Chapter 2 The Generalisation and Solving of Timetable Scheduling Problems

Colin D. Green

2.1. Introduction

Timetabling at present is still very much a manual process, usually involving a draught timetable which is modified over a period of time as problems and limitations are uncovered. The final timetable from this method is often acceptable to the people who have to follow the timetable, however, much time and effort is spent and some improvements may still be possible.[10]

2.1.1 Aims

With this project I hope to create an application on a PC which will allow the user to build a model of their timetabling problem, the application will then use the model to produce a usable timetable.

Timetable schedules are used to co-ordinate use of resources in many areas, some of these are;

- Transport timetables; Buses, trains, planes, etc.
- Manufacturing; Planning the use of resources in a manufacturing environment, be it machines, raw material and/or personnel.
- University, School timetables etc.

In this project I will attempt to look into as many types of timetable problem as possible and build a general model capable of representing each of them.

I will then be looking at known techniques used to solve or at least, create suitable solutions to the problems and assessing the relevance of each technique to the general model I have created.

Finally I will choose a technique or a number of techniques most suited to the problem and implement them in an application on a

PC. The application will request all relevant data from the user to build up a complete timetabling problem. The user will then be able to output the finished timetable solution in a number of differing formats, relevant to different entities in the timetable e.g. In the case of a bus timetable, there are timetables for passengers but the data could be represented from the point of view of a driver or a bus, stating where a driver or bus should be at key times.

The possibility of modifying the timetable at a later date to accommodate changes should be allowed for.

2.2. Introduction to Timetabling

2.2.1 Definition of a Timetable

A timetable can be loosely defined as being something which describes where and when people and resources should be at a given time. Most people are familiar with the school timetable which can be presented as a table of days of the week and time slots, e.g.

	9am - 10am		10am - 11am	11am - 12pm
Monday	MATHS Mr. Green Room 1	ENGLISH Mrs. White Room2	SCIENCE Mrs. Teacher Room 2	
Tuesday				

Figure 2.1 Example of a Timetable

You can see that each day is split into time-slots, each time-slot has a list of what subjects are being taught, by who and where. The timetable can be represented in a number of different ways, each student will have there own timetable depending on which subjects they study, as will each teacher and each room, these are all different perspectives on the same timetable.

Other situations where timetables are required are:

1. Manufacturing - production lines, project planning.
2. Travel - trains, buses, etc.
3. University/School examinations.
4. University Lecturing.
5. School timetable.
6. T.V./Radio/Media Schedules.
7. Conferences/Meetings.

These situations require timetables of varying complexity depending on the number of resources we are trying to timetable, the number of time-slots and locations.

2.2.2 Problems Related With Producing Timetables

At first sight timetable problems can seem simple, using the school as an example we can use a simple algorithm to group subjects with no common students, and assign classes to rooms in whatever order is convenient. We can then assign teachers to classes depending on what qualifications they have.

Some problems become apparent. If a student is given a choice of subjects to study we increase the chance that clashes will occur between the subjects [7 et al]. A clash is defined as an instance when a student or other person is scheduled to be in more than one place at some time.

A clear explanation of this problem would be a situation where a student is taught a group of subjects and each subject is only taught once a week, if you miss the lesson there is no repeated lesson. If two or more of these subjects occur at the same time the student will be forced to miss a lesson, there is a clash in the schedule. This problem worsens when we increase the number of students, subjects or the number of subjects taken by each student and also in modularised courses where subjects are taught to students on different courses.

Other problems that could occur are:

- Two or more subjects which have no students common between them require a single teacher so that, again, we cannot place them in the same time slot.

- Some classes may need specific limited resources, such as lab equipment, or a class may be so large that it will be limited to a few large rooms.

These problems are described for a teaching environment, the same problems occur in other timetabling problems and there may well be additional domain specific constraints. When combined, the constraints can make the production of a timetable far from trivial.

2.2.3 NP-Hardness of a Problem

For every problem that can be solved with a well defined algorithm there is a relationship between the size of the problem and the amount of time required to solve it [1, 19 et al]. Obviously if we implement an algorithm within a computer, the speed at which the computer runs the algorithm directly affects how fast we find a solution. New computer hardware may well run an algorithm faster than before and find solutions in less time, but the problem has remained the same.

We could calculate the number of computer processor cycles needed to find a solution, assuming that a faster computer can run more cycles in a given amount of time but will need to complete the same number of cycles as a slower machine. Unfortunately this type of measurement is also prone to variations dependent on computer hardware.

The solution is to use the concept of time complexity of an algorithm. A time complexity function exists for any algorithm. The function describes how the algorithm's time requirements change with respect to the size of the problem the algorithm is solving, but it does not explicitly state how long it will take to find a solution [1,19 et al].

An example time complexity function could be $t=v$, where v is problem size and t is time complexity, this simply means that if we double the problem size then the time required to solve it will double.

Simple algorithms such as sorting algorithms generally have time complexity functions such as; $t=v$, $t=2v$, $t=v2$

As we increase v (the problem size), t does not increase too dramatically, these algorithms are said to have 'polynomial time complexity' because they have the same order as a given polynomial p(v) e.g. v+v2+v3 [1,19]

In problems such as the travelling salesman problem, graph colouring(see section 4.2), scheduling and timetabling problems, as we increase the size of the problem the time required to solve it increases exponentially. These problems are said to have non-polynomial time complexity and are described as being NP-hard [1,19].

NP-Hard problems become effectively not solvable as we increase the problem size. This is because the amount of time required to solve them becomes astronomical. This is demonstrated in figure 2.2.

Although time complexity functions don't actually tell us how long a problem will take to solve, for any given problem size they will have only one value which can be scaled to real time depending on how fast we know we can implement the algorithm. In the following table we will assume that the time complexity function returns the number of computer processor cycles required to find a solution, if we also assume that one cycle takes 1 microsecond(0.000001 seconds) we can now calculate how long an algorithm of a given time complexity will take to solve a problem of a given problem size.

Taken from [20, Garey, M.R & Johnson D.S (1979) "Computers and Interactability : A guide to the theory of NP-Completeness" (Figure 1.2, pg. 7)]

Time Complexity Function	Problem size					
	10	20	30	40	50	60
v	0.00001 seconds	0.00002 sec.	0.00003 sec.	0.0004 sec.	0.0005 sec.	0.0006 sec.
$\mathbf{v^2}$	0.0001 sec.	0.0004 sec.	0.0009 sec.	0.0016 sec.	0.0025 sec.	0.0036 sec.
$\mathbf{v^5}$	0.1 sec.	3.2 sec.	24.3 sec.	1.7 min.	5.2 min.	13 min.
$\mathbf{v^{10}}$	2.7 hrs.	118.5 days	18.7 yrs.	3.3 centuries	30.1 centuries	192 centuries
$\mathbf{2^v}$	0.001 sec.	1 sec	17.9 min.	12.7 days	35.7 yrs.	366 centuries
$\mathbf{3^v}$	0.59 sec.	58 min	6.5 yrs.	3855 yrs	$2.28 * 10^8$ centuries	$1.3 * 10^{13}$ centuries
v!	3.6 sec.	771 centuries	$8.4 * 10^{16}$ centuries	$2.6 * 10^{32}$ centuries	$9.6 * 10^{48}$ centuries	$2.6 * 10^{66}$ centuries

Figure 2.2 Time Complexity Function

For each of the time complexity functions in the left hand column, the table shows the time taken to solve various sizes of problems. The problem size is just a figure and could represent the number of parameters to an algorithm or the range of an input value. What these values represent is not important. If we look at the way in which the amount of time required to solve each problem increases we can see the difference between NP-hard problems and those which are not.

For the first four time complexity functions the increase in time needed is much lower than that for the later three functions. Although the fourth function does seem to be borderline, if we were to increase the problem size further we would see the time required to solve the NP-hard problems increase at a much higher rate [1].

2.3. A General Model for Timetabling Problems

In this section I will look into different types of timetable, identify similarities and from this produce a general model for the representation of timetable problems. This model will define what data we need to give an algorithm before we can even attempt to find a solution.

2.3.1 Generalising Timetable problems

Timetabling problems I have identified are :

1. Manufacturing - production lines, project planning.

2. Travel - trains, buses, etc.

3. University/School examinations.

4. University Lecturing.

5. School timetable.

6. T.V./Radio/Media Schedules.

7. Conferences/Meetings.

There are some key commonalties which can be immediately identified. Each of the above has a resource or number of resources which need to be assigned to a time and/or location.

Although I am attempting to build a general model I would like to describe the scope of the model more accurately here by omitting some of the above timetabling problem type. Firstly, there are many techniques specifically for building manufacturing/project schedules which go into far more detail than is possible in a general model [19,22,23]. Travel timetables are also hard problems to fit into a general model because of the large amount of data needed before we can build a solution e.g. varying number of passengers during the day, data pertaining to locations such as travel times, positions, route information.

T.V./Radio scheduling is not really a problem for automation. When assigning a programme there is usually no choice to be made regarding which channel it will be broadcast. The time of the broadcast is chosen to fit social patterns, not something easily reduced to a set of data or mathematical function!

The remaining set of timetable problems that I will be building a model to represent are as follows:

1. University/School examinations.

2. University Lecturing.

3. School timetabling.

4. Conference/Large Meeting timetables.

More generally, any timetable problem where different groups of entities need to come together one or more times, and have varying resources available each time. Also, the number of times and locations available for timetabling is limited.

