GENETIC ALGORITHMS FOR TIMETABLE GENERATION

A

THESIS

Presented to the Department of

Computer Science

African University Of Science And Technology

In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

By

Walusungu Gonamulonga Gondwe

Abuja, Nigeria

December 2014

GENETIC ALGORITHMS FOR TIMETABLE GENERATION

By

Walusungu Gonamulonga Gondwe

A THESIS APPROVED BY THE COMPUTER SCIENCE DEPARTMENT

RECOMMENDED: .....
Supervisor, Prof. Lehel Csato

.....
Head, Computer Science Department

APPROVED: .....
Chief Academic Officer

.....
Date

Abstract

Timetabling presents an NP-hard combinatorial optimization problem which requires an efficient search algorithm. This research aims at designing a genetic algorithm for timetabling real-world school resources to fulfil a given set of constraints and preferences. It further aims at proposing a parallel algorithm that is envisaged to speed up convergence to an optimal solution, given its existence. The timetable problem is modeled as a constraint satisfaction problem (CSP) and a theoretical framework is proposed, which guides the approach used to formulate the algorithm. The constraints are expressed mathematically and a conventional algorithm is designed that evaluates solution fitness based on these constraints. Test results based on a subset of real-world, working data indicate that convergence on a feasible (and optimal/Pareto) solution is possible within the search space presented by the given resources and constraints. The algorithm also degrades gracefully to a workable timetable if an optimal one is not located. Further, a SIMD-based parallel algorithm is proposed that has the potential to speed up convergence on multi-processor or distributed platforms.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Prof. Lehel Csato, for the guidance rendered during my research. I also would like to thank the Head of Computer Science Department at AUST, Prof. Mamadou Kaba Traore for working tirelessly to provide us with the necessary direction during the MSc program. My gratitude also extends to all faculty and visiting professors for the lessons, both inside and outside the lecture room.

I am also grateful to my employer, Chancellor College of the University of Malawi, for granting me a study passage and for all the financial support rendered during my time as a postgraduate student. In the same light, I also thank the Directorate of Technical Cooperation in Africa (DTCA) and the African Development Bank for the role they played in funding my scholarship.

Last but not least, thanks to the whole AUST community for the friendships, partnerships, lessons and support.

You are all appreciated.

DEDICATION

To God be all the glory for seing me through. To my family, this is yet another milestone for us. I could not have made it this far without your support. I love and appreciate you. To all my friends in Malawi, Nigeria and beyond who supported me during my studies, I will forever be grateful.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Research Context

Timetabling is a well known NP-Hard combinatorial optimization problem that has not yet been solved in polynomial time using a deterministic algorithm. Several techniques are used to solve the timetabling problem including manual construction, search heuristics (tabu search, simulated annealing and genetic algorithms), neural networks and graph colouring algorithms. Most timetabling problems have application specific peculiarities and hence, the use of domain-specific patterns together with most of the aforementioned techniques to improve computational efficiency is not uncommon (see [9], [18]).

However, despite the considerable success of the aforementioned techniques, the timetabling problem still remains a challenge especially when dealing with large data sets with many constraints. This research investigates the suitability of using genetic algorithms (GAs) to locate an optimal school timetable in a large search space. Our work is set apart from previous studies by the prior development of a theoretical framework as a basis for convergence of the proposed algorithm. In addition, our investigation targets real-time data sets governed by potentially conflicting constraints, a goal that is seldom seen in most similar past research efforts. In particular, the work endeavours to achieve the following objectives:

1. Explore a theoretical framework for using GAs for timetable construction
2. Design and prototype a genetic algorithm to solve the timetabling problem and test it using a trial dataset
3. Propose a distributed timetabling GA based on the results of objective (2)

1.2 Methodology

The research sets out by modeling the timetable problem and proposing a theoretical framework as a basis for the convergence of the proposed algorithm. Secondly, a serial algorithm is designed and prototyped to furnish a timetable from a subset of real-world university student data with the aim of investigating the effects of various parameters on its convergence behaviour. This is followed by application of the algorithm to a bigger data set to investigate its scalability properties. Finally, a parallel/distributed GA is proposed with the goal of exploiting current distributed/parallel architectures to enhance performance when applied to real-world data sets.

The rest of this paper is organised as follows: The next section (2) briefly introduces critical timetable and GA concepts with the aim of laying a foundation for understanding subsequent discussion. The section also includes a review and analysis of literature on GA-based scheduling algorithms and heuristics. The analysis relates the proposed research topic to the state of the art and endeavours to situate it in the context of already existing work. Section 3 documents and discusses the theoretical framework and the proposed genetic algorithm and analyses the results obtained from the prototype and a test data set. The section concludes with details of the proposed distributed genetic algorithm. Finally, Section 4 summarizes the results of the analysis, explores the limitations of the algorithm and suggests directions for future work.

Chapter 2

State of the Art

2.1 The Timetable Problem

A school timetable is a combinatorial optimization problem set up as follows: Given a set of resources (lecture rooms, labs etc), and a set of student groups, along with a set of teachers, how can these three entities be arranged in time so that given constraints are met and optimality conditions are also satisfied. Perhaps the most complex timetables are found in universities where the number of students and lecturers is large and enrollment into courses is guided by route maps. In such settings, allocation of courses and their respective lecturers to time slots and rooms requires that a set of potentially conflicting constraints be satisfied.

Most literature recognize two categories of constraints; *hard* and *soft* constraints. The former are those that must be satisfied for the timetable to be feasible (applicable) while the latter may be satisfied to enhance the quality of the timetable. Examples of hard constraints include conflicts or clashes (a lecturer cannot teach more than one course at the same time, students can only attend one class at a time, a room cannot be allocated to two classes at the same time) and capacity (a class must be allocated a room with enough capacity). Soft constraints may include administrative needs or individual/departmental preferences. Examples include class location and timing preferences, departmental room allocation preferences and class spacing.

2.2 Genetic Algorithm Concepts

Genetic algorithms are a stochastic search mechanism that uses principles of natural selection to evolve and search for solutions to complex combinatorial problems . GAs are typically initialized with an initial random set of potential solutions (typically called a

population of chromosomes) that evolve through iterative application of *genetic operators* until a given optimal value or a maximum number of generations is observed. The most common genetic operators include *selection*, *crossover (recombination)* and *mutation*. Selection is the mechanism of choosing parents that will produce the next population. Selected parents are allowed to crossover (mate or recombine) to produce offspring. Mutation is the introduction of minute random alterations to a chromosome to induce diversity into the population.

More advanced GAs use additional concepts such as *elitism* and *migration* that allow for more robust searching. Elitism ensures that the best chromosome is maintained between successive generations by *artificially* inducing it. In multi-population, distributed/parallel GAs, migration allows exchange of individuals among isolated populations in order to introduce new genetic material, in effect, moving the isolated searches to different regions of the global search space. A fitness function based on the optimization objective(s) is typically used to evaluate a chromosome's *fitness value*, which in turn determines the chromosome's suitability for reproduction (crossover) and survival into the next generation.

Several factors affect the efficiency, convergence time and overall performance of a GA. These include the chromosome encoding scheme, mutation rate, crossover rate, selection mechanism and migration parameters in distributed GAs. The crossover rate or probability is the likelihood that two parents will mate and produce offspring after selection. A GA's mutation rate indicates the probability that an offspring will mutate after crossover. The selection mechanism determines how the algorithm selects parents to reproduce offspring for the next generation and also determines the population's *selection pressure*. High pressure implies that individuals with better fitness values have higher chances of being selected for crossover than those with lower values. Low pressure implies uniform probability for selection across the population. Setting the optimal pressure ensures that the algorithm does not converge prematurely and that it avoids local optima. Several selection mechanisms have been proposed but the two most commonly used are *fitness proportionate methods* and *tournament-based methods*.

For distributed GAs, additional factors that have a bearing on the algorithm's efficiency include subpopulation number and size, migration rates, topology and migration timing.

2.3 Literature Review

This section reviews several influential works in the areas of GAs and distributed/parallel GAs as applied to school timetabling and scheduling in general. The approach used aims at critically examining each piece of literature in the context of the research objectives

outlined in the introduction. Particularly, the survey analyzes the extent to which GAs have been applied to real-world school timetabling problems and also developments made in terms of leveraging current hardware and software technologies to parallelize genetic algorithms in general. It is envisaged that by the end of this survey, the research topic, approach and objectives will have been justified and situated in the context of existing related work.

Considerable research on scheduling/timetabling using genetic algorithms has been conducted, despite minimal literature documenting cases of satisfactory applications to real-world, large data sets. Corne and Ross [19] explored a successful arbitrary lecture timetabling approach using (serial) evolutionary algorithms. Their approach was applicable to data sets of considerable sizes and scaling up was left to further research. Burke et al. [6] proposed a hybrid genetic algorithm for highly constrained timetabling problems to solve a university exam timetabling problem. They proposed generation of an initial population of feasible timetables using graph coloring methods and further refinement of these solutions using genetic operators. The algorithm was tested successfully on a randomly generated test problem but took considerably long to converge on real world data. Similar approaches can be seen in [9], [15] and [17].

Colorni et al. [9] further explore the problem of generating infeasible solutions after application of genetic operators (mutation and crossover). They propose repair strategies, heuristics and filters to guide the GA and allow it to only explore promising sections of the search space. Using data from an Italian high school, they were able to produce feasible timetables of better quality compared to handmade ones and those produced by simulated annealing. However, their results proved to be inferior to solutions obtained using tabu search and test cases were reported to take 8 hours to complete.

Another effort of comparable success was the work of Lukas et al. [18] who conducted a case study using a combination of GA and a search heuristic to solve a timetabling problem for a university. The GA was used to salvage feasible course combinations and sequences which in turn were fed into a search heuristic that allocated the course combinations to time slots. Experimental results showed a successful generation of a feasible timetable for a representative real data set. However, the approach taken used a GA as a helper tool for generating feasible course groups rather than a dominant generator of feasible timetables. In addition, other resources (such as rooms) were not taken into account and inclusion was left to further research.

Other works have endeavored to employ advanced genetic operators to improve the performance and convergence time of GA-based timetabling algorithms. Beligiannis et al. [3] proposed an adaptive GA approach to high-school timetabling in Greece. The approach assigned weights to constraints to allow need-based dynamic reconfiguration. The mutation rate was incremented with each successive generation to avoid convergence on

local optima and elitism was used to preserve the best individuals between generations. Notably, their approach favored generation of an initial population of semi-feasible individuals over the usage of repair strategies, purporting that the latter could cause the GA to be trapped in local optima, which is in contrast to the approach taken by Colorni et al. in [9]. Similarly, Sigl et al. [5] solved the timetable problem by applying improved genetic operators with significant improvements in performance. They used binary encoding of timetable chromosomes and used improved tournament selection with modified uniform crossover designed to minimize constraint violation between consecutive generations. The algorithm was tested on both large and small data sets with noticeable improvements in both clash minimization and convergence time.

The literature reviewed reveals that the amount work done by previous researchers to improve genetic algorithms in general and application to the timetabling problem in particular has not been matched by efforts to design GA-based approaches that can leverage current distributed/parallel architectures. Most research has dwelled on designing parallel/distributed GA approaches with no application to substantial real world timetabling problems. Perhaps the most notable effort to parallelize a GA specifically for timetabling purposes is the work of Abramson and Abela [1]. Their work explored areas in a basic genetic algorithm that could easily be parallelized on a shared memory multiprocessor with minimal synchronization (inter-process communication) overhead. They noted that selection and crossover were the two most promising areas for parallelization and distributed the crossover operation among several worker threads. Experimental results based on 9 data sets showed a maximum speedup of 9.3 on 15 processors. However, despite recording considerable speedup, their work only involved basic exploitation of instruction level parallelism besides being based on an outdated multiprocessor platform.

