Exploring the applicability of Self Organization Principles of Biological Systems to Solve the University Course Timetabling Problem

A Thesis
Submitted to the Department of Computer Science
Faculty of Natural Sciences
UNIVERSITY OF GUYANA

in partial fulfilment of
the requirements for the
degree of

Bachelor of Computer Science

Department of Computer Science

by

Sheldon Tappin
&
Imran Mohamed

August 2010
UNIVERSITY OF GUYANA
FACULTY OF NATURAL SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

THESIS PANEL

__________________________________________
Malcolm Williams, MSc.

__________________________________________
Sherlock Licorish, MSc.

__________________________________________
Roger Gajraj, BSc.

__________________________________________
Eldon G Marks, MSc.
Primary Advisor

Students:
Sheldon Tappin
Imran Mohamed
ACKNOWLEDGEMENTS

First of all we would like to thank the Almighty for granting us health and fortitude throughout the duration of this project.

Also, we would like to express our gratitude to Mr Eldon Marks, whose insights, guidance and contributions were priceless and without which we would not have succeeded.

Mention must be made of Mr Malcolm Williams and Mr Roger Gajraj for providing insightful comments and suggestions.

Finally, to our family and friends for their unwavering support and encouragement, thank you.

Sheldon Tappin
Imran Mohamed
ABSTRACT

Several researches have explored numerous approaches to solve the university course timetabling problem. To date, none of these approaches have adequately solved the problem. The reasons range from the NP-complete nature of the timetabling problem, through to the non-cohesiveness of the problem and existing approaches. We have observed that the ontology of timetabling solutions resemble a self-organizing biological system. As a result, we explored the applicability of self-organization principles of biological systems to solve the university timetabling problem. An evaluation of timetabling efficiency showed a statistically significant improvement in excess of 100 percent over existing approaches.

Keywords: self-organization principles, biomimetics, sexual selection, university timetabling, mate selection
TABLE OF CONTENTS

THESIS PANEL .................................................................................................................. II

ACKNOWLEDGEMENTS ....................................................................................................... III

ABSTRACT .............................................................................................................................. IV

LIST OF TABLES ................................................................................................................... VII

LIST OF FIGURES ................................................................................................................ VIII

CHAPTER 1. INTRODUCTION ............................................................................................... 1

1.1 The Problem ..................................................................................................................... 1
1.2 A Brief Review of Existing Approaches ........................................................................... 2
1.3 Motivations for this Research .......................................................................................... 3
1.4 The Solution ..................................................................................................................... 4
1.5 Thesis Outline .................................................................................................................... 4

CHAPTER 2. BACKGROUND .................................................................................................. 5

2.1 The University Course Timetabling Problem .................................................................. 5
2.2 Existing Solutions & Their Limitations ............................................................................. 6
2.3 Self Organization Principles ............................................................................................. 8

CHAPTER 3: SOUTS – THE CONCEPT ................................................................................ 10

3.1 Decomposition of the Timetabling Problem .................................................................. 10
3.2 Similarities between the University Course Timetabling and a Natural System .............. 10
3.3 The self-organizing pattern of Mate Selection .................................................................. 11
3.4 The Formulated Concept ................................................................................................. 12

CHAPTER 4: DESIGN .......................................................................................................... 13

4.1 Static Design of the System ............................................................................................. 13
4.1.2 Analogies of Hard Constraints ...................................................................................... 15
4.2 Interaction of entities within the System .......................................................................... 17
4.2.1 Finding Potential Rooms at Timeslots ......................................................................... 17
4.2.2 Successful bonding of a course to a room at timeslot .................................................. 18
4.2.3 Competition ................................................................................................................ 19
4.2.4 Dropping Constraints .................................................................................................. 20

CHAPTER 5. IMPLEMENTATION ....................................................................................... 21

5.1 Architecture ................................................................................................................... 21
5.1.1 Entity .......................................................................................................................... 21
5.1.2 Knowledge Base .......................................................................................................... 22
5.1.3 Decision Engine .......................................................................................................... 22
5.1.4 Room at Timeslot ....................................................................................................... 23
5.1.5 Courses ....................................................................................................................... 23
5.1.6 The Populator ............................................................................................................. 24
5.1.7 The Environment ........................................................................................................ 24
5.2 Technologies Used & Justification .................................................................................. 25
5.3 Process Model Used & Justification ................................................................................. 26
LIST OF TABLES

Table 6.1: Categorization of Data into Small, Medium & Large Datasets ......................................................... 34

Table 6.2: Results of Experiment ......................................................................................................................... 35

Table 7.1: Details of Anova: Two-Factor without Replication ................................................................................. 37

Table 7.2: Details of Anova: Two-Factor without Replication showing P Value ................................................. 38
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>A Static View of the System</td>
<td>13</td>
</tr>
<tr>
<td>4.2</td>
<td>A Course searches for potential mates</td>
<td>17</td>
</tr>
<tr>
<td>4.3</td>
<td>Course bonds with a suitable room at timeslot</td>
<td>18</td>
</tr>
<tr>
<td>4.4</td>
<td>Competition between two courses</td>
<td>19</td>
</tr>
<tr>
<td>4.5</td>
<td>Dropping of Constrains by a course</td>
<td>20</td>
</tr>
<tr>
<td>5.1</td>
<td>An entity’s knowledgebase</td>
<td>22</td>
</tr>
<tr>
<td>5.2</td>
<td>An entity’s decision engine</td>
<td>22</td>
</tr>
<tr>
<td>5.3</td>
<td>The creation of entities by the Populator</td>
<td>24</td>
</tr>
<tr>
<td>5.4</td>
<td>Entities reside in an Environment</td>
<td>24</td>
</tr>
<tr>
<td>5.5</td>
<td>Class diagram showing relationships</td>
<td>27</td>
</tr>
<tr>
<td>5.6</td>
<td>KnowledgeBase Class</td>
<td>28</td>
</tr>
<tr>
<td>5.7</td>
<td>Program Class</td>
<td>28</td>
</tr>
<tr>
<td>5.8</td>
<td>DecisionEngine Class</td>
<td>28</td>
</tr>
<tr>
<td>5.9</td>
<td>Entity Class</td>
<td>29</td>
</tr>
<tr>
<td>5.10</td>
<td>Percept Class</td>
<td>29</td>
</tr>
<tr>
<td>5.11</td>
<td>ColourCode Class</td>
<td>29</td>
</tr>
<tr>
<td>5.12</td>
<td>Populator Class</td>
<td>29</td>
</tr>
<tr>
<td>5.13</td>
<td>Rules Class</td>
<td>30</td>
</tr>
<tr>
<td>5.14</td>
<td>Course Class</td>
<td>30</td>
</tr>
<tr>
<td>5.15</td>
<td>RoomAtTimeSlot Class</td>
<td>30</td>
</tr>
<tr>
<td>5.16</td>
<td>Timeslot Class</td>
<td>31</td>
</tr>
<tr>
<td>5.17</td>
<td>Lecturer Class</td>
<td>31</td>
</tr>
<tr>
<td>5.18</td>
<td>Timeslot Class</td>
<td>31</td>
</tr>
<tr>
<td>5.19</td>
<td>Environment Class</td>
<td>32</td>
</tr>
<tr>
<td>5.20</td>
<td>Room Class</td>
<td>32</td>
</tr>
<tr>
<td>5.21</td>
<td>DataBridge Class</td>
<td>32</td>
</tr>
</tbody>
</table>
Figure 5.22: DynamicControlGeneration Class ................................................................. 33