For now though I will refer to the above list of timetable problems. For each of these problems I have identified the main entities:

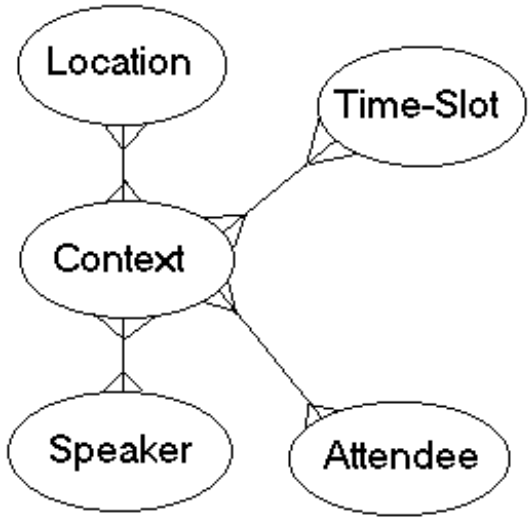
Problem Type	**Entities**
1. The exam timetable -	Supervisor, Student, Subject, Location, Time-slot.
2. University Lecturing -	Lecturer, Student, Subject, Location, Time-slot.
3. School timetable -	Teacher, Pupil, Subject, Location, Time-slot.
4. Conferences/Meetings -	Speaker, Attendees, Topic, Location, Time-slot.

These are the entities that will need to be directly assigned in a completed timetable. Other entities, their attributes and relationships between them will influence the assigning of the above entities but are not actually visible on the completed timetable. I will identify these other entities, but first I will examine these main entities.

It is easy to see the similarities between the four type of timetabling problem I have listed, the entities can be grouped and given a general name as follows:

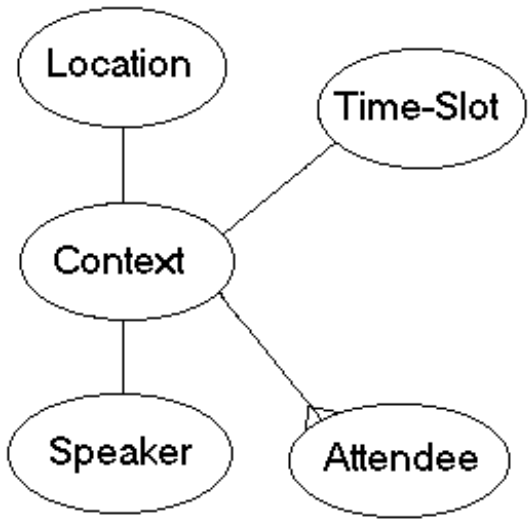
	Problem Type			
General Entity Name	**Exam**	**Lecturing**	**School**	**Conference**
Speaker	Supervisor	Lecturer	Teacher	Speaker
Attendee	Student	Student	Pupil	Attendee
Context	Subject	Subject	Subject	Topic
Location	Location	Location	Location	Location
Time-slot	Time-slot	Time-slot	Time-slot	Time-slot

We can apply some simple relationships to the entities:



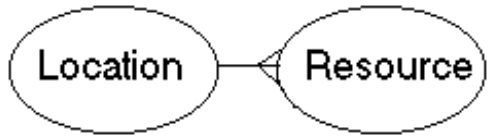
Many people could be qualified to be the speaker for a given subject or context, also a speaker may speak on many different topics or contexts. Similarly, many Attendees will want to be present for a meeting of a particular context and also may want to be present at meeting of many different contexts. Etc.

This is an overall relationship model where all the relationships are many to many. Now here is the model for one timetable assignment.



This is a more useful model. It shows limitations which should be enforced for each timetable assignment. An assignment in the timetable must somehow reference all of the above entities either directly or indirectly. The Speaker and the attendees are meeting under a specific context at a single location and single time-slot. These limitations must be met and are therefore called hard constraints, these are discussed in more detail in section 3.2.

Earlier I stated that other entities not referenced in the end timetable will effect how we can assign the main entities described above. Actually I will introduce just one more entity, a **Resource.**



Each location has an associated list of resources. These are the resources available at each location, such as seats or machinery.

It is envisaged that these simple relationships are sufficient for the general model, more complex relationships can be built around this. e.g. where a subject is taught by a team made up of lecturers

in a university, data pertaining to teams is not needed by the scheduler, only that a lecturer has been marked as a teacher of a subject.

2.3.2 Hard Constraints on Resource Allocation

Each of the timetable problem types I am basing the general model on have common constraints limiting how we allocate the Speakers and Attendees to a Location and Time-slot. The most obvious of these is that no-one can be at more than one location at any given time. A physical constraint such as this is know as a 'Hard' constraint, a timetable that violates hard constraints is not feasible in the real world [7 et al].

The hard constraints are:

A Speaker can only be assigned to one location at any given time.

An Attendee can only be assigned to one location at any given time.

A Location can only have one Context assigned to it at any given time.

4. Attendees cannot be allocated so as to exceed a Locations maximum capacity.

Note that the third constraint is not necessarily applicable to the exam timetable, where exams for many subjects may be held in the same Location.

2.3.3 Soft Constraints on Resource Allocation

Soft constraints are defined as being limits on allocation of resources which, if violated, still result in a feasible timetable, but we would like to avoid them if possible [7 et al].

e.g. We do not want to set consecutive exams for students, and we want to spread exams out over the whole examination period as much as possible.

Some of the things we might want to achieve in a timetable with respect to any one person are:

1. Minimise the occurrence of consecutive timetable assignments.

2. If consecutive assignments do occur, minimise the number of assignments in a row.

3. Maximise gaps between timetable assignments. (Especially applicable to examinations)

4. Minimise gaps between timetable assignments.

5. Minimise travel times/distances.

The desirable characteristics in a timetable depend what type of timetable we are dealing with, points 3 and 4 conflict. Point 3 is desirable for examinations, point 4 for day to day timetables.

2.3.4 Timetable Templates

So far we have assumed that before any allocation was performed that all Speakers, Attendees and Contexts are available for any time slot and can be assigned to any location. In reality people have other commitments and resources may be unavailable for any number of reasons. We need a method of describing when people and resources are available.

To service this need I will define a template which describes when and where people can be allocated.

The template will directly describe unavailability and also allow daily/weekly slots to be made unavailable. A more implicit description, de-allocating slots dependent on which slots have already been (de)allocated will not be used on the grounds that it would over-complicate the final application. Also it is not feasible to describe a template for every resource and attendee, so I will only be providing templates for Speakers.

2.4. An Investigation Into Techniques for Producing Timetables

2.4.1 Job-Shop Scheduling

A lot of work on scheduling is concerned with manufacturing processes and much of the terminology in this field reflects this. The job-shop scheduling problem refers to the assignment of jobs to machines in a workshop, however, the model also applies to many scheduling problems. For instance, a job could be a patient in a hospital and the machine could be a doctor. The basic template

for a job-shop problem is that we have a number of entities which need processing (the jobs) and some other entities which do the processing (the machines) [1].

The general case where there are n jobs to be processed on m machines with each job having it's own sequence of processing is called the general job-shop. This model can be used to represent all job-shop problems, but more specifically defined problems can be represented using four parameters: from [1]

n: The number of jobs

m: The number of machines

A: Describes the flow of jobs through machines. When m=1(only one machine) this parameter is omitted.A can be any one of the following;F - The flow shop. Each job goes through the same sequence of machines.

P: The permutation flow-shop. As the flow-shop, but here the job order for each machine is also the same.

G: The general job-shop

B: Describes the performance measure by which the schedule is evaluated.

The last parameter is used to specify performance measures we wish to minimise or maximise when generating a schedule, e.g. total time the schedule takes, elimination of quiet periods in the workshop or keeping an expensive machine working as much as possible, etc.The job-shop as it stands can not always represent a timetable completely. For instance, timetabling in a school needs to address not only which room a group of pupils should be in but also who will teach them and what subject they should be taking.

If we assume that all class sizes are the same and that all subjects can be taught in all rooms then we could represent the problem as classes being jobs and timetable slots being machines. Allocating the classes to a teacher can then be treated as a separate job-shop problem in which classes(now allocated a timetable slot) are jobs and teachers are machines.

So in the last case we have broken the timetable problem into two job-shop problems, however the case still assumes that classes are

made up of a fixed group of pupils and that all subjects can be taught in all rooms. These extra variables limit the way we can allocate jobs to machines, they are constraints which vary with each instance of a timetable problem, otherwise known as domain dependent constraints.

These domain dependent constraints are not all allowed for in standard job-shop scheduling techniques, some modification would be needed. Also, by splitting the timetabling problem into two separate job-shop problems we may not find feasible timetables e.g.

If we again take the example of the school timetable. The first problem was to allocate classes to timetable slots. If we assume we have successfully solved this problem and now go on to the second problem, allocating the class a teacher. We may find that this problem cannot be solved because some classes may need specific teachers, and these teachers may have already been allocated a class. If this occurs we need to go back to the first job-shop problem and alter the allocation of timetable slots to classes.

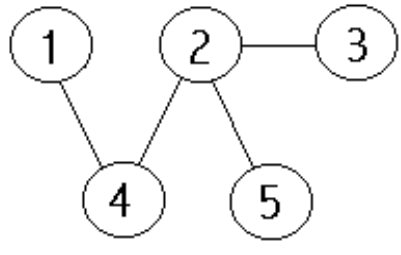
So the two problems are not completely independent, and therefore we really need a single technique which can deal with all the variables.

2.4.2 Graph Colouring

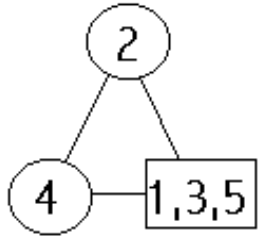
If we take a university timetable as an example, the problem of grouping subjects so that student clashes are eliminated is similar to the mathematical problem of colouring the vertices of a graph[2 et al].

