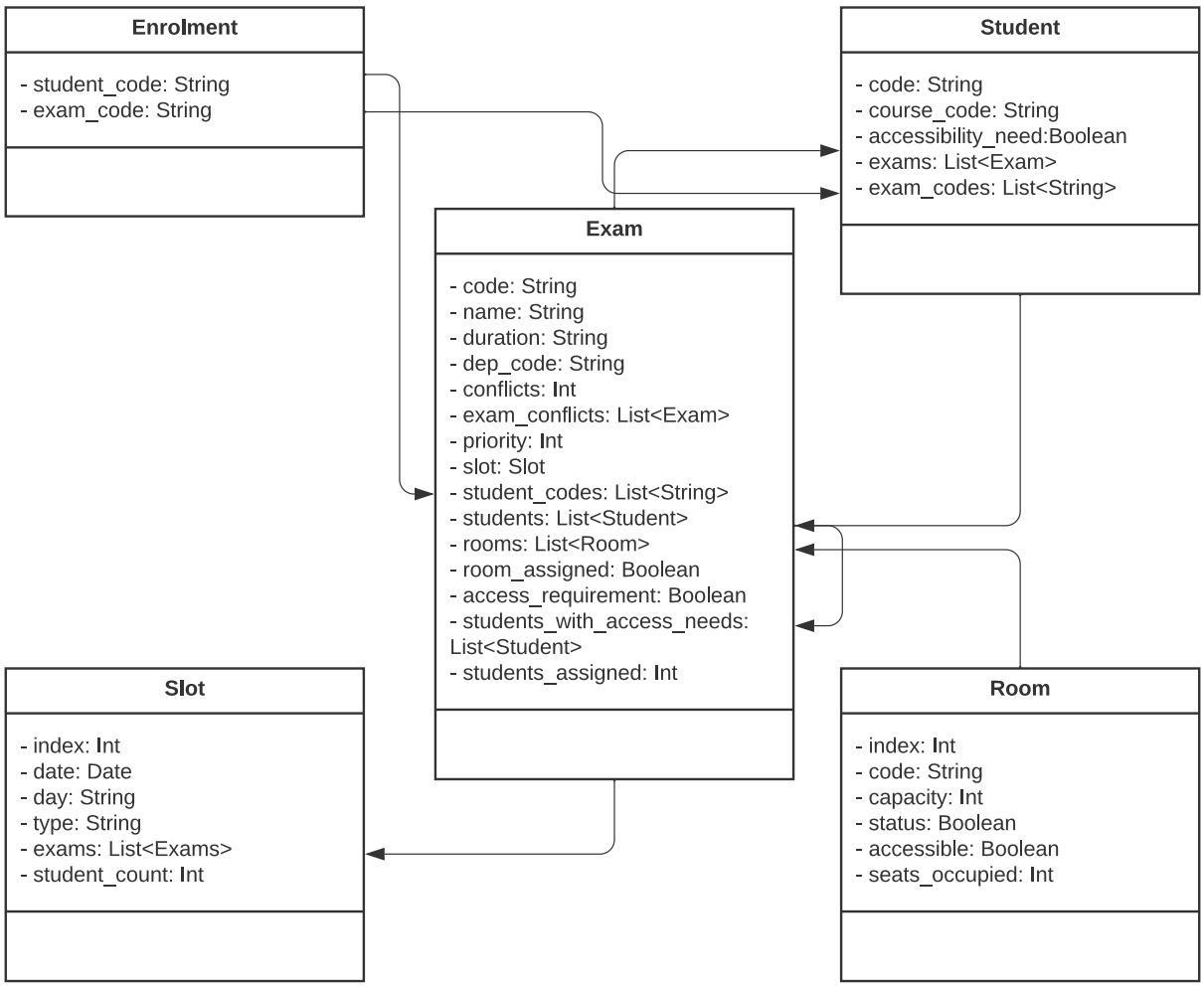

Figure 5: UML class diagram

4.3 Model Definition

The model can be defined as a tuple of four elements:

1. A set S of n students : $\{s_1, s_2, ..., s_n\}$
2. A set E of d exams : $\{e_1, e_2, ..., e_d\}$
3. A set R of c rooms : $\{r_1, r_2, ..., r_c\}$
4. Finally, A set T of s timeslots : $\{t_1, t_2, ...t_s\}$ as the time-slots available for each exam. For ex: t_1 could be the period starting from 9:30 am on 9th December.

The nomenclature mentioned above is used to describe the algorithms applied for the purpose of this project.

4.4 Max-Min Ant System

The general implementation and structure of the MAX-MIN Ant system for this project is largely inspired from the paper produced by (Socha [20] *et al., 2002*) with exclusion of local search procedures applied after the initial solution construction phase. Construction of the timetable is carried out by the ants, resulting in a global best solution which is updated after making a comparison with the iteration best solution.

Initially, the data files associated with the problem are imported. The data is pre-processed to initialise the parameters and variables associated with the various objects. Parameters such as Iteration Count, evaporation rate (ρ), max pheromone value (τ_{max}) and min pheromone value (τ_{min}) are configured. The pheromone matrix τ (E x T x R) is initialised with all the values set to the max pheromone value τ_{max}, and the heuristic matrix η (E x T x R) is initialised with values set to 1. Then, computations are performed to order the most difficult exam to schedule, based on various factors such as common student ratio, total number of students attending the exam, duration, and finally the alphabetic ordering of the exam code or name of the exam.

The exams are then added to a sorted list of exams, E_{sorted} based on the priority value assigned after performing the difficulty computation, d(e). Finally, After all the ants have finished constructing their solutions, the best assignment carried out among the ants is chosen as the iteration best solution ($A_{iterbest}$). The solution with least score, given by the number of constraint violations, is assigned as the $A_{globalbest}$, which then updates the pheromone trails globally.

The algorithm for the same is as follows: Here, **K** is the total number of ants and "totalConflictsForExams" is the procedure employed to calculate the priority value of each exam.

