BIRZEIT UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Examination Timetabling

Prepared by

Ahmad Zaid

Mahmoud Abdelkareem

Mohammad Sehweil

Supervised by

Dr. Ahmad Afaneh

A graduation Project submitted to the Department of Electrical and Computer Engineering in partial fulfillment of the requirements for the degree of B.Sc. in Computer Engineering

BIRZEIT

January – 2018

This page intentionally left blank

Abstract

Time is one of the most important factors in our everyday lives, this importance shines when there is an urgent task on hand, a critical situation or even when time is needed to study for an exam and to psychologically prepare for it. This report and research is dedicated towards contributing to the solution of the examination-timetabling problem. Having the right schedule not only gives the students the needed time to prepare and study for their exams, it also gives them the right psychological state to relax and undergo their exams without much pressure.

When looking at examination schedules, and how packed they can be, one starts to think if it is fair enough to decide more than 50% of the semester's grade in a matter of hours. In some cases, students had to take more than one exam in a single day, sometimes with not a single minute between the two. Typically, a university has over 10,000 students taking over 500 courses (total of all the courses), and this huge number of variables makes it near impossible to come up with a perfect exam schedule in an exam period that is less than 2 weeks long.

The method that was used to solve this problem was the Constraint Satisfaction Problem (**CSP**), which relied on hard "obligatory" constraints and soft "preferred" constraints. CSP depends on considering each exam as a variable and the timeslots as values that can be assigned to these variables. The goal of the project is to come up with the best method to assign these values (timeslots) to the variables in order to come up with the best schedule, without leaving an exam empty valued.

المستخلص

يعتبر الوقت من العناصر المهمة في الحياة اليومية لكل فرد منا, وتبرق شدة هذه الاهمية في حالة وجود حالة حرجة ,قضية عاجلة ,مهمة يجب ان تنجز او حتى في حالة الحاجة اليه للدراسة من اجل اختبار معين والتحضير نفسيا له .يتجه هذا البحث والتقرير بالاهتمام والتركيز على ايجاد حل لمعضلة (مشكلة) جدولة الامتحانات. إنّ الحصول على الترتيب الجيد والصحيح لمواعيد جدول الامتحانات لا يوفر للطالب فقط فرصة الدراسة والتحضير الجيد للامتحان، وإنما يساعد في توفير حالة التحضير النفسي الجيد للطالب من أجل حصوله على الاسترخاء المرجو, وتقديم امتحاناته دون شعوره بالضغط.

عند النظر الى جداول الامتحانات والى كيفية تجميعهم وضغطهم، يبدأ اي منا بالتساؤل عن مدى عدالة تقرير اكثر من نصف معدل الفصل في غضون ساعات. هناك بعض الحالات التي يضطر فيها الطلاب الى تقديم اكثر من امتحان واحد خلال يوم، احيانا اخرى يكون هناك فارق لا يتجاوز البضع دقائق بينهم. فعليا ،إنّ الجامعة تتكون من ما يتجاوز العشرة آلاف طالب مسجلين لأكثر من خمسمئة مساق (المجموع الكلي للمساقات). وهذه الكمية الهائلة من المساقات التي يمكن ان يرمز لها ب (المتغيرات) تجعل من تصميم جدول امتحانات متقن ومناسب للجميع خلال فترة الامتحانات التي لا تتعدى الاسبوعين امرا صعبا او شبه مستحيلا احيانا.

إنّ الخوارزمية المتبعة لحل هذه المشكلة تدعى (Constraint Satisfaction Problem). والتي تعتمد على قيود الزامية معقدة بالإضافة الى قيود أخرى اقل تعقيدا تعتبر مفضلة للأخذ او التطبيق. إنّ الخوارزيمة تعتمد على النظر الى كل اختبار على انه متغير , وعلى النظر الى كل وقت متوفر وفارغ لأي اختبار على انه قيمة من الممكن أن تُسنَد الى هذه المتغيرات. إنّ الهدف الأساسي من هذا البحث هو الحصول على أفضل الطرق الممكنة من أجل إسناد القيم الى تلك المتغيرات وبالتالي الحصول على أفضل جدول امتحانات كناتج دون ترك أي اختبار من غير وقت مخصص له.

Table of Contents

List of Tables

List of figures

Acknowledgment

It has been a great honor working on this project, which will hopefully give back to the educational community in general and to Birzeit University in particular. None of this would have ever been possible without the endless help of our supervisor Dr. Ahmad Afaneh. In addition, we would like to thank Dr. Aziz Qaroush, who never stopped giving us helpful feedback and attention. We sincerely thank Birzeit University for the support and help they offered us, and the test data we received from them. Finally, nothing is possible without those in our everyday lives, the ones who never stopped encouraging us, our families, friends and colleagues.

Table of Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
BZU	Birzeit University
CP	Constraint Programming
CSP	Constraint Satisfaction Problem
ETP	Examination timetabling problem
FOL	First Order Logic
OPL	Optimization programming language
OOP	Object-Oriented

Chapter I: Introduction

1.1 Timetabling Problems Overview - A generic timetabling system.

Timetabling in general is a computationally difficult problem, which is concerned with scheduling. The main goal of the timetabling problem is to find slots to achieve certain tasks that use up limited resources. For example, scheduling courses in a university is a timetabling problem where the goal is to find the appropriate schedule (time-slots) and place them in an appropriate room where they are taught by a teacher (the resources). Timetabling mainly relies on the constraints of the problem; moreover, the outcome of the problem (solution) can differ with different constraints. These constraints are put with objectives in mind, these objectives can be to minimize the length of the tasks to be scheduled, use as little resources as possible, not to violate any constraints and many other objectives, which can be taken into consideration.

Timetabling can happen in any setting or institution, but the focus of it is in educational institutions. Where the main goal is to satisfy the scheduling of courses for the students and their teachers. This problem has a tremendous amount of research regarding it; this is due to the fact of it having great significance and difficulty, where sometimes the goal becomes to just find a feasible solution. Moreover, this problem is very difficult to generalize, this is because it is different from one setting to another and different educational institutions prefer different objectives. Schools for example, have continues periods or time-slots, whereas universities are more flexible and prefer empty gaps between courses. Timetabling problems can be grouped into two main categories that are course timetabling and examination time tabling, both will be discussed in greater depth in the following section.

1.1.1 School timetabling

School timetabling is all about a regular school schedule, for example the schedule of the 12th grade in a certain school in Palestine or whatsoever. The school timetabling problem is a problem with a number of researches and studies done regarding it. However, the advance in it has not been that great; this is because these researches have been done independent of each other and because researchers in different places define the problem differently.

Due to the problem having multiple definitions, it can be generalized that this timetabling problem is one that is defined in terms of the number of grades, number of teachers, number of lessons that a single class is taught and the set of constraints put by the educational institution. The clear solution of this problem is the allocation of class, teacher and room tuples so that they satisfy the constraints of the problem.

1.1.2 Course timetabling

In course scheduling, the time period is of fixed length, most of the time it is one week. The goal here is to find a feasible weekly schedule that does not violate a set of constraints. One of the main constraints addressed in this type of scheduling is that the student cannot take more than one class period at the same time. In this type of problem, it is sometimes more helpful to split the problem into many sub problems, some of which are solved on the department level, while others are solved on the whole institution level. For example, one could start by decomposing the problem into sub problems like scheduling large university lectures (courses taken by large number of students in the educational institution), and then solve other sub problems like scheduling small lectures in small lecture rooms (department by department lectures).

1.1.3 Examination timetabling

Examination Timetabling deals with spreading a known number of exams over a known period of time (exam period) all of which are subject to certain constraints. In many cases, the main goal of this type of problem is to achieve a feasible schedule with no conflicts in the shortest period of time. In other cases, the goal is to come up with the “best schedule” that has some preferred characteristics, which may be:

- Maximize the period (gap) between two consecutive exams for students.
- Schedule exams with large number of students at the beginning to reduce the risk of conflicts and allow more time for grading
- Schedule harder exams at the beginning and so on.

Examination and Course scheduling can be both similar and different. For instance, in both of them, the student cannot be at two places at the same time, meaning he cannot take more than one exam at the same time nor can he attend more than one class at the same time. On the

other hand, course scheduling is always of fixed time periods, whereas examination scheduling is not.

1.2 Examination Timetabling

The following section will solely focus on Examination Timetabling, their constraints, types, optimality and complexity.

1.2.1 Hard and Soft Constraints

Examination timetabling problem (ETP) appears as a complex problem due to the number of variations it must deal with. ETP variables fall in of these sets:

- Exams set ($E=e_1, e_2,..$).
- Rooms Capacity set ($C = c1, c2,\ldots$).
- Available time-slots ($T = t1, t2, \ldots$).

In this type of problems, the project deals with different constraints that may overlap, which increases the complexity to find a solution. Mainly, constraints can be divided into two types: **hard** and **soft** constraints. Hard constraints must be satisfied and cannot be violated such as not allowing three exams in the same day or having one student take two exams at the same time. A timetable, which satisfies all the hard constraints, is considered a **feasible timetable**[1]. On the other hand, soft constraints are the desirable, but not necessary, satisfying them leads to an improved solution. For example, having two days before any exam is a soft constraint. In real life problems, it is about impossible to find a solution that satisfies both hard and soft constraints. Table 1-1 show some examples of hard and soft constraints.