Figure 5.23: Entity Relationship Diagram ................................................................. 33

Figure 7.1: Results after using a small dataset ......................................................... 36

Figure 7.2: Results after using a medium dataset ..................................................... 37

Figure 7.3: Results after using a large dataset ......................................................... 37
CHAPTER 1. INTRODUCTION

Since time immemorial, there has always been the need for planning and scheduling. A response from the software world was to build real world applications that encompass scheduling. A few examples of such applications would be the ones that are employed in the areas of production planning, timetabling or personnel planning. It must be noted that there are efficient algorithms for certain classes of well-defined problems. However, these algorithms are often very specific and slight changes in the problem definition raises difficulties in the adaptation of these special purpose algorithms (Legierski, 2002).

In response to the numerous problem specific algorithms, various generalized algorithms have been developed and several attempts have been made to solve a wider scope of scheduling problems. However, to date, there is not a generalized solution that can be applied to fully solve any scheduling problem and more specifically, the University Course Timetabling Problem, which is the focus of this thesis (Legierski, 2002), (Burke, Marecek, Parkesa, & Rudováb, 2009), (Dimopoulou & Miliotis, 2004).

1.1 The Problem

According to Schaerf (1999), timetabling involves the scheduling of a sequence of lectures between teachers and students in a predefined period of time, satisfying a set of constraints of various types. It was stated by Kostuch (2005) that the timetabling problem involves the placing of entities into a timetable made up of a limited number of atomic units called time slots.

As universities grow and the teaching programs become more complicated, the already complex task of scheduling courses is becoming even more complex and time consuming (Dimopoulou & Miliotis, 2004). Currently the majority of approaches used as timetabling
solutions are computationally based in nature. This means the outcome produced would always be the result of some mathematical computation which is based on some predefined mathematical model. The outcomes of these calculations are always predetermined by some formula which cannot dynamically change to meet the requirements of the dynamic nature of a timetabling system. Computational approaches are defined within a context and the components of the system are governed by certain rules which cannot be broken (ASAP Research Group Seminars 2004 - 2005).

There are a few approaches that are patterned after natural or biological systems. However, these approaches also have limitations as well (Marczyk, 2004).

1.2 A Brief Review of Existing Approaches

According to Norberciak (2006), approaches to the University Course Timetabling problem can be categorized under four broad headings. These would include sequential methods, cluster methods, constraint based methods and meta-heuristic methods.

Sequential methods order events using domain heuristics and then places events sequentially into valid timeslots so that no events conflict with each other in the same period. Generally, the most difficult or constrained events are assigned to timeslots first (Norberciak, 2006).

With cluster based approaches, a cluster can contain any two events that do not conflict with each other. According to Zibran (2007) the cluster method divides the events into groups which satisfy the hard constraints and then the groups are assigned to time periods to satisfy the soft constraints.
Constraint-based methods are another approach that attempts to solve the University course timetabling problems. Generally, in a constraint-based approach, a problem is represented as a set of variables with a restricted domain. A number of constraints are fulfilled by assigning values to the variables. Constraint logic programming is often used to obtain a feasible initial timetable and serves as a starting point for another search method since it is able to quickly generate a solution. Constraint based approaches have been used quite effectively when combined with other local search methods (Abdullah, 2006).

Meta-heuristic methods begin with one or more initial solutions and employ search strategies that try to avoid local optima. An initial feasible solution is then used in an attempt to find an optimal solution by iteratively searching in its neighborhood. A neighborhood is a set of feasible solutions obtained by changing one parameter of the current solution (Zibran, 2007).

1.3 Motivations for this Research

Many studies suggest that the university course timetabling problem is almost impossible to solve (Pongcharoena, Promtettb, Yenradd, & Hicksd, 2008), (Melício, Caldeira, & Rosa). Numerous attempts have been made to fully and optimally solve the timetabling problem, however, to date; there is no single solution that satisfies all of both hard and soft constraints (Abdennadher).

All of the existing solutions have limitations and by shifting the solution from a mathematical and computational to a more natural realm, maybe this problem can be solved. By using nature which is self-sustaining and has self-optimized over billions of years as a model which the timetabling problem can be patterned after, a decision was taken to explore the applicability of self-organizing principles derived from a natural system and apply it to the
timetabling problem. Hence, this research seeks to explore the feasibility and applicability of principles found in self-organizing biological systems to the timetabling problem.

1.4 The Solution

A web application – SOUTS; was developed using a natural system and the principles of that natural system as a model. The timetabling problem was decomposed into discrete, autonomous entities, resources, behaviors and attraction. By doing this decomposition, the entities of the timetabling problem were abstracted and their respective parallels in a natural system were found. The main objective of this approach is to mimic a self-organizing natural system and in doing so, produce an optimal, clash free solution.

1.5 Thesis Outline

This thesis documents the creation, implementation and testing of SOUTS - a system that utilizes self-organizational principles of biological systems to solve the University Course Timetabling Problem. Chapter 2 presents relevant background information followed by Chapter 3, in which the proposed concept of SOUTS is presented and discussed. Chapter 4 presents the design of the system based on the concept. Chapter 5 documents the implementation of the model discussed in the previous chapter. Chapter 6 describes the method and evaluation exercise and presents the results from the experiment. This is followed by a discussion of the results from the evaluation in Chapter 6. Finally, Chapter 8 concludes with insights and recommendations for future work.
CHAPTER 2. BACKGROUND

In this chapter, the University Course timetabling problem is explained in detail, followed by a detailed overview of existing solution and their limitations. Finally, an explanation of the concept of self-organization is presented.