Algorithm 2 Max-Min Ant System Algorithm

Instantiate the problem and parameters
$\tau_{max} \leftarrow \frac{1}{\rho}$
τ (e,t,r) $\leftarrow \forall$ (e,t,r) $\in$ E x T x R
totalConflictsForExams(e) $\forall$ e $\in$ E
$E_{sorted} \leftarrow$ sort(E)
while stopping criteria not met **do**
  for k = 1 to |K| **do**
    S $\leftarrow \emptyset$
    for i = 0 to |E| **do**
      Select a time-slot t and room r or a sub-set of rooms following procedure in 4.6.2
      S $\cup$ {(e_i, t, r)}
    end for
    Evaluate constraint violations for solution A
    $S_{iterbest} \leftarrow$ better(S, $S_{iterbest}$)
  end for
  Evaluate constraint violations for solution S_{iter_best}
  $S_{globalbest} \leftarrow$ better($S_{iterbest}$, $S_{globalbest}$)
  Global pheromone update for τ using $S_{globalbest}$, τ_{min} and τ_{max}
end while
return best solution

4.5 Heuristic Influence and Pheromone Update

This section describes the configuration of pheromone and heuristic matrices that directly effect the selection of the slot and rooms in the model. The strategies employed to update both the factors are also briefly described.

4.5.1 Pheromone Matrix

The pheromone matrix is designed to be 3-dimensional representing rows as exams($e \in E$), columns as time-slots($t \in T$) and the layers or sheets as rooms($r \in R$). Initially, the pheromone matrix is initialised with the τ_{max} values. After which, the best solution obtained since the beginning of the solutions construction process is used to update the pheromone trails which is given as:

$$\tau(e,t,r) = \begin{cases} (1-\rho) * \tau(e,t,r) + 1, & \text{if } \textit{best solution placed exam e in time} - \textit{slot t and room r} \\ (1-\rho) * \tau(e,t,r), & \text{otherwise} \end{cases} \tag{3}$$

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

Following from the Max-Min Ant System, The pheromone trails is forced to be in the range [τ_{min}, τ_{max}]

$$\tau(e,t,r) = \begin{cases} \tau_{min}, & \text{if } \tau_{min} > \tau(e,t,r) \\ \tau_{max}, & \text{if } \tau_{max} < \tau(e,t,r) \\ \tau(e,t,r), & \text{otherwise} \end{cases} \tag{4}$$

4.5.2 Heuristic Information

Similar to the pheromone matrix, The heuristics matrix is designed to be 3-dimensional representing rows as exams($e \in E$), columns as time-slots($t \in T$) and the layers or sheets as rooms($r \in R$). The heuristic matrix is updated during the process of slot and room selection by the ant. The basic equation representing the update strategy for heuristic matrix is provided below:

$$\eta(e,t,r) = \frac{1 + numIncentive(e,t,r)}{1 + numAdditionalViolations(e,t,r)} \tag{5}$$

The heuristic information is affected by values obtained from two sub-procedures which are described below. The first procedure called "numIncentive" incentivises the ant towards a specific slot based on criteria specified. For this project, the ant was incentivised to choose an evening slot for exams with duration less than or equal to 1:30.

The second procedure, "numAdditionalViolations", inversely affects the updation of heuristic value. A score based on number of hard and soft constraint violations, arising due to a choice of specific slot or room for a given exam, defines the heuristic value for that node. Violations associated with hard constraints are given a score of 4, while those associated with soft constraints violations are assigned a score of 1. These scores are aggregated for each slot and room selection.

4.6 Important Procedures

This section describes the notable procedures used in the model such as priority based ordering of exams, slot and room selection.

4.6.1 Ordering Exams by Priority

Ordering of exams based on a calculated difficulty factor is important in order to assign slots and rooms for such exams early as presented in [13]. This invariably improves the quality of generated solution as described further in evaluation. Computations are performed to order the most difficult exam to schedule, based on various factors such as common student ratio, total number of students attending the exam, duration, and finally the alphabetic ordering of the exam code or name of the exam. Finally, The exams are added to a sorted list of exams in descending order, E_{sorted} based on the priority value assigned after performing the difficulty computation. The exam with the highest priority value is queued first for assignment of time-slot and the rooms. An overview of the algorithm is provided below:

Algorithm 3 Ordering Exams by Priority

```
for i = 0 to |E| do
    for j = 0 to |E| do
        if conflicts(i) > conflicts(j) then
            priority(i) ← priority(i) + 1
        else if conflicts(i) < conflicts(j) then
            priority(j) ← priority(j) + 1
        end if
        if len(i.students) > len(j.students) then
            priority(i) ← priority(i) + 1
        else if len(i.students) < len(j.students) then
            priority(j) ← priority(j) + 1
        end if
        if i.duration > j.duration then
            priority(i) ← priority(i) + 1
        else if i.duration < j.duration then
            priority(j) ← priority(j) + 1
        end if
        if i.code < j.code then
            priority(i) ← priority(i) + 1
        else if i.code > j.code then
            priority(j) ← priority(j) + 1
        end if
    end for
end for
```

4.6.2 Time-Slot Selection and Allocation of Rooms

The process for choosing time-slot and rooms, although convoluted, performs satisfactorily to select time-slot and rooms for each exam. The function takes the index of the exam, exam object, time-slots, rooms, α and β as its parameters. These are required to carry out computations related to slot and room selection. A 2 Dimensional array called, desirability is initialised with the dimensions T x R (length of time-slots and rooms), which will store the probability value of the ant choosing an edge.

Then a list of forbidden slots is created for the given exam, calculated based on the exams it is in conflict with. This essentially means that some students enrolled in the given exam are also appearing for other exams which have been allotted time-slots. Another matrix given the name "roomCapacityMatrix" (T X R), represented as **rc** in the algorithm below, is used to track the seats allocated to each room for the given time-slot. This matrix is required in-order to manipulate the heuristic value. If a room for the specified time-slot has all it's seats allocated, then an extremely low value is assigned to the heuristic factor(η). This reduces the desirability of the edge and hence, an ant is less likely to follow this edge. In another instance, desirability value for the same can be directly manipulated and assigned a negative score to completely negate the chances of an ant ever choosing that edge.