Table 1-1: Examples of hard and soft constraints

Hard Constraints	Soft Constraints
The number of students assign to any exam must not be over the capacity of the exam room	Large exam should be at the beginning of the exams period to give student enough time to study
For any student, no two exams must be at the same time	Two days before any exams for all students

[1] A feasible timetable is one which satisfies all the hard constraints, and it is capable of being carried out and implemented since all exams are assigned time-slots.

1.2.2 Feasibility, Optimality, and Complexity

As mentioned before the solution is feasible if it satisfies all hard constraints. One of the metrics that distinguishes between different solutions is satisfying the soft constraints. Depending on that, an optimal solution is one that satisfies both hard and soft constraints. However, it is almost impossible to satisfy all the ETP soft constraints, because of the huge number of students and limited resources (i.e. limitation in time and rooms). Therefore, the best table is one that satisfies the maximum number of soft constraints, in addition to it's feasibility.

ETP problem can be considered as a complex problem because it has a large set of manipulated variables and constraints. The main issue is that the solution must assign a non-overlapping schedule to each student who takes different courses, which means the number of student, must be considered as a metric of complexity. In addition to that, time to find a solution is also a metric of complexity and this means that the solution must implement a good solving algorithm.

1.3 Problem Statement

This work focuses on solving a real-world examination-timetabling problem at Birzeit University (BZU). Since Birzeit University has a large number of students (approximately 13000) students, it must have an efficient program or algorithm to create schedules for all students. The main goal of this work is to make the timetabling process efficient, automated and takes into consideration the overall situation for students taking exams. Moreover, most timetabling problems focus only on the hard constraints, for example, in BZU not having three exams in the same day, this work will take into account other constraints "soft constraints". These constraints make the timetabling process more complicated but in return, it creates schedules that are more efficient for students and gives them the needed time.

1.4 Methodology

The following section will talk about the general methodology of the proposed work. The length of the examination periods are 12 days (in Birzeit University). Each week has 6 days (Saturday to Thursday). Each day has three timeslots (8.00 – 11.00, 11.00 – 2.00 and 2.00 – 5.00).

Table 1-2 shows the complete time-slot vector representation:

Table 1-2: Timeslot vector

SAT {1}	SUN {2}	MON {3}	TUE {4}	WED {5}	THUR {6}	SAT {8}	SUN {9}	MON {10}	TUE {11}	WED {12}	THUR {13}
{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}	{[8-11], [11-2], [2-5]}

Note that Thursday is represented by 6 and Saturday is represented by 8 so the difference by this operation can be found when calculating the spaces between exams, which is 2 days.

Room specifications must be taken from the database **[Room, Capacity]**. Each examination should be assigned to a single room, unless this cannot be avoided. In exceptional cases, i.e. if no room is available to accommodate the exam, then the exam can be assigned to multiple rooms, but the room location should be close to each other.

Birzeit University has some constraints that must be achieved in any schedule, which are called **hard constraints.** Table 1-3 show the hard constraints and soft constraints in Birzeit University:

Table 1-3: Hard and soft constraints in BZU

Hard constraints	Soft Constraints
The number of students assign to any exam must not be over the capacity of the exam room	Spread exams as even as possible
For any student, no two exams must be at the same time	Schedule largest exams, as early as possible
No more than two exams in the same day for any student	Number of students % room capacity nearly zero
No more than three exams in two consecutive days for any student	The exam that assigned to multiple rooms, its room location should be close to each other

The general algorithm described below, this algorithm ensures that the solution is improved if needed until an optimal solution is achieved and all constraints are satisfied:

- Identify all possible constraints and group them in the order of hard to soft constraints.

- Select the hardest constraint and apply solution.
- Check if constraint is satisfied.
- If constraints are satisfied, then repeat process for all other constraint, otherwise improve solution.

1.5 Report Outline

The rest of the report will be as follow: Related work in Chapter 2. Chapter 3 will present proposed work. Next, the report will show how the system was implemented in chapter 4. Finally, the last chapter will be a conclusion of the report.

Chapter II: Related Work

2.1 Examination Timetabling Solution Approaches

In order to solve ETP problem, researches used different algorithms such as constraint satisfaction problems (CSP), local search, graph based algorithms and population techniques. CSP attracted researcher because they can easily model it. In CSP, there are two main features, which are variables and values. The variables [2] were the exams and timeslots is the set of values [3] that can be assigned to the variables. Local search techniques depended on building a tree of the possible solutions and starting with an initial condition then the technique starts to search the neighbored to find the best solution that satisfies the problem's constraints. In this technique a good heuristic is needed to evaluate the neighbored.

Graph based algorithms add a significant improvement to solve the problem by make a graphical representation and heuristic to model the problem. It represents exams as vertices and edges that represent the hard constraints. To solve the problem, a color must assign to each vertices where no two adjusting have the same color. In this algorithm each solution is evaluated by the percentage of satisfying soft constraints. On the other hand, population based techniques mainly use genetic algorithm, which repeatedly chooses two applicable solutions from a set of solutions to produce two other solutions by crossing them over in order to find better solution. Moreover, researchers improve it by using local search to find the optimal local solution.

2.1.1 Constraint Based Techniques

Early research focused on finding feasible solutions, (i.e. satisfying all hard constraints). One of the papers that has been published in 1998[14] mentioned that partial solutions were first obtained since time complexity was crucial. Based on these partial solutions, particular local repairing strategies were employed successively to obtain complete solutions and make

[2] Variables are the entities that cannot be left without a value at the end of our solution and they represent the general entity. Whereas [3] Values are what get assigned to the variables to get to our final solution. For example, each exam (variable) must take a timeslot (value).

improvements. The approach was run several times with different initial assignments to reduce the chance of missing good solutions.

Another paper that obtained good results is **Merlot et al**[15], which was published in 2003. An optimization programming language (OPL) was used, then Simulated Annealing and hill climbing methods were added to improve the solutions. Variables (exams) were ordered by the sizes of their domains (available timeslots) and scheduled into the earliest timeslots, one by one.

2.1.2 Local Search Based Techniques

Local search is one of the approaches that can solve the timetabling problem. By forming a tree then finding a feasible solution. This is done by searching in neighborhoods to find the best solution that satisfies the soft constraints. Different local search techniques were used in order to solve it such as Tabu and simulated annealing.

Tabu search depends on keeping a list of previous moves so they will not be visited again. Therefore, in order to improve the solution it may move to worse solution, which means the algorithm will not be stuck in a local optimal solution. Di Gapero and Schaerf used this algorithm in 2001 to solve the problem[1].

2.1.3 Graph Based Techniques

A paper published by **Welsh and Powell** in 1967[16] focused on this kind of technique. To the regular human eye, probably one of the easiest ways to explain the timetabling problem is through showing it in a graph. This is what the research presented in the published paper. It did that through bridging graph coloring and timetabling altogether. The courses or exams were represented as nodes or **vertices** and the hard constrains between them was represented as edges between these vertices. This transformed the problem into a graph-coloring problem, where assigning different colors to adjacent nodes meant that hard constraints of our problem were met.

To explain the graph-coloring technique in more depth, have a look at figure 2-1. As seen, at the beginning, each node starts with all the available colors, where each color represents a time-slot in the exam period. Hard constraints are represented through the vertices between the nodes, and a solution exists if every adjacent nodes in the graph have different colors. The figure below shows the initial graph, a solution (the middle graph) and a conflict in the problem because two adjacent nodes (the red ones) have the same color.

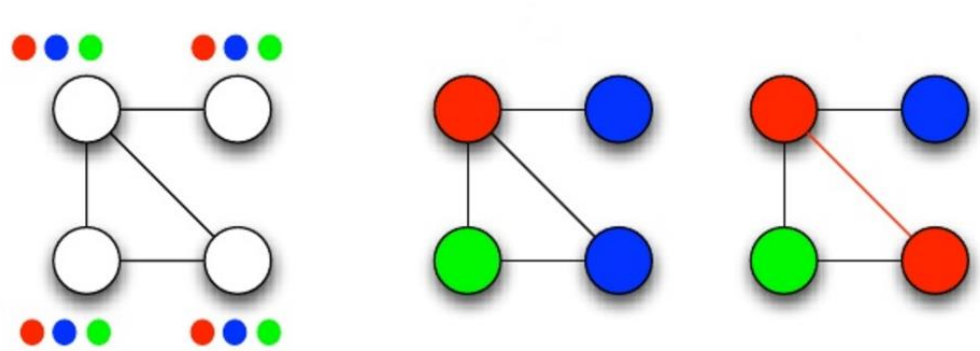

Figure 2-1: Graph colouring example

It was a bit difficult to Define and measure soft constraints using this technique. They were measured and evaluated outside the graph coloring and then they added as an additional feature or measure to the problem.