2.1 The University Course Timetabling Problem

Every year, the university administration invests large amounts of human and material resources to schedule timetables and solve the many problems associated with such a process (Abdullah, 2006). The results of manual timetabling may be unsatisfactory; hence, considerable work has been done over the past few decades starting with Gotlieb, (1963) to alleviate the problems. Many applications have also been developed and employed with good success (Schaerf, 1999).

Wren (1996) defines timetabling as follows: “Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives” (Abdullah, 2006). Abdullah, (2006) classified educational timetabling into three main groups i.e. school timetabling, examination timetabling and university timetabling. Timetabling at universities can be broken down into two categories: examination and course timetabling. Even though they have distinct differences in some instances, they share many common characteristics such as requiring rooms at a particular time etc. One common difference is that examination timetables can accommodate more than one exam per room whilst course timetables do not allow this (Zibran, 2007). For a timetabling system to be feasible, it has to satisfy “hard” constraints and try to satisfy as much “soft”
constraints as possible (Legierski, 2002). Pongcharoena et al (2008) listed some hard and soft constraints. They are as follows:-

**Hard Constraints:**

- Students can only have one lecture at a time.
- A lecturer can only deliver one lecture at a time.
- A classroom can only be used for one lecture at a time.
- Allocated rooms must be large enough to accommodate the students.
- No two courses must be scheduled at the same time in a particular room.

**Soft Constraints:**

- Events should be scheduled at locations within their locality.
- There shouldn’t be two consecutive events i.e. one exactly after the other.
- There should be at most two events per day.
- Students should have consecutive lectures in the same building.
- The number of lectures by a particular lecturer should be limited.

### 2.2 Existing Solutions & Their Limitations

According to Filipe & Albano, many authors believe that the timetabling problem cannot be fully automated. One reason is that there are details that make one timetable better than another one which cannot easily be expressed in an algorithm. Another reason is that, since it usually involves a huge search space, human intervention may bias the search toward promising directions that the system by itself may be not able to find. Another limitation is that some algorithms work well for earlier searches but can produce clashes later on (Zibran, 2007). As mentioned earlier, approaches to the timetabling can be classified under four broad groups.
These would include sequential methods, cluster methods, constraint based methods and metaheuristic methods.

In sequential methods, graphs are usually used to represent timetabling problems where events are represented as vertices, while conflicts between the events are represented by edges. For example, if a lecturer has to attend to two events there is an edge between the nodes which represent this conflict. The construction of a conflict-free timetable can therefore be modelled as a graph colouring problem where each time period in the timetable corresponds to a colour in the graph colouring problem and the vertices of a graph are coloured in such a way so that no two adjacent vertices are coloured by the same colour (Müller, 2005). A known limitation of sequential approaches is that they might not be able to produce a high quality solution with respect to the satisfaction of the soft constraints (Abdullah S., 2006).

Cluster methods uses groups of events and tries to satisfy hard constraints. The groups are then assigned to time periods to satisfy soft constraints. With this approach, a solution may be found quickly but in some cases they may result in a poor timetable if there are many dependencies between events of different clusters (Zibran, 2007). Another drawback is that clusters of events are formed and fixed at the beginning and may result in a poor quality timetable (Norberciak, 2006). Another limitation lies with assigning which cluster to which period. In most cases, the assignment of clusters to periods is done so as to minimise the number of adjacent conflicts. However, by doing this, it may not be possible to construct a good quality timetable once clusters are formed (Burke, Newall, & Weare, 1998).

With constraint based methods, a timetabling problem is modeled as a set of variables (i.e., events) to which values (i.e., resources such as rooms and time periods) have to be assigned to satisfy a number of constraints. A number of rules are defined for assigning resources to events. When no
rule is applicable to the current partial solution, a backtracking is performed until a solution is found that satisfies all constraints (Müller, 2005). An example of a constraint based approach is constraint logic programming. Constraint based algorithms also have limitations. This can include the lack of ability to further enhance the quality of the generated solution. According to Abdullah (2006), Brailsford et al. (1999) said that pure constraint-based approaches cannot compete with the state of the art local search methods.

Meta heuristic based algorithms present a general algorithmic framework for addressing intractable problems. In many instances, they are often inspired by processes occurring in nature, e.g. annealing, collective behaviour of ants etc. Some meta-heuristic based methods include simulated annealing, tabu search and genetic algorithms (Abdullah, 2006). With this approach, many initial solutions are generated and then through the process of swapping, infeasible solutions are truncated. This happens until one solution remains (Zibran, 2007). Typically, meta-heuristics are approximation algorithms – they cannot always produce provably optimal solutions, but they do have the potential to produce good solutions in short amounts of time (if used appropriately) (Rhyd). Meta-heuristic methods can produce high quality solutions but may have a high computational overhead (Zibran, 2007).

2.3 Self Organization Principles

In the last few years, nature-inspired self-organization has been applied to the field of computer science. The ever increasing complexity and dynamic nature of software makes it impossible to think in advance of the complex interactions, and the cascading side-effects arising in complex distributed applications. For this reason, software can no longer be managed with outmoded techniques. Components must be able to anticipate and cope with dynamic changes and complicated situations. Nature, which is self-organizing and self-sustaining provides an ideal
example of a system with these capabilities. The amazing and spectacular organization of insect colonies and biological cells provides us with ample evidence that natural systems are able to organize their activities and achieve tasks with exceptional robustness despite harsh and dynamic environments (Mamei, Menezes, Tolksdorf, & Zambonelli, 2006).

According to Camazine et al. (2001), “self-organization is a process in which a pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern”. The self-organizing principles of biological and natural systems have been used by computer scientists to enhance their understanding of certain problems. This enhanced understanding better equips them in their bid to find better solutions to certain problems. One such system is neural networks. In neural networking, the “neurons” are either directly or indirectly connected, but none is in control. Distributed control enables the system to make sense out of complex patterns of input (Heylighten, 2009). Self-organization principles have also been applied to computer systems design. European researchers have developed a system where there are many small modules, each made from chips with an inbuilt ability to learn. They evolve to suit the task at hand by acting on information about their environment, thus improving efficiency (Bio-Inspired Computer Networks Self-Organize and Learn, 2010). This phenomenon of taking principles from nature and applying them to a real world problem in a bid for a solution is captured in the term biomimetics. Biomimetics is defined as “the examination of nature, its models, systems, processes, and elements to emulate or take inspiration from in order to solve human problems” (Biomimicry, 2010).
CHAPTER 3: SOUTS – THE CONCEPT

In this chapter, the timetabling problem is decomposed and the similarities between the university course timetabling problem and natural systems are presented. This is followed by an explanation of mate selection, which is a pattern of self-organization. This chapter concludes with a presentation of the developed concept.