Following calculation and updation of the heuristic factor, desirability associated with the given edge is calculated, and desirability for the forbidden slots is updated to a negative value such as -1. The desirability matrix is sorted in descending order and the indexes are retrieved. A slot is randomly chosen from the sorted list of indexes with highest probability. This introduces some randomness to the model, allowing the ants to possibly improve upon the previous best solutions. Based on the chosen time-slot and seats allocated to the rooms, as visible in "roomCapacityMatrix"(rc), an apt room or a set of rooms are selected. The procedure returns the chosen slot and chosen set of rooms to be evaluated by the fitness function. A pseudo-mathematical representation of the algorithm is given below. Here,

- t represents the chosen time-slot
- R' represents the sub-set of Rooms selected from R
- T' represents the sub-set of forbidden Time-slots selected from T
- rc represents the room capacity matrix (T x R)

Algorithm 4 Choosing a Time-Slot(t) and suitable Rooms(R') for Exam (e)

$t \leftarrow 0$
$R' \leftarrow \emptyset$
$desirability \leftarrow np.zeros((len(T), len(R)))$
$T' \leftarrow findForbiddenSlots(e \in E)$

for i = 0 to |T| **do**
  for k = 0 to |R| **do**
    if rc(t,r) <= 0 **then**
      $\eta(e,t,r) \leftarrow 1E-20$
    else if rc(t,r) > 0 **then**
      $\eta(e,t,r) \leftarrow \frac{1+numIncentive(e,t,r)}{1+numAdditionalViolations(e,t,r)}$
    end if

    $desirability(t,r) \leftarrow \tau(e,t,r)]^{\alpha} * \eta(e,t,r)^{\beta}$

    for j = 0 to |T'| **do**
      if $t \in T \cap T'$ **then**
        $desirability(t,r) \leftarrow -1$
        continue
      end if
    end for
  end for

  $argSortedList \leftarrow sorted(argsort(desirability))$
  $t \leftarrow random(argSortedList)[0]$

  for index in argSortedList **do**
    if $len(e.students) - e.students\_assigned == 0$ **then**
      break
    end if
    if $index == t$ **then**
      $r \leftarrow index[1]$
      $R' \leftarrow r \in R$
      if $len(e.students) - e.students\_assigned >= rc(t,r)$ **then**
        $e.students\_assigned \leftarrow e.students\_assigned + rc(t,r)$
        $rc(t,r) \leftarrow 0$
      else if $len(e.students) - e.students\_assigned < rc(t,r)$ **then**
        $e.students\_assigned \leftarrow len(e.students) - e.students\_assigned$
        $rc(t,r) \leftarrow rc(t,r) - e.students\_assigned$
      end if
    end if
  end for
end for
return **t**,**R'**

5 Evaluation

In this section, we investigate the performance of the developed Ant algorithm for different parameter configurations, algorithm tweaks and variations in data-sets, such as, for different time-slot numbers and addition of students with special access needs. Each comparison is followed by an aggregated line chart visualisation representing the performance of the model for each iteration.

5.1 Problem Description

The following section provides a quantitative comparison of the different problem instances tackled for the project.

Instance	Description	Number of Exams	Number of Students	%age of Students with accessibility requirement	Number of Rooms	Number of Slots
1	Original Data-Set given in section 4.1	800	7896	0	34	27
2	Reduced Slots	800	7896	0	34	20
3	Increased Slots	800	7896	0	34	36
4	5% students with accessibility requirements	800	7896	5	34	27
5	20% students with accessibility requirements	800	7896	20	34	27
6	No Incentive Factor	800	7896	0	34	27
7	No priority based ordering	800	7896	0	34	27

Table 6: Comparison of different Problem Instances

5.2 Scoring Criteria

The evaluation score used below for the comparison represents the total number of violations (hard + soft) caused by the model and its variants. Violations associated with hard constraints were weighed higher than violations for soft constraints with a ratio of 4:1 (Hard Constraints:Soft Constraints). A model with a lower score is stated to perform better.

5.3 Parameter Configuration

The original data-sets are first evaluated based on four different parameter configurations. This helps us in defining a base line, which can be further used to compare the models involving algorithm tweaks and other instances mentioned in Table **??**. The better model out of the given four parameter configurations will be used as the baseline estimate for further evaluation.

It must be noted that the model can be additionally tested for numerous parameter configurations, hence, the evaluation below may not represent an optimal parameter set for the specified MMAS algorithm. These parameters were chosen based on readings from papers published for the same and experience gained while testing the algorithm. The time and technology based constraints associated with the project limit such exploration.

The four parameter configurations named **ACO1**, **ACO2**, **ACO3** and **ACO4** are given in the table below.

Parameter	ACO1	ACO2	ACO3	ACO4
Number of Ants	3	3	6	6
Evaporation Rate	0.2	0.2	0.2	0.2
Maximum Pheromone	5	5	5	5
Minimum Pheromone	0.019	0.019	0.019	0.019
Pheromone influence	1.0	1.0	1.0	1.0
Heuristics Influence	2.0	4.0	2.0	4.0
Hard constraint violation score	4	4	4	4
Soft constraint violation score	1	1	1	1
Total Iterations	10	10	10	10
Evaluation Score	182	188	161	180

Table 7: Evaluation of various parameter configurations

These parameters guide the main functioning of the developed algorithm and hence form the core part of the model. They were applied to the origin data-set as described in section 4.1. The figures below present the total number of hard and soft constraint violations associated with each parameter configuration.