Another research endeavor worth noting is the work of Pospichal et al. [16]. Their research focused on mapping multi-population GA (using the *island model* as described by the authors) for execution on a Graphics Processing Unit (GPU) through the Compute Unified Device Architecture (CUDA) model. Standard GA benchmarking algorithms were used to compare speedups on two types of GPUs; the GTX 285 (30 multiprocessors / 240 cores) and GTX 260-SP216 (27 multiprocessors / 216 cores) against the Intel Core i7 92. Average speedups of seven thousand were observed without compromising the quality of results. This promises a great potential for the timetabling problem to be efficiently solved by employing similar techniques and technologies.

Other notable literature on parallel GAs is seen in [2], [7] and [16]. El-daily et al. [2] compared three different DGA approaches based on their ability to maintain diversity in all subpopulations post migration. The paper surveyed *DGA with Diversity Guided Migration*, which migrated a special individual and replaced clones in adjacent subpopulations; *DGA with Automated Adaptive Migration* and *DGA with Bi-coded Chromosomes*.

The last two approaches replaced the worst individuals in subpopulations during migration. Test results showed that DGA with Diversity Guided Migration was superior to the other two approaches in the majority of test cases. It is worth noting that the comparison criterion used was not based on the fundamental goals of distributed approaches such as speedup and reduced convergence time. In addition, their work was generalized and did not have any direct application to the timetabling/scheduling domain.

Similar work was done by Cantú-Paz [7] who conducted a survey of parallel GA-based algorithms and classified them into four major categories; *Master-slave (global) Scheme*, a single population scheme with distributed computation of fitness function on slave processors; *Single Population Fine Grained Scheme* involving a single population with selection and crossover occurring within restricted neighborhoods; *Multiple-population (multi-deme) Coarse Grained Scheme*, which uses multiple populations distributed to multiple processors with possible migration (communication) and the *Hierarchical scheme*, a combination of the three schemes to produce a hierarchical, parallel approach.

Perhaps the scheme with the most potential for applicability is the multiple-population scheme. The author notes that the migration rate, frequency and communication topology in the multiple-deme scheme affects the efficiency and convergence time of the algorithm and has implications on the quality of the optimal solution. In addition, the author also purports that the number of demes (isolated populations) affects overall algorithm efficiency and that there exist a theoretical optimum number of demes for each problem category over which the communication overhead of the parallel algorithm begins to overshadow the speed up due to parallelism. Further analysis of this scalability issue led by the same author can be seen in [8].

In addition to the main literature which has a direct bearing on the topic of research, there are other publications that focus on specific aspects of genetic algorithms. Of some significance to the topic at hand is the work of Xie and Zhang [23], which focused on adaptive tuning of selection pressure when tournament selection is used in a GA. In particular, they distinguished purely stochastic selection mechanisms (low selection pressure) from guided mechanisms (high selection pressure) and examined a novel adaptive selection mechanism that adjusts the selection pressure during the course of evolution based on what they termed as a *Fitness Rank Distribution.* They observed that populations undergo different fitness distributions during evolution (*uniform, reverse-quadratic, random and quadratic*) and that that tournament size itself is not a sufficient factor to consider when tuning selection pressure and that more intelligence is required in the course of execution. It was concluded that the intelligent selection mechanism yields better results for their GA as it adapted the selection pressure to the changing fitness distribution of the population.

Chapter 3

Contribution and Analysis

3.1 Theoretical Framework

3.1.1 Model Formulation

The timetabling problem will be modeled as a *Constraint Satisfaction Problem* (CSP). It is an NP-hard problem with no known polynomial-time algorithm and we will work under the assumption that $\mathsf{P} \neq \mathsf{NP}$ and, therefore, it may not be solved efficiently in polynomial time using a deterministic algorithm. This, in turn implies solving using other means e.g. search heuristics. Given a timetabling problem T, then modeled as a CSP:

$$\mathsf{T} = \langle \mathsf{X}, \mathsf{D}, \mathsf{Q}, \mathsf{H}, \mathsf{S} \rangle$$

where:

- X is a set of four variables; L, C, R and P for *lecturer*, *course*, *room* and *time slots (or periods)* respectively
- $\mathsf{D} = \{\mathsf{D}_l, \mathsf{D}_c, \mathsf{D}_r, \mathsf{D}_p\}$ is a finite set of domains of all variables in X
- H is a finite set of hard constraints i.e. relations specifying the *admissible or feasible* combinations of values over sets in X
- $\mathsf{Q} \subset \mathsf{H}$ is a set of relations that partition D_c into disjoint and collectively exhaustive sets of student groups, G, such that:

  $\forall g \in \mathsf{G} \wedge \forall a, b \in \mathsf{D}_c, (a, b \in g \wedge a \neq b)$ implies a and b do not have a common student and are not taught by a common lecturer
- S is a finite set of soft constraints i.e. relations specifying the *preferable* combinations of values over sets in X

Note that this extends the classical definition of a CSP by partitioning the set of constraints into two disjoint sets, H and S.

Given this definition, if we let V denote all possible variable combinations over their respective domains, given by the Cartesian product:

$$V = \prod_{x \in X} x \tag{3.1}$$

then we can define a *feasible* timetable, τ as a subset of V that at least satisfies H. The *quality* of τ is determined by the extent to which it satisfies S.

3.1.2 Nature of Search Space

The cardinality of V is given by:

$$|V| = \prod_{x \in X} |x| \tag{3.2}$$

If we let w denote the total number of contact hours for all courses, then the number of all possible timetables is given by the set of w-ary[1] subsets of V with cardinality:

$$\binom{|V|}{w} = \frac{|V|!}{w!(|V| - w)!} \tag{3.3}$$

This presents a large (factorial) finite search space for variable sets of moderate sizes (e.g. a typical university has more than 300 courses, 100 classrooms, 200 lecturers and 40 weekly time slots).

3.1.3 Convergence of a GA-based Timetable Search

Given the complex nature of real-world constraints, it is likely that $\nexists \tau \subset V$ such that τ is an optimal solution that is complete and consistent w.r.t both H and S. A fitness function should be defined based on H and S to evaluate the feasibility and quality of candidate solutions.

Ideally, in the case where no feasible solution exists, the GA should settle for the so-called *Pareto solution*, an optimal (and usually infeasible) allocation where further optimization of one constraint negatively affects one or more other constraints. The following conditions are sufficient for non-existence of a feasible solution:

[1] Here we assume that one time slot is equivalent to one contact hour and this can be extended, without loss of generality, to cover scenarios where this condition does not hold

Condition 1: $w > |D_p| \times |D_r|$. i.e. the total number of contact hours (w) for all courses is greater than the number of all possible room-time slot combinations.

Condition 2: $|G| > |D_p|$ i.e. the number of student groups is greater than the number of available time slots. This is can be shown by noting that given any $a, b \in D_c$ and if $a \in g_i$ and $b \in g_j$ where $g_i \cap g_j = \emptyset$, then a and b cannot be allocated the same time slot since they share either a student or lecturer. But each group $g \in G$ contains at least one course, therefore if $|G| > |D_p|$ then the timetable needs more than $|D_p|$ timeslots to be feasible.

Condition 3: $\sum_{g \in G} g_{max} \leq |D_p|$, where g_{max} is the maximum number of contact hours among the courses in each group. This can be shown by noting that the resulting timetable should allocate at least g_{max} hours for each $g \in G$. This is a direct consequence of the fact that only one contact hour for a course can occupy a given time slot even in different locations. Thus, a course with x contact hours requires at least x separate timeslots. We can then conclude in a similar fashion to *condition 2* that the total sum of the g_{max} should be at most equal to $|D_p|$.

To increase the likelihood of convergence to a globally optimal solution in polynomial time, the initial population should be restricted to a region that is most promising. Two theories will constitute a framework for convergence of the proposed GA:

1. *Holland's Schema Theorem* [14]
2. *Markov Chain Analysis of Canonical Elitist-based GAs* [22, 13, 20]

Holland [14, p. 15] defined schema as the generalization (or common pattern) of a set of chromosomes, and their associated operators. In basic terms, he observed that certain sets of chromosomes have common patterns of alleles and if these patterns are of above-average fitness, they are most likely to be maintained and, in turn, produce above-average offspring. The schema theorem as generalized by Goldberg's in [11] takes the following form:

$$m(H, t+1) \geq m(H, t)\theta(H, t)[1 - \epsilon(H, t)] \tag{3.4}$$

where $m(H, t)$ is the number of instances of schema H at time t, $\theta(H, t)$ is the ratio of average fitness of instances of schema H to the average fitness of the whole population at time t and $\epsilon(H, t)$ is the '*error factor* 'that accounts for the stochastic disturbances to H introduced by genetic operators.

The proposed GA uses the following two strategies to ensure introduction of above-average schemas in the timetable gene population:

(i) Prior knowledge of Q *and the student groups that follow from partitioning (*G*)*

(ii) Prior assignment of lectures to courses before optimization

The first strategy can be achieved by using an auxiliary partitioning algorithm or heuristic (e.g. graph coloring) prior to running the GA. The latter is easily achievable since most school schedules fix lecture-course assignments before generating timetables.

To further enhance the likelihood of convergence, the GA will be formulated within the framework of Markov chain analysis of GA convergence that suggests sufficient conditions that guarantee convergence to an optimal solution. It should be noted that GAs are a class of randomized search heuristics that possess the *Markov property* i.e. evolution in GAs is *memoryless* and *stochastic*. Given an initial population, θ_i, the probability of another population θ_j being generated from θ_i (denoted by $P(\theta_i|\theta_j)$) depends only on the state of θ_i. Markov chain analysis of the convergence behavior of canonical GAs using transitional matrices of conditional probabilities can be seen in [22, 13, 20]. This theory applies to canonical GA whose transitions are triggered by genetic operators of selection, mutation and crossover. The analysis also proves that convergence is almost always guaranteed if the GA implements *elitism* or any variation of it i.e. if the best individual is always guaranteed selection into the next generation. Our approach employs canonical genetic operators with repair and elitist strategies.

3.1.4 Genetic Operators

The two theories that form the basis of our work are founded on canonical GA operators, namely, (uniform) selection, crossover and mutation. Further, as already seen, elitism increases the odds of locating an optimal solution in a multi-dimensional, noisy search space. It is also worth noting that the generalized schema theorem as given in [11] is independent of the choice of genetic operators. This research employs the following operator strategies:

Selection Mechanism

Given the nature of the timetabling problem, *tournament selection* presents a more suitable selection method as opposed to both fitness proportionate and rank-based mechanisms. Given a population of size N, tournament selection works by selecting n random individuals ($n \leq N$) from which two parents are selected for crossover in a tournament fashion. This is repeated until the required number of offspring for the next generation is satisfied. This technique eliminates the need for fitness scaling techniques that are used in the latter two schemes. Scaling is done to prevent premature convergence when there are large gaps in fitness values between individuals in the early stages of evolution

and to prevent stagnation when there is little variance in fitness values. In addition, tournament selection is highly amenable to parallelization which can be exploited in a parallel/distributed GA.

A critical parameter to consider when applying tournament selection to the timetabling problem is the tournament size (n), which determines the behavior of the selection scheme. At any point during evolution, selection pressure (p_s) is directly proportional to n. Our work will adopt the formula used by Blickle and Thiele in [4] that estimates[2] the dimensionless value of p_s by:

$$p_s \approx \sqrt{2(\ln(n) - \ln(\sqrt{4.14 \ln(n)}))} \tag{3.5}$$

This estimation will be utilized in this work where necessary to dynamically adjust p_s across generations (by varying n) to allow more control over the search process.

Crossover and Mutation

Crossover and mutation operators will be defined with repair strategies to further guide the search towards an optimal solution and increase the chances of polynomial-time convergence. Mutation and crossover rates will be modifiable between evolutions to allow adaptation of the algorithm to different 'environments'.

Elitist Strategy

Two important factors to consider when implementing elitism are *(i) elite size* and *(ii) replacement strategy*. Elite size is the number of fittest chromosomes maintained between consecutive generations. Elite replacement strategy defines how an elite chromosome (or chromosomes) from a previous generation is inserted into a new population. Two approaches are commonly used; the first one replaces a *random* chromosome in the current generation with the elite of the previous one and the second one replaces the *worst* chromosome. Our approach uses an elite size of 1 (one) and leaves the choice of replacement strategy to the implementer to allow flexibility.