Graph colouring algorithms will group together subjects so that no student has to be in more than one place at any time.

The graph itself represents subjects as vertices and the clashes between subjects as connections between the vertices. Here is a simple graph:



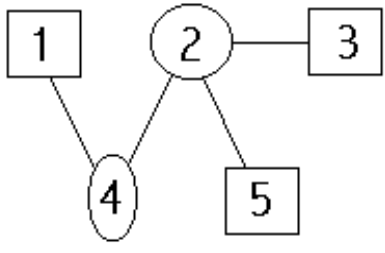
A connection between two vertices indicates there are students who study both of these subjects, it could be just one student or a hundred. The process of colouring the graph refers to assigning colours or symbols to vertices indicating that they can be assigned the same timetable slot. This can be done by merging vertices that do not clash. So for our simple graph we can merge vertices 1,3 and 5. This results in the following graph:



We have now coloured vertices 1,3 and 5 to the same colour. No more vertices will merge with 1,3 or 5 so we can remove the new vertex and continue to colour the remaining vertices.

Clearly only vertices 2 and 4 are left and they are connected, so they are given different colours.

Applying the colours or groupings to the original graph we get:



Vertices of the same colour (here I have used different shapes) represent subjects which can be time-tabled at the same time. In the above graph there are three different symbols, therefore we need a minimum of three time-slots.

In colouring the graph, we have calculated the minimum number of time slots needed to prevent any student clashes. The number of time slots needed may be more than there are available. Also, if many subjects are grouped together there may not be sufficient locations or other resources to actually allocate all of these subjects in one time slot.

The graph colouring problem is NP-complete[2, section 2.3]. There exist various algorithms for colouring a graphs of different structures or with different colouring requirements which can be utilised[2 et al]. This is a useful technique but it does not address the remaining problem of allocating the subjects to timetable slots which is also an NP-complete problem.

2.4.3 Genetic Algorithms

The genetic algorithm works on the same principal as natural evolution. We have a population of candidate solutions to a problem, we select solutions based on how good they are and then we create new solutions from those which were selected [15].So a genetic algorithm acts as a directed search algorithm, or a heuristic algorithm, with the search being performed on the set of possible solutions to a problem.

Genetic algorithms have proven to effective at tackling many problems where no other algorithms exist, usually where the problem is NP-Hard [26, 28]. Among these are timetabling problems [5,6,7,8,9,15 et al].

2.4.4 Choice of Algorithm for the General Timetable Problem

I have decided to use a genetic algorithm to solve the general timetabling problem. The general timetable problem has many different parameters and therefore has a particularly complex set of possible solutions. A genetic algorithm is capable of searching through a complex search space such as this[16]. Also, if extra constraints on the timetable are required at a later date, all that is

required is a new evaluation function which describes the new search space.

2.5. Genetic Algorithms

2.5.1 Introduction to Evolution and Genetics.

Genetic algorithms or evolutionary computing is based on biological genetics and natural evolution. In nature every living organism has a unique set of chromosomes which describes how to build the organism. A chromosome is just one long molecule of DNA, however, different parts of a chromosome describe how to make different things, one part may describe how to build an eye or a limb or just a simple protein. These different sections of the chromosome are called genes [29].

When an organism reproduces, it passes on copies of it's chromosomes. The process of copying the chromosomes can result in errors(mutations) which occasionally can give rise to useful features and better offspring [29].

In sexual reproduction chromosomes from two parents are combined producing offspring with a mixture of the two parents chromosomes, this may lead to combinations of good genes coming together, again producing better offspring. This key process here is know as crossover which refers to the crossing over of genes between chromosomes [15, 29 et al, and section 5.2.4.1].

In the previous paragraphs I have used the phrase 'better offspring', so what is it that makes some organisms 'better' than others? In nature, a species which generates offspring with good genetic information will survive, but organisms of the same species will not all have the same genetic information, there will be variations due to mutation and crossover described above. These variations will make some individuals better able to survive and reproduce, they are better or fitter.

The results is that fitter individuals reproduce more and increase in population. Similarly, individuals that are not as fit will not reproduce as much, if at all, and will have a smaller population and may die off altogether. Over time the build up of good genes, maybe coming together via sexual reproduction, will result in individuals which have adapted very well to their environment.

So evolution can be broken down into two basic steps:

1. Selection of individuals based on their fitness.
2. Reproduction of the selected individuals.

These two steps are repeated continuously, one cycle being a generation.

2.5.2 Evolutionary Computing

We can use the concept of natural evolution to solve problems in the computer. Possible solutions make up the population and better solutions, equivalent to fitter organisms in nature, are more likely to reproduce and pass on their genetic information to the next generation. Over time we could expect that good solutions are evolved just as organisms have evolved in nature.

To solve a problem using genetic algorithms, otherwise know as evolutionary computing, we need the following:

1. A system of encoding the possible solutions or chromosome structure,
2. An initial population of solutions.
3. A function to evaluate a solutions fitness.
4. A method of selecting solutions to be used to produce new solutions.
5. Recombination and Mutation operators to create new solutions from those existing.

2.5.2.1 Solution Representation - The Chromosome Structure

Typically a bit-string or a string of some other kind is used to represent a solution in evolutionary computing. The string data structure is most similar to the natural chromosome and therefore we can manipulate the strings in similar ways to natural chromosomes[15 et al].

2.5.2.2 Initial Population

Before we can begin evolving solutions we must have an initial population of solutions which we can evolve.

Initial solutions are sometime created randomly or by using an algorithm to produce solutions, previously evolved solutions can also be used. It is possible to use any combination of the these three techniques together, increasing the diversity of the solutions in a population will give a better chance of finding good solutions.

2.5.2.3 Evaluation Functions

This function takes a single solution as a parameter and returns a number indicating how good the solution is. The number can be integer or real and have any range. By itself the number returned means nothing, only when we compare the values returned by all of the possible solutions can we select the better solutions.

2.5.2.4 Selection

Each generation we must produce new solutions from the current population of solutions. How many of the current population are used to generate the new solutions, which ones we select and which of the current population should we erase to make room for the new solutions all need to defined.

Some types of selection are:

Roullette wheel - Imagine that each solution in the population has a slice of a roulette wheel. Better solutions have proportionately larger sections of the wheel and therefore there is a greater chance they will be selected. If we want to select two solutions we spin the wheel twice. We then have the choice of removing selected solutions from the wheel before we run again or we can leave them on the wheel and therefore good solutions may be selected more than once.

Tournament Selection - picks 2 or more solutions at random and uses a tournament strategy, where a number of rounds are played. In each round a number of solutions come together to compete, and only the fittest solution will win and be selected.

Elitism - Not really a selection strategy, but other strategies can employ elitism. This is where the best solution in the population is guaranteed to survive to the next generation.

2.5.2.5 Recombination and Mutation operators

After selecting solutions we need a technique or techniques of combining these solutions in a manner that will produce good, new solutions. These are known as recombination operators.

Mutation operators alter the new solutions in a totally random manner, this helps maintain diversity in the population, and results in solutions that otherwise could not have been produced by recombination alone.

2.5.2.5.1 Crossover Operators

Crossover operators will normally take two parents and create offspring with a mixture of both parents genetic information. The common forms of crossover are, one-point, two-point, n-point and uniform crossover.

In 1-point crossover a random point is chosen along each parents chromosome, each half of the chromosomes is then swapped over e.g. if we have two parents:

parent1 1234|5678

parent2 8765|4321

If we perform crossover at the mid-points (show by the bar) we get two new offspring:

offspring1 12344321

offspring2 87655678

One-point crossover is inspired by biological processes [15].

Two Point crossover is similar to one-point, but there are two crossover points. This results in one parent retaining the head and tail of it's chromosome, gaining a new mid-section. The other parent will retain it's mid-section and gain a new head and tail.

N-point crossover is the same technique again but we select N crossover points, swapping parts of the chromosomes between every other crossover point, starting at the head and the first crossover point for parent one and parent two respectively.

Uniform crossover selects x number of points in the chromosomes, where x is a random number less than the chromosome length. Each selected point is then swapped over.

2.5.2.5.2 Mutation

When we create offspring using whatever technique we must copy genetic information from the parents. In nature, duplicating DNA can sometimes result in errors, DNA is also prone to damage in day to day existence which also results in errors. These errors or mutations can sometimes result in good features, these features can then come together via sexual reproduction and eventually lead to new species[29]. Therefore errors in DNA perform a vital role in natural evolution.

In Evolutionary computing we can imitate mutation by generating errors in the offspring. A point in the offspring's chromosome can be set to a random value, however this may be an invalid value and so it can be beneficial to replace the value with one which is valid.

2.6. A Genetic Algorithm For Our General Timetabling Problem.

2.6.1 Chromosome Structure

Each chromosome represents a complete timetable. The chromosome is made up from a string of genes and a gene consists of four numbers. These four numbers represent a Subject, Location, Lecturer and a Time-slot respectively. One gene therefore assigns a subject and a lecturer to a time and location.

So the gene 2,5,7,23 assigns lecturer 7 to teach subject 2 at location 5 in time-slot 23.

2.6.2 The Initial Population

Each chromosome in the initial population is generated by a simple algorithm which allocates as many timetable slots as possible without creating subject, lecturer or room clashes in a time-slot. If a lecturer defined as being able to teach a subject is available they will be selected, otherwise any other available lecturers are chosen.

Subject clashes are not considered.