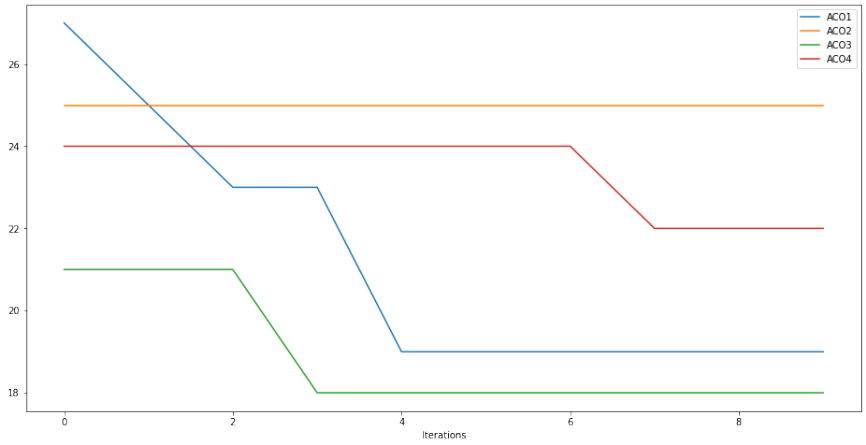

Figure 6: Total hard constraint violations per iteration

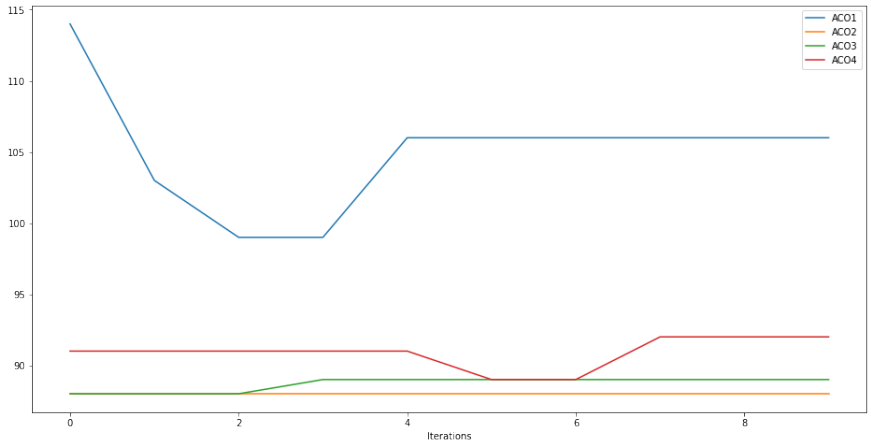

Figure 7: Total soft constraint violations per iteration

5.4 Algorithm Tweaks

The original model developed for the MMAS algorithm consists priority based exam ordering and an added incentive factor, part of heuristic influence to incentivise an ant towards a certain edge. In this section, the original model is compared with the models tweaked to remove any added incentive and pre-ordering of exams.

In order to evaluate the performance of these variations, the model was tested with all the aforementioned parameter configurations mentioned above in section 5.3. Here, **T1** is the model without an added incentive, and **T2** is the model without any pre-ordering of exams.

Algorithm Variation	ACO1	ACO2	ACO3	ACO4
Original	182	188	161	180
T1 (No added incentive)	127	132	136	125
T2 (No pre-ordering of exams)	306	315	294	312

Table 8: Evaluation of different algorithm tweaks

The figures below present the total number of hard constraint violations associated with the algorithm tweaks T1 and T2.

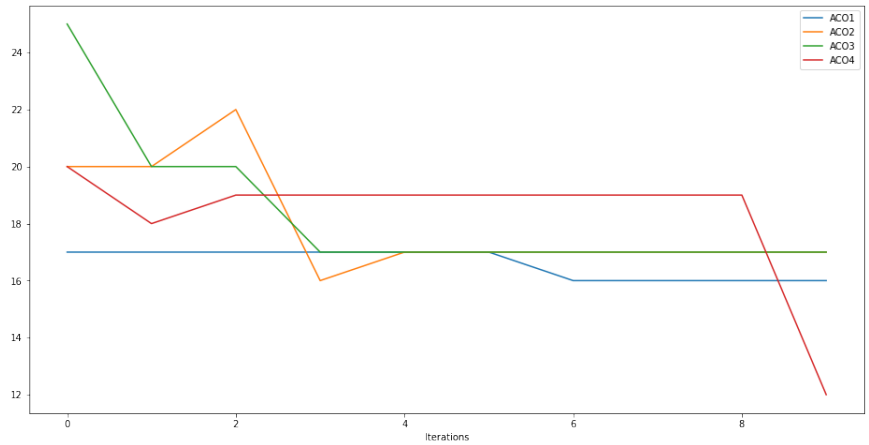

Figure 8: Total hard constraint violations per iteration for T1

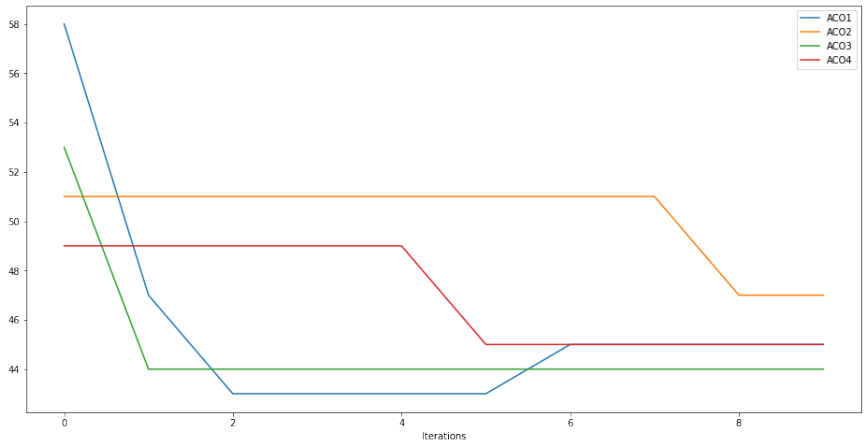

Figure 9: Total hard constraint violations per iteration for T2

The evaluations described in the sections below are carried out after removing the incentive factor and using the parameter configuration **ACO3** as the base model as shown in table 5.3.

5.5 Data-set Variations

In this section, we attempt to study the change in performance of the model when data-sets are modified. This can be used to present a more realistic outlook of the model when it is applied to different data-sets. The model is tested with limited variations of the original data-set and no other external data-set is used for the evaluation. The variations are described in detail in the sections below.

5.5.1 Variations in Slots Data

One of the most important part of the experimentation for the examination timetabling, apart from the various parameters required for ant algorithms, is the total number of available slots and the constraints associated with them. The performance of the model seems to be directly correlated to the number of available slots.

Thus, to evaluate the performance of the model for different slot arrangements, we modify the original slots data-set (D0) to extract another time-slot data-set with increased number of available slots. The number of available slots was increased from 27 to 36.

The model is tested with the **ACO3** parameter configuration mentioned above in section 5.3 and the result of its evaluation is tabulated below.

Data-set Variation	Number of Slots	ACO3 Score
D0(Original slots)	27	133
D1(Increased Slots)	36	45

Table 9: Evaluation of different slots variants

The figures below present the total number of hard constraint violations associated with the data-set variations D0, D1 and D2.