2.1.4 Population Based Techniques

Population based algorithms are algorithms are ones that maintain multiple assignments, in contrast with local search that maintain just a single assignment. Each assignment corresponds to a unique point in the search space of the problem. The general shape that this algorithm follows explained in figure 2-2:

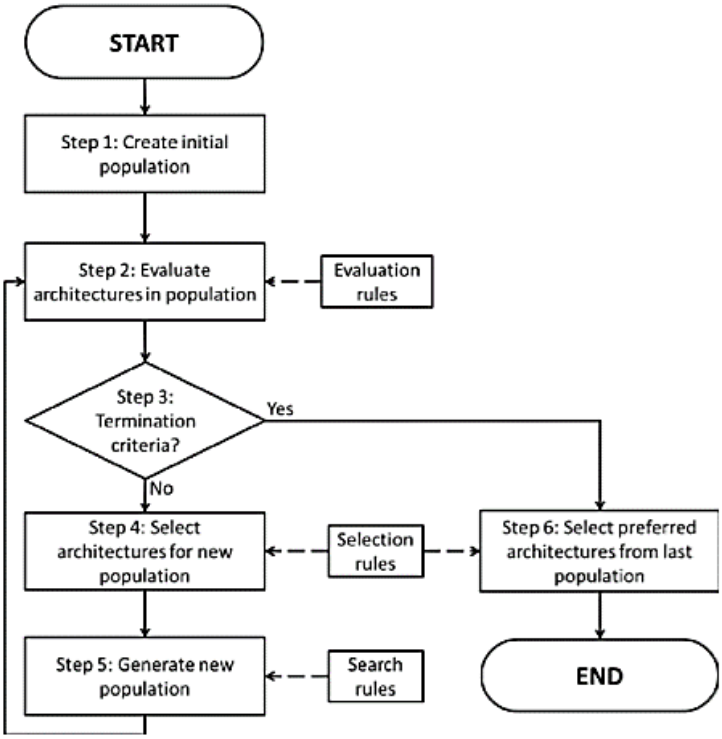

Figure 2-2: Population-based search algorithm developed for the framework

From figure 2-2, it is shown that first, an initial population is randomly created. Then the second step is to evaluate the population of architectures. After that, from the population of architectures, select a subnet using selection operators. The selection operators are defined according to the problem. Then, generate a new population by applying a set of operators to the selected architectures. The algorithm continues iteratively until the termination criteria is achieved. So now, the population matches the termination criteria, which means this is the requested population, the last step is to select preferred architectures from last population.

There are many algorithms fall under **Population Based Algorithms.** The first method, beam search, maintains the best *k* assignments. The next algorithm, stochastic beam search, in this algorithm, first select which assignments to propagate stochastically. In genetic algorithms, which are inspired by biological evolution, the *k* assignments forming a population interact in various ways to produce the new population. In these algorithms, a total assignment of a value to each variable is called an **individual** and the set of current individuals is a **population**.

Chapter III: Proposed System

This part will talk particularly about timetabling in Birzeit University depending on the University's and student's criteria. Techniques like graph coloring and local search were implemented and used in previous researches. In this work, the problem will be solved using the concept of Constraint Satisfaction Problem (CSP) with minimum remaining value beside heuristics to compare between problem variables and values.

One of the main disadvantages that have to do with CSP is special constraints need to be dealt with, which makes the process more complex and less time efficient. Moreover, this is the so-called NP-complete decision problem; therefore, it cannot be decided in a deterministic polynomial time. This makes it difficult to know if there is a given feasible solution or not. Despite these disadvantages, CSP shines in many aspects. Some of the main advantages is that "helper algorithms" can be applied to help with the complexity and the time of the problem. One of these ways is backtracking, which continuously verifies the validity of the restrictions directly after they are assigned. If any constraint is violated, assignments of variables are returned to the last valid instance that has another alternative assignment. Backtracking performs a depth-first search. The other technique used is Propagating Constraints, which is done by the process of propagation, which aims to reduce the search tree in a way that removes values that do not contribute to the solution.

3.1 Problem Formalization

In this part, the problem will be formalized using first order logic (FOL) in order to transform it to a mathematical form, which will help in illustrating the coding approach. Mainly, the following symbols will be used in formalization Birzeit timetabling problem:

- N is the number of course exams. I.e. variable that must assign value to it.
- E_i is i^{th} exam where $i \in \{1,\ldots,N\}$;
- S_i is the set of students who take i^{th} course.
- D is the number of days.
- R is the set of rooms.
- T_i is time-slot for exam E_i, where $T_i \in \{1,..,T\}$.

- R_i is room for exam E_i, where $R_i \in \{1,..,R\}$.
- $A_t = |T_i - T_j|$ is the timeslot different between exam E_i and E_j.
- C is the set of constraints that will be used in the problem (pools of constraints)
- C_{Si} is the i^{th} soft constraint, where $C_{si} \in$ {the pool of soft constraints}.
- C_{Hi} is the i^{th} hard constraint, where $C_{Hi} \in$ {the pool of hard constraints}.

After the formalization of the problem, the main ingredients of any CSP problem can be defined as follows, which are the *constraints*, *variables* and the *values*.

3.1.1 Constraints

The main goal of this problem is to come up with the best schedule for all the students in the University or at least one that gives them enough time to study (sub-optimal). The algorithm can always find the optimal solution for every single student in the University if unlimited resources were provided, meaning unlimited time, unlimited rooms and all the other resources needed for the exam period. Unfortunately, this is impossible and can never happen, so the solution considered will be the semi-optimal solution or the best solution possible. To do so a set of constraints needs to be followed, which can be split into two groups, hard constraint and soft constraint. Remember that hard constraints are the ones that need to be applied and soft constraints are the ones that would have an advantage or increased optimality if they were applied. Below are the hard and soft constraint for the problem from the University's point of view and the student's point of view.

Hard constraints (university point of view):

1- There is a prescribed exam interval, usually two or three weeks. All exams must hold on in this period.
2- All exams (i.e. variables) must be scheduled and each exam must has only one timeslot.
3- The capacity for any exam room must be around twice the number of students taking the exam.
4- There is only one exam for each course.

Soft constraints (university point of view):

1- Each exam should has minimum number of rooms that accommodate the course student to minimize the number of monitors.

2- Minimize the examination period.
3- Exams that are assigned to multiple rooms should be in the same building.

Hard constraints (student point of view):

1- Each student can have one exam only at a given slot.
2- Each Student can at most have two exams at one day.

Soft constraints (student point of view):

1- The average exams per day for exams period should be convergent.
2- Schedule largest exams (exams with largest number of students), as early as possible.
3- At most one exam per day for each student.

For the solution to be as feasible as possible, a survey was conducted and distributed to the students at Birzeit University. From the results of the survey, a pool of soft and hard constraint were used for the problem that would combine the constraints for both the university and for the students, the two pools are listed below.

Pool of hard constraints used

1- There is a prescribed exam interval, usually two or three weeks. All exams must hold on in this period.
2- All exams (i.e. variables) must be scheduled and each exam must has only one timeslot.
3- The capacity for any exam room must be around twice the number of students taking the exam.
4- There is only one exam for each course.
5- Each student can have one exam only at given slot.
6- Each Student at most can have two exams at one day.

Pool of soft constraints used

The pool of soft constraints is ordered depending on the priority of these constraints for the students and the university.

1- The average exams per day for exams period should be convergent.
2- At most one exam per day for each student.
3- Schedule largest exams (exams with largest number of students), as early as possible.
4- Exams that are assigned to multiple rooms should be in the same building.

3.1.2 Variables

The second ingredient of our CSP problem is the variable. The variable can be thought of as a node, which can be assigned to a certain value. This node or variable is linked with other variables using the predefined constraints of the problem. In our exam-timetabling problem, the courses will be considered as variables. Courses are linked together when they have one student taking these two courses, which means constraints between these two courses have to be verified. In the form of FOL that can be represented as the following:

$$\exists\ Student\ k, \qquad k\ \in (S_j)\ and\ k\ \in (S_q)$$

Where S_j is the set of students who take course (j) and S_q is the set of students who take course (q), meaning that the student (k) takes both courses so they are linked together. Moreover, a solution is defined when all variables are set to a certain value then the problem can be considered a feasible one.

3.1.3 Values

The final ingredient of our CSP problem is the value. This is what is assigned to each variable, having a variable left without any value means that the algorithm has reached a dead end and the problem cannot be solved. This shows the importance of choosing the right value for all the variables, which leads to each variable taking a unique value in the problem. In the timetabling problem, there was double values for each variable. This lead to each variable being assigned two unique values for each exam. The first value was the time-slot each exam was assigned, in our university, the exam period usually takes 12 days and each day has three finals exam slot. Each one of the variables would have to take one of these 36 times slots, and the time-slot needs to be unique to the students taking a certain exam. The second value that had to be assigned is the room, which the exam will be held in. This is because our problem does not depend on the time as the only resource for the problem, but in addition, the algorithm considers the room-slots for the exams as another constraining value since there is a need for enough rooms to conduct the exams. Reaching a variable left without any values, either time-slot values or room values means a dead end, and back propagation needs to be done to the last assignment made and choose another one that might lead to better solution path leading to all variables taking their right values.