2.6.3 Evaluation Function

The evaluation function calculates a punishment value for each chromosome. Good chromosomes have low punishment values, a value of zero indicates the chromosome meets all the criteria described in the evaluation function.

The evaluation function is made up from many different evaluations which all produce a punishment value which accumulates to give the final punishment.

The following sections describe how a chromosome is evaluated:

2.6.3.1 Timetable Clashes

The following clashes between entities are evaluated and will increase the punishment value if they are violated.

Subject clashes

Where two subjects are allocated at the same time the number of students common to both subjects is added to the punishment value.

Location clashes

Where two or more subjects are allocated to a location at the same time.

Punishment is increased by (subjects allocated-1).

Lecturer clashes

Where two or more lecturers are allocated to a location at the same time.

Punishment is increased by (allocations-1).

Lecturer template violations

If a lecturer is assigned during a time-slot for which they are not available punishment is increased by 1000.

2.6.3.2 Subject slot satisfaction

Here we evaluate how well a chromosome satisfies a subjects requirements, these requirements are the number of time-slots each subjects requires and also the resources that are required for each slot.

Surplus and Insufficient Timetable Allocations

If a chromosome allocates too many or too few timetable slots to a subject.

Punishment is increased by the quantity of surplus resources or the quantity of resources not allocated, multiplied by the number of time-slots the surplus/shortfalls effect.

Surplus and Insufficient Resources

For all slots that a chromosome assigns to a subject we punish for any surplus of resources that are available at the assigned location and also for any resources that are required but are not available at the assigned location.

2.6.3.3 Weighting of Evaluations

Running the genetic algorithm with just the clash evaluations resulted in a population of zero length chromosomes. This is because if nothing is assigned then no clashes can occur. Similarly, using just the slot satisfaction evaluations resulted in long chromosome with many clashes, but most of the subject slots where assigned a timetable slot.

By adding together the punishment values from all of the evaluations the genetic algorithm tended towards short chromosomes, avoiding clashes at the expense of not assigning timetable slots.

To alleviate this problem some of the evaluation punishment values are multiplied by an independent value. This increases or weights the importance of some evaluations with respect to the other evaluations. This prevents important factors from being ‘drowned’ by less important factors.

2.6.4 Selection

We use roulette wheel selection. Chromosomes with small punishment values have a greater chance of being selected. This is done by finding the highest of the punishment values, a new value is then calculated for each chromosomes which is it's punishment value subtracted from the highest punishment value. This value divided by the highest punishment value now gives us the probability a chromosome will be selected.

The selection is weighted so that a very good chromosome can not have a selection probability of more than 90%. This is called selection pressure, the pressure to select good chromosomes. By reducing the selection pressure we prevent a few good chromosomes from dominating the population of solutions and reducing the effectiveness of the genetic algorithm.

The offspring produced from the selected chromosomes replace the worst chromosomes in the population. Therefore this strategy guarantees the best solution survives in each generation, it is elitist.

2.6.5 Recombination Operator incorporating Mutation

When two chromosomes have been selected as parents they are used to produce offspring using the crossover operator. Here we will use one-point crossover.

The length of a chromosome is variable so we must choose the crossover point in each of the parents separately since one parent may be longer than the other. We then produce two offspring. The first offspring has the first half of parent one (up to the crossover point) and the second half of parent two. The second offspring has the first half of parent two and the second half of parent one.

As we copy each gene from a parent to the offspring each part of the gene has a 1% chance of being mutated. If a value is mutated it is replaced with a valid value for that part of the gene, so if we decide to mutate the subject value it will be replaced with a random value between 0 and the number of subjects minus 1. Any clashes that may occur as a result are not considered, a clash will result in a higher punishment value from the evaluation function.

2.7. Conclusions

2.7.1 Assessment of the Applications Usefulness

To test the genetic algorithm initially I created a very simple problem. The problem consists of nine subjects, each requiring one timetable slot. There are 21 students each studying two subjects, the subjects have been selected so that they form three main groups (1,2,3), (4,5,6) and (7,8,9). Subjects within a group have common students and therefore should not be assigned in the same time slot.

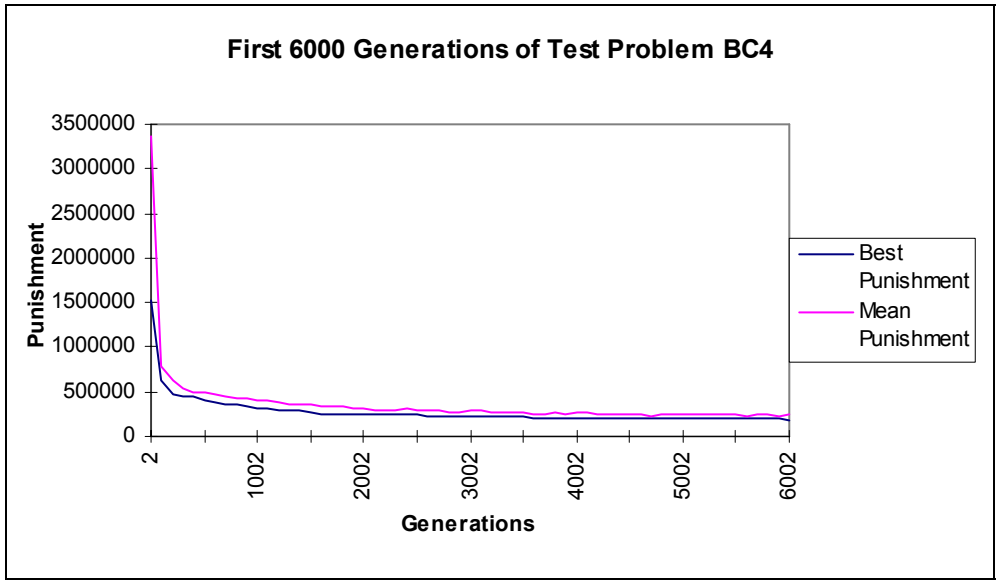
Three hours, three locations and three lecturers are available for timetabling, effectively giving 9 hours of teaching time, this is the exact amount required and so there should be no free timetable slots. Finally, each lecturer is available for teaching all of the subjects

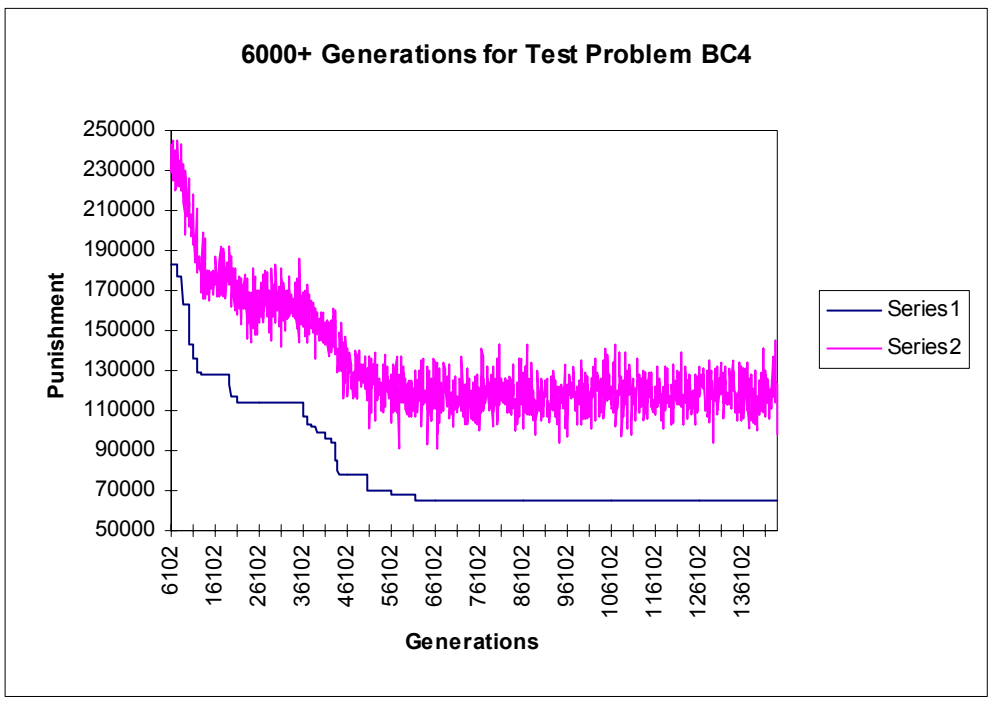
A satisfactory solution to this problem is indeed found within a few minutes. A population of 100 was satisfactory.

I then created a more complex problem in test file ‘bc4.ttd’. This file describes the 23 subjects currently being taught on the final year computer science/information technology degree courses at Staffordshire university computing school. All 270+ students and their options are listed in the STUDENT_DATA block.

From then on I have constructed requirements for the timetable. Each subject requires 2 hours a week and only enough seats for all students on a subject must be present. Therefore we need 46 timetable slots. I have defined 50 timetable slots made up from 2 days, 5 slots a day, five lecturers and 5 locations. Again, each lecturer is available for all subjects and are always available.

The performance of the genetic algorithm is shown in the following graphs:





We can see that both the current best weight and the mean weight fall rapidly early on. This indicates that the genetic algorithm is

finding fitter solutions and is resulting in a fitter population as a whole.

The actual timetable produced described by the best solution from this run is show in file ‘BC4.MST’ in appendix D. The timetable contains no location clashes, no lecturer clashes(although these are not visible on a finished timetable). The two problems with the time table are that only 43 slots from the 46 that are required are time-tabled and there are 41 subject clashes from a possible 73. On the positive side, the subject clashes do tend to have low numbers of common students, less than ten in most cases. To see if this problem can be alleviated I increased the punishment on student clashes by a factor of ten.