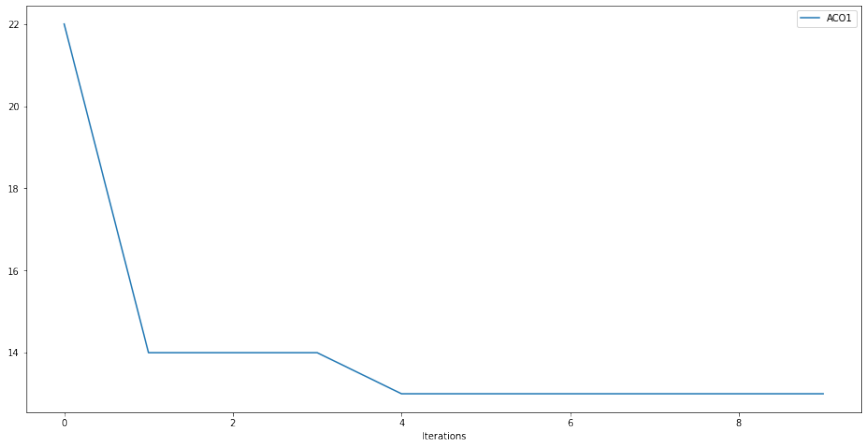

Figure 10: Total hard constraint violations per iteration for D0

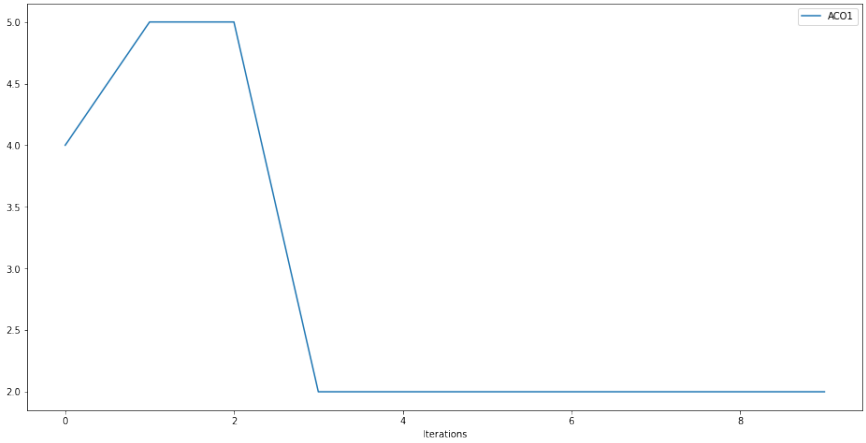

Figure 11: Total hard constraint violations per iteration for D1

5.5.2 Variations in Students Data

The original student data-set (DS0) lacks any information related to the accessibility requirements for students who will need access to additional assistance and facilities in order to successfully attend exams. As per the **World Health Organisation**, About 15% of the world's entire population is afflicted with some type of disability, in which around 2-4% face serious difficulties in performing routine activities, including reading and writing[22].

Considering the WHO study, two variants of the original student data-set were generated. A total of 5% of the students were randomly sampled and assigned Boolean "need_accessibility" status for the first generated data-set (DS1). And, For the second generated data-set (DS2), 20% students were randomly sampled and assigned the Boolean "need_accessibility" status. Similarly, the original rooms data-set was modified to include accessibility status for each room. So, in our case, rooms which lack access to lifts were assigned a false boolean value to denote that accessibility requirements were not met.

The model is tested with the **ACO3** parameter configuration mentioned above in section 5.3 and the result of its evaluation is tabulated below.

Data-set Variation	%age of Students with accessibility requirement	ACO3 Score
DS0	0	180
DS1	5	205
DS2	20	148

Table 10: Evaluation of different variations in students data

The figures below present the total number of hard constraint violations associated with the data-set variations DS1 and DS2.

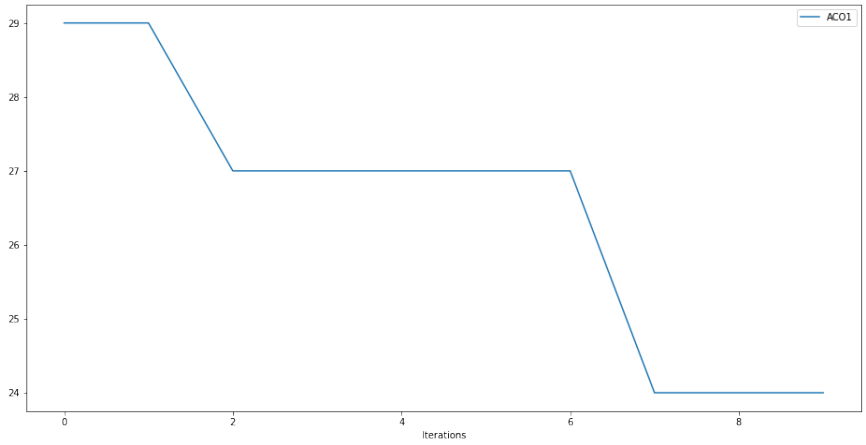

Figure 12: Total hard constraint violations per iteration for DS1

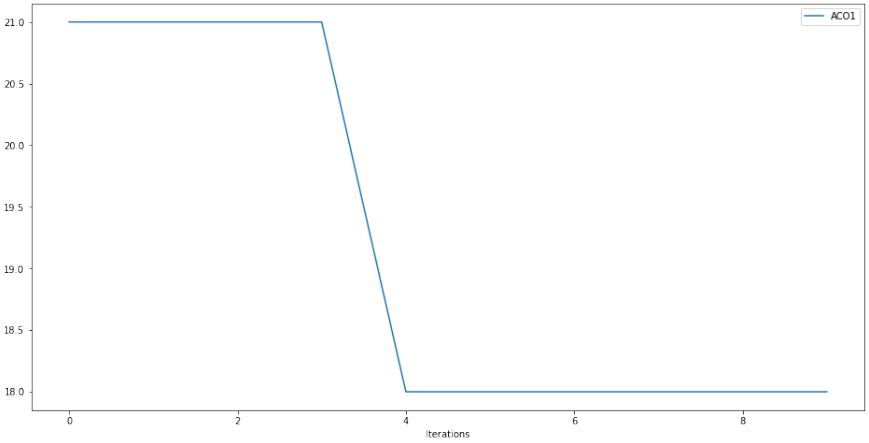

Figure 13: Total hard constraint violations per iteration for DS2

6 Summary

In summary, A number of conclusions can be drawn from the experiments carried out, which leaves room for further improvement and modifications. Min-Max Ant Colony Optimisation (ACO) algorithm was used to tackle the University Examination Timetabling Problem, which is a combinatorial optimisation problem. Appropriate heuristics and pheromone models were devised to tackle the problem. Some notable additions like incentivising ants to move towards a certain node produced better results. The model was evaluated and compared with different parameter configurations, data-set and algorithmic variations. Finally, the study carried out in this project can prove that meta-heuristic algorithms such as Ant Colony Optimisation, if tuned properly for a problem domain can possibly produce effective and acceptable results.