[2]This assumes a Gaussian (Normal) distributed population with mean 0 and standard deviation 1 ($G(0,1)$) but the authors purport that it can still be used as an approximation for populations that are not normally distributed

3.2 Proposed Algorithm

3.2.1 Serial Algorithm Overview

The algorithm will search for a solution in two phases. The first phase allocates student groups to different rooms with the following objectives:

- Eliminate or optimize (minimize) student/lecturer clashes (i.e. no two or more courses belonging to different student groups are scheduled at the same time)
- All allocations satisfy the capacity constraints i.e. all classes in the group should optimally fit in the allocated room
- All required contact hours for the group are satisfied (allocated)
- Eliminate or minimize room clashes

Note that this phase assumes prior availability of student group information in addition to courses, rooms, lecturers and time slots. Allocation of specific time slots in each room for each group will be done in the second phase using a secondary GA to satisfy the following objectives:

- Allocate peer courses for each group based on the optimized group allocation matrix from first phase
- Optimize group timetables to satisfy departmental/administrative preferences

The algorithm searches within the boundaries of the following constraints:

Hard constraints (H)

- All required contact hours for each course are scheduled
- No clashes i.e. no student or lecturer can be in more than one class. Here class means a combination of course and room and time slot (period)
- Each course is allocated a room that is available at that particular time
- Each course must be scheduled in a room with sufficient capacity
- A lecturer should only be assigned the courses he/she is eligible to teach

Soft constraints (S)

- Departmental rooms should be given priority when scheduling a course. This includes lab sessions, which require special rooms with appropriate equipment

- All courses that require multiple consecutive time slots should be scheduled appropriately e.g. lab sessions
- Students/lecturers should be given breaks between classes (a good spread of classes)

3.2.2 Phase 1: Group Matrix Optimization

Chromosome Encoding

Let C be the set of all courses, R be the set of all rooms, P be the set of all timeslots (or periods) and G be the set of all student groups. If we let w_i denote the total sum of contact hours for all courses in group $g_i \in G$ and let $m = |G|$ and $n = |R|$ and $k = |P|$ denote the cardinalities of the respective sets, then we can represent individual group allocation chromosome as an $m \times n$ matrix of the form:

$$A = [a_{ij}], \text{where } a_{ij} \in \mathbb{Z} \cap [0, w_i], 0 < i \leq m, 0 < j \leq n \tag{3.6}$$

Thus $a_{ij} = x$ indicates student group i is allocated x hours in room j. Therefore, A represents possible allocations of student group contact hours to rooms[3]. With this encoding scheme, we can express the the hard constraints mathematically as follows:

Clashes:

Given that student groups are non-overlapping (i.e. peer courses in each group can be scheduled at the same time in different spaces) then the three requirements to ensure no clashes are:

1. Each column of A should add up to at most the total number of available time slots for the given period (e.g. week). If the total exceeds this number then room j contains a clash of two or more classes, i.e.

$$\sum_{i=1}^{m} a_{ij} \leq k, \forall j \in \mathbb{Z} \cap [1, n] \tag{3.7}$$

2. The total number of concurrent classes in each group (each row vector) should be bound by the number of available class rooms (with enough capacity). Given our encoding scheme, this can be mathematically expressed as:

[3]The matrix representation lends itself to almost effortless parallelism, a factor that will be exploited in the design of the distributed algorithm

$$\sum_{j=1}^{n} a_{ij} \leq nk, \forall i \in \mathbb{Z} \cap [1, m] \tag{3.8}$$

Note that equation 3.8 should ideally be a strict equality instead of an inequality to preserve the exact number of required weekly hours for all courses in a group. This is the approach that will be taken in our implementation in order to restrict the search to a 'promising' solution space. Therefore, in the implementation, the condition changes to:

$$\sum_{j=1}^{n} a_{ij} = w_i, \forall i \in \mathbb{Z} \cap [1, m] \tag{3.9}$$

where w_i is the total number of hours for group i

3. Assuming *condition 3* for non-existence of a feasible timetable is not met (see section 3.1.3), the total of the maximum allocated hours for each group should not exceed the number of available time slots. Mathematically,

$$\sum_{i=1}^{m} max(a_i) \leq k \tag{3.10}$$

where a_i denotes the i^{th} row vector of A and $max(a_i)$ is the largest entry in the vector.

The necessity of this restriction stems from the fact that any two courses from disjoint student groups have either at least one common student or are taught by a common lecturer and hence, they cannot be scheduled at the same time even in different locations. Hence the maximum number of concurrent classes that can be scheduled for each group (at different locations) is given by $max(a_i)$ and if each group is allocated this number of time slots then the total sum of these maximum values should not exceed the total number of available time slots given by $|P| = k$.

Capacity

Let l_i denote size of the largest class in group i and c_j denote the capacity of room j. Then,

$$\forall a_{ij}, c_j \geq l_i \tag{3.11}$$

i.e. each group should be allocated a room with adequate capacity to contain the largest class

Fitness Evaluation For Group Matrices

The fitness function for the first given by $f_A : A \rightarrow [0, 1]$ is constructed from the following components:

Let:

$$\alpha(a_{ij}) = \begin{cases} 1 & \text{if } c_j \geq t_\alpha \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

where c_j is the capacity of room j and t_α is a parameter called the α-*threshold*. This threshold is a user-defined k^{th} percentile of all group course sizes and it, in turn, determines which entries of the i^{th} row of A are α-valid (i.e. satisfy the capacity constraint). The capacity constraint defined in equation 3.11 is a special case where the threshold is defined as the 100^{th} percentile with $t_\alpha = l_i$ (l_i being the maximum class size of group g_i).

Then we can define the α-component of f_A as:

$$\alpha = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} \alpha(a_{ij})}{m \times n} \tag{3.13}$$

This component represents the proportion of group-to-room allocations that are valid based on the α-threshold for each group (proportion of α-valid entries of A)

Similarly, let

Let:

$$\beta(j) = \begin{cases} 1 & \text{if } \sum_{i=1}^{m} a_{ij} = k \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

then the β-component of f_A can be defined as:

$$\beta = \frac{\sum_{j=1}^{n} \beta(j)}{n} \tag{3.15}$$

i.e. the proportion of all column vectors of A whose elements sum up to less than or equal to total available timeslots (β-valid columns).

Thirdly, let:

$$\lambda(i) = \begin{cases} 1 & \text{if } \sum_{j=1}^{n} a_{ij} = w_i \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

then the λ-component of f_A can be defined as:

$$\lambda = \frac{\sum_{i=1}^{m} \lambda(i)}{m} \tag{3.17}$$

i.e. the proportion of all row vectors of A whose elements sum up to exactly the required group total contact hours (λ-valid rows)

And finally we can define the ρ-component of f_A as:

$$\rho = \begin{cases} 1 & \text{if } \sum_{i=1}^{m} max(a_i) \leq |P| \\ 0 & \text{otherwise} \end{cases} \tag{3.18}$$

Here ρ indicates whether the total time slot allocations for the different groups is feasible according to the third clash constraint i.e. the total maximum group allocations does not exceed the number of available time slots.

Therefore, f_A can be defined as the weighted sum of the above components:

$$f_A = \omega_\alpha \alpha + \omega_\beta \beta + \omega_\lambda \lambda + \omega_\rho \rho \tag{3.19}$$

where $\omega_\alpha, \omega_\beta, \omega_\lambda$and ω_ρ are the respective weights for each component.

Satisfying the above constraints in the initial phase of the GA produces a matrix containing feasible allocations of required group hours to appropriate rooms. This matrix is the framework from which actual group timetables will be constructed.

3.2.3 Phase 2: Group-local Timetable Optimization

This phase deals with the actual distribution of courses for each room to optimize additional (soft) constraints. This can be done (serially or in parallel) for each group using a GA or other assignment means. Our work uses a secondary GA to finalize the class-to-time slot allocation and to optimize the individual group timetables.

Chromosome Encoding

Let $T_i \subset T$ denote a random subset of time slots allocated to group g_i, with:

$$|T_i| = h_i = \begin{cases} g_{max} & \text{if } g_{max} > max(a_i) \\ max(a_i) & \text{otherwise} \end{cases} \tag{3.20}$$

(g_{max} and $max(a_i)$ are as previously defined in section 3.1.3 and 3.2.3 respectively)

Then we can represent a group timetable, Θ_i as an $h_i \times n$ matrix of the form:

$$\Theta_i = [c_{kj}], \text{where } c_{kj} \in g_i, 0 < k \leq h_i, 0 < j \leq n \tag{3.21}$$

($n = |R|$ as previously defined in section 3.23)

More concisely, if we let R_i denote the set of non-zero entries in the i^{th} row of A and let $n_i = |R_i|$, then Θ_i can be re-written as the dense matrix:

$$\Theta_i = [c_{kj}], \text{where } c_{kj} \in g_i, 0 < k \leq h_i, 0 < j \leq n_i \tag{3.22}$$

Thus $c_{ij} = a$ indicates course a is scheduled in room j during time slot k. Therefore, Θ_i represents a possible final group timetable based on the group allocation matrix A. The overall timetable will be aggregated from the component group timetables.

Fitness Evaluation for Group Timetables

For the second phase, there are two evaluations that will contribute to the fitness function $f_\Theta : \Theta_i \rightarrow [0, 1]$ as follows:

Let θ_k denote the k^{th} row of Θ_i. We define

$$\mu(\theta_k) = \begin{cases} 1 & \text{if } \forall x, y \in Z \cap (0, n_i], x \neq y \implies \theta_{kx} \neq \theta_{ky} \\ 0 & \text{otherwise} \end{cases} \tag{3.23}$$

i.e. every entry in each row is unique. This prevents the GA from scheduling the same course during the same timeslot. Then we can define the μ component of f_θ as:

$$\mu = \frac{\sum_{k=1}^{h_i} \mu(\theta_k)}{h_i} \tag{3.24}$$

This gives the ratio of μ-valid rows for the group timetable.

Further, let ϵ be defined as:

$$\epsilon(c_{kj}) = \begin{cases} 1 & \text{if course } c_{kj} \text{ is schedule in } \textit{preferred} \text{ room} \\ 0 & \text{otherwise} \end{cases} \tag{3.25}$$

Here preferred room is indicated by course requirements. If the course has no preferences, any room allocation evaluates to 1. Then the ϵ component of f_θ can be defined as:

$$\epsilon = \frac{\sum_{k=1}^{h_i} \sum_{j=1}^{n} \epsilon(c_{kj})}{h_i \times n_i} \tag{3.26}$$

Therefore,

$$f_\Theta = \omega_\mu \mu + \omega_\epsilon \epsilon \tag{3.27}$$

Where, as in equation 3.19, ω_μ and ω_ϵ are user defined weights.

Phase 1 Pseudocode

Procedure 1 Optimize Group Allocation Matrix

Input: genetic_parameters: list of key-value pairs of genetic parameters
Output: group matrix chromosome

```
1: procedure GroupMatrixGA(genetic_parameters)
2:     population ← generateInitialPopulation(parameters.pop_size)
3:     if empty(population) then
4:         return null
5:     end if
6:     elite ← getFittest(population)
7:     runs ← 0
8:     while runs ≤ parameters.generations & elite.fitness ≤ parameters.optimal_fitness do
9:         population ← evolveMatrix(population, parameters)
10:        if parameters.elitism = True then
11:            replace(population, elite, parameters.replacement_policy)
12:        end if
13:        elite ← getFittest(population)
14:        runs ← runs + 1
15:    end while
16:    return elite
17: end procedure
```

Phase 2 Pseudocode

Procedure 2 Group-Local Timetable Optimization

Input: group_matrix: group matrix chromosome, timeslots: list of time slot values, student_groups: set of student group partitions, parameters:list of key-value pairs of genetic parameters

Output: NULL

```
1: procedure GroupTimetableGA(group_matrix, timeslots, student_groups, parameters)
2:     timeslots ← randomize(timeslots)
3:     start_index ← 0
4:     for all group in student_groups do
5:         group.allocation_vector ← group_matrix[group.id]
6:         slot_count ← max(getGmax(group), getMaxCourseHours(group))
7:         end_index ← start_index + slot_count
8:         if start_index > timeslots.size then
9:             assignTimeslots(group, timeslots[timeslot.size-count : timeslot.size])
10:        else
11:            assignTimeslots(group, timeslots[start_index : end_index])
12:            start_index ← start_index + count
13:        end if
14:        group_timetable = evolveTimetable(group, parameters)
15:        write(group_timetable)
16:    end for
17:    return NULL
18: end procedure
```

The evolveTimetable algorithm is the same for both phases (Procedure 1) with modifications made to the mutation and crossover algorithms (see Appendix A).