3.1 Decomposition of the Timetabling Problem

In order to better understand the University Course Timetabling Problem, it was decomposed. When a problem is decomposed, it is broken down into smaller sub problems which should be simpler than the large problem (Jackson & Jackson, 1995). The problem was decomposed into generic entities, relationships, behaviours, resources and interactions. It was noted that it drew a parallel to a natural system because of the autonomous nature of entities both in the timetabling and natural systems. Also, there are relationships among entities, which compete for scarce resources in an attempt to satisfy their wants. In the timetabling problem, there is this pairing off that happens where a course has to be paired off with rooms at different timeslots. This pairing off can be seen in natural / biological systems and can be encapsulated in a concept called mate selection.

3.2 Similarities between the University Course Timetabling and a Natural System

A timetabling system is one that is initially dynamic in nature, but eventually becomes stable if it is successful in its quest to attain equilibrium, i.e. when a feasible timetabling solution is produced. In this regard, the timetabling problem domain can be viewed as an environment in which various entities interact. These entities include courses and rooms at specific timeslots. The eventual organization of the structure of the timetabling system is governed by a number of
rules known as hard constraints and soft constraints. A characteristic of the timetabling system would be the need for entities in the timetable to be organized in such a way as to avoid conflicts. Another characteristic is that before an entity makes a decision, it tests its neighborhood space to see if it is feasible to occupy; if the feedback is positive it then occupies it, else, it does not. There is also competition among entities of the system for very scarce resources in order to bring about a resolution of rightful ownership of the resource in question. The entities within the system iteratively testes for the aforementioned characteristics and if possible, rearranges themselves until they are optimally placed in their rightful position within the system. These characteristics and behaviors can also be found in natural systems.

3.3 The self-organizing pattern of Mate Selection

Mate selection also known as sexual selection is a “special case” of natural selection and can be defined as an organism's ability to obtain (often by any means necessary!) or successfully copulate with a mate (Evolution 101: Sexual Selection). Mate selection is very evident in nature, for example; female peacocks choose their mate (the male) who has brightest, longest, and most decorated tail. Whenever more than one male is attracted to a female, the males compete with each other by spreading their brightly feathered tail to attract potential mates (females) (Ehrlich, Dobkin, & Wheye, 1988). Another example of mate selection in nature involves deer. Females are attracted and ultimately mates with the strongest male who possess the largest antlers they come in contact with. If there are more than one male deer attracted to a female, they faces off and the winner gets to mate with the female (Deer Antlers; it’s not all about sex, 2007). The aforementioned examples reiterate the concept of mate selection in nature and the many similarities it shares with the timetabling problem.
It is evident that mate selection exemplifies the principles of self-organization and thus falls under the umbrella of self-organization in which autonomous entities find mates based on their own instinct without external interference (Hoelzer, Smith, & Pepper, 2006), (Heylighten, 2009)

3.4 The Formulated Concept

The timetabling problem is analogous to mate selection. To prove this fact, the timetabling problem is made of courses (males) that have to be paired off with particular rooms at timeslots which are typically scarce. Because of this scarcity of resources, competition will come into play. Courses have to scan through a list of rooms to make sure that it meets its requirements. Requirements such as a course with a huge student population will need a room that can contain its entire class. Given that a suitable room is found, an appropriate timeslot must be agreed upon. A course is required to be held many times a week according to its credits. This phenomenon is analogous to the polygamous behaviour of many animals. The pairing off process described above deeply resembles the activities carried out by animals during the process of finding mates.

In SOUTS, a course represents a male in a natural system, a room at a specific timeslot is presented as the female and the hard and soft constraints are presented as the laws of attraction of males and females in natural systems.
CHAPTER 4: DESIGN

Creating a timetabling system based on natural systems required exploring two major research areas, namely self-organization and the timetabling problem. The timetabling problem had to be decomposed in such a way as to draw parallels to a natural system. After these parallels were drawn, a specific self-organizing system had to be chosen and a model of that natural system had to be created. This chapter describes the work done to design the Self Organizing University Timetabling System (SOUTS).

4.1 Static Design of the System

The diagram in Figure 4.1 represents the overall system and its components. A course represents a male, a room at a particular timeslot represents a female and attraction encapsulates...
the hard and soft constraints. A compromise is seen as the lowering of soft constraints in order to find mates.

Courses are presented as males because in numerous natural systems, the male is the entity that actively pursues a mate or potential mates. In a timetabling system the course has to be assigned to a specific room at timeslot. The room at a specific timeslot is presented as the female because in nature, the female in most cases relies on the male to pursue her. The same can be seen in the timetabling system where a room at a specific timeslot is “given” a course. In nature, a male and female are attracted based on some law of attraction. There are many factors that are encapsulated under that law, such as strength and physical characteristics etc. In timetabling, there are hard and soft constrains that would govern the attraction of courses to room at timeslots. These hard and soft constraints of the timetabling system are presented as the laws of attraction based on the aforementioned similarities.

In SOUTS, a course is attracted to rooms at specific timeslots based on the hard and soft constraints. It will bond with a specific room at timeslot once it satisfies the constraints criteria of that room at timeslot. If another course is already bonded to the room at timeslot the course wants to bond with, there is competition where the constraints of both courses are weighed off against each other and the one that is better suited for the room at timeslot in question wins the competition. The loser of the competition then has to go and search for other rooms at timeslots. If there is a case where courses share a common lecturer of students, only one of the courses can bond with the room at timeslot in question while the other has to search for another.
4.1.2 Analogies of Hard Constraints

The hard constraints considered for this system were presented as a series of analogies that made them easier to understand in relation to a natural system.

- A selected classroom should be large enough for the assigned class.
  - This is analogous to mate selection based on attraction where the female possesses certain characteristics.

- Up to one course can be taught in a room at a given timeslot.
  - This is analogous to mating where only one male can mate with one female at a time.