6.1 Results

From the results of the evaluation, It can be gleaned that the parameter configuration ACO3, as given in Table 5.3 performs consistently better when compared to other parameter configurations. The removal of incentive factor, from heuristic influence, seemed to improve the performance of the model overall as evident from the evaluation score in the table 8 and the reduced number of hard constraint violations as seen in Figure 8. Since, the base model instances in Section 5.3 were tested with the incentive factor in addition, the outcome of these instances could be inadvertently improved by removing the incentive factor. Alternatively, other strategies can be devised to properly study and establish the performance of the model when such incentive mechanism is employed.

Unlike the first algorithm tweak, when priority based pre-ordering of the exams was removed, the model seemed to perform considerably worse as evident from the higher overall evaluation score, as observed in table 8 and increased number of hard constraint violations as seen in Figure 9. This proves that pre-processing of similar nature is very important when it comes to tackling constraint based optimisation or scheduling problems.

The variations in data-set were evaluated with the ACO3 parameter configuration, and no added incentive factor. As observed in table 9, When the number of slots were increased from the original 27 to 36, the number of violations associated with hard constraints and those with soft constraints were reduced significantly as seen in Figure 11 and 18, when compared to the original number of slots, "26" the performance of which is shown in figure 10 and 17. Thus, It can be stated that there is a negative correlation between the number of slots and performance of the model.

When the constraints associated with accessibility requirements were introduced, a general increase in total number of constraint violations is observed. However, there is no significant correlation between the increase in percentage of students with special accessibility needs and the performance of the tested ant model. This can be partially gleaned form the results obtained in Table 10 and Figures 12, 13 and 19, the evaluation score for DS2 is less than that of DS1. This is possibly due to the fact that enough accessible rooms were available in the rooms data-set. Increasing the number of students with accessibility needs further would potentially degrade the performance of the model.

6.2 Future Work

The results produced in this work leave space for further research and exploration when it comes to tackling exam timetabling using Ant colony algorithms. The model presented here can be combined with other search techniques such as local search, tabu search or hill climbing techniques to analyse its hybridised implementation which are known to produce better results as observed in [1]. The pheromone and heuristics scheme employed in this work can be tested with other Ant colony algorithm variants like Ant Colony System, Rank based ant system etc. In Computer Science, Constraint programming is another area which is directly relevant to the scheduling problem where constraints are involved. It is an effective way to model and solve constraint optimisation problems. An approach that hybridises the developed model with constraint programming languages would be particularly innovative.

Parallelisation of Ant colony algorithms has seen an increasing amount of interest off late as evident from the numerous research published in the previous decade such as [23]. Hence, an attempt can be made to parallelise the model taking advantage of the increasing number of cores in CPU and GPU. The incentive approach utilised for updating heuristic information also requires further exploration where different strategies can be employed to provide such incentives. Finally, the model can be tested against other benchmark data-sets to scientifically establish its performance and where it stands amongst other algorithms and strategies employed to tackled exam timetabling.

6.3 Challenges and Limitations

Although, an enlightening experience, the implementation of this work faced several obstacles such as time constraints and technical equipment constraints. Ant colony is one of the algorithms in heuristics domain which lacks an established framework, and this proved to be a major challenge. Majority of the code implementation carried out in this project was done from scratch, which required careful planning and thinking. A big proportion of the time for implementing such a work is devoted to reading relevant research published before, and an efficient model can only be devised using the knowledge gained from such readings.

The project outcome could certainly improve if more time is devoted to it. Technical constraints like limited CPU capacity proved to be major barriers in efficiently writing and testing the code.

7 Appendix I

7.1 Directory Structure

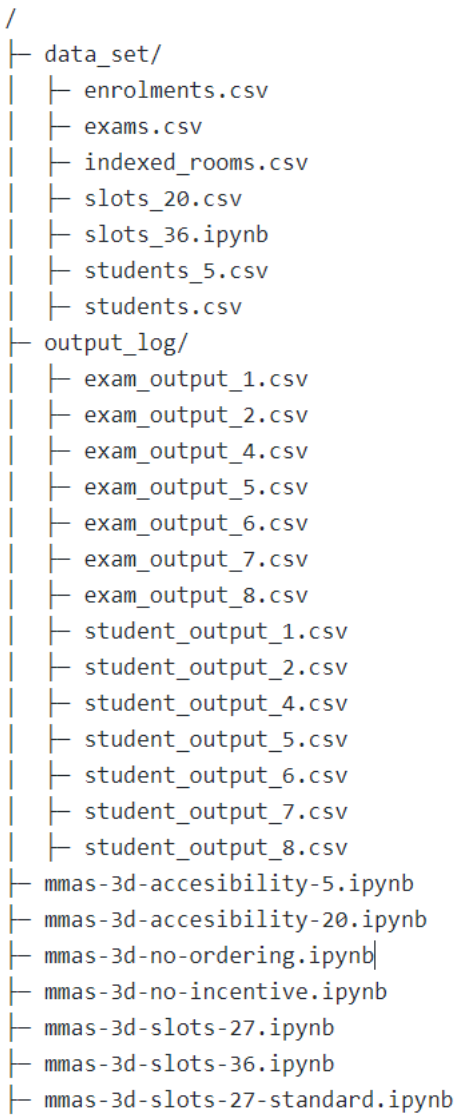

Figure 14: Directory Structure

Index	File Name	Description	Format
1	mmas-3d-accessibility-5.ipynb	Evaluation of DS1 from section 5.5.2	ipynb
2	mmas-3d-accessibility-20.ipynb	Evaluation of DS2 from section 5.5.2	ipynb
3	mmas-3d-accessibility-no-ordering.ipynb	Evaluation of section 5.4	ipynb
5	mmas-3d-slots-27-no-incentive.ipynb	Evaluation of section 5.4	ipynb
6	mmas-3d-slots-27.ipynb	Evaluation of D0 from section 5.5.1	ipynb
7	mmas-3d-slots-36.ipynb	Evaluation of D2 from section 5.5.1	ipynb
8	mmas-3d-slots-27-standard.ipynb	Just a standard run of ACO3 parameter configuration 5.3	ipynb

Table 11: Description of Python Notebooks

File Name	Description	Format
enrolments.csv	Enrolments data containing student and exam codes	csv
exams.csv	Exams data containing exam codes, name, duration and course codes	csv
indexed_rooms.csv	Rooms data containing name, capacity and accessibility status for each room	csv
slots_36.csv	Time slot data with date, day and type of slot limited to 36 data items	csv
slots.csv	Time slot data with date, day and type of slot limited to 27 data items	csv
students_5.csv	Students data with 5% students having accessibility needs	csv
students_20.csv	Students data with 20% students having accessibility needs	csv
students.csv	Students data containing student code, course code and accessibility requirement for each student	csv