3.2.4 Implementation and Results

A prototype of the algorithm was developed using Python 3.4 as a programming language with Numpy 1.91 for matrix manipulation and MySQL 5.6.12 as a database backend. The choice of language was influenced by Python's consiseness, prototyping prowess and availability of well supported third-party packages. The prototype was first tested[4] using a subset of real-world data obtained from the University of Malawi, Chancellor College. The data consisted 46 courses partitioned into 16 student groups to be allocated into 19 rooms and a total of 45 weekly hours (9 hours per day, 5 days a week). The prototype was then run on a larger data set (100 courses, 41 rooms, 22 student groups and 60 weekly hours) to test its scalability. Sample timetables were generated in both cases.

In the former case, the prototype was run to satisfy H on four different values of α-threshold; 75^{th}, 80^{th} and 100^{th} percentile. As outlined in section 3.2.2, a feasible timetable with x^{th} percentile for α-threshold implies that $x\%$ of the classes in the group are expected to fit in all of the group's allocated rooms. As an example, for the 80^{th} percentile, an allocation of h_i hours of group g_i in room r_i indicates that at least 80% of the courses in that group will fit in that room given that the group allocation has fitness of 1.0. This is to allow more flexibility in the algorithm. All runs were performed with the following genetic parameter values: *(i) Populations size: 50, (ii) Crossover rate: 0.2, (iii) Mutation rate: 0.1, (iv) Tournament size: 10, (v) Generations: 1500, (vi) Elitism: true, (vii) Elitist replacement policy: random.* Weights for f_A were assigned based on relative importance of the components. On the other hand, f_Θ only considered one component due to lack of preference information necessary to for the inclusion of ϵ. The fitness functions used were as follows:

$$f_A = 0.3\alpha + 0.15\beta + 0.05\lambda + 0.5\rho \tag{3.28}$$

$$f_\Theta = 1.0\mu \tag{3.29}$$

Below are the data statistics and trends obtained from the different runs:

[4]Testing was done on a Zinox computer, with Intel's CORE i7 processor (2 physical cores, 4 logical cores, 4GB RAM running Windows 7 Professional Version.

Group ID	Courses	Max Hours	Total Hours	Largest Course
1	7	8	32	85
2	8	8	34	50
3	3	8	17	40
4	4	8	16	50
5	4	8	16	180
6	4	4	13	180
7	4	4	12	150
8	1	2	2	150
10	2	4	6	150
11	2	4	6	180
12	2	3	5	100
13	1	2	2	120
17	1	4	4	180
19	1	2	2	100
20	1	4	4	170
22	1	4	4	170

Figure 3.1: Student group statistics

Group ID	Courses	Max Hours	Total Hours	Largest Course
1	10	8	42	137
2	10	8	42	65
3	4	8	21	100
4	5	8	18	50
5	9	8	32	180
6	8	4	27	180
7	6	4	18	240
8	3	4	10	150
9	2	4	6	70
10	5	4	17	150
11	2	4	6	180
12	5	4	16	100
13	3	4	9	120
14	3	4	11	110
15	2	4	7	47
16	3	4	10	60
17	5	4	18	185
18	3	4	12	80
19	3	4	10	100
20	5	4	20	170
21	2	4	7	60
22	2	4	6	170

Figure 3.2: Student group statistics (larger data set)

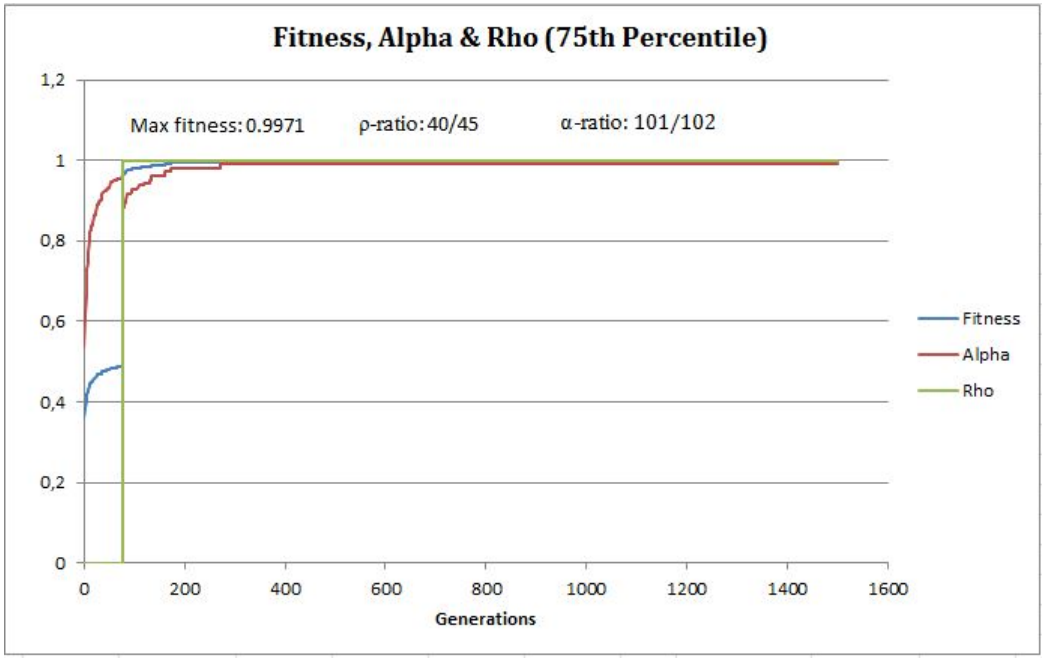


Figure 3.3: Trend: (α-threshold at 75^{th} percentile)

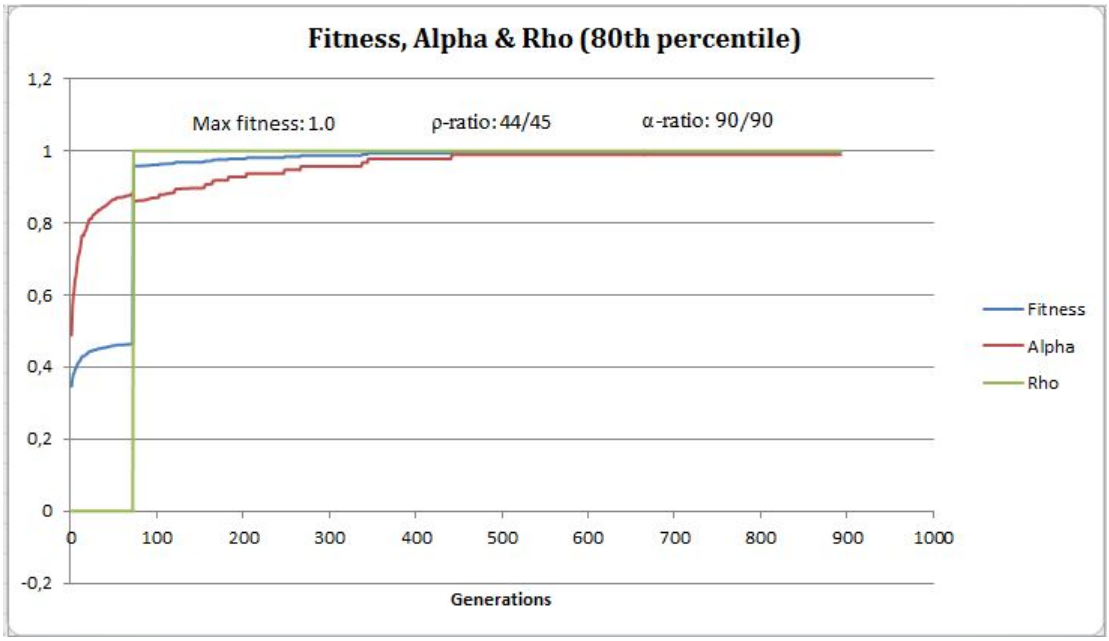


Figure 3.4: Trend: (α-threshold at 80^{th} percentile)

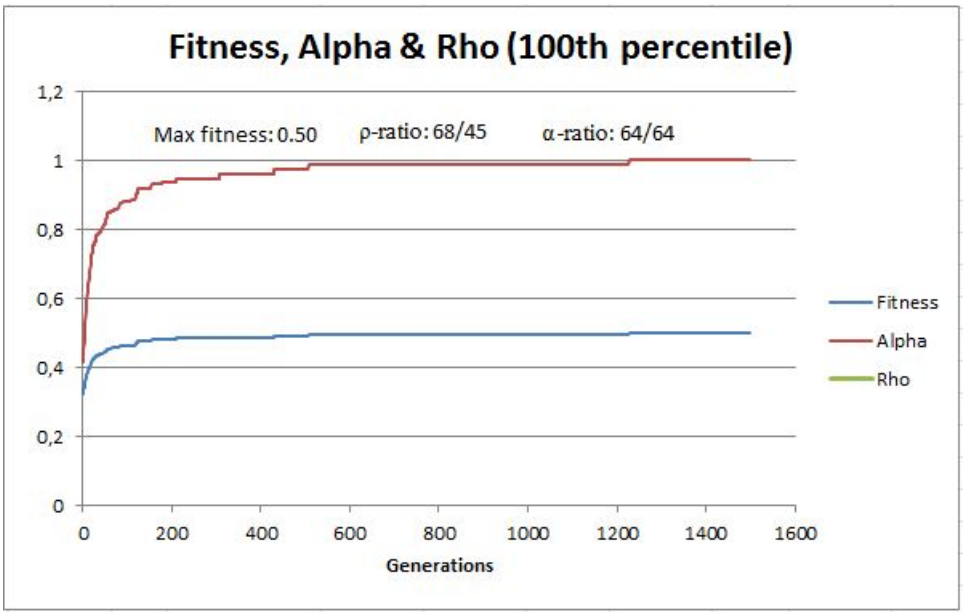


Figure 3.5: Trend: (α-threshold at 100^{th} percentile)

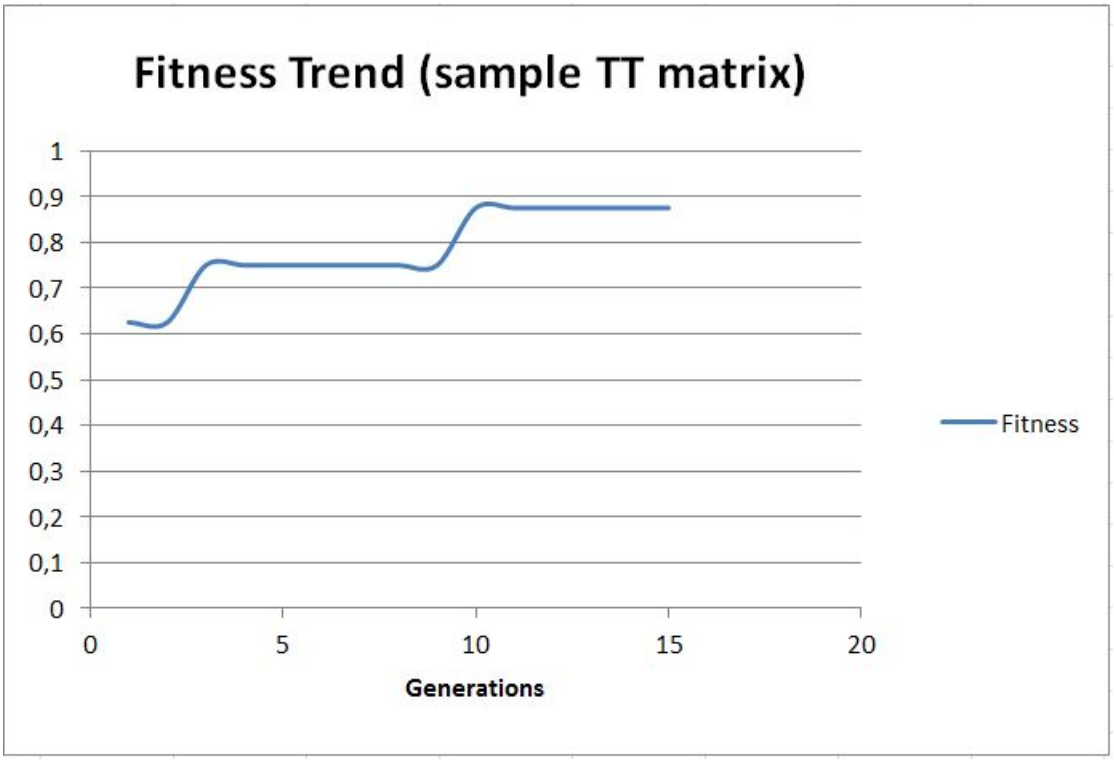


Figure 3.6: Trend (sample group timetable generation)