- A lecturer can teach up to one course in a given timeslot.
  - This is analogous to bigamy where one male has more than one mate that doesn’t know about each other. In this scenario, the male can only be with one of his mate at a time.

- A class can attend up to one course in a given timeslot.
  - This scenario can be analogous to the outcome of a bigamous relationship. The children can only reside with one set of families at a time.

- A class should be scheduled based on the requirements of that class (if a special room needs to be used etc.).
  - This is analogous to mate selection based on attraction where the female possesses certain characteristics that the male desires.

- Up to one session is allowed for a course per day (a session can be a 1 or 2 hr. timeslot).
  - This is analogous to a male that can only mate once daily.
• Each lecturer must deliver a specified number of lectures per week.
  
  o This is analogous to a case where a male should only mate a specified number of times per week.

• A given course allotted to a timeslot must be assigned a room or rooms that accommodate the class of the said course.
  
  o This is analogous to the mate selection criteria if the class can find a room to hold it in one timeslot. However, if that is not the case, a course must fragment (split itself on class size). This is done initially and is analogous to embryo dividing to create twins. They look alike but make use of different mates.
4.2 Interaction of entities within the System

This section presents the dynamic design of the system where the entities are interacting. These interactions are presented in the form of sequence diagrams.

4.2.1 Finding Potential Rooms at Timeslots

Interactions within the system start off with Courses seeking potential RoomAtTimeSlots (see Fig 4.2). Before a RoomAtTimeslot can be deemed as potential mate, it must meet or exceed the Course’s expectations, which are laws of attraction that the Course must abide by. This is a process that all Courses must go through to make sure that that they identify each and every RoomAtTimeslot that meets their criteria.

![Diagram](image.png)

**Figure 4.2:** A Course searches for potential mates
4.2.2 Successful bonding of a course to a room at timeslot

Given that the Course has found its best potential mate, its natural instinct would be to solicit that RoomAtTimeslot; hence the Course will try to solicit the RoomAtTimeslot. If the RoomAtTimeslot accepts the Course’s solicitation, the Course will be informed and the two Entities can proceed to bond (see Fig 4.3). The bonding process can be likened to a natural bonding of entities in the real world where each has an intimate knowledge of the other after bonding.
4.2.3 Competition

![Diagram of Competition between two courses]

**Figure 4.4:** Competition between two courses

There are cases where the RoomAtTimeSlot rejects the course’s solicitation and if the reason for the rejection is because the RoomAtTimeSlot is already bonded, then it will be the in course’s interest to find out who the RoomAtTimeSlot is bonded to. Once the Course gets that information, it will confront the other Course and initiate a competition (see Figure 4.4). In the system, Courses compete by weighing off their constraints against each other, and whichever Course best fits the RoomAtTimeSlot’s expectations will be given the chance to bond with the said RoomAtTimeSlot and the looser will have to search for another RoomAtTimeSlot.
4.2.4 Dropping Constraints

There will be cases in which a Course searches the entire Environment and cannot find a single potential mate. In this case, the Course will have to drop one of its expectations and begin searching the Environment again to see if it can find potential mates. If the Course searches again and still cannot find potential mates then another of its expectations is dropped. This procedure is repeated until the Course finds potential mates (see Fig 4.5).

Figure 4.5: Dropping of Constrains by a course
CHAPTER 5. IMPLEMENTATION

In order to evaluate whether self-organization principles of natural system can be applied to solve the university course timetabling; the designed model was implemented in the form of a web application named the Self Organizing University Timetabling System (SOUTS). This chapter describes the implementation of SOUTS with respect to its interface and architecture as well as the technologies and process model used and justifications for their use.

5.1 Architecture

The developed architecture of SOUTS is made up of seven major components based on the design model. These components are: (1) entity (2) the knowledge base, (3) the decision engine, (4) rooms at timeslots, (5) courses, (6) the populator and (7) the environment.

5.1.1 Entity

An Entity is an Object that is composed of a Knowledgebase and a Decision Engine along with a unique identifier so that it can be distinguished from all other entities within the system. This combination gives an entity the ability to learn, think, and make decisions based on past and current events as any typical real world entity would.
5.1.2 Knowledge Base

![Knowledge Base Diagram](image)

**Figure 5.1:** An entity’s knowledgebase

The *Knowledgebase* knowledge (see Figure 5.1) is a software component used by entities to store information about themselves, other entities and the environment. It has the capabilities to gain new found knowledge, update existing knowledge and also forget previously stored.

5.1.3 Decision Engine

![Decision Engine Diagram](image)

**Figure 5.2:** An entity’s decision engine
The Decision Engine (see Figure 5.2) is a software component that is used by entities to make decisions. It uses existing knowledge stored in its Knowledgebase along with a perception of the environment and or other entities to formulate a decision.

5.1.4 Room at Timeslot

The RoomAtTimeslot (female) is an entity that is composed of a Timeslot Object and a Room Object; hence it is a combination of the two objects along with heredity from the Entity class.

- The Timeslot class has properties such as day, startTime and endTime along with methods that manipulate these properties.
- The Room class consists of the name of a particular room, its location and its maximum capacity, along with its methods.

5.1.5 Courses

A Course (male) is an entity that exemplifies the properties of a typical course at any university such as a course’s credits, code, and title. The Course Object is created with a natural instinct to solicit RoomAtTimeSlots, and if it is confronted by any other course for that particular RoomAtTimeslot then the only natural thing to do is for the two courses to compete. The Course makes use of its Knowledgebase in accessing and selecting the most suitable mate.
5.1.6 The Populator

The first action that occurs within SOUTS is the creation of entities. This is done through the Populator class; it accepts raw information about courses such as title, credits, etc. and rooms such as capacity, location, etc. from an external data source and create entities (see Figure 5.3). There are two types of entities within the system; they are Courses (Males) and RoomAtTimeSlots (Females). After these entities are created they are placed in the Environment.

5.1.7 The Environment

![Diagram of entities in an Environment](image_url)
The Environment (see Figure 5.4) is a software component that acts as a container. All entities reside in the environment in their respective niches. Courses reside in the Program niche and RoomAtTimeSlots reside the Timeslot niche. Entities interact and settle their disputes, such as competition, in the Environment.

5.2 Technologies Used & Justification

The choice of a programming language is a very important decision. Understanding the nature of the application to be developed is critical in choosing the technologies to be used for the development of the application in question. After careful analysis, the technologies chosen to implement the system were Visual Basic.Net (programming language), ASP.Net (web interface generator), IIS server and the database used was MSSQL 2005.