Table 12: Description of Input Data Files

File Name	Type of Output	Related Notebook Index	Format
exam_output_1.csv	Exam Schedule	6	csv
exam_output_2.csv	Exam Schedule	7	csv
exam_output_4.csv	Exam Schedule	1	csv
exam_output_5.csv	Exam Schedule	2	csv
exam_output_6.csv	Exam Schedule	3	csv
exam_output_7.csv	Exam Schedule	5	csv
exam_output_8.csv	Exam Schedule	8	csv
student_output_1.csv	Students Schedule	6	csv
student_output_2.csv	Students Schedule	7	csv
student_output_4.csv	Students Schedule	1	csv
student_output_5.csv	Students Schedule	2	csv
student_output_6.csv	Students Schedule	3	csv
student_output_7.csv	Students Schedule	5	csv
student_output_8.csv	Students Schedule	8	csv

Table 13: Description of Output Data Files and its association with Table 11

Two types of output files are generated. First, "exams_output.csv" providing exam specific assignment of slots and rooms. And second, "student_output.csv" providing exam specific assignment of slots and rooms.

8 Appendix II

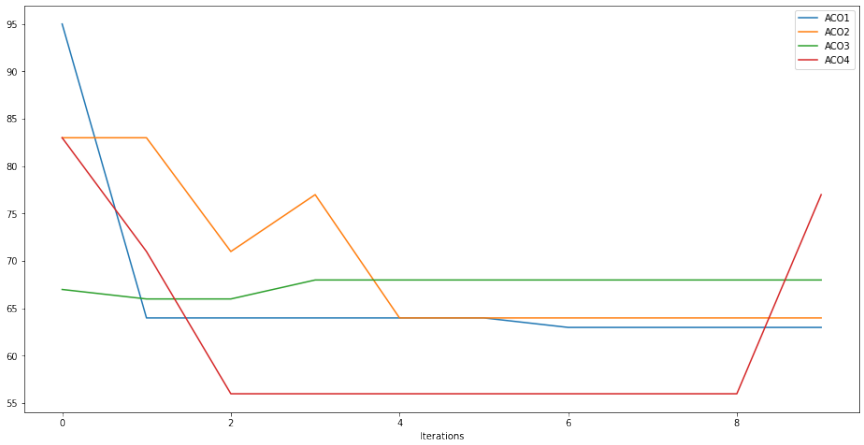

Figure 15: Total soft constraint violations per iteration for T1

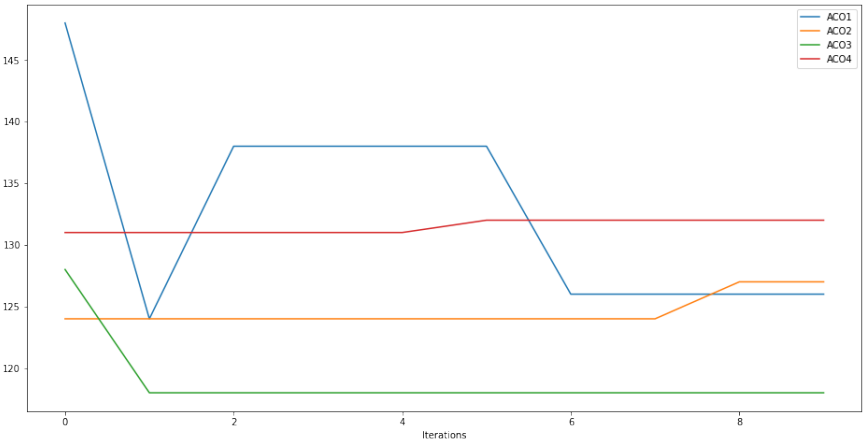

Figure 16: Total soft constraint violations per iteration for T2

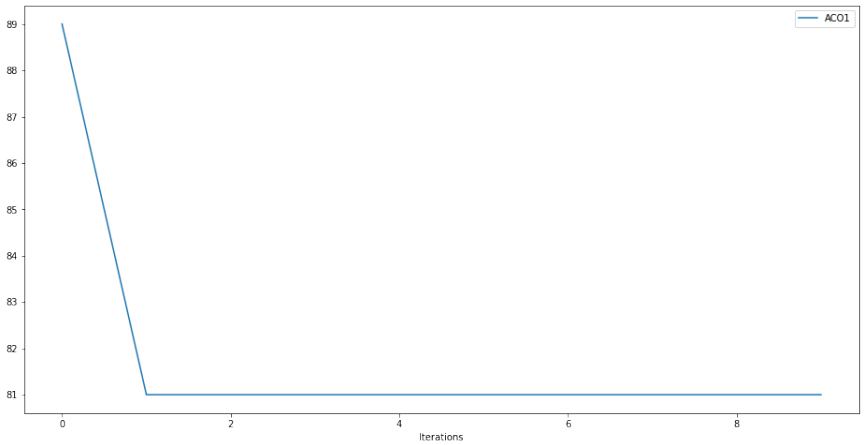

Figure 17: Total soft constraint violations per iteration for D0

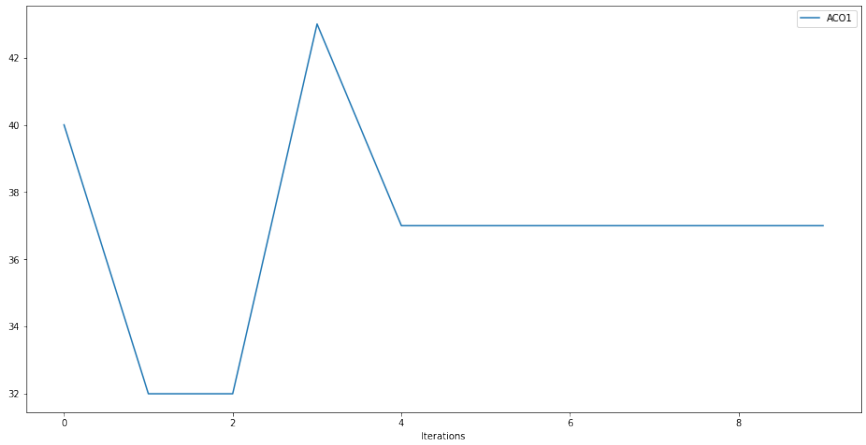

Figure 18: Total soft constraint violations per iteration for D1

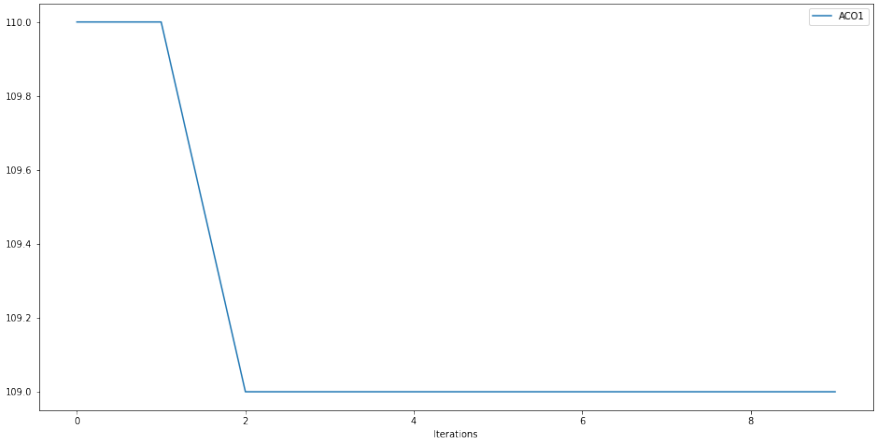

Figure 19: Total soft constraint violations per iteration for DS1

9 Bibliography

[1] Zahra Naji Azimi. "Comparison of metaheuristic algorithms for examination timetabling problem". In: *Journal of Applied Mathematics and Computing* 16.1-2 (2004), pp. 337–354. DOI: 10.1007/bf02936173.

[2] *Benchmark exam timetabling datasets*. URL: http://www.cs.nott.ac.uk/~pszrq/data.htm.

[3] Bernd Bullnheimer, Richard F. Hartl, and Christine Strauß. *A new rank based version of the Ant System. A computational study.* English. WorkingPaper 1. SFB Adaptive Information Systems et al., 1997.

[4] E. Burke et al. "Automated University Timetabling: The State of the Art". In: *The Computer Journal* 40.9 (Jan. 1997), pp. 565–571. ISSN: 0010-4620. DOI: 10.1093/comjnl/40.9.565. eprint: https://academic.oup.com/comjnl/article-pdf/40/9/565/981064/400565.pdf. URL: https://doi.org/10.1093/comjnl/40.9.565.

[5] E.K. Burke and J.P. Newall. "Solving examination timetabling problems through adaption of heuristic orderings". In: *Annals of Operations Research* 129.1-4 (2004), pp. 107–134. DOI: 10.1023/b:anor.0000030684.30824.08.

[6] Edmund K Burke, James P Newall, and Rupert F Weare. "A memetic algorithm for university exam timetabling". In: *international conference on the practice and theory of automated timetabling*. Springer. 1995, pp. 241–250.

[7] Edmund K. Burke et al. *Optime: Integrating Research Expertise with Institutional Requirements*. 2006.

[8] Michael W. Carter, Gilbert Laporte, and Sau Yan Lee. "Examination timetabling: Algorithmic strategies and applications". In: *Journal of the Operational Research Society* 47.3 (1996), pp. 373–383. DOI: 10.1057/jors.1996.37.