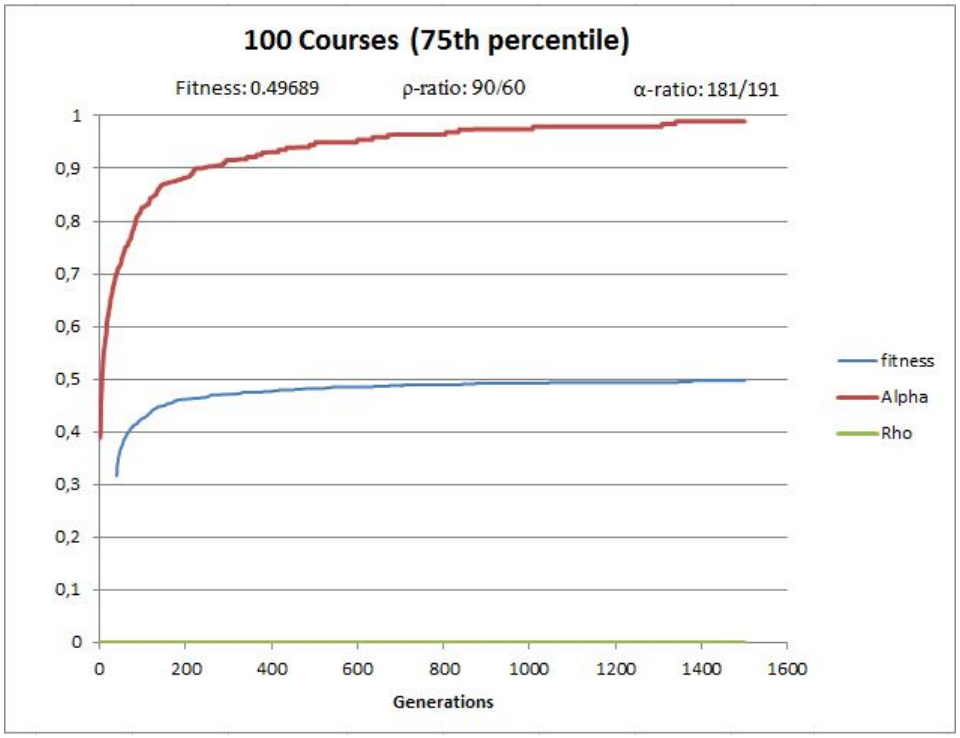


Figure 3.7: Trend (larger data set)

3.2.5 Performance Analysis

The results for the smaller data set show that the algorithm is able to attain the maximum fitness value of 1.0 for moderate α-threshold values (e.g. 50^{th} and 80^{th}). The two-dimensional nature of chromosome encoding implies that the most intensive computations (e.g. crossover and mutation) are $O(n^2)$. This can be clearly seen in the mutation and crossover algorithms (see Appendix A). Holding the population size constant, the computational complexity of the algorithm should scale quadratically with increasing input data (number of courses and rooms). The running time averages 4 minutes and 7 minutes for the smaller and larger data sets respectively on a quad-core, hyper-threaded Intel Core i7 processor. Given the vast nature of the space, this reduced time may be attributed to the 'pruned' search space within which the algorithm operates.

Figures 3.3 to 3.5 show that the general trend of evolution for all trials follows an initial quadratic increase in fitness with a steep rise in the trend that coincides with a jump in ρ value. The graphs also show that the search proceeds steadily without a drop in fitness between generations. This general increasing trend is a direct consequence of elitism, which maintains the best chromosome between consecutive generations. However, periodic samples of chromosome generations for all cases (see Appendix A) indicate a constant value of 1.0 for both β and λ components. This implies that the initial chromosome populations satisfies all $\mathsf{hours-per-room}$ allocation constraints and that all hours for each group are fully scheduled.

Additionally, a closer examination of the data will reveal an inverse relationship between α-threshold and the probability of convergence on an optimal fitness. The higher the α-threshold value, the less likely it is to locate a feasible allocation. This is an instance of conflicting constraints i.e. the stricter the rule on room capacity, the harder it is to find a feasible allocation with each allocated course fitting into its respective room.

An instance of this scenario can be seen in the results obtained for α-threshold value of 100^{th} percentile as shown in Figure 3.5. A number of runs were performed at this threshold with no significant improvement in the final optimal fitness value (approx. 0.50). A similar trend was also observed for the larger data set (Figure 3.15). This is a consequence of the inadequate time slots relative to the number of courses that require scheduling (condition 3 necessary for non-existence of feasible timetable). The effect of this is seen in the output timetables where a number of (peer) courses register time clashes.

Finally, Figure 3.6 shows a sample trend generated by a secondary GA that is optimizing a group timetable in phase 2. It is worth noting that the trend only displays fitness values solely based on μ. This is due to the unavailability of departmental preference information in the source data. This case shows a maximum fitness value of 0.875, which indicates 0.115 odds of finding a time clash among peer courses within the group.

3.2.6 Parallel/Distributed Algorithm

The approach and representation chosen for the serial algorithm offer opportunities for *single-instruction, multiple data* (SIMD) parallelization that can be exploited to speed up convergence on parallel/distributed architectures. The most promising parts for parallelization include:

1. Fitness evaluation, mutation and crossover for group allocation matrix and group timetables
2. Group timetable optimization

For group-local timetable optimization, each i^{th} row of A (for group g_i) will be mapped as a separate input split onto a processing node for group timetable generation. Results will be written to separate file for each group and merged either at runtime or manually after execution. The parallelization strategy can be diagrammatically represented as in Figure 3.8 below:

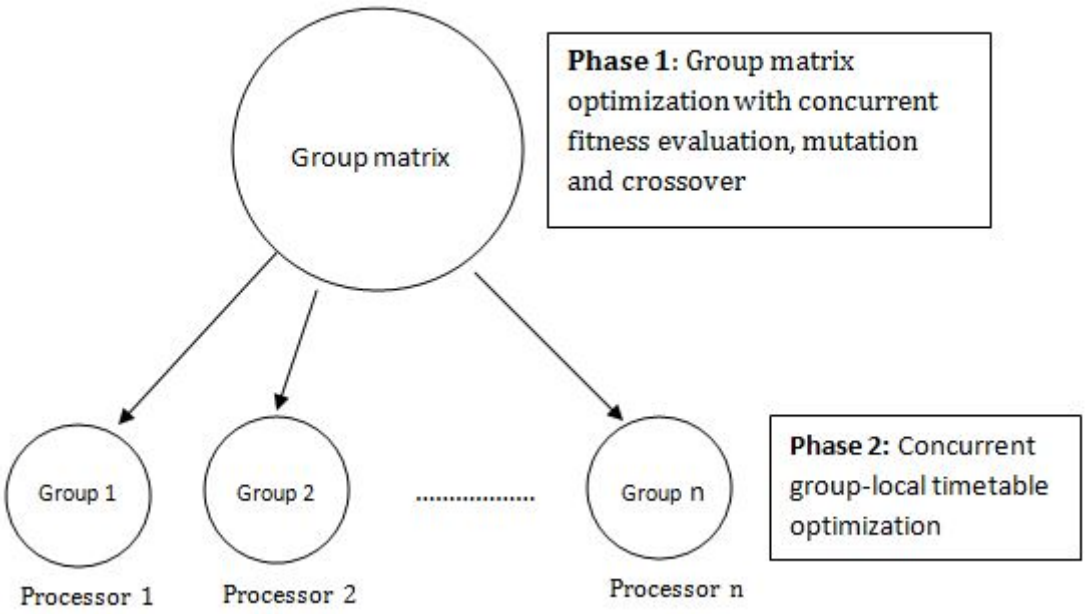


Figure 3.8: Parallel/distributed GA Model

This parallelization scheme offers a great deal of flexibility in terms of implementation technology and platform. It can be implemented on a distributed computing platform or on a single node with multi-core processors depending on resource availability. It is worth noting that the level of parallelization in the proposed scheme is limited by the serial phase (phase 1) i.e. generation of the group allocation matrix. Since this phase represents approximately 50% of the whole algorithm, by Amdahl's law, the speedup achieved by an embarrassingly parallel implementation of phase 2 *only* should be at most:

$$\lim_{n\to\infty} \frac{1}{0.5 + \frac{1-0.5}{n}} = 2 \tag{3.30}$$

where n is the number of processing elements used.

However, given the nature of the chromosome encoding scheme (and the genetic operator algorithms), phase 1 can also be parallelized to take advantage of multi-processing architectures thereby increasing the speedup beyond the predicted theoretical limit of 2.

Below is the pseudocode for the main parallel/distributed algorithm. Appendix A contains pseudocode for the procedures called by the algorithm.

Pseudocode For Parallel/Distributed Algorithm

Procedure 3 Main DGA/PGA

```
1: procedure ParallelGA(constraints)
2:     parameters ← getParametersInput()
3:     group_matrix ← GroupMatrixGA(parameters)
4:     if group_matrix.rho_value > 0 then
5:         initializeGroups(group_matrix, constraints.timeslots, constraints.student_groups)
6:         for all group in constraints.student_groups in PARALLEL do
7:             group_timetable ← evolveTimetable(group)
8:             group_file ← group.unique_id
9:             write(group_timetable, group_file)
10:        end for
11:    else
12:        print(NoOptimalSolutionError)
13:    end if
14: end procedure
```

Procedure 4 Initialize Student Groups Using Group Matrix

```
1: procedure initializeGroups(group_matrix, timeslots, student_groups)
2:     timeslots ← randomize(timeslots)
3:     start_index ← 0
4:     for all group in student_groups do
5:         group.allocation_vector ← group_matrix[group]
6:         slot_count ← max(getGmax(group), getMaxCourseHours(group))
7:         end_index ← start_index + slot_count
8:         if start_index > timeslots.size then
9:             assignTimeslots(group, timeslots[timeslot.size-count : timeslot.size])
10:        else
11:            assignTimeslots(group, timeslots[start_index : end_index])
12:            start_index ← start_index + count
13:        end if
14:    end for
15: end procedure
```

Chapter 4

Conclusion

4.1 Summary

Timetabling presents a complicated constraint satisfaction problem that requires efficient algorithms to solve. Genetic algorithms offer a promising mechanism due to their ability to handle complicated search spaces. Our work proposes a theoretical framework to guide application of GAs to the timetabling problem. It is observed that, with this framework as the basis, an efficient algorithm can be designed that will converge to an optimal or Pareto solution in polynomial time. The work also shows that the choice of chromosome encoding has a great impact on algorithm performance and scalability. Our work adopts a matrix-based encoding scheme which lends itself to almost effortless parallelization, a factor that is exploited in the proposed parallel/distributed algorithm. The scheme also allows for simple mathematical expression of constraints for efficient evaluation.