ASP.NET and the .NET Framework simplify web development by separating the application logic and presentation logic making it easier to maintain the code. The presentation logic and the application logic are separate, thus eliminating the need to mix HTML with logic code. ASP.NET can also handle the details of maintaining the state of the controls, such as contents in a textbox and between calls to the same ASP.NET page (Advantages of .NET Framework). Visual Basic.Net was used because of our familiarity with the language and its native support for Object Oriented Programming (OOP).

There was need for rapid development and as such the .NET Framework was used because it supports rapid application development. MSSQL 2005 server was used because it is natively supported by the .NET Framework. Finally, the development environment used was Microsoft Visual Studio 2008. This IDE is a powerful, feature rich development environment.
5.3 Process Model Used & Justification

Developing reliable software within the time scheduled is a difficult task for many software developers. Any flaws or late delivery of a system means a great deal for many individuals involved. It is indeed vital to produce reliable software right on schedule to avoid inconveniences for the developers, vendors and users. The software community places great hope on software modelling notations and techniques to ease various software development challenges. One of the challenges is the requirement to creatively analyse and design problem-solving technique with a highly coordinated development team within a complex environment (Atan, Ghani, Selamat, & Mahmod, 2007).

Software process modelling (SPM) is one of the techniques used to creatively define and analyse significant aspects, which can be adapted into intricate application development and also can be used to structure a strategic co-ordination for the development team (Atan, Ghani, Selamat, & Mahmod, 2007). The process model chosen for this system was Extreme Programming (XP). Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation (Extreme Programming: A Gentle Introduction, 2009).

Extreme Programming emphasizes teamwork. Customers and developers are all equal partners in a collaborative team. Extreme Programming implements a simple, yet effective environment enabling teams to become highly productive. The team self-organizes around the problem to solve it as efficiently as possible.
Extreme Programming improves a software project in five essential ways; communication, simplicity, feedback, respect, and courage. Extreme Programmers constantly communicate with their customers and fellow programmers. The design is kept simple and clean. Feedback is gotten by testing of the software starting on day one. The system is delivered to the customers as early as possible and changes are implemented as suggested. Every small success deepens their respect for the unique contributions of each and every team member (Extreme Programming: A Gentle Introduction, 2009).

5.4 Class Diagram

The diagram below shows the relationships that exist among the classes that makes up SOUTS.

**Figure 5.5:** Class diagram showing relationships
5.4.1 Class Diagram 2

The diagrams below show all the classes of the system as well as their properties and methods.

<table>
<thead>
<tr>
<th>KnowledgeBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>- knowledge : HasTable</td>
</tr>
<tr>
<td>+ KnowledgeBase(): void</td>
</tr>
<tr>
<td>+ addKnowledge(key : String, value : Object): void</td>
</tr>
<tr>
<td>+ getKnowledge(key : String): Object</td>
</tr>
<tr>
<td>+ forgetKnowledge(key : String): Void</td>
</tr>
</tbody>
</table>

**Figure 5.6:** KnowledgeBase Class

<table>
<thead>
<tr>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>- programYear : HasTable&lt;Course&gt;</td>
</tr>
<tr>
<td>- programID: Integer</td>
</tr>
<tr>
<td>+ Program(programID): void</td>
</tr>
<tr>
<td>+ getProgramID(): Integer</td>
</tr>
<tr>
<td>+ getYear(index : Integer): ArrayList&lt;Course&gt;</td>
</tr>
<tr>
<td>+ getYearCount(): Integer</td>
</tr>
<tr>
<td>+ addCourse(course: Course): Void</td>
</tr>
<tr>
<td>+ getCourse(course: String): Course</td>
</tr>
<tr>
<td>+ removeCourse(course: Course): Void</td>
</tr>
</tbody>
</table>

**Figure 5.7:** Program Class

<table>
<thead>
<tr>
<th>DecisionEngine</th>
</tr>
</thead>
<tbody>
<tr>
<td>- entityName: String</td>
</tr>
<tr>
<td>- evaluatedResults: HasTable</td>
</tr>
<tr>
<td>+ DecisionEngine(entityName: String)</td>
</tr>
<tr>
<td>+ evaluate(percept: Percept, expectation: HasTable&lt;Object&gt;, entity: Entity): HasTable</td>
</tr>
<tr>
<td>+ analyse(roomAtTimeSlot: Timeslot, evaluatedRoomProperties: HasTable&lt;Object&gt;): Boolean</td>
</tr>
<tr>
<td>+ examineExpectation(results: HasTable&lt;Object&gt;)</td>
</tr>
<tr>
<td>+ dropExpectation(expectation: HasTable&lt;Object&gt;, course: Course): Boolean</td>
</tr>
<tr>
<td>+ resetEvaluatedResults(): Void</td>
</tr>
</tbody>
</table>

**Figure 5.8:** DecisionEngine Class
<table>
<thead>
<tr>
<th>Entity</th>
<th># id: String</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- knowledgeBase: KnowledgeBase</td>
</tr>
<tr>
<td></td>
<td>- decisionEngine: DecisionEngine</td>
</tr>
<tr>
<td></td>
<td>+ Entity(entityName: String)</td>
</tr>
<tr>
<td></td>
<td>+ getID(): String</td>
</tr>
<tr>
<td></td>
<td>+ setID(id: String): Void</td>
</tr>
<tr>
<td></td>
<td>+ getKnowledgeBase(): KnowledgeBase</td>
</tr>
</tbody>
</table>

**Figure 5.9:** Entity Class

<table>
<thead>
<tr>
<th>Percept</th>
<th>- perception: Hashable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+ addPercept(perceptKey: String, Percept: Object): Void</td>
</tr>
<tr>
<td></td>
<td>+ getPercept(perceptKey: String): Object</td>
</tr>
<tr>
<td></td>
<td>+ getPerception(perceptName: String, perceptValue: Object): Void</td>
</tr>
</tbody>
</table>

**Figure 5.10:** Percept Class

<table>
<thead>
<tr>
<th>ColourCode</th>
<th>- dataSource: DataBridge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- programList: HashTable&lt;OBJECT&gt;</td>
</tr>
<tr>
<td></td>
<td>+ colourCode(): Void</td>
</tr>
<tr>
<td></td>
<td>+ colourCodeProgram(): Void</td>
</tr>
<tr>
<td></td>
<td>+ highlightProgram(programName: String): ArrayList&lt;String&gt;</td>
</tr>
</tbody>
</table>