3.1.4 The problem as a CSP graph

At the end of the day, any CSP problem can be seen as a graph of nodes (variables) linked together with some set of constrains and each one of these nodes takes a certain value (room and time-slot).

Representing the node (variable)

Figure 3-1 shows how the variable (course) will look in our problem, consider "A" a course in the university taken by a set of students.

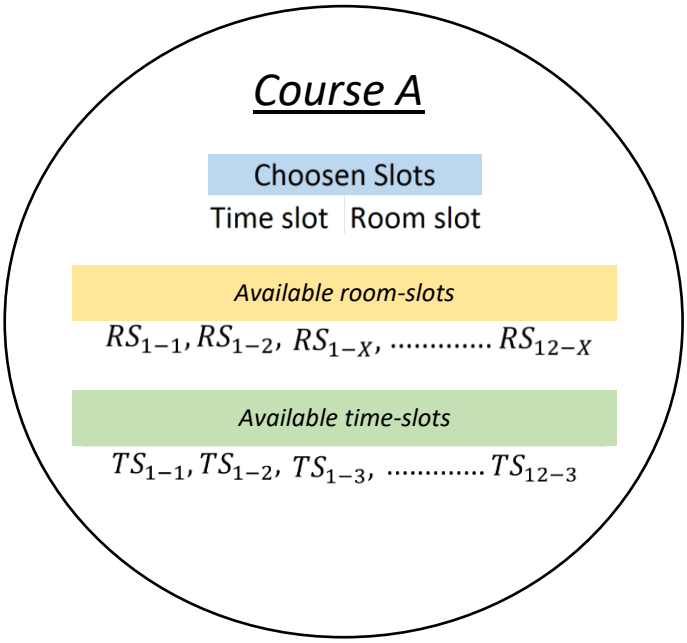

Figure 3-1: Visual repersentation of each node

For each course, one of the available time-slots is chosen and assigned, and then one of the available room-slots is chosen and assigned to the variable as well. To understand the representation more, consider the following notes:

- Time-slot TS_{x-y} means the y^{th} time-slot in the x^{th} day. An example would be TS_{1-1}, which means the first time-slot on the first day.

- Room-slot RS_{j-k} means the k^{th} room on the j^{th} day, since rooms are reserved for a day only, they can be re-reserved each day. An example would be, RS_{10-5}, which means the fifth room on the tenth day.

Representing the links between variables

Consider the following table (table 3-1) of courses and students that take these courses.

Table 3-1: Table of students taking certain courses

Student ID	Course ID
1	A
2	B
3	A
4	C
1	C
2	A
4	B

The best illustration of the problem can be in the form of a graph as shown in the figure 3-2.

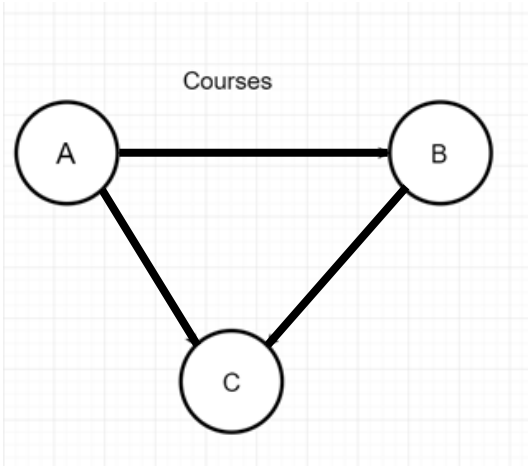

Figure 3-2: Example of the connection edges between vertices

The problem can be considered as graph connection problem where nodes or courses are connected to each other if they have one student who takes both of these courses meaning that the student can cause a conflict in our value assignment for the variable. In our example course

A is connected to course B since student 2 takes both courses A and B so, a link between A and B exists.

Representing the constraints

Constraints can be split into two groups represented in figure 3-3. Let us call the first type linking-constraint, which are constraint developed because of links between two courses (one student takes both courses). The other type of constraint are self-constraint and these are from the students who take this course in particular. Examples of self-constraint are making sure that a student has two exams maximum in a single day another example would be making sure that no student has more than 3 exams in two consecutive days.

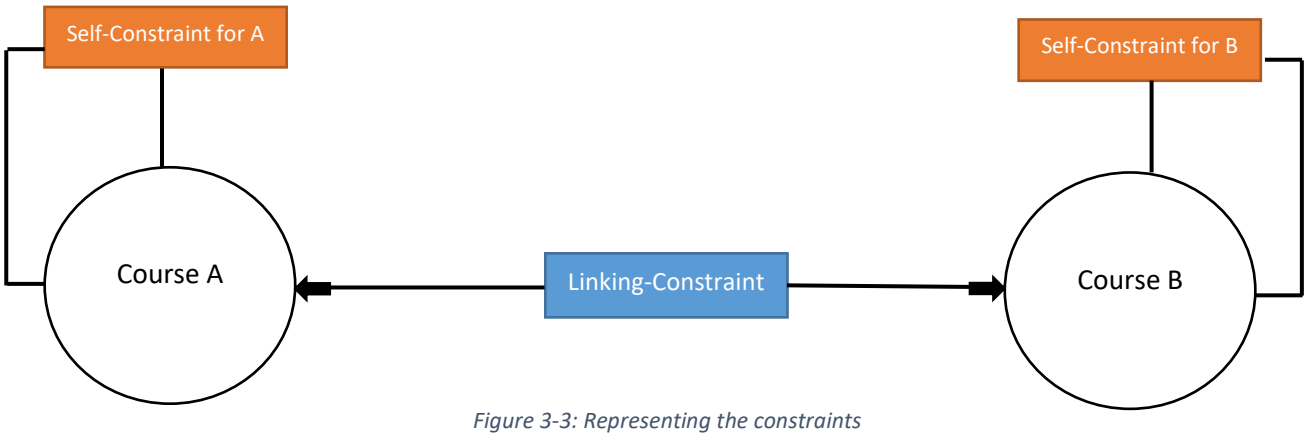

Figure 3-3: Representing the constraints

3.2 Search Strategies

This part of the report will talk about search strategies that are used to offer the best selection for both the variable (course) and the value (time-slot), more to follow on how the room selection was implemented and separated it into a subproblem in chapter 4. One could argue that since all the solutions are found or a huge number of them, why is a search strategy needed at all? This is because of the finite amount of time available and this will speed up the time to reach the best solution. Variable search strategy determines which course will get the chance to be assigned first depending on different measurements. On the other hand, value search strategies choose the best time-slot to assign to this exam (variable). Search strategies in this project aim to satisfy maximum number of soft constraints to find the best solution since

there needs to be a validation system that ensures that all the solutions are always valid hard-constraint wise. Search strategies rely on giving variables and values that satisfy the maximum number of soft constraints the highest score.

3.2.1 Choosing the Variable (Course Selection)

Having a big number of variables in any CSP problem can make the search graph huge, as an example in Birzeit university having over 700 courses that need to be assigned into 36 times slots leads to a search graph that's interlocked and impossible to traverse, and even if one was to take one millionth of all the possible solutions this number would still be significantly huge. Therefore, having a search strategy leads to good results that are both efficient and quick. This search strategy might not go over all the solutions, but it guarantees that a **good solution**[4] is found.

The variable search strategy depended on two factors or two scores. A static score that's given one time at the beginning of the strategy and another Dynamic score that's updated as a program continues to run. The static score builds upon two factors which were the **intersection factor**[5] and the number of students who take this course. The dynamic score was updated depending on the number of remaining time-slots available for a certain variable, meaning that the minimum remaining value property would be satisfied. So, a score that has one time-slot available left would have a high exponential score guaranteeing that it will not be left out with no time-slots to be assigned to it. At the beginning of the program and since all the variables had equal time-slots to be distributed to them, they all had the dynamic score of zero and the score that could distinguish between them was the static score. The flowchart in figure showcases the variable selection process.

[4] A good solution here is one that commits to all the hard constraints and maximizes the soft ones. It also means that it has the highest possible score value of all discovered solutions.
[5] The Intersection Factor is how linked this note is in the CSP graph with other nodes or courses.