In addition, we note that timetabling requires balancing several parameters given that most real-world problems present inherently insufficient resources and conflicting constraints and preferences. In this case, an algorithm should be able to settle for a Pareto allocation of resources i.e. an optimal equilibrium where one resource cannot be reallocated without negatively affecting other allocations.

Finally, our work on parallel/distributed algorithm reveals that speedup can be increased by a theoretical factor of two if only the second phase of the algorithm is parallelized. Given that this factor is too small to justify the parallelization efforts, we suggest further increasing the parallel footprint of the algorithm by implementing the basic genetic operators on multiprocessing platforms. This approach has the potential to boost the theoretical speedup limit by a significant factor.

4.2 Limitations and Future Work

The approach explored in this paper is applicable within a number of limitations. Perhaps the most important limitation is the assumption of existence of student group information prior to generation of the group allocation matrix. In most cases, the problem of generating student groups information is separated from the timetabling problem and it is usually done prior to timetable generation. However, in the absence of such prior information, the algorithm requires a supplementary partitioning algorithm (e.g. graph colouring) to generate the initial input.

Another limitation is the serial nature of phase 1 of the algorithm. This arises due to the need to have all student group information available in one 'space' to generate a feasible group allocation matrix. Massive parallelization of this phase would require intricate levels of inter-process interaction and synchronization, two factors that may lead to communication overheads and reduced efficiency. However, future work may explore this option to test different interaction and synchronization schemes and their effects on efficiency and convergence.

Bibliography

[1] D Abramson and J Abela. A Parallel Genetic Algorithm for Solving the School Timetabling Problem. In *Division of Information Technology, C.S.I.R.O*, pages 1–11, 1992.

[2] Arwa Al-Edaily, Nada Al-Zaben, Sharefa Al-Ghamdi, and Aboalsamh Hati. Improved Distributed Genetic Algorithms Based on Their Methodology and Processes. In *in Proceedings of the European Computing Conference*, pages 415–420, 2011.

[3] Grigorios N. Beligiannis, Charalampos N. Moschopoulos, and Spiridon D. Likothanassis. A Genetic Algorithm Approach to School Timetabling. *The Journal of the Operational Research Society*, 60(1):23–42, 2009.

[4] Tobias Blickle and Lothar Thiele. A Comparison of Selection Schemes used in Genetic Algorithms. Technical report, Computer Engineering and Communication Networks Lab, Swiss Federal Institute of Technology, Gloriastrasse, Zurich, Switzerland, December 1995.

[5] Sigl Branimir, Marin Golub, and Vedran Mornar. Solving Timetable Scheduling Problem by Using Genetic Algorithms. In *Int Conf Information Technology Interfaces IT1*, pages 519–524, 2003.

[6] Edmund K. Burke, Dave G. Elliman, and Rupert F. Weare. A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 605–610. Morgan Kaufmann, 1995.

[7] Erick Cantu-Paz. A Survey of Parallel Genetic Algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10, 1998.

[8] Erick Cantu-Paz and David E. Goldberg. On the Scalability of Parallel Genetic Algorithms. *Journal of Evolutionary Computation*, 7(4):429–449, 1999.

[9] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. A Genetic Algorithm to Solve the Timetable Problem, 1993.

[10] Stephanie Forrest. Genetic Algorithms: Principles of Natural Selection Applied to Computation. *Science, New Series*, 261(5123):872–878, 1993.

[11] David E. Goldberg and Kalyanmoy Deb. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[12] Randy L. Haupt and Sue Ellen Haupt. *Practical Genetic Algorithms.* John Wiley & Sons, Hoboken, New Jersey, 2nd edition, 2004.

[13] Jun He, Feidun He, and Xin Yao. A Unified Markov Chain Approach to Analysing Randomised Search Heuristics. Technical report, Department of Computer Science, Aberystwyth University, Aberystwyth, SY23 3DB, U.K., December 2013.

[14] John H. Holland. *Adaptation in Natural And Artificial Systems. An Introductory Analysis with Applications to Bilogy, Control, and Artificial Intelligence.* MIT Press, 5th edition, 1998.

[15] Omar Ibrahim Obaid, MohdSharifuddin Ahmad, Salama A. Mostafa, and Mazin Abed Mohammed. Comparing Performance of Genetic Algorithm with Varying Crossover in Solving Examination Timetabling Problem. *Journal of Emerging Trends in Computing and Information Sciences*, 3(10):1427–1434, 2012.

[16] Petr Pospichal, Jiri Jaros, and Josef Schwarz. *Parallel Genetic Algorithm on the CUDA Architecture, EvoApplications 2010, Part I*, pages 442–451. Springer-Verlag, 2010.

[17] Rushil Ranghavjee and Nelishia Pillay. Use of Genetic Algorithms to Solve the South African School Timetabling Problem. In *Second World Congress on Nature and Biologically Inspired Computing*, pages 286–292, 2010.

[18] Olympia Roeva, editor. *Solving Timetable Problem by Genetic Algorithm and Heuristic Search Case Study: Universitas Pelita Harapan Timetable, Real-World Applications of Genetic Algorithms*, chapter 15, pages 303–316. InTech, 2012.

[19] Peter Ross, Dave Corne, and Hsiao lan Fang. Successful Lecture Timetabling with Evolutionary Algorithms. In *Proceedings of the ECAI 94 Workshop on Applications of Evolutionary Algorithms.* Springer, 1994.

[20] Gunter Rudolph. Convergence Analysis of Canonical Genetic Algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, 1994.

[21] S.N. Sivanandam and S.N. Deepa. *Introduction to Genetic Algorithms.* Springer, Berlin Heidelberg, USA, 2008.

[22] Joe Suzuki. A Markov Chain Analysis of Genetic Algorithms: Large Deviation Principle Approach. *Journal of Applied Probability*, 47(4):967–975, 2010.

[23] Huayang Xie and Mengjie Zhang. Tuning Selection Pressure in Tournament Selection. Technical Report ECSTR-09-10, School of Engineering and Computer Science. Victoria University of Wellington, New Zealand, 2009.

Appendix A

Algorithms and Sample Output

A.1 Pseudocode

Procedure 5 Phase 1: Mutation Algorithm

Input: chromosome: group matrix chromosome, mutation_rate: float value
Output: NULL

```
1: procedure mutateMatrix(chromosome, mutation_rate)
2:     re-evalute_flag ← FALSE
3:     if randomFloat(0,1) ≤ mutation_rate then
4:         re-evaluate_flag ← TRUE
5:         for all row in chromosome do
6:             for all column in row do
7:                 if chromosome[row][column] ≠ 0 then
8:                     delta ← randomInteger(0, chromosome[row][column])
9:                     chromosome[row][column] ← chromosome[row][column] - delta
10:                    repair_allele := getRandomAllele(chromosome)
11:                    chromosome.repair_allele ← chromosome.repair_allele + delta
12:                end if
13:            end for
14:        end for
15:    end if
16:    return NULL
17: end procedure
```

Procedure 6 Phase 1: Crossover Algorithm

Input: parent1, parent2: group matrix chromosome, crossover_rate: float value
Output: NULL

```
1: procedure crossoverMatrix(parent1, parent2, crossover_rate)
2:     if crossover_rate = 0 and parent1.size ≠ parent2.size then
3:         return
4:     end if
5:     for all row in parent1 do
6:         for all column in row do
7:             if parent1[row][column] != parent2[row][column] AND randomFloat(0,1) <=
   crossover_rate then
8:                 if parent1[row][column] > parent2[row][column] then
9:                     diff ← parent1[row][column] - parent2[row][column]
10:                    parent1[row][column] ← parent2[row][column]
11:                    repair_allele ← getRandomElement(parent1)
12:                    repair_allele ← repair_allele + diff
13:                else
14:                    diff ← parent2[row][column] - parent1[row][column]
15:                    parent2[row][column] ← parent1[row][column]
16:                    repair_allele ← getRandomAllele(parent2)
17:                    repair_allele ← repair_allele + diff
18:                end if
19:            end if
20:        end for
21:    end for
22:    return NULL
23: end procedure
```

Procedure 7 Phase 2: Crossover Algorithm

Input: parent1, parent2: group timetable chromosome, crossover_rate: float value
Output: NULL

```
1: procedure crossoverTimetable(parent1, parent2, crossover_rate)
2:     if crossover_rate = 0 or parent1.size ≠ parent2.size then
3:         return
4:     end if
5:     for all column in parent1 do:
6:         if randomFloat(0,1) ≤ crossover_rate then
7:             swap(parent1.column, parent2.column)
8:         end if
9:     end for
10:    return NULL
11: end procedure
```

Procedure 8 Phase 2: Mutation Algorithm

Input: chromosome: group timetable chromosome, mutation_rate: float value
Output: NULL

```
1: procedure mutateTimetable(chromosome, mutation_rate)
2:     re-evaluate_flag ← FALSE
3:     for all column in chromosome do:
4:         if randomFloat(0,1) ≤ mutation_rate then:
5:             re-evaluate_flag = TRUE
6:             randomize(column)
7:         end if
8:     end for
9:     return NULL
10: end procedure
```

Procedure 9 Evolve Group Allocation Matrix Population

Input: population: list of group matrix chromosomes, parameters: list of key-value pairs of genetic parameters

Output: list of group matrix chromosomes

```
1: procedure evolveMatrix(population, parameters)
2:     new_population:LIST
3:     while space ≥ 0 do
4:         parent1 ← tournamentSelect(population, parameters.tournament_size)
5:         parent2 ← tournamentSelect(population, parameters.tournament_size)
6:         crossoverMatrix(parent1, parent2, parameters.crossover_rate)
7:         mutateMatrix(parent1, parameters.mutation_rate)
8:         mutateMatrix(parent2, parameters.mutation_rate)
9:         parent1.fitness = evaluateFitness(parent1)
10:        parent2.fitness = evaluateFitness(parent2)
11:        new_population.append(getFittest(parent1,parent2))
12:    end while
13:    return new_population
14: end procedure
```

A.2 Output data samples

		Periodic Sample					
Generation	Fitness	Alpha	Beta	Lambda	Rho	Valid Alphas	Valid Rhos
1	0,361	0,5366	1	1	0	22/41	45/114
50	0,4797	0,9324	1	1	0	69/74	45/60
100	0,9786	0,9286	1	1	1	91/98	45/43
150	0,9885	0,9615	1	1	1	100/104	45/42
200	0,9942	0,9806	1	1	1	101/103	45/45
250	0,9942	0,9806	1	1	1	101/103	45/41
300	0,9971	0,9902	1	1	1	101/102	45/42
350	0,9971	0,9902	1	1	1	101/102	45/44
400	0,9971	0,9902	1	1	1	101/102	45/43
450	0,9971	0,9902	1	1	1	101/102	45/38
500	0,9971	0,9902	1	1	1	101/102	45/45
550	0,9971	0,9902	1	1	1	101/102	45/44
600	0,9971	0,9902	1	1	1	101/102	45/41
650	0,9971	0,9902	1	1	1	101/102	45/40
700	0,9971	0,9902	1	1	1	101/102	45/40
750	0,9971	0,9902	1	1	1	101/102	45/44
800	0,9971	0,9902	1	1	1	101/102	45/42
850	0,9971	0,9902	1	1	1	101/102	45/43
900	0,9971	0,9902	1	1	1	101/102	45/44
950	0,9971	0,9902	1	1	1	101/102	45/43
1000	0,9971	0,9902	1	1	1	101/102	45/44
1050	0,9971	0,9902	1	1	1	101/102	45/44
1100	0,9971	0,9902	1	1	1	101/102	45/41
1150	0,9971	0,9902	1	1	1	101/102	45/43
1200	0,9971	0,9902	1	1	1	101/102	45/44
1250	0,9971	0,9902	1	1	1	101/102	45/43
1300	0,9971	0,9902	1	1	1	101/102	45/43
1350	0,9971	0,9902	1	1	1	101/102	45/39
1400	0,9971	0,9902	1	1	1	101/102	45/43
1450	0,9971	0,9902	1	1	1	101/102	45/44
1500	0,9971	0,9902	1	1	1	101/102	45/40