**Figure 5.11:** ColourCode Class

<table>
<thead>
<tr>
<th>Populator</th>
<th>- dataSource: DataBridge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- environment: Environment</td>
</tr>
<tr>
<td></td>
<td>+ Populator()</td>
</tr>
<tr>
<td></td>
<td>+ EntityCreationGene(getQuery: String): String</td>
</tr>
<tr>
<td></td>
<td>+ generateEntities(): ArrayList(entities)</td>
</tr>
<tr>
<td></td>
<td>+ populate(): Void</td>
</tr>
<tr>
<td></td>
<td>+ getEnvironment(): Environment</td>
</tr>
</tbody>
</table>

**Figure 5.12:** Populator Class
Figure 5.13: Rules Class

<table>
<thead>
<tr>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>- ruleOperator: String</td>
</tr>
<tr>
<td>- ruleValue: Object</td>
</tr>
<tr>
<td>+ Rules(ruleValue: Object, ruleOperator: String)</td>
</tr>
<tr>
<td>+ setOperation(ruleOperation: String)</td>
</tr>
<tr>
<td>+ setValue(ruleValue: Object)</td>
</tr>
<tr>
<td>+ getOperation(): Object</td>
</tr>
<tr>
<td>+ getValue(): String</td>
</tr>
</tbody>
</table>

Figure 5.14: Course Class

<table>
<thead>
<tr>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>- courseCode: String</td>
</tr>
<tr>
<td>- courseCredits : Integer</td>
</tr>
<tr>
<td>- courseTitle: String</td>
</tr>
<tr>
<td>+ Course(): Void</td>
</tr>
<tr>
<td>+ Course(courseCode: String, courseTitle: String, courseCredits: Integer): Void</td>
</tr>
<tr>
<td>+ getCode(): String</td>
</tr>
<tr>
<td>+ getCredits(): Integer</td>
</tr>
<tr>
<td>+ getTitle(): String</td>
</tr>
<tr>
<td>+ setCode(courseCode: String): Void</td>
</tr>
<tr>
<td>+ setCredits(courseCredits: Integer): Void</td>
</tr>
<tr>
<td>+ setTitle(courseTitle: String): Void</td>
</tr>
<tr>
<td>+ bond(roomAtTimeSlot: RoomAtTimeSlot): Void</td>
</tr>
<tr>
<td>+ communicate(percept: Percept): Void</td>
</tr>
<tr>
<td>+ compete(atPercept: Percept): Double</td>
</tr>
<tr>
<td>+ findPotentialMates(roomAtTimeSlot: RoomAtTimeSlot, expectations: Rules): ArrayList&lt;RoomAtTimeSlot&gt;</td>
</tr>
<tr>
<td>+ getBestMate(potentialMates: ArrayList&lt;RoomAtTimeSlot&gt;): RoomAtTimeSlot</td>
</tr>
<tr>
<td>+ setExpectations(programs: Hashtable&lt;Programs&gt;): Hashtable&lt; Rules&gt;</td>
</tr>
<tr>
<td>+ solicitBond(roomAtTimeSlot: RoomAtTimeSlot): Boolean</td>
</tr>
<tr>
<td>+ unbind(roomAtTimeSlot: RoomAtTimeSlot): Void</td>
</tr>
</tbody>
</table>

Figure 5.15: RoomAtTimeSlot Class

<table>
<thead>
<tr>
<th>RoomAtTimeSlot</th>
</tr>
</thead>
<tbody>
<tr>
<td>- room: Room</td>
</tr>
<tr>
<td>- timeslot: Timeslot</td>
</tr>
<tr>
<td>+ RoomAtTimeSlot(): Void</td>
</tr>
<tr>
<td>+ RoomAtTimeSlot(room: Room, timeslot: Timeslot): Void</td>
</tr>
<tr>
<td>+ bond(course: Course): Void</td>
</tr>
<tr>
<td>+ communicate(sender: Course, message: Percept): Boolean</td>
</tr>
<tr>
<td>+ expectations(): Hashtable&lt;OBJECT&gt;</td>
</tr>
<tr>
<td>+ getRoom(): Room</td>
</tr>
<tr>
<td>+ getTimeslot(): Timeslot</td>
</tr>
<tr>
<td>+ unbind(): Void</td>
</tr>
<tr>
<td>Timeslot</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>- day : String</td>
</tr>
<tr>
<td>+ Timeslot(day : String, startTime : Integer, duration : Integer)</td>
</tr>
<tr>
<td>+ getDay() : String</td>
</tr>
<tr>
<td>+ isEqual(timeSlot : TimeSlot) : Boolean</td>
</tr>
</tbody>
</table>

**Figure 5.16:** Timeslot Class

<table>
<thead>
<tr>
<th>Lecturer</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- department : Integer</td>
<td>- firstName: String</td>
<td>- lastName: String</td>
<td>- lecturerID: String</td>
</tr>
<tr>
<td>- timeAvailable : ArrayList&lt;TimeSlot&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ Lecturer() : Void</td>
<td>+ Lecturer(lectorID: Integer, lName: String, fName: String, dept: integer, timeNotAvailable: ArrayList&lt;TimeSlot&gt;): Void</td>
<td>+ getFirstName() : String</td>
<td></td>
</tr>
<tr>
<td>+ getLastName(): String</td>
<td>+ getFullName(): String</td>
<td>+ getLecturerID(): String</td>
<td></td>
</tr>
<tr>
<td>+ getTimesNotAvailable(): ArrayList&lt;TimeSlot&gt;</td>
<td></td>
<td>+ setTimesNotAvailable(ArrayList&lt;TimeSlot&gt;): Void</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.17:** Lecturer Class