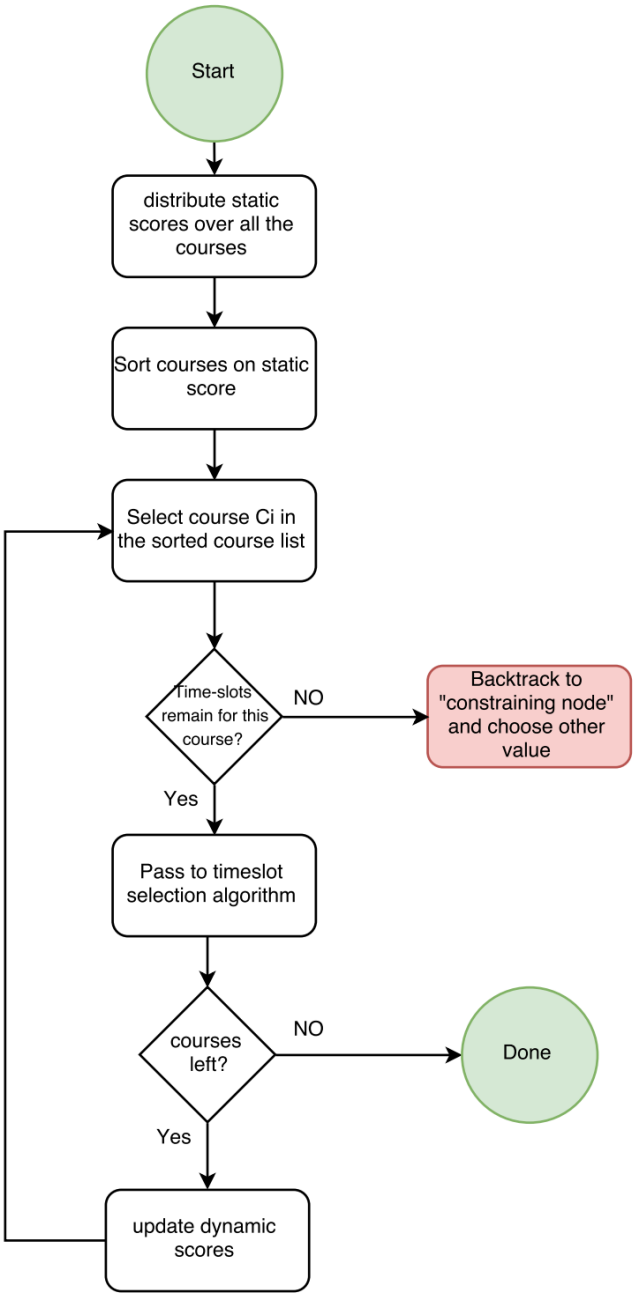

Figure 3-4: Variable Selection flowchart

3.2.2 Choosing the Value (Time-slot Selection)

The value selection strategy seems less important than the variable selection strategy when in fact it is more important. Having a huge graph with thousands of nodes connected can be altered by the slightest modification meaning selecting time-slot A instead of B could change the output of the system greatly (even though A and B are right after each other). Therefore,

having an efficient and effective tim-slot selection strategy was important. Moreover, improper time-slot selection meant that the algorithm was removing values from other nodes and leaving them without time-slots to be selected for them, which raised the probability of backtracking and this led to inefficiency and an increase in the time needed to reach a solution.

Thus, choosing the perfect time-slot for a certain variable meant that the program must be careful with other neighboring variables, so this would not affect the system in a big way. To prevent such a situation, the algorithm starts with the value that minimizes the number of removed time-slots for other variables (the neighboring variables linked with this variable in the CSP graph), this led to a reduction in the number of backtracking occurrences since this opened up the opportunity for time-slots to be left up for grabs. Moreover, to find the best solution the algorithm finds the value that will maximize the average space between exams for students taking this course. The average space is the average amount of time (days) between two exams and relies on reducing the occurrence of having two exams in the same day and spacing out the exams as much as possible. This implied that the system would have to filter out the time-slots and sort them depending on "some score" that would be calculated using the two previous points and take this score as our sorting mechanism to know which time-slot works best for this course and for the students taking this course. The following two factors would select the best value for any variable:

1- The number of time-slots that would be affected (removed) from other linked variables due to the selection of a time-slot.
2- The average space between exams that would be left for students due to the selection of a time-slot. The average space was affected by the following properties:
    A- Two in Four occurrences.
    B- Back to Back exams, which are exams that occur right after each other for any given student.
    C- Overall spacing of exams.

3.3 Methodology and procedure

This section will talk about the algorithms that are generally implemented in the CSP problem and how they relate to the project. It talks about two ways for the algorithm, one of them takes less time but its feasibility is not that great. On the other hand, the second enhanced algorithm uses arc consistency to increase the feasibility, but takes more time and space.

3.3.1 First Proposed Algorithm

The first proposed algorithm depends on assigning the value of the variable without any additional add-ons or calculations (Arc consistency). The goal of this algorithm is to make the **calculation** or variable assignment **quick** and it does that since there is no arc consistency calculation to go through many nodes in the graph. The first algorithm runs as following:

1- Courses are defined as variables for problem and both timeslot and rooms as values that the exam should have.
2- Running CSP with most constringing variable, which is the course. For all courses (variable), each course has a score that referring the order of assigning values for variables. In the previous section, the heuristic of how to calculate the score were shown. The algorithm sorts the courses depending on this score so that the most constraining course is chosen first to be assigned.
3- Choose the best value for the variable depending on value heuristic.
1- If the algorithm reaches a variable that has no available time-slots, then back track until the variable get at least one available value.
2- If all courses have values then the solution is found, if not, choose the next variable depending on variable heuristic then go to step 3.

3.3.2 Second Proposed Algorithm (Enhanced CSP with Arc Consistency)

The second enhanced algorithm, which uses arc consistency, runs as follows:

1- Define the courses as variables for problem and both timeslot and rooms as values that the exam should have.
2- Running CSP with most constringing variable, which is the course. For all courses (variable), each course has a score that referring the order of assigning values for

variables. In the previous section, the heuristic for the score was shown. The algorithm sorts the courses depending in this score so the most constraining course is chosen first to be assigned.

3- After choosing the exam (i.e. variable), a value for it needs to be picked. To find the best value students, the algorithm do the following:
    A. Each variable has a set of all available timeslot for this exam. The available time exam, which is the timeslots that satisfies the hard, constrains.
    B. If there is no available timeslots, the algorithm should backtrack one-step, and remove the previous assigned value for the previous variable and choose the second best value. Because the best value makes the solution unfeasible.
    C. Each available timeslot has a score depending on the value measurement (previous section). The slot that has the maximum score is assigned to the chosen variable.
4- After choosing the value for the variable, the available values for all variables are updated (i.e. update all available time-slots for courses) and if there is a variable that has only one available timeslot when assign it directly. In addition, the dynamic score is updated for each variable.
5- If a course has an empty set of available time-slot, backtrack one-step, remove the value of the previous variable and assign a new value for it after remove the value that make a conflict from available timeslots.
6- In order to assign value for the room, find all the available rooms, which are not reserved at this time. Then find the best fit for this corresponding course, which is around twice the number of students taking the exam. More details in room heuristic.
7- For each student that has this course the exam schedule needs to be updated.

3.3.3 Flowchart for the algorithm

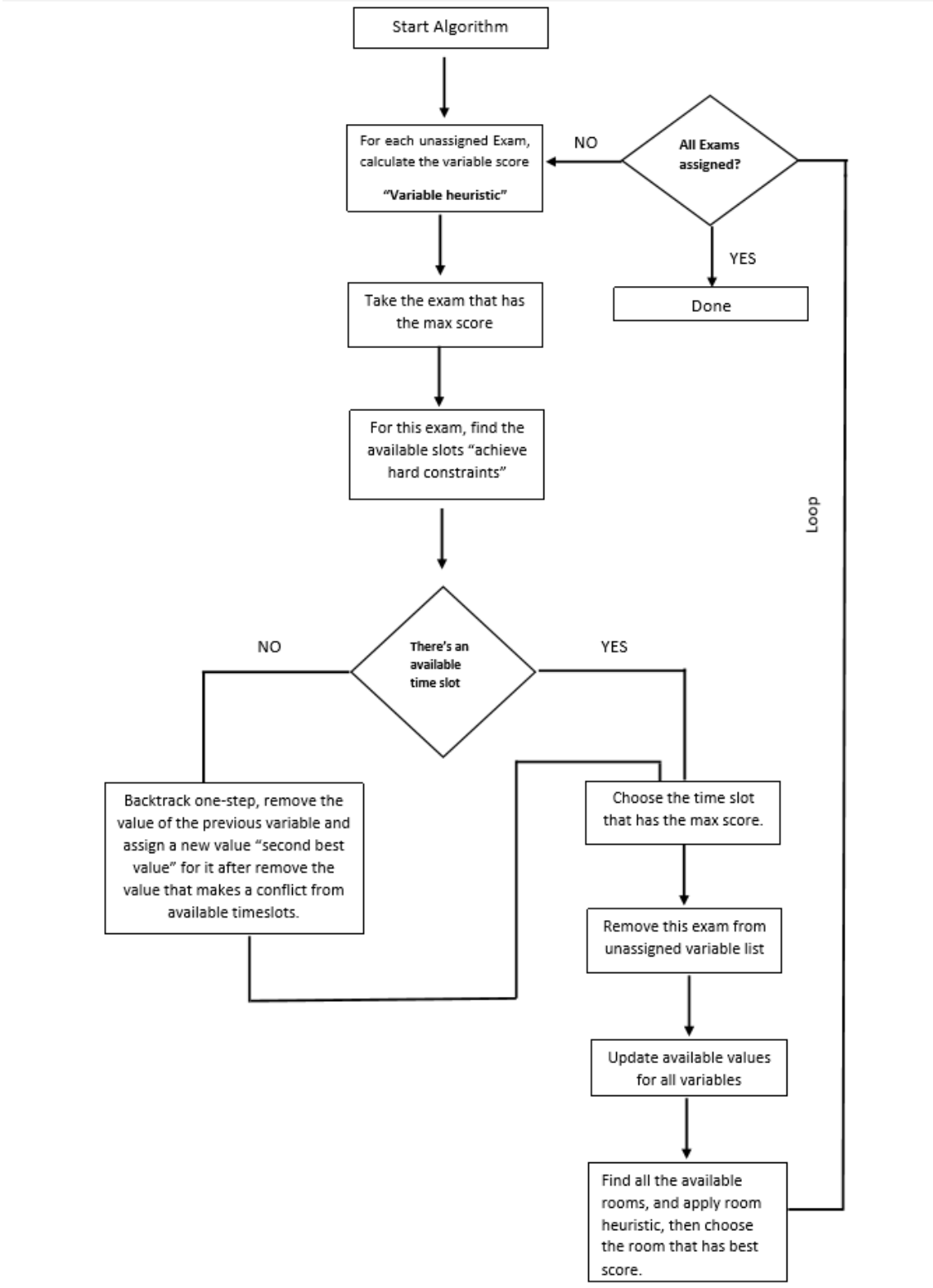

Figure 3-5: Flowchart for the algorithm with arc consistency

Chapter IV: System Implementation

This chapter will talk about the system implementation including the dataset that was used for the project, the programming language and solver used, the modelling of the problem and the results achieved.