Figure A.1: Periodic data sample (α-threshold at 75^{th} percentile)

		Periodic Sample					
Generation	Fitness	Alpha	Beta	Lambda	Rho	Valid Alphas	Valid Rhos
1	0,3465	0,4884	1	1	0	21/43	45/115
50	0,4595	0,8649	1	1	0	64/74	45/58
100	0,9613	0,871	1	1	1	81/93	45/42
150	0,9691	0,8969	1	1	1	87/97	45/39
200	0,9784	0,9278	1	1	1	90/97	45/43
250	0,9842	0,9474	1	1	1	90/95	45/45
300	0,9872	0,9574	1	1	1	90/94	45/44
350	0,9935	0,9783	1	1	1	90/92	45/45
400	0,9935	0,9783	1	1	1	90/92	45/45
450	0,9967	0,989	1	1	1	90/91	45/45
500	0,9967	0,989	1	1	1	90/91	45/45
550	0,9967	0,989	1	1	1	90/91	45/43
600	0,9967	0,989	1	1	1	90/91	45/44
650	0,9967	0,989	1	1	1	90/91	45/44
700	0,9967	0,989	1	1	1	90/91	45/43
750	0,9967	0,989	1	1	1	90/91	45/43
800	0,9967	0,989	1	1	1	90/91	45/44
850	0,9967	0,989	1	1	1	90/91	45/45
893	0,9967	0,989	1	1	1	90/91	45/44

Figure A.2: Periodic data sample (α-threshold at 80^{th} percentile)

		Periodic Sample					
Generations	Fitness	Alpha	Beta	Lambda	Rho	α-ratio	ρ-ratio
1	0,3244	0,4146	1	1	0	17/41	45/120
50	0,4446	0,8154	1	1	0	53/65	45/66
100	0,4645	0,8816	1	1	0	67/76	45/57
150	0,476	0,92	1	1	0	69/75	45/60
200	0,4805	0,9351	1	1	0	72/77	45/64
250	0,4846	0,9487	1	1	0	74/78	45/65
300	0,4846	0,9487	1	1	0	74/78	45/63
350	0,4883	0,961	1	1	0	74/77	45/70
400	0,4883	0,961	1	1	0	74/77	45/64
450	0,4921	0,9737	1	1	0	74/76	45/67
500	0,4921	0,9737	1	1	0	74/76	45/63
550	0,496	0,9867	1	1	0	74/75	45/65
600	0,496	0,9867	1	1	0	74/75	45/70
650	0,496	0,9867	1	1	0	74/75	45/74
700	0,496	0,9867	1	1	0	74/75	45/71
750	0,496	0,9867	1	1	0	74/75	45/64
800	0,496	0,9867	1	1	0	74/75	45/73
850	0,496	0,9867	1	1	0	74/75	45/69
900	0,496	0,9867	1	1	0	74/75	45/66
950	0,496	0,9867	1	1	0	74/75	45/66
1000	0,496	0,9867	1	1	0	74/75	45/69
1050	0,496	0,9867	1	1	0	74/75	45/69
1100	0,496	0,9867	1	1	0	74/75	45/67
1150	0,496	0,9867	1	1	0	74/75	45/63
1200	0,496	0,9867	1	1	0	74/75	45/63
1250	0,5	1	1	1	0	73/73	45/72
1300	0,5	1	1	1	0	67/67	45/71
1350	0,5	1	1	1	0	66/66	45/72
1400	0,5	1	1	1	0	64/64	45/69
1450	0,5	1	1	1	0	66/66	45/66
1500	0,5	1	1	1	0	64/64	45/68

Figure A.3: Periodic data sample (α-threshold at 100^{th} percentile)

Generation	Fitness	μ	Valid μ
1	0,625	0,625	5
2	0,625	0,625	5
3	0,75	0,75	6
4	0,75	0,75	6
5	0,75	0,75	6
6	0,75	0,75	6
7	0,75	0,75	6
8	0,75	0,75	6
9	0,75	0,75	6
10	0,875	0,875	7
11	0,875	0,875	7
12	0,875	0,875	7
13	0,875	0,875	7
14	0,875	0,875	7
15	0,875	0,875	7

Figure A.4: Group timetable data (sample group timetable generation)

			Periodic sample				
Generation	**Fitness**	**Alpha**	**Beta**	**Lambda**	**Rho**	**Valid Alphas**	**Valid Rhos**
1	0,3167	0,3889	1,0000	1,0000	0	28/72	60/221
50	0,4162	0,7206	1,0000	1,0000	0	98/136	60/123
100	0,4477	0,8258	1,0000	1,0000	0	128/155	60/100
150	0,4612	0,8706	1,0000	1,0000	0	148/170	60/93
200	0,4649	0,883	1,0000	1,0000	0	166/188	60/73
250	0,4708	0,9026	1,0000	1,0000	0	176/195	60/79
300	0,4748	0,9158	1,0000	1,0000	0	185/202	60/72
350	0,4765	0,9216	1,0000	1,0000	0	188/204	60/72
400	0,4792	0,9307	1,0000	1,0000	0	188/202	60/76
450	0,4819	0,9397	1,0000	1,0000	0	187/199	60/80
500	0,4835	0,945	1,0000	1,0000	0	189/200	60/81
550	0,4849	0,9497	1,0000	1,0000	0	189/199	60/84
600	0,4863	0,9543	1,0000	1,0000	0	188/197	60/92
650	0,4878	0,9594	1,0000	1,0000	0	189/197	60/92
700	0,4893	0,9643	1,0000	1,0000	0	189/196	60/96
750	0,4893	0,9643	1,0000	1,0000	0	189/196	60/97
800	0,4893	0,9643	1,0000	1,0000	0	189/196	60/89
850	0,4922	0,974	1,0000	1,0000	0	187/192	60/92
900	0,4923	0,9742	1,0000	1,0000	0	189/194	60/89
950	0,4923	0,9742	1,0000	1,0000	0	189/194	60/87
1000	0,4923	0,9742	1,0000	1,0000	0	189/194	60/88
1050	0,4938	0,9793	1,0000	1,0000	0	189/193	60/87
1100	0,4938	0,9793	1,0000	1,0000	0	189/193	60/90
1150	0,4938	0,9793	1,0000	1,0000	0	189/193	60/83
1200	0,4938	0,9793	1,0000	1,0000	0	189/193	60/83
1250	0,4938	0,9793	1,0000	1,0000	0	189/193	60/83
1300	0,4938	0,9793	1,0000	1,0000	0	189/193	60/84
1350	0,4969	0,9895	1,0000	1,0000	0	189/191	60/84
1400	0,4969	0,9895	1,0000	1,0000	0	189/191	60/87
1450	0,4969	0,9895	1,0000	1,0000	0	189/191	60/87
1500	0,4969	0,9895	1,0000	1,0000	0	189/191	60/90

Figure A.5: Periodic sample (larger data set)

A.3 Sample output matrices

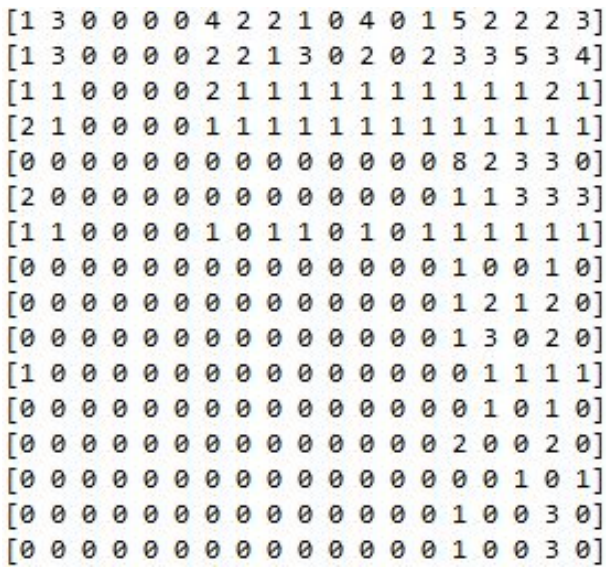


Figure A.6: Group allocation matrix (α-threshold at 75^{th} percentile)

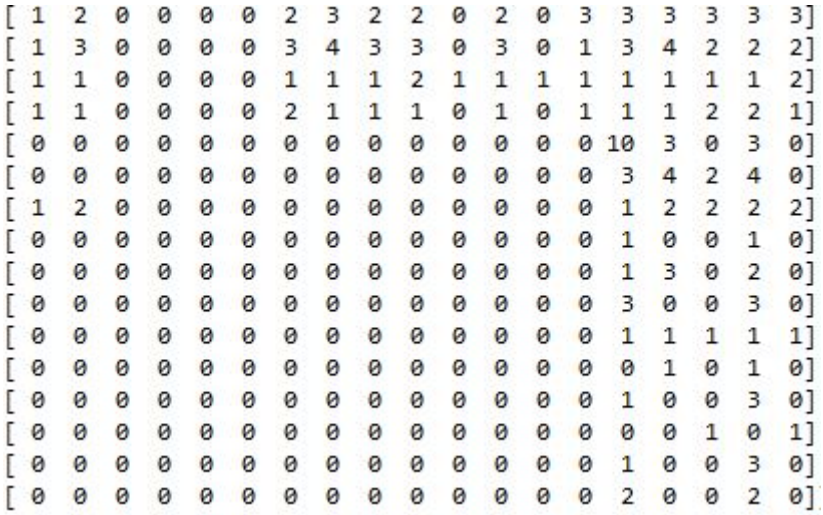


Figure A.7: Group allocation matrix (α-threshold at 80^{th} percentile)

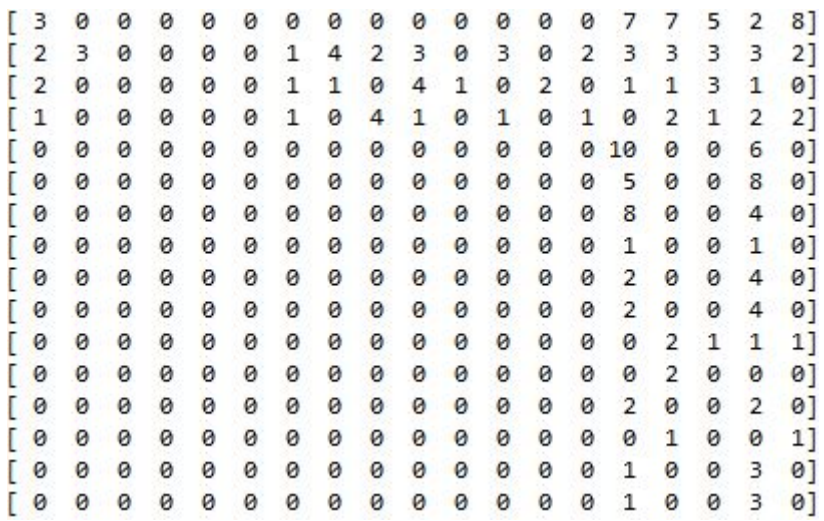


Figure A.8: Group allocation matrix (α-threshold at 100^{th} percentile)