[9] D. de Werra. "An introduction to timetabling". In: *European Journal of Operational Research* 19.2 (1985), pp. 151–162. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(85)90167-5. URL: https://www.sciencedirect.com/science/article/pii/0377221785901675.

[10] M. Dorigo and L.M. Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 53–66. DOI: 10.1109/4235.585892.

[11] M. Dorigo, V. Maniezzo, and A. Colorni. "Ant system: optimization by a colony of cooperating agents". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.1 (1996), pp. 29–41. DOI: 10.1109/3477.484436.

[12] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. "Ant colony optimization". In: *IEEE Computational Intelligence Magazine* 1.4 (2006), pp. 28–39. DOI: 10.1109/MCI.2006.329691.

[13] Marco Dorigo and Krzysztof Socha. "An Introduction to Ant Colony Optimization". In: (May 2006).

[14] Johan Havås et al. *Modeling and optimization of university timetabling - A case study in Integer Programming*. Oct. 2013. URL: https://gupea.ub.gu.se/handle/2077/34259.

[15] Yael Heyman et al. "Ants regulate colony spatial organization using multiple Chemical Road-signs". In: *Nature Communications* 8.1 (2017). DOI: 10.1038/ncomms15414.

[16] Liam T. G. Merlot et al. "A Hybrid Algorithm for the Examination Timetabling Problem". In: *Practice and Theory of Automated Timetabling IV*. Ed. by Edmund Burke and Patrick De Causmaecker. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 207–231. ISBN: 978-3-540-45157-0.

[17] R. Qu et al. "A survey of search methodologies and automated system development for examination timetabling". In: *Journal of Scheduling* 12.1 (2008), pp. 55–89. DOI: 10.1007/s10951-008-0077-5.

[18] R. Qu et al. "A survey of search methodologies and automated system development for examination timetabling". In: *Journal of Scheduling* 12.1 (2008), pp. 55–89. DOI: 10.1007/s10951-008-0077-5.

[19] Harikrishnan Rajendran et al. "Ants resort to majority concession to reach Democratic consensus in the presence of a persistent minority". In: *Current Biology* 32.3 (2022). DOI: 10.1016/j.cub.2021.12.013.

[20] Krzysztof Socha, Joshua Knowles, and Michael Sampels. "A MAX-MIN Ant System for the University Course Timetabling Problem". In: *Ant Algorithms*. Ed. by Marco Dorigo, Gianni Di Caro, and Michael Sampels. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 1–13. ISBN: 978-3-540-45724-4.

[21] Thomas Stützle and Holger H. Hoos. "MAX–MIN Ant System". In: *Future Generation Computer Systems* 16.8 (2000), pp. 889–914. ISSN: 0167-739X. DOI: https://doi.org/10.1016/S0167-739X(00)00043-1. URL: https://www.sciencedirect.com/science/article/pii/S0167739X00000431.

[22] WHO. *World Report on Disability*. URL: https://www.who.int/publications/i/item/9789241564182.

[23] Yi Zhou et al. "Parallel ant colony optimization on multi-core SIMD CPUs". In: *Future Generation Computer Systems* 79 (2018), pp. 473–487. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.09.073. URL: https://www.sciencedirect.com/science/article/pii/S0167739X16304289.