4.1 Dataset

The dataset that was used in the project was provided by the computer center at Birzeit University. This dataset consists of over 1,000 courses taken by over 12,000 students. The dataset was provided as tables, which was transformed into a relational database. The database consists of:

1. Course table: This table contains all information about the courses such as: Id, course label, course title, department that responsible about course.
2. Room table: This table was used in the rooming system, which contains information about room such us: capacity and location
3. Student table: This table holds student's data such as: id, department and the current year level.
4. Student_Course table: Which was created to maintain the relations between courses and students since the relation between them is many to many.
5. University solution table: This table was used to compare between the project and the university's results.
6. Project Solution: This table stores the final solution for the project.

4.2 Programming language

The algorithm for the problem was explained in Chapter 3. Constraint programming is a problem-solving technique that works by incorporating constraints into a programming environment. CSP is NP complete problem that is solved in an exponential time, in addition, the huge number of possible variables and values make the time a major factor in this problem. Java language seems to have many advantages compared with other languages, for example, it supports Object-Oriented (OOP), and java is faster than traditional interpretation. In addition to the excellent background and documentation provided for JAVA.

Constraint Programming (CP) is a programming paradigm, which provides useful tools to model and efficiently solve constraint satisfaction and optimization problems. For this problem, Choco solver will be used. Choco is a Free Open-Source Java library dedicated to Constraint Programming. The user models its problem in a declarative way by stating the set of constraints that need to be satisfied in every solution. Then, the problem is solved by alternating constraint filtering algorithms with a search mechanism. Moreover, Choco allows for huge customization where the user could implement his own set of values, variables and even constraints. Some of the advantages of using Choco are:

1- Allows for easy modelling of the problem
2- Easy to tweak
3- Uses state-of-the-art strategies.
4- Choco is among the fastest CP solvers on the market

4.3 Modelling the problem using Choco

The key component of using Choco solver is the Model component. One could think of it as the problem or the "base" for the problem where highly-configurable layers can be added such as the variables and constraints. It is the cornerstone to the Choco solver and it should be the first step to define this Model. Afterwards, the variables, values and constraints can all be then defined. It's very important to mention that Choco is NOT specialized to be a university scheduling solver, where as other solvers are specialized for this aspect solely. But the advantages Choco provides were bigger, even though that meant the programmer needed to translate the problem into one that Choco can understand, that's the reason Choco was chosen over the all the other specialized solvers.

4.3.1 Modelling the Variables

In Choco solver, or any other CSP solver, variables need to be added to the model for the problem to be a real CSP problem, after all that's what a CSP problem is, a bunch of variables dueling over a set of values while being attached to some constraints. The data set previously mentioned was used to define these variables then add them to the Model one by one. Each variable was saved uniquely by the course label it refers to, this way duplicate variables do not exist.

There are many types of variables such as integer variables, boolean variables and real variables, basically they are all the same they just differ in the values they take and in this problem the chosen type of variables was the integer variable since all the variables would be assigned integer values that refer to certain time-slots.

4.3.2 Modelling the Values

In Choco, whenever a variable is defined, the set of allowed values it can take need to be attached to it in the definition statement. These values can be separate integers or a set of incremental numbers from a starting point to an endpoint. In this problem the set of integers, since integer variables are used, from 1 to the last predefined time-slot (usually the university assigns the number of days for the examination period) were used to represent the time-slots. Meaning that one corresponds to the first time-slot in the first day, and for example 3 corresponds to the third time-slot in the first day and 12 corresponds to the third and final time-slot in the fourth day. If the set of integers allowed was not assigned, then Choco finds the least possible number of time-slots to achieve the solution.

Integer values were chosen to replace days since it's much easier to do mathematical operations on numbers and then map them to slots and days whereas using predefined days and dates meant that the system cannot be modular, nor can it be general. Moreover, using integer values instead of days and dates helps keep track of off days and neglect them, so far example if Friday was an off day for finals exams and 6 was the last time-slot on Thursday then 7 would be the first time-slot on the next available finals exams day (either Saturday or Sunday depending on what the university preferred).

4.3.3 Modelling the Constraints

This was the toughest part of the modeling process. Choco is not an examination timetabling solver which meant that it does not offer the predefined features and constraints of other solvers. The programmer must model his own constraints, some of them are simple while others are really tough, and they have to be designed from scratch so that Choco can understand them. An example of an easy constraint that can be implemented in Choco is the hard constraint that does not allow for a student to have two exams in the same time-slot. This was done by

attaching the **AllDifferent**[6] constraint to any two exams that had common students in them this can be seen in the CSP graph as the vertices between any two courses.

The other type of constraint, was the second hard constraint, which does not allow a student to have three exams in a single day. Choco does not understand this constraint as it is not an examination solver, so it cannot be injected into the solver as simple input rather it needs to be designed by the programmer. This constraint was designed by implementing the constraint class and manually defining the constraint and designing the propagator that follows this constraint. For the second hard constraint, a ThreeInOne constraint class was created for this purpose. This class takes three courses that a student takes as input and adds a constraint to the model that these three cannot have the same value. To achieve this for any given student, the system needs to find all the possible combinations of length three (three exams) and add them to the model as a constraint. For an example, if a student takes the following set of courses {A, B, C, D} then the system would add three constraints being:

1- Constraint on {A, B, C}
2- Constraint on {A, B, D}
3- Constraint on {B, C, D}

4.4 Implementing the system with Choco

After the overall modeling of the system was defined as mentioned previously, it was time to implement the search strategies defined in chapter 3, where Choco allows for a value selector and a variable selector to be injected into the model. The goal of these two selectors is to choose the best variable then choose the best value to be assigned to this variable just as described in both strategies. Afterwards, the problem can be solved. Choco provides all the possible solution for all the variables within the given constraints, this meant that one could see the output of every single solution if the computing power was there and so was the patience as these solutions tend to take time to be visible to the user due to the huge computational power done to achieve the solution. After the solution was received from the solver, the output needed

[6] AllDifferent is predefined in Choco and takes the set of variables (exams) that the programmer wants to enforce this constraint on as input.

to be passed on to three subsystems that were designed for comparing and validating the solution. They can be summed into the three subsystems below:

1- The rooming system. Which is a subsystem that distributes the exams among the set of available rooms.

2- The validation system. Which is a system to make sure that all the time and room-slot assignments that were made did not violate any hard constraints.

3- The scoring system. The purpose of the system was to know when a better solution was reached. This helped know when to stop looking for more solutions by putting a threshold of some value.

The overall run of the system can be summarized in figure 4.1 below.