The resulting timetable is shown in file BC4_2.MST in appendix D. The time-table does have less subject clashes (31 from a possible 58), but also only time-tables 39 hours from the 46 that are required. Of course we do not know if it is possible to timetable all of the subjects 5 at a time so that no clashes occur, a graph colouring algorithm could be employed at the initialisation to estimate the possibility that a good solution will be found and also to seed the initial population with better solutions.

2.7.2 Assessment of the Bibliography

[1] “Sequencing and Scheduling - An introduction to the Mathematics of the Job shop”

by S. French (1982). Provided a good introduction to the job-shop problem and traditional techniques used to produce solutions and systems of evaluating problems. Also a good introduction to the mathematics behind scheduling problems, the NP-Hardness etc. There is very little reference to timetabling problems, however it has given an insight into scheduling which has shown that the use of job-shop scheduling techniques alone are not appropriate to the timetabling problem.

There are a number of papers on timetabling by the Timetabling Research group at the Department of Computer Science, University of Nottingham. These papers almost exclusively deal with examination timetable problems, with a view to expanding the examination techniques to day to day university timetabling.

There are also a number of papers from the Evolutionary Computing group at Edinburgh university. These deal with the use of evolutionary computing for a range of applications, some of which are timetabling problems.

These two sets of papers have been the main source of information regarding the implementation and abilities of genetic algorithms.

Another paper from Edinburgh worth mention is [15] "Investigating Genetic Algorithms for Scheduling" by Hsiao-Lan Fang, this paper also deals with examination timetabling. The early sections on genetic algorithms provide a concise description of the various components of a genetic algorithm.

2.7.3 Conclusions

Although the solutions found above are not particularly good they do show promise. There is a lot of scope for investigation of the genetic algorithm. Some areas which could help the search for solutions are:

Chromosome structure - The structure can affect how sections of good solution come together.

Recombination operators - In this project I have only used one-point crossover, the other types of crossover discussed in section 5.2.5.1 may be beneficial. Also, more intelligent crossover/mutation operators which avoid producing non-viable timetables effectively reduce the search space by eliminating the non-viable timetables. Reducing the search space should increase out chance of finding a good solution.

Selection Methods - Tournament selection, and also using different selection pressures could be effective.

There are many variables within the genetic algorithm which effect path of the search through the set of possible solutions. Those specified above and also simple variables such as mutation and crossover rates, different values could be used or even variable rates.

Bibliography

[1] French S(1982) “Sequencing and Scheduling. An introduction to the Mathematics of the Job shop”

[2] EK Burke, DG Elliman and RF Weare, "A University Timetabling System based on Graph Colouring and Constraint Manipulation", Journal of Research on Computing in Education, vol 27, no 1, pp 1-18, 1994.

[3] EK Burke, DG Elliman and RF Weare, "Automated Scheduling of University Exams", proceedings of the IEE Colloquium on Resource Scheduling for Large Scale Planning Systems (10th June 1993), digest No. 1993/144, section 3, 1993.

[4] EK Burke, DG Elliman and RF Weare, "Extensions to a University Exam Timetabling System", proceedings of the IJCAI-93 Workshop on Knowledge-Based Production, Planning, Scheduling and Control (Chambery, France, 29th Aug 1993), pp 42-48.

[5] EK Burke, DG Elliman and RF Weare, "A Genetic Algorithm for University Timetabling", proceedings of the AISB Workshop on Evolutionary Computing (University of Leeds, UK, 11th-13th April 1994), Society for the Study of Artificial Intelligence and Simulation of Behaviour (SSAISB).

[6] EK Burke, DG Elliman and RF Weare, "A Genetic Algorithm based University Timetabling System", proceedings of the 2nd East-West International Conference on Computer Technologies in Education (Crimea, Ukraine, 19th-23rd Sept 1994), vol 1, pp 35-40, 1994.

[7] EK Burke, DG Elliman and RF Weare, "The Automation of the Timetabling Process in Higher Education", Journal of Educational Technology Systems, vol 23, no 4, pp 257-266, Baywood Publishing Company, 1995.

[8] EK Burke, DG Elliman and RF Weare, "A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems", proceedings of the 6th International Conference on Genetic Algorithms (ICGA'95, Pittsburgh, USA, 15th-19th July 1995), pp 605-610, Morgan Kaufmann, San Francisco, CA, USA.

[9] EK Burke, DG Elliman and RF Weare, "The Automated Timetabling of University Exams using a Hybrid Genetic Agorithm", proceedings of the AISB (Artificial Intelligence and Simulation of Behaviour) workshop on Evolutionary Computing (University of Sheffield, UK, 3rd-7th April 1995).

[10] EK Burke, DG Elliman, PH Ford and RF Weare, "Examination Timetabling in British Universities - A Survey", proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT'95, Napier University, Edinburgh, UK, 30th Aug - 1st Sept 1995), pp 423-434.

[11] EK Burke, JP Newall and RF Weare, "A Memetic Algorithm for University Exam Timetabling", proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT'95, Napier University, Edinburgh, UK, 30th Aug - 1st Sept 1995), pp 496-503.

[12] EK Burke, DG Elliman, PH Ford and RF Weare, "Specialised Recombinative Operators for the Timetabling Problem", to appear in proceedings of the AISB (Artificial Intelligence and Simulation of Behaviour) Workshop on Evolutionary Computing (University of Sheffield, UK, 3rd-7th April 1995), Lecture Notes in Computer Science, Springer-Verlag.

[13] RF Weare, "Automated Examination Timetabling", PhD Thesis, Department of Computer Science, University of Nottingham, UK, June 1995.

[14] Paul Field (1996) “A Multary Theory for Genetic Algorithms: Unifying Binary and Non-binary Problem Representations”, Submitted to the university of London for the degree of Doctor of Philosophy in Computer Science.

[15] Hsiao-Lan Fang (1992) “Investigating Genetic Algorithms for Scheduling”, Msc Dissertation, Department of Artificial Intelligence, University of Edinburgh.

[16] Dave Corne, Peter Ross, Hsiao-Lan Fang “Evolutionary Timetabling: Practice, Prospects and Work in Progress” , Department of Artificial Intelligence, University of Edinburgh. Presented at the UK Planning and Scheduling SIG Workshop, Strathclyde, September 1994.

[17] "Dynamic Training Subset Selection for Supervised Learning in Genetic Programming", Chris Gathercole, Peter Ross. To be published in PPSN-III, Springer-Verlag, 1994.

[18] "Applications of Genetic Algorithms", Peter Ross and Dave Corne, AISB Quarterly No. 89, Autumn 1994, Special Theme on Evolutionary Computing, Terry Fogarty (Ed.), ISSN 0268-4179, pages 23--30.

[19] “Resource Allocation Problems. Algorithmic Approaches”, Toshihide Ibaraki and Naoki Katoh. (1988).

[20] “Computers and Intractability : A guide to the theory of NP-Completeness”, Garey M.R and Johnson D.S. (1979)

[21] "Solving the Module Exam Scheduling Problem with Genetic Algorithms", Dave Corne, Hsiao-Lan Fang, Chris Mellish", In Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Gordon and Breach Science Publishers, Chung, Lovegrove, Ali (eds), pages 370--373, 1993. This is also available as DAI Research Report No. 622.

[22] "A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems", Hsiao-Lan Fang, Peter Ross, Dave Corne, in Proceedings of the Fifth International Conference on Genetic Algorithms, S. Forrest (ed), San Mateo: Morgan Kaufmann, pages 375--382, 1993.

[23] "Successful Lecture Timetabling with Evolutionary Algorithms", Peter Ross, Dave Corne, Hsiao-Lan Fang, Appears in Workshop Notes, ECAI'94 Workshop W17: Applied Genetic and other Evolutionary Algorithms.

[24] "Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation", Peter Ross, Dave Corne, Hsiao-Lan Fang, to appear in PPSN III, Springer Verlag, 1994. Also, DAI Research Paper No. 707.

[25] "Fast Practical Evolutionary Timetabling", Dave Corne, Peter Ross, Hsiao-Lan Fang, in Evolutionary Computing: AISB Workshop 1994, Selected Papers, Springer Verlag Lecture Notes in Computer Science 865, T. Fogarty (ed), Springer Verlag, 1994. Also DAI Research Paper No. 708.

[26] Spears, William M. (1990). Using Neural Networks and Genetic Algorithms as Heuristics for NP-Complete Problems. Masters Thesis, George Mason University, Fairfax, Virginia.

[27] Spears, William M. (1995). Simulated Annealing for Hard Satisfiability Problems. To appear in Proceedings of the 2nd DIMACS Implementation Challenge, David S. Johnson and Michael A. Trick (eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science.

[28] De Jong, Kenneth A. and William M. Spears (1989). Using Genetic Algorithms to Solve NP-Complete Problems. In Proceedings of the Int'l Conference on Genetic Algorithms, 124-132.

[29] “The Evolution of Life”, Macdonald Educational Ltd. (1980).

Appendix A: User Documentation

1. Overview of Programs

There are two programs which make up the complete application, they are called “TT_GA” and “TT_OUT”. Both are text based and run under Microsoft Windows.

The TT_GA program reads in a description file (with extension .ttd) and actually runs the genetic algorithm to find a solution to the timetable problem. When the genetic algorithm is terminated the user has the option of saving the best solution to a timetable save file (with extension .tts). This file contains the raw data describing the timetable.