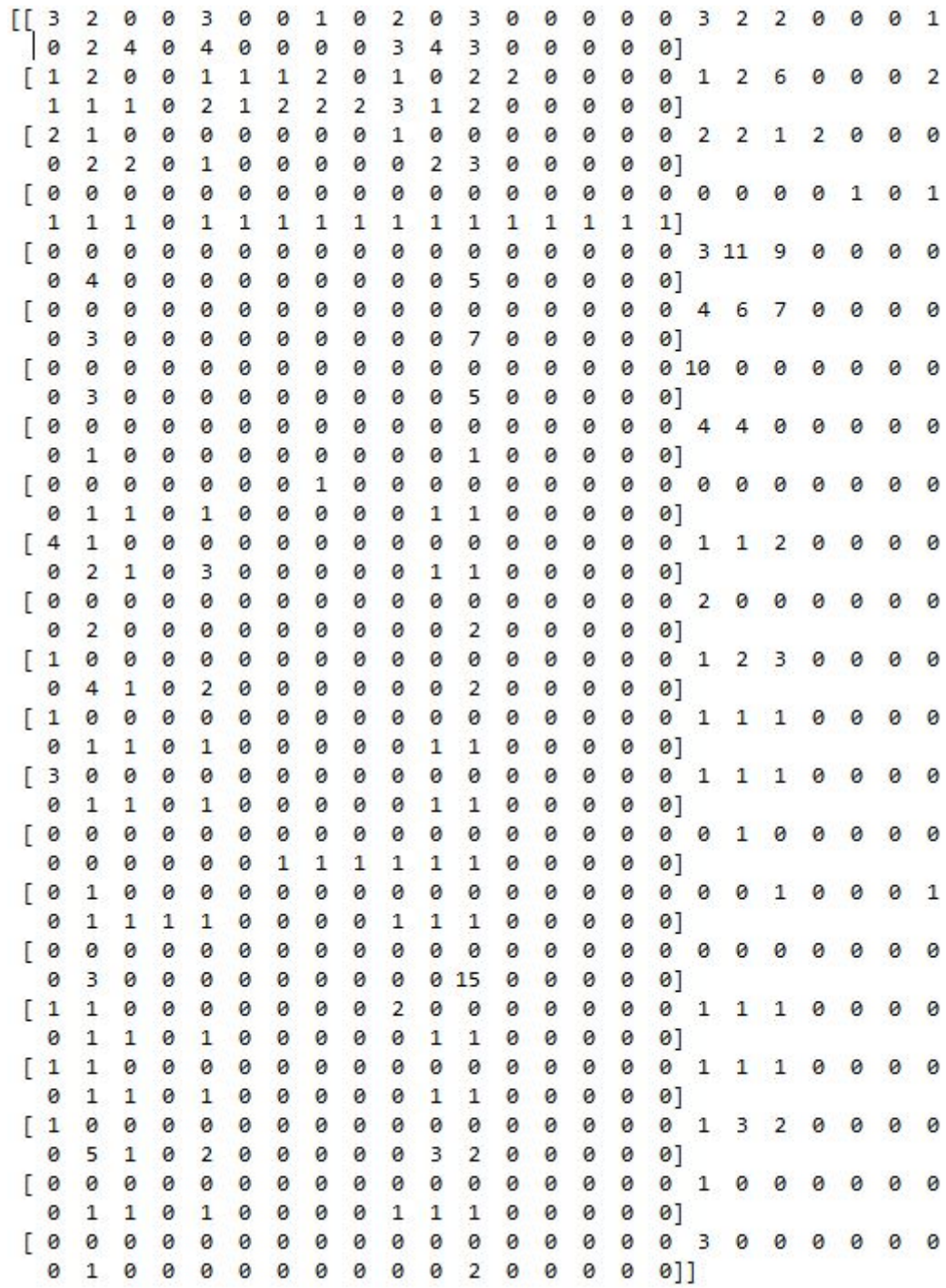
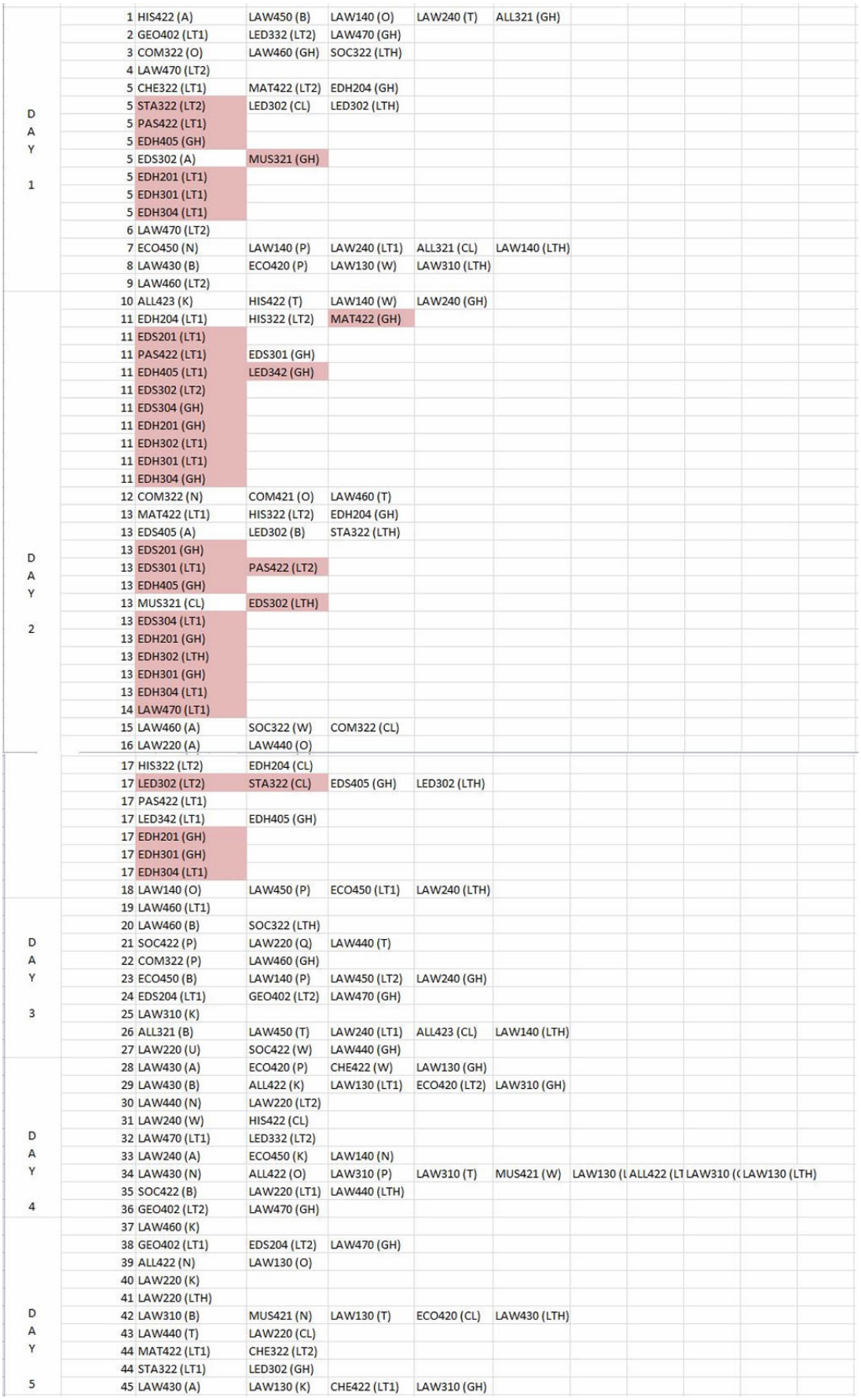


Figure A.9: Group allocation matrix (larger data set)



	1	HIS422 (A)	LAW450 (B)	LAW140 (O)	LAW240 (T)	ALL321 (GH)					
	2	GEO402 (LT1)	LED332 (LT2)	LAW470 (GH)							
	3	COM322 (O)	LAW460 (GH)	SOC322 (LTH)							
	4	LAW470 (LT2)									
	5	CHE322 (LT1)	MAT422 (LT2)	EDH204 (GH)							
	5	STA322 (LT2)	LED302 (CL)	LED302 (LTH)							
D	5	PAS422 (LT1)									
A	5	EDH405 (GH)									
Y	5	EDS302 (A)	MUS321 (GH)								
1	5	EDH201 (LT1)									
	5	EDH301 (LT1)									
	5	EDH304 (LT1)									
	6	LAW470 (LT2)									
	7	ECO450 (N)	LAW140 (P)	LAW240 (LT1)	ALL321 (CL)	LAW140 (LTH)					
	8	LAW430 (B)	ECO420 (P)	LAW130 (W)	LAW310 (LTH)						
	9	LAW460 (LT2)									
	10	ALL423 (K)	HIS422 (T)	LAW140 (W)	LAW240 (GH)						
	11	EDH204 (LT1)	HIS322 (LT2)	MAT422 (GH)							
	11	EDS201 (LT1)									
	11	PAS422 (LT1)	EDS301 (GH)								
	11	EDH405 (LT1)	LED342 (GH)								
	11	EDS302 (LT2)									
	11	EDS304 (GH)									
	11	EDH201 (GH)									
	11	EDH302 (LT1)									
	11	EDH301 (LT1)									
	11	EDH304 (GH)									
	12	COM322 (N)	COM421 (O)	LAW460 (T)							
	13	MAT422 (LT1)	HIS322 (LT2)	EDH204 (GH)							
	13	EDS405 (A)	LED302 (B)	STA322 (LTH)							
	13	EDS201 (GH)									
D	13	EDS301 (LT1)	PAS422 (LT2)								
A	13	EDH405 (GH)									
Y	13	MUS321 (CL)	EDS302 (LTH)								
2	13	EDS304 (LT1)									
	13	EDH201 (GH)									
	13	EDH302 (LTH)									
	13	EDH301 (GH)									
	13	EDH304 (LT1)									
	14	LAW470 (LT1)									
	15	LAW460 (A)	SOC322 (W)	COM322 (CL)							
	16	LAW220 (A)	LAW440 (O)								
	17	HIS322 (LT2)	EDH204 (CL)								
	17	LED302 (LT2)	STA322 (CL)	EDS405 (GH)	LED302 (LTH)						
	17	PAS422 (LT1)									
	17	LED342 (LT1)	EDH405 (GH)								
	17	EDH201 (GH)									
	17	EDH301 (GH)									
	17	EDH304 (LT1)									
	18	LAW140 (O)	LAW450 (P)	ECO450 (LT1)	LAW240 (LTH)						
	19	LAW460 (LT1)									
	20	LAW460 (B)	SOC322 (LTH)								
D	21	SOC422 (P)	LAW220 (Q)	LAW440 (T)							
A	22	COM322 (P)	LAW460 (GH)								
Y	23	ECO450 (B)	LAW140 (P)	LAW450 (LT2)	LAW240 (GH)						
	24	EDS204 (LT1)	GEO402 (LT2)	LAW470 (GH)							
3	25	LAW310 (K)									
	26	ALL321 (B)	LAW450 (T)	LAW240 (LT1)	ALL423 (CL)	LAW140 (LTH)					
	27	LAW220 (U)	SOC422 (W)	LAW440 (GH)							
	28	LAW430 (A)	ECO420 (P)	CHE422 (W)	LAW130 (GH)						
	29	LAW430 (B)	ALL422 (K)	LAW130 (LT1)	ECO420 (LT2)	LAW310 (GH)					
	30	LAW440 (N)	LAW220 (LT2)								
	31	LAW240 (W)	HIS422 (CL)								
D	32	LAW470 (LT1)	LED332 (LT2)								
A	33	LAW240 (A)	ECO450 (K)	LAW140 (N)							
Y	34	LAW430 (N)	ALL422 (O)	LAW310 (P)	LAW310 (T)	MUS421 (W)	LAW130 (L	ALL422 (LT	LAW310 ((	LAW130 (LTH)	
	35	SOC422 (B)	LAW220 (LT1)	LAW440 (LTH)							
4	36	GEO402 (LT2)	LAW470 (GH)								
	37	LAW460 (K)									
	38	GEO402 (LT1)	EDS204 (LT2)	LAW470 (GH)							
	39	ALL422 (N)	LAW130 (O)								
	40	LAW220 (K)									
	41	LAW220 (LTH)									
D	42	LAW310 (B)	MUS421 (N)	LAW130 (T)	ECO420 (CL)	LAW430 (LTH)					
A	43	LAW440 (T)	LAW220 (CL)								
Y	44	MAT422 (LT1)	CHE322 (LT2)								
	44	STA322 (LT1)	LED302 (GH)								
5	45	LAW430 (A)	LAW130 (K)	CHE422 (LT1)	LAW310 (GH)						

Figure A.10: Sample timetable for smaller data set (red marks indicate clashes)