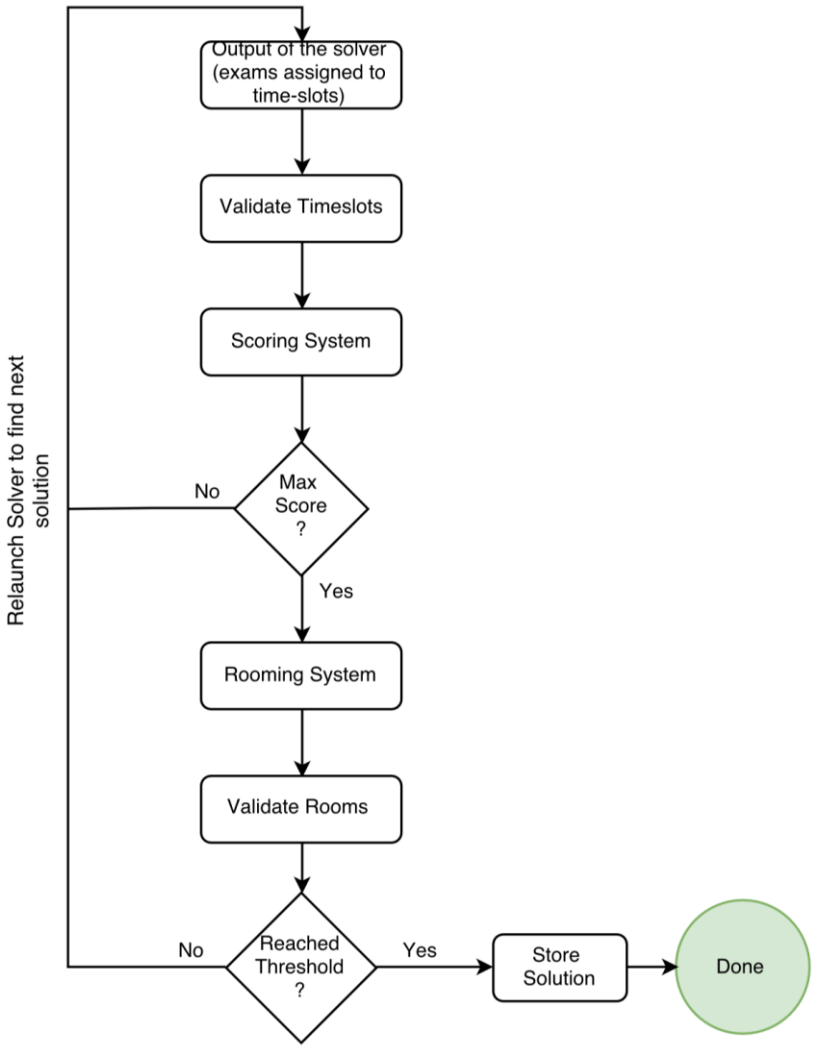

Figure 4-1: Flowchart for the overall run of the system

4.4.1 Rooming system

The last component in the constraint problem is the rooming system. As mentioned in the previous chapter, the rooming system was designed as a subsystem and not implemented in the original system to make it easier and quicker to achieve Solutions. It is impossible to add room-related constraints, since one does not know the constraints on the rooms based on the exams. So, the output of the solver was basically all the exams with their given time-slots now rooms for these exams needed to be found.

This system is responsible for finding the best and efficient distribution for exam rooms among different time-slots. After checking the validity of the solution in the time-slot validation system (i.e. satisfy all hard constraints), the rooming system also adds one more constraint which has to be checked . A valid solution in this system is the solution that allows exams to be held in rooms with twice the capacity of the student count (number of students taking a certain exam) or as the university defines the ratio to be.

The rooming system algorithm is summarized in figure 4.2 on the page 31.

4.4.2 Validation System

Looking at the final solution, the visible output to the regular user is thousands and thousands of records in the database, but there's more than one solution! There are millions of solutions. Because of that, it's impossible to check the solutions or even just one solution manually. The validation system looks at the constraints as filters or gates and solution can be a valid solution only if it passes all these filters. The filters are grouped into the two groups which are time selection validation and room selection validation.

Time Selection Validation

Time selection validation is achieved by making sure that the pool of hard constraints is achieved for every single course and for every single student. Meaning not a single student can have two exams in a single time-slot and nor can he have three exams in one day. In addition, making sure that every course is assigned to a single time-slot is essential.

Room Selection Validation:

Room selection validation is achieved by making sure that the set of predefined conditions related to the rooms are all achieved. That means that every exam needs to be

assigned into one or more rooms. Moreover, the system makes sure duplicate room assignments (two exams having the same room) are not present.

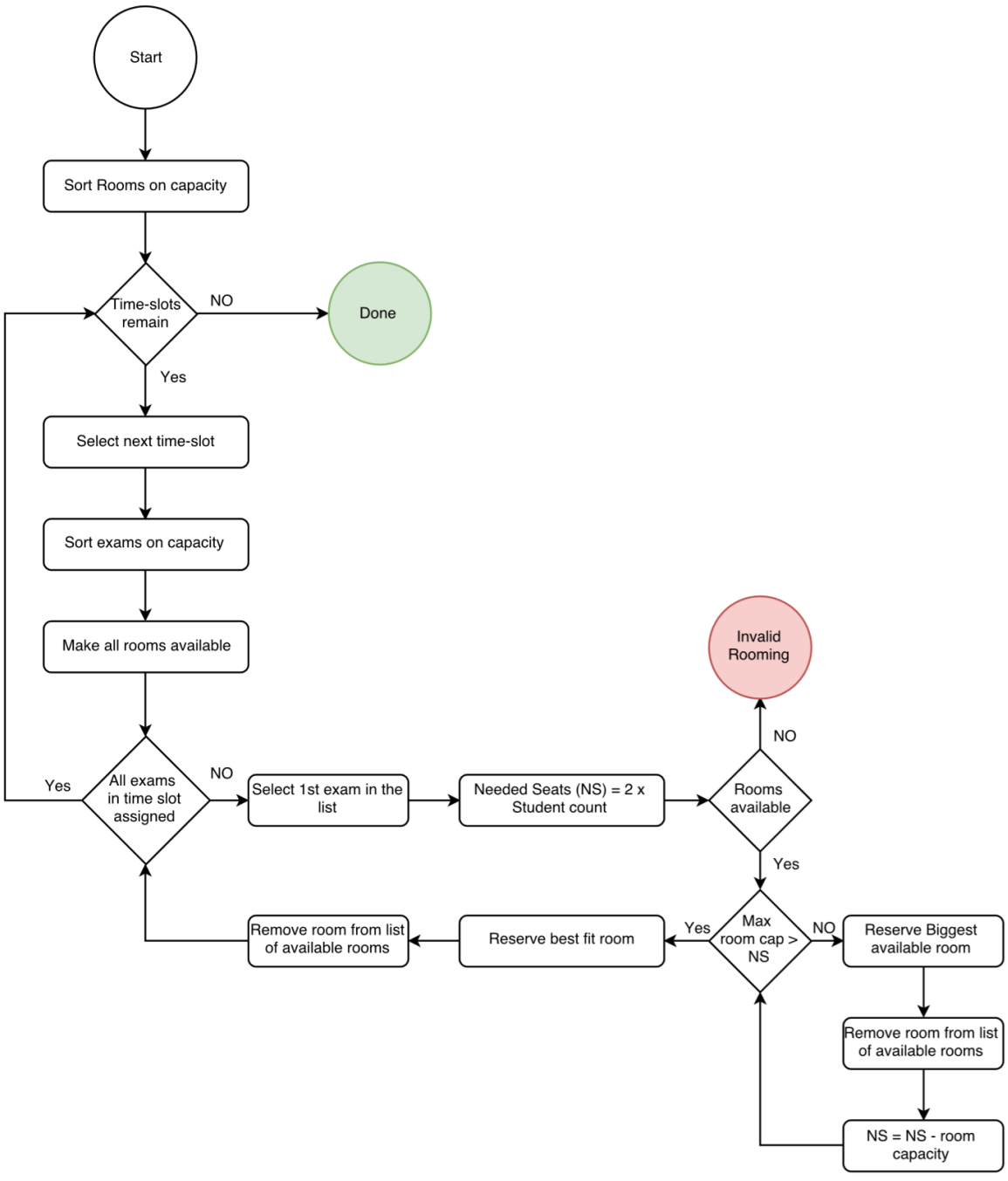

Figure 4-2: Flowchart for the rooming system

4.4.3 Scoring system

The number of solutions in the problem are very huge so there needed to be a way to distinguish between all these solutions and answer the question, which solution is better than the other? For that, a scoring system needed to be implemented. In the value and variable selection algorithms, one can guarantee that a good selection is made, but that selection is not always the best selection, or it might not lead to a maximization in the scoring criterions. Therefore, a scoring system needed to be developed, to ensure that solutions are always compared with each other and with other solutions from other possible systems (Birzeit examination system) based on standardized criterions and then choose the highest scoring one. In addition, it is needed to mention that the scoring system needs to be developed in an environment that's isolated from any other system or any other solutions in order to guarantee that it is always a fair scoring system.

The scoring system emulates a student. The student wants to get the greatest number of hours to study for any given exam, so the most important factor is spreading the exams over the interval of exams period. The scoring system takes in consideration these factors:

1- The more spaces between exams that a solution provides, the greater it's score is. This is until a certain threshold, where after that threshold the increase of the score becomes less by a factor of 1/x.
2- The ratio of consecutive exams to the total number of exams, as it increases, the score decreases.
3- Back to Back exams, as having exams in the same day or in consecutive time-slots reduces the score.

4.5 Experiment and Results

This section will present the output/results of the system. These results were obtained using the dataset that was provided by Birzeit University, the results were then compared to the solution achieved by Birzeit. It is important to point out that the comparison is conducted after the whole system (with its subsystems) is finished. This way, the program does not just stop when it finds a solution that it thinks is better than the university's, rather, it achieves the best solution possible using the algorithms and then compares it to the one the university uses. The fact that the compared solution has better results than the university's is connected to the

deployment of the algorithms into the system and not due to some stopping criteria (reaching a solution that's better than the university's).

There are many aspects that one could use to compare between different solutions, the system's solutions with each other or even comparing them to the results of other systems like the one used by Birzeit University, but some main aspects were taken into account depending on the logical interpretation of the problem and based on a survey conducted in Birzeit University that aimed to understand what the demands of students were when it comes to their finals exams schedule, and they can be summarized as follows:

1- The average space between exams needs to be as big as possible.
2- Reduce the occurrence of having two exams in one day to a minimum.
3- Avoid exams that happen back-to-back, which are exams that come in neighbouring time-slots and that includes exams at last slot in the day followed by exams at the start of the next day.
4- Avoid four-in-two occurrences. In some cases, some students were taking four exams in two days and that meant they had to take all their exams (or most of them) in 48 hours or less.