The TT_OUT program reads in a timetable save file and allows the user to save a text file containing a readable form of the timetable solution. The user is then free to load this text file into any other application for output to a printer or to perform further manipulation of the timetable.

The following sections will assume that all files and programs pertaining to the application are installed under MS-Windows.

2. The Timetable Problem Definition File (.ttd file)

Before you can run TT_GA to find a solution to your timetable problem it is necessary to create a file describing the timetable problem. We will call this file the ‘timetable definition file’ and it will always have the filename extension ‘.ttd’. Example .ttd files can be found in appendix C.

The timetable problem files contain key words which denote what is being described at that point in the file. The primary key words are:

DAYS, SLOTS, RESOURCE, LOCATION, SUBJECT,

LECTURER, STUDENT_DATA .

DAYS, SLOTS

These two keywords must be present, and before any other text. Both should be followed by an integer value like so :

DAYS 5

SLOTS 8

This defines how many days the timetable covers and how many distinct time slots there are in each day. Time slots are therefore all the same length and are not allowed to overlap.

RESOURCE

We use this keyword to describe different types of resources. These resources are later used in the LOCATION definitions to define what resources are present at each location. Therefore resources must be declared before locations.

Typical resource definitions would be:

```
RESOURCE SEAT     {  "  "    }
RESOURCE TERMINAL { "A standard dumb
terminal" }
```

After the resource keyword you must specify a mnemonic which will be used to identify the resource throughout the rest of the file. After the mnemonic a text string between speech marks and parenthesis must be specified. This is string is a description of the resource which the user can refer to in later editing of the file.

LOCATION

A typical LOCATION definition would be:

```
LOCATION KC15
{
   TERMINAL 20
   SEAT 20
}
```

The LOCATION keyword must be followed by a mnemonic which is used to refer to the location in the final output timetable produced by the second program TT_OUT (see section 4). Within the brackets you can now list the resources that are present at the location using the resource mnemonics defined earlier. Each resource mnemonic must be followed by an integer which defines the quantity of each resource at the location.

SUBJECT

A typical SUBJECT definition would be:

```
SUBJECT AF
{
    SLOT
    [   RESOURCE SEAT 0
        HOURS 2    ]
    SLOT
    [   RESOURCE TERMINAL 0
        RESOURCE SEAT 0
        RESOURCE DEC5500 0
        HOURS 1
    ]
}
```

The SUBJECT keyword must be followed by a mnemonic which is used to refer to the subject in the STUDENT_DATA section of the .ttd file and also in the final timetable output produced by the second program TT_OUT (see section 4).

Within a set of curly brackets we now define how many timetable slots the subject requires and the resources that are required on each occasion.

A slot or set of slots which require the same set of resources can be defined with one SLOT statement. After the SLOT keyword you can define the resources that are required using the RESOURCE keyword followed by a resource mnemonic and finally the quantity

of the resource that is required. If a resource quantity of zero is given then it is assumed that the quantity of the resource that is required is equal to the number of students for that subject.

Also in the SLOT statement you must use the HOURS keyword to specify for how many time-slots in the timetable this slot definition should be assigned. So e.g. if a subject requires two hours of lectures a week we would write something like :

SLOT

[

RESOURCE SEAT 0

HOURS 2

]

Finally each SLOT definition must be with a set of Square brackets as shown above.

LECTURER

Defines a lecturer, a list of subjects s/he is available for teaching on and when s/he is unavailable for teaching.

A typical LECTURER definition would be:

```
LECTURER JFB
{
   NAME "Joe F. Bloggs"
   SUBJECT CNDS MTOS ;
   TEMPLATE 0 0 EVERY 1 0
   TEMPLATE 1 0 THROUGH 2 8
}
```

After the LECTURER keyword you must specify a mnemonic that is used to refer to the lecturer in the final output timetable produced by TT_OUT(see section 4).

Within a set of curly brackets you must specify the name of the lecturer using the NAME keyword followed by the lecturers name within speech marks. This should be followed by the SUBJECT

keyword which is followed by a list of subject mnemonics(defined earlier) and terminated with a semi-colon.

Finally the optional keyword TEMPLATE can be used to describe when a lecturer is unavailable. A single TEMPLATE statement can take one of two forms :

TEMPLATE <int> <int> EVERY <int> <int>

or

TEMPLATE <int> <int> THROUGH <int> <int>

where <int> refers to an integer value.

In both cases the first integer specifies a day and the second integer a time-slot within that day. In the first case the lecturer is made unavailable for this first time-slot and then repeatedly after the number of days and time-slots defined by after the EVERY keyword.

So if we wanted to make a lecturer unavailable for the first slot of every day we would use the template:

TEMPLATE 0 0 EVERY 1 0

If we where to use the template:

TEMPLATE 0 0 EVERY 1 1

then the lecturer is unavailable for the 1st slot on day 1, the 2nd slot on day two and so on.

In the second form of the TEMPLATE statement second set of integers after the THROUGH statement also describe a day and time-slot, the first and second day/time-slot are made unavailable and also all of the time-slots in between.

Multiple TEMPLATE statements can be used to build a detailed template of a lecturers availability.

STUDENT_DATA

This section defines which subjects each student is teaching. From this a great deal of usefull information is calculated, such as the total number of students, the number of students on each subject and the number of students common to any given set of subjects.

An example STUDENT_DATA block:

STUDENT_DATA

{

SUB1 SUB2 SUB3 ;

SUB2 SUB4 SUB5 ;

}

Each line here contains a list of subjects terminated by a semi-colon. These are the subjects taken by one student. Each students subjects must be given in order to build an accurate model of the timetable problem. Each line must be terminated by both a space and a semi-colon.

3. The TT_GA program

To run the program simply click on the 'TT_GA' program icon in the relevant window. A new window will appear in which text will be displayed and text can be typed into.

The new window will immediately show the following:

> Enter name of timetable problem definition file,
>
> .ttd extension will be added if not specified
>
> :

You must enter the name of a pre-written timetable problem definition file before the genetic algorithm can be run. On pressing <return> the file will be parsed, as the various sections of the input file are parsed messages indicating the parsing of each section will be displayed. If the input file has parsed successfully you will see the following:

> Input file parsed successfully
>
> Do you wish to load a previously saved solution?(Y/N)

If you have run the genetic algorithm previously and saved a solution in a timetable save file (with .tts filename extension) then you can read in the solution. It will be used in the search for a better solution by the genetic algorithm.

Also, if a good solution has been found but the problem definition file has been altered since, the previous solution may provide a good starting point for the search for a new solution.

So if you now press ‘Y’ for yes, load a solution you will be asked to give the name of a timetable save file, enter an appropriate filename and press return, or just press return to use the same name as the problem definition file but with a .tts extension rather than a .ttd extension.

If you have successfully read in a previous solution or selected ‘no’ at the previous prompt the genetic algorithm will begin to run.

The window should now show a number of values which are being continually updated:

```
Now  Searching for timetable

| Generation  | Best weight  | Mean weight  | Mean
Length  | Length of best  |

|-------------|--------------|--------------|----------
----|-----------------|

| 10          | 200000       | 400000       | 21
| 34              |
```

These values are:

Generation - How many generations have been performed so far.

Best Weight - The weight of the best solution that has been found so far. As calculated by the evaluation function.

Mean weight - The mean weight of the entire population of solutions.

Mean length - The mean length of the entire population of solutions.

Length of best - The best solutions chromosome length.

As the algorithm progresses through the generations the best and mean weights should be seen to fall. When they no longer fall then the genetic algorithm is struggling to find better solutions. At this point press ‘Q’ to stop searching.

You will now be asked if you wish to save the best solution. If you select ‘yes’ you will be asked for a save timetable filename, at this

point you can press <return> to use the same file name as the timetable description file, but with a .tts file extension rather than a '.ttd'.

4. The TT_OUT Program

To run the program simply click on the 'TT_OUT' program icon in the relevant window. A new window will appear in which text will be displayed and text can be typed into.

The new window will immediately show the following:

> Enter name of timetable problem definition file,
>
> .ttd extension will be added if not specified
>
> :

As with the TT_GA program you must enter the name of a pre-written timetable problem definition file. The file will be parsed as before and if successful you will be prompted for the name of a timetable save file. Again, by pressing <return> the timetable definition file name will be used but with the relevant '.tts' extension. Otherwise you can type in any other file name.

> Select an output format for the timetable:
>
> 1) Complete master timetable
>
> 2) Quit

If everything has been successful the window will show the following text:

Currently there is only one output format for the timetable. Press '1' and you will be asked to enter a filename to save the timetable to. The timetable will be saved as a text file with days and slots across the top and lecturers listed on the left hand column. An example master timetable can be found in appendix D.

Appendix C: Example Timetable Problem Definition Files

QUICK1.TTD - A small test problem

```
DAYS 1
SLOTS 3

RESOURCE SEAT     {   " Most resources are obvious "     }

LOCATION RED
{
  SEAT 3
}

LOCATION BLUE
{
  SEAT 3
}

LOCATION GREEN
{
  SEAT 3
}

SUBJECT SUB1
{
  SLOT
```

```
  [  RESOURCE SEAT 0
     HOURS 1   ]
}

SUBJECT SUB2
{
  SLOT
  [  RESOURCE SEAT 0
     HOURS 1   ]
}

SUBJECT SUB3
{
  SLOT
  [  RESOURCE SEAT 0
     HOURS 1   ]
}

SUBJECT SUB4
{
  SLOT
  [  RESOURCE SEAT 0
     HOURS 1   ]
}

SUBJECT SUB5
{
  SLOT
```

```
  [  RESOURCE SEAT 0
     HOURS 1   ]
}

SUBJECT SUB6
{
  SLOT
  [  RESOURCE SEAT 0
     HOURS 1   ]
}


SUBJECT SUB7
{
  SLOT
  [  RESOURCE SEAT 0
     HOURS 1   ]
}

SUBJECT SUB8
{
  SLOT
  [  RESOURCE SEAT 0
     HOURS 1   ]
}

SUBJECT SUB9
{
```

```
  SLOT
  [  RESOURCE SEAT 0
     HOURS 1   ]
}

LECTURER LEC1
{
  NAME "Lec1"
  SUBJECT SUB1 SUB2 SUB3 SUB4 SUB5 SUB6 SUB7 SUB8
SUB9 ;
}

LECTURER LEC2
{
  NAME "Lec2"
  SUBJECT SUB1 SUB2 SUB3 SUB4 SUB5 SUB6 SUB7 SUB8
SUB9 ;
}

LECTURER LEC3
{
  NAME "Lec3"
  SUBJECT SUB1 SUB2 SUB3 SUB4 SUB5 SUB6 SUB7 SUB8
SUB9 ;
}



STUDENT_DATA
{
```

SUB1 SUB2 ;
SUB1 SUB2 ;
SUB1 SUB2 ;

SUB2 SUB3 ;
SUB2 SUB3 ;

SUB1 SUB3 ;
SUB1 SUB3 ;

SUB4 SUB5 ;
SUB4 SUB5 ;
SUB4 SUB5 ;

SUB5 SUB6 ;
SUB5 SUB6 ;

SUB4 SUB6 ;
SUB4 SUB6 ;

SUB7 SUB8 ;
SUB7 SUB8 ;
SUB7 SUB8 ;

SUB8 SUB9 ;
SUB8 SUB9 ;

SUB7 SUB9 ;

```
SUB7 SUB9 ;
}

Test file BC4.TTD - Student data from the 4th year computing degree course at Staffordshire University.

DAYS 2
SLOTS 5
MINFLAG 0

RESOURCE SEAT { " Most resources are obvious " }
RESOURCE TERMINAL { " see what I mean" }
RESOURCE DEC5500 { "DEC Vaxstation 5500's" }
RESOURCE PC-BCC { "PC with BorlandC++" }
RESOURCE PC-LINUX { "PC with linux" }

LOCATION RED
{
  SEAT 220
}

LOCATION BLUE
{
  SEAT 180
}

LOCATION GREEN
{
  SEAT 180
}

LOCATION D116
{
  SEAT 200
}

LOCATION D118
{
  SEAT 200
}

SUBJECT AF
{
  SLOT
  [ RESOURCE SEAT 0
    HOURS 2 ]
}

SUBJECT AI
{
  SLOT
  [ RESOURCE SEAT 0
    HOURS 2 ]
}

SUBJECT ASAD
{
  SLOT
  [ RESOURCE SEAT 0
    HOURS 2 ]
}
```

```
SUBJECT CBO
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT CLTI
{
  SLOT
  [ RESOURCE
SEAT  0
    HOURS 2  ]
}

SUBJECT CNDS
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT DB
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT DP
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT FR
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT GE
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT GR
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT HCI
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT IDE
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT IP
{
  SLOT
  [ RESOURCE
SEAT 0
```

```
    HOURS 2  ]
}

SUBJECT IRM
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2   ]
}

SUBJECT MI
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT MTOS
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2 ]
}

SUBJECT NN
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2
  ]
}

SUBJECT OR
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}
SUBJECT PIO
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT RTES
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT SE
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}

SUBJECT SIM
{
  SLOT
  [ RESOURCE
SEAT 0
    HOURS 2  ]
}


LECTURER PVB
{
  NAME "Pete
Best"
  SUBJECT AF
AI ASAD  CBO
CLTI CNDS   DB
DP   FR   GE   GR
HCI  IDE   IP
IRM   MI MTOS
NN   OR  PIO
RTES   SE  SIM ;
```

```
}

LECTURER
DKB
{
  NAME "Di
Bishton"
  SUBJECT AF
AI ASAD  CBO
CLTI CNDS   DB
DP   FR   GE   GR
HCI  IDE   IP
IRM   MI MTOS
NN   OR  PIO
RTES   SE  SIM ;

}

LECTURER CM
{
  NAME "Chris
Mills"
  SUBJECT AF
AI ASAD  CBO
CLTI CNDS   DB
DP   FR   GE   GR
HCI  IDE   IP
IRM   MI MTOS
NN   OR  PIO
RTES   SE  SIM ;

}

LECTURER PG
{
  NAME "Pete
Gittins"
  SUBJECT AF
AI ASAD  CBO
CLTI CNDS   DB
DP   FR   GE   GR
HCI  IDE   IP
IRM   MI MTOS
NN   OR  PIO
RTES   SE  SIM ;

}

LECTURER PC
{
  NAME "Phil
Cornes"
  SUBJECT AF
AI ASAD  CBO
CLTI CNDS   DB
DP   FR   GE   GR
HCI  IDE   IP
IRM   MI MTOS
NN   OR  PIO
RTES   SE  SIM ;

}

STUDENT_DAT
A
{
IRM   HCI   AF
DB    ;
AI    NN    HCI
MTOS ;
HCI   IRM   SE
DB    ;
AI    MTOS
CNDS  HCI   ;
ASAD  PIO   HCI
IRM   ;
ASAD  PIO   DB
AI    ;
HCI   DB    IRM
CBO   ;
DB    AF    IRM
HCI   ;
GR    IDE
CNDS  SIM   ;
CNDS  MTOS  DB
      HCI   ;
IRM   HCI
      ASAD
      CBO   ;
ASAD  PIO
      HCI
      IRM   ;
MTOS  IP
      CNDS
      RTES  ;
```

MTOS IDE SE
CNDS ;

DP AI IP
SIM ;

ASAD PIO DB
IRM ;

CNDS ASAD DB
HCI ;

ASAD PIO
IRM DB
;

ASAD HCI
IRM
CBO ;

DB CNDS AF
IRM ;

IDE MTOS
RTES
CNDS ;

MI MTOS
CNDS
RTES ;

RTES DP DB
SIM ;

IRM HCI
ASAD
CBO ;

ASAD PIO
IRM
HCI ;

ASAD DB
IDE DP
;

DB AF
HCI
IRM ;

IRM DB ;

ASAD PIO
HCI
IRM ;

DP DB
RTES
SIM ;

FR CNDS IP
MTOS ;

SE CNDS
RTES AI
;

DB MTOS
IDE
CNDS ;

MTOS IP DB
IDE ;

CNDS AF DB
IRM ;

HCI CNDS MI
IP ;

IRM CNDS DB
OR ;

HCI CNDS GR
AI ;

HCI CNDS DB
OR ;

ASAD CBO OR
IRM ;

DB CNDS SE
IRM ;

HCI CNDS DB
IRM ;

HCI OR DB
ASAD ;

ASAD CBO DB
IRM ;

AF CNDS DB
IRM ;

ASAD GE
CBO
IDE ;

CNDS IP
RTES GR
;

IP AF
HCI
IRM ;

MTOS CNDS DB
HCI ;

DB IRM
HCI
CBO ;

ASAD PIO
HCI
IRM ;

GR MI
MTOS IP
;

MTOS GR
CNDS IP
;

```
MTOS CNDS
     HCI
     RTES ;
HCI  IRM  DB
     AF   ;
DB   IRM
     HCI
     CNDS ;
MTOS GR   IP
     AI   ;
DB   GR
     HCI
     CNDS ;
DB   HCI
     IRM
     CBO  ;
AI   MI
     MTOS
     CNDS ;
ASAD PIO
     HCI
     IRM  ;
RTES IDE
     MTOS DP
     ;
DB   HCI
     CNDS AF
     ;
HCI  CNDS DB
     OR   ;
CNDS MI
     MTOS
     CBO  ;
GR   AI
     MTOS
     HCI  ;
IDE  AF
     SIM
     CLTI ;
AF   GR   DB
     CLTI ;
ASAD FR
     CBO  DB
     ;
CNDS MTOS
     IRM  SE
     ;
ASAD PIO  DB
     HCI  ;
CNDS IRM
     HCI  DB
     ;
NN   AI   DP
     SIM  ;
NN   CNDS
     MTOS GR
     ;
DB   ASAD
     SIM
     CBO  ;
CNDS DB
     RTES
     MTOS ;
IRM  AF
     HCI  DB
     ;
ASAD CBO
     IRM  DB
     ;
DB   CNDS
     RTES
     MTOS ;
RTES CNDS
     CLTI
     SIM  ;
ASAD DB   IP
     OR   ;
DB   IRM
     HCI
     CBO  ;
DB   HCI
     ASAD
     IRM  ;
CNDS HCI  OR
     IRM  ;
IRM  HCI  DB
     PIO  ;
CNDS IP   AF
     SIM  ;
MTOS GR   IP
     IDE  ;
AI   MTOS
     CNDS NN
     ;
HCI  IRM  MI
     CBO  ;
MTOS RTES IP
     MI   ;
```

```
ASAD GE
     CBO
     IDE ;
SIM  MTOS GR
     CNDS ;
CNDS DB
     IRM
     ASAD ;
GR   IP
     CNDS
     HCI ;
ASAD CBO DB
     IRM ;
DB   RTES MI
     GR ;
CNDS MTOS
     IRM
     HCI ;
ASAD PIO DB
     IRM ;
HCI  GR IP
     MI ;
IP   OR DB
     HCI ;
AI   CNDS DB
     OR ;
AF   DB OR
     IP ;
HCI  IRM DB
     CBO ;
SIM  IP
     RTES GR
     ;
```

```
ASAD PIO
     HCI DB
     ;
DB   CNDS AF
     IP ;
AF   CNDS DB
     HCI ;
GR   CNDS IP
     DB ;
CNDS IRM OR
     DB ;
IP   CLTI DB
     AI ;
AI   DP GR
     IP ;
DB   AF
     HCI OR
     ;
CNDS OR
     ASAD
     IRM ;
ASAD CBO DB
     HCI ;
HCI  CNDS DB
     AF ;
OR   CNDS DB
     ASAD ;
ASAD CBO DB
     IRM ;
ASAD HCI DB
     OR ;
AF   CNDS DB
     IRM ;
```

```
ASAD CBO
     HCI MI
     ;
GR   CNDS AF
     IRM ;
DB   HCI AF
     IRM ;
OR   IRM DB
     HCI ;
GR   IP MI
     MTOS ;
HCI  IRM DB
     CBO ;
GR   IDE DB
     RTES ;
DB   GR
     RTES
     MTOS ;
OR   AF DB
     HCI ;
PIO  ASAD
     IRM
     HCI ;
HCI  DB SE
     GR ;
GR   IDE
     RTES IP
     ;
FR   AI NN
     IP ;
IP   GR MI
     NN ;
```