The previous four points were derived from the survey that was conducted in the University. They all basically aim to spread the exams as evenly as possible throughout the exam period. On the other hand, other aspects were taken into consideration when comparing the results with each other. One of those main aspect was the variance of average space between exams. Everyone agrees that a high average space between exams is a good thing, but that's not always the case as the average was not the only important measure here, but rather the variance was also a significant measure as well. There is no good in a high average if its variance is high, this would mean that the values are mostly either in the highs or lows. Continuing with the aspects, the required amount of rooms and chairs used for the exams were taken into consideration, but their weight was rather smaller than the other aspect as rooms and chairs are often available and they do not form a high constraint on the solution.

To understand the main differences between the output of the system and the solution used by the university, let us first look at the university's system and see how it works.

The university uses a program dedicated towards examination timetabling which is a good thing but it's not a great thing. Using a specialized program limits the customization and modularity of the program and that's what seems to happen with their solution. The program they use is called UniTime which is a comprehensive educational scheduling system that supports developing course and exam timetables. The program allows for inputs to be the list of students currently enrolled in the set of courses, and the list of available rooms. It also allows for a set of constraints to be defined and given their respective weights. Returning to the way the university treats the problem, they do not define any hard constraints, instead, they use a weighting system that sometimes must compromise and break the hard constraints, yes that's true, they break some of the hard constraints and that explains why 18 cases of students taking three exams in a day or even to exams at one time-slot were found in the dataset given by the university. They start off with high initial weights for the hard constraints, but during the solving process these weights are updated and sometimes they are updated in a manner that favors soft constraints over the hard constraints leading to such cases.

What's more surprising, is that they start with an initial solution and let the program look for a neighboring solution near this initial one. This means if the choice of initial solution was poor the program could look for a neighboring solution that has somewhat the same characteristics, leading to endless search time going in the path of nowhere. Moreover, they have a time limit cap for running the program, which is around five hours which is astonishing since there is billions and billions of solutions and five days are not enough to go through all of them let alone five hours.

Pointing out the flaws the university solution does not mean in any way that the solution of the implemented system is perfect or error-free. Nor does it mean that the implemented system has the capacity to go over all the solutions which would need huge computational power and an endless amount of time, rather the whole goal of this problem is to create an effective search strategy that reaches the best available searchable solution in the endless search graph. To defeat the problem of falling in a local maximum, a restart strategy was used with the solver. This strategy made the solver span wider in the search graph and avoid spending endless useless hours in neighboring solutions that have the same quality.

To compare numerically between the two solutions, algorithms to compute the average space between exams with its variance and count the back-to-back and four-in-two occurrences were developed. Let's first start with the most important aspect and that's how evenly spread the exams were in the schedules. The average space between exams of the University's solution was 1.22 days, while on the other hand, the system developed reached an average space of 1.4 days. One might think of this as a small increase, but as a matter of fact it's huge. The average student has six exams and the examination period is usually twelve days, so no matter how spaced-out that exams are a 0.18 day/exam increase is a huge one. When looking at this number from a more general angle, it is easy to see why it increased the total number of study hours for all the students by a quarter million hour and that's important time that the students can use to improve their finals exams achievements. In addition, the variance was previously mentioned to be very important along with the average. The system implemented saw a decrease of 26.2% of the university's original variance and this means that not only does it have a higher average, but also that the average is more realistic and fairly distributed among students. Table 4.1 compares between the two solutions using the most important measures that are of great value to both the students and the university.

Table 4-1: Comparing between Birzeit University's solution and the one obtained by the implemented system

Measure / System	Birzeit University System	Implemented System
Solution is valid?	**No**. 18 cases that violate hard constraints were found	Solution is **always valid**
Average space between exams	1.22 days/exam	1.4 days/exam
Variance of the space	2.17	1.6
Four-in-two count	9	40
Back-to-back count	3500	4000
Number of rooms used	1215	1219
Average number of chairs between students	1.25 chairs between each two students	2.25 chairs between each two students
Minimum number of chairs used between students	0 chair between each two students	1 chairs between each two students
Number of excess chairs[7]	14779 chairs	6098 chairs

To get rid of the subjectivity and make the comparison as fair as possible, a survey was conducted to help select which was the better solution. The survey was hosted on a public domain and allowed all students at Birzeit University to submit their responses. The survey shows the finals schedule for a single student (the same student) using the university solution and the solution from the system implemented. It allows contributors to select which schedule, in their opinion, is better than the other. The survey randomly selects 1 out of 10000 possible schedules. The survey obtained around 2500 votes and the results are shown in Table 4-2 on the next page.

[7] Excess chairs are chairs that offered more than 1 chair between each two student, this is to show the number of chairs that were wasted and were not needed.

Table 4-2: The results of the survey conducted regarding which system is better

System	Number of Votes
Birzeit University solution	1007
Implemented system solution	1538

This meant that the implement system solution received around 66% of the votes, making it more likely to appeal to the demands of students and their study needs.

Chapter V: Conclusion and Future Work

Time scheduling is an important factor in everyday tasks, and one of the most important schedules out there are examination schedules or timetables. This project discusses the main factors that lead to a fair and evenly distributed schedule. In addition, it relies on constraint satisfaction problem in reaching the target goal. The project shows how to utilize the three main ingredients of CSP, which are the variable, value and the constraints, in order to give the exams the appropriate time-slots which leads to the best schedule possible. Moreover, enhancements were added to the CSP algorithm to reduce the space and time used by the algorithm. One of these helper algorithms was Arc Consistency, which makes sure that there is an available time-slot for all the variables out there, by back propagating as soon as a variable becomes empty valued. It also showcases the many aspects and advantages of using a CSP solver like Choco Solver, which simplifies the problem and allow for high modularity and customization. A project like this helps the students receive the required time they need study for their finals exams so that their grade is not decided in a matter of two days when it can be decided in five or more days in a better distributed schedule. Also, having a less compact schedule reduces the psychological pressure on the students undergoing the exams at their university, since having more time to study makes the students more relaxed.

As for future work, the A-star algorithm will be implemented trees the best available room-slots distribution system. The system will be tested using a bigger dataset, meaning more variables which will need lots of additional computational power and this is where a chance for the search strategy to shine will appear. Moreover, threads will be used to turn this single thread problem into a multi-thread one meaning action in time and this allows for bigger traversal of the search graph. In addition, the system will be implemented with adding difficulty level to the variable scoring algorithm.

References

1. *Schaerf, A. A survey of automated timetabling. Amsterdam: Centrum voor Wiskunde en Informatica, 1995.*

2. *Marie-Sainte Souad Larabi An Automatic IT Examination Timetable Tool. 2013*

3. *Reis Luís Paulo, Oliveira Eugénio, A Language for Specifying Complete Timetabling Problems. 2000*

4. *Pillay Nelishia, survey of school timetabling research, 2013*

5. *Burke E. K and Lee S.Y, Automated System Development for Examination Timetabling, 2004*

6. *Moreira José Joaquim, A System for Automatic Construction of Exam Timetable Using Genetic Algorithms, 2008*

7. *Chunbao Zhu and Nu Tha, An Intelligent, Interactive & Efficient Exam Scheduling System (IIEESS v1.0), 2012*

8. *Qu R., Burke E. K., McCollum B., Merlot L.T.G. and Lee S.Y., A Survey of Search Methodologies and Automated System Development for Examination Timetabling, 2009*

9. *Timetabling Problems , CALIFORNIA STATE UNIVERSITY LONG BEACH, http://web.csulb.edu/~obenli/DSS/node2.html (Retrieved April 14, 2017)*

10. Final Exam Scheduling Timetable a Case Study, Hebron University http://elearning.hebron.edu/EPortfolio/artefact/file/download.php?file=2216&view=4 (Retrieved April 15, 2017)

11. http://www.unitime.org/papers/patat07.pdf *(Retrieved April 15, 2017)*

12. *Murray Keith, M¨uller Tom´a˘s and Rudov´a Hana, Modeling and Solution of a Complex University Course Timetabling Problem, (2007).*

13. *Comprehensive University Timetabling System, University Timetabling Comprehensive Academic Scheduling Solutions, http://www.unitime.org/uct_description.php (Retrieved May 2 2017).*

14. *P. David. A constraint-based approach for examination timetabling using local repair techniques. In: E.K. Burke and M.W. Carter (eds) (1998). Practice and Theory of Automated Timetabling: Selected Papers from the 2nd International Conference. Lecture Notes in Computer Science, vol. 1408. 169-186.*

15. *R. Miles (1975). Computer timetabling: A bibliography. British Journal of Educational Technology. 6(3): 16-20.*

16. *D.J.A. Welsh and M.B. Powell (1967). The upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal. 11: 41-47.*

17. http://www.unitime.org/index.php?tab=0 *(Retrieved Jan 1, 2018)*