ASAD CBO
HCI
IRM ;

GR IP
HCI MI
;

ASAD PIO
HCI
IRM ;

GR NN
CNDS DP
;

CNDS MTOS
CLTI MI
;

AI DP
MTOS IP
;

CNDS ASAD
IDE DP
;

ASAD PIO NN
SE ;

ASAD PIO DB
HCI ;

MI GR
MTOS IP
;

MTOS DB
RTES
CNDS ;

IRM DB ;

DB HCI IP
SIM ;

IP GR
MTOS
CNDS ;

HCI AF DB
IRM ;

DB ASAD IP
OR ;

ASAD DB
CNDS
IRM ;

CNDS DB
HCI
ASAD ;

HCI CNDS DB
IRM ;

ASAD CBO
IRM
HCI ;

CNDS AF
HCI
ASAD ;

HCI CNDS DB
OR ;

DB OR
HCI
IRM ;

OR CNDS DB
IRM ;

MTOS CNDS MI
DB ;

IRM HCI DB
AF ;

ASAD IRM
HCI DB
;

IRM CNDS DB
OR ;

MI CNDS
MTOS
RTES ;

AI NN IP
GR ;

GR IDE DB
RTES ;

MTOS GR IP
MI ;

IP MTOS
CNDS GR
;

AF CNDS DB
IRM ;

IRM DB
ASAD
CBO ;

DB HCI
MTOS
CNDS ;

IP GR
HCI
RTES ;

HCI MI
IRM AF
;

ASAD PIO DB
HCI ;

CNDS MTOS DB
IP ;

ASAD PIO DB
HCI ;

IP CNDS GR
DB ;

IRM DB
HCI
CBO ;

DP GR MI
MTOS ;

IP CLTI
MTOS AI
;

ASAD CBO
IRM
HCI ;

HCI DB
IRM
CNDS ;

CNDS DB
HCI AF
;

CNDS MI DP
IP ;

GR CNDS MI
IP ;

MTOS CNDS DB
CLTI ;

DB SE
HCI GR
;

OR IP DP
AI ;

GR HCI IP
DB ;

MTOS CNDS AF
DB ;

FR CNDS AI
GR ;

IP MTOS
CNDS GR
;

GR IP MI
MTOS ;

CNDS SIM AI
DP ;

GR CNDS IP
NN ;

ASAD PIO DB
IDE ;

CNDS IDE
CLTI DB
;

GR CNDS IP
IRM ;

ASAD CBO
IRM
HCI ;

AF HCI DB
IRM ;

ASAD PIO
HCI
IRM ;

HCI CNDS DB
OR ;

OR HCI DB
IP ;

CNDS DB
HCI AF
;

DB CNDS
IRM
HCI ;

ASAD CBO
IRM
HCI ;

GR HCI
CNDS IP
;

ASAD CBO
IRM
HCI ;

ASAD CBO
IRM
HCI ;

GR MTOS
RTES
SIM ;

GR SIM IP
RTES ;

HCI CBO IP
IRM ;

GR CNDS
MTOS
IRM ;

NN CNDS
MTOS
SIM ;

GR SIM IP
AI ;
GR SIM
MTOS
IDE ;

OR DB
CNDS IP
;
MTOS GR NN
RTES ;
ASAD CBO
HCI MI
;
CNDS DB AI
HCI ;
HCI CNDS
IRM
CBO ;
CNDS DB
IRM OR
;
IP IRM MI
CNDS ;
MTOS CNDS MI
RTES ;
OR CNDS DB
IRM ;
CNDS DB OR
IRM ;
ASAD CBO
IRM OR
;

CNDS HCI
CBO DB
;
MTOS DB IP
CNDS ;
CNDS AF DB
IRM ;
ASAD HCI AF
IRM ;
HCI CNDS DB
OR ;
ASAD PIO
HCI
IRM ;
SE AF DB
IRM ;
ASAD DB
IRM OR
;
MTOS CNDS IP
DB ;
HCI IP GR
DB ;
CNDS DB
CBO IP
;
MTOS DB
HCI MI
;
MTOS HCI
IRM MI
;

ASAD PIO
IRM DB
;
IP SIM NN
GR ;
HCI AI
CNDS
SIM ;
HCI IRM DB
PIO ;
ASAD CBO
HCI DB
;
HCI IP
MTOS DB
;
MTOS CNDS
IDE DB
;
IRM HCI DB
CBO ;
HCI SIM AF
DB ;
ASAD PIO
HCI
IRM ;
ASAD PIO
IRM
HCI ;
IP IDE
CNDS
RTES ;
FR DB AF
HCI ;

```
SE      CNDS
        IDE
        MTOS ;
ASAD    PIO
        HCI
        IRM     ;
ASAD    SE
        HCI     DB
        ;
CNDS    MTOS    AI
        DP      ;
ASAD    CBO     DB
        IRM     ;
HCI     CNDS    DB
        OR      ;
DB      IRM     IP
        OR      ;
DB      CNDS
        IRM
        HCI     ;
CNDS    DB
        SIM     IP
        ;
DB      CNDS    OR
        AF      ;
DB      IRM     IP
        OR      ;
ASAD    DB      OR
        HCI     ;
HCI     IRM
        CBO     SE
        ;
RTES    DB      IP
        CNDS    ;
CNDS    ASAD    DB
        OR      ;
OR      CNDS    DB
        IRM     ;
ASAD    CBO
        IRM
        HCI     ;
HCI     DB
        IRM     AF
        ;
CNDS    AF      DB
        HCI     ;
ASAD    CBO     DB
        IRM     ;
CNDS    DB
        IRM     AF
        ;


}
```

Appendix D: Master Timetable Files

BC4.MST

```
    ---------------------------------------------------------
    |DAY 0                  |DAY 1                  |
    ---------------------------------------------------------
    |0   |1   |2   |3   |4   |0   |1   |2   |3   |4   |
-------------------------------------------------------------------
|PVB |OR   |IP   |CBO |    |CLTI |DB   |    |    |    |CLTI |
|    |BLUE |RED  |RED |    |D116 |D118 |    |    |    |D116 |
-------------------------------------------------------------------
|DKB |AF   |SIM  |OR   |ASAD |MTOS |RTES |IRM  |MI   |AI   |SIM  |
|    |D116 |GREEN|D118 |D118 |BLUE |BLUE |D116 |GREEN|GREEN|BLUE |
-------------------------------------------------------------------
|CM  |MTOS |MI   |RTES |CNDS |DP   |    |GR   |DB   |AF   |IDE  |
|    |D118 |D118 |D116 |BLUE |D118 |    |BLUE |BLUE |D118 |RED  |
-------------------------------------------------------------------
|PG  |ASAD |IRM  |IP   |FR   |HCI  |    |AI   |PIO  |CNDS |HCI  |
|    |GREEN|BLUE |BLUE |GREEN|GREEN|    |RED  |D116 |RED  |D118 |
-------------------------------------------------------------------
|PC  |NN   |SE   |PIO  |SE   |    |NN   |IDE  |GR   |CBO  |DP   |
|    |RED  |D116 |GREEN|D116 |    |GREEN|GREEN|D118 |BLUE |GREEN|
```

BC4_2.MST

```
    ---------------------------------------------------------
    |DAY 0                  |DAY 1                  |
    ---------------------------------------------------------
```

|0 |1 |2 |3 |4 |0 |1 |2 |3 |4 |

|PVB | |CBO |CNDS |SIM |GR |MI | | |AF |AF |

| | |D118 |D116 |GREEN|RED |BLUE | | |GREEN|D118 |

|DKB |MI | | |MTOS | |IRM |NN |HCI | |IP |

| |RED | | |D118 | |RED |D116 |RED | |BLUE |

|CM |DB |CNDS |CBO | | |IDE |IP |SIM |ASAD | |

| |GREEN|BLUE |BLUE | | |D118 |D118 |D116 |D118 | |

|PG |GE | |PIO |HCI |DB |RTES |IDE |OR |OR |ASAD |

| |BLUE | |GREEN|BLUE |BLUE |D116 |GREEN|GREEN|D116 |GREEN|

|PC |AI |PIO |DP |GE |DP |AI |IRM |MTOS |GR |RTES |

| |D118 |GREEN|RED |D116 |GREEN|GREEN|BLUE |BLUE |BLUE |D116 |