<table>
<thead>
<tr>
<th>Timeslot</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- day : String</td>
<td>- duration : Integer</td>
<td>- startTime : Integer</td>
<td></td>
</tr>
<tr>
<td>+ Timeslot(day : String, startTime : Integer, duration : Integer)</td>
<td>+ Timeslot(day : String, startTime : Integer)</td>
<td>+ equal(timeSlot : TimeSlot) : Boolean</td>
<td></td>
</tr>
<tr>
<td>+ getDay() : String</td>
<td>+ getDuration() : Integer</td>
<td>+ getStartTime() : Integer</td>
<td></td>
</tr>
<tr>
<td>+ isEqual(timeSlot : TimeSlot) : Boolean</td>
<td>+ isEqual(timeSlot : TimeSlot) : Boolean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.18:** Timeslot Class
### Environment
- **dataSource**: `dataBridge`
- **entities**: `ArrayList<Objects>`
- **programs**: `Hashtable<Program>`
- **RoomAtTimeSlot**: `Hashtable<RoomAtTimeSlot>`

```java
+ Environment(programs: HashTable, roomAtTimeSlots: HashTable, entities: ArrayList<Entity>): void
+ getProgram(a Program: Program): Program
+ startCourse(): Void
+ interaction(course: Course): Void
+ generateTimeTable(): Void
+ courseStatistics(courseStatisticsAbout: String): Integer
+ census(entity: String): Integer
```

**Figure 5.19**: Environment Class

### Room
- **roomName**: `String`
- **roomSize**: `Integer`
- **roomLocation**: `Integer`

```java
+ Room()
+ Room(roomName: String, roomSize: Integer, roomLocation: Integer)
+ setName(roomName: String): Void
+ setSize(roomSize: Integer): Void
+ setLocation(roomLocation: Integer): Void
+ getName(): String
+ getSize(): Integer
+ getLocation(): Integer
```

**Figure 5.20**: Room Class

### DataBridge
- **adapter**: `sqlDataAdapter`
- **conn**: `sqlConnection`
- **dataset**: `DataSet`
- **row**: `dataRow`

```java
+ DataBridge(): Void
+ doesFieldExist(tableName: String, sql: String, fieldName: String): Boolean
+ executeSelect(sql: String, tableName: String): DataTable
+ executeSQL(sql: String, tableName: String): DataTable
+ fetchTable(adapter: sqlAdapter, tableName: String, sql: String): DataTable
+ find(tableName: String, id: Integer): DataRow
+ getRow(position: Integer, table: DataTable): DataRow
+ getTable(tableName: String): DataTable
+ initDataSet(): Void
+ initDataTable(dt_table: String): DataTable
+ insert(sql: String): Void
```

**Figure 5.21**: DataBridge Class
DynamicControlGeneration
- chBox: CheckBox
- dataSource: DataBridge
+ generateCheckBox(sld: String, tableName: String, id: String, phHolder: Placeholder): CheckBoxList
+ generateDropDownList(sld: String, tableName: String, field1: String, field2: String, dDListId: String, phHolder: Placeholder): DropDownList
+ generateDropDownList(name: String, dDListId: String, phHolder: Placeholder): DropDownList
+ generateDropDownList(sld: String, tableName: String, field1: String, field2: String, field3: String, field4: String, dDListId: String, phHolder: Placeholder): DropDownList
+ generateDropDownList(sld: String, tableName: String, field1: String, field2: String, field3: String, dDListId: String, phHolder: Placeholder): DropDownList
+ generateListBox(sld: String, tableName: String, field1: String, field2: String, dDListId: String, phHolder: Placeholder): ListBox
+ generateTextBoxphHolder: Placeholder, amount: Integer, id: String):

Figure 5.22: DynamicControlGeneration Class

5.4.2 Entity Relationship Diagram

Figure 5.23 show the relationship of tables in the database.

Figure 5.23: Entity Relationship Diagram
CHAPTER 6. EVALUATION

To assess whether our concept – SOUTS, can be applied to solve the University course timetabling problem, experiments were carried out. The aim of the experiments was to see how it compared with other systems with regards to time taken and quality of timetable produced. This chapter outlines the details of the experiment and hardware environment used in the experiment as well presenting the results obtained from the experiment.

6.1 Method

6.1.1 Apparatus

Tests were done using a Dell Dimension 3100 Machine with an Intel Pentium 4, 2.8 GHz Processor and 1GB DDR2 RAM. The Operating System used was Windows XP SP3 Professional Edition.

6.1.2 Experiment Benchmarks

The experiments for the course timetabling problem discussed in this paper were done using the benchmark (see Table 6.1) for course timetabling problems proposed by the Metaheuristics Network that needs to schedule 100-400 courses into a timetable with 45 timeslots corresponding to 5 days of 9 hours each, whilst satisfying a number of constraints (Abdullah, Shaker, McCollum, & McMullan, 2010).

<table>
<thead>
<tr>
<th>Category</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Courses</td>
<td>100</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>Number of Rooms</td>
<td>5</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Number of Students</td>
<td>80</td>
<td>200</td>
<td>400</td>
</tr>
</tbody>
</table>

Table 6.1: Categorization of Data into Small, Medium & Large Datasets
6.2 Caveats

A few problems arose during the experiment which may affect the results. Firstly, the hardware specifications of the system used might not be comparable to the system that was used to test the other algorithms. Secondly, our approach used a different set of constraints than the existing approaches, which can lead to inaccurate results.

6.3 Results

The intention of this research was to explore whether self-organization principles of biological systems can be applied to solve the University Course Timetabling Problem. This section presents the results of the experiment that was done.

6.3.1 Time taken to produce Timetable

The time taken to generate a timetable based on three differently sized datasets (see Table 6.1) is presented in this section.

Table 6.2: Results of Experiment

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Our Method</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
<th>M8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>0.6</td>
<td>0.8</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Medium</td>
<td>16.6</td>
<td>163</td>
<td>197</td>
<td>161</td>
<td>419</td>
<td>313</td>
<td>147</td>
<td>105</td>
<td>130</td>
</tr>
<tr>
<td>Large</td>
<td>54.6</td>
<td>738.6</td>
<td>912</td>
<td>-</td>
<td>1068</td>
<td>-</td>
<td>529</td>
<td>730</td>
<td>876</td>
</tr>
</tbody>
</table>

M1: Great Deluge and Tabu Search by Abdullah et al.
M2: Genetic algorithm and local search by Abdullah and Turabieh (2008)
M3: Randomized iterative improvement algorithm by Abdullah et al. (2007a)
M4: Graph hyper heuristic by Burke et al. (2007).
M5: Variable neighborhood search with tabu by Abdullah et al. (2007b)
M6: Hybrid evolutionary approach by Abdullah et al. (2007c)
M7: Extended great deluge by McMullan (2007).
M8: Nonlinear great deluge by Linda-Silva and Obit (2008)
CHAPTER 7. DISCUSSION

The results obtained from the experiment indicates that SOUTS produced timetabling solutions in considerably less times than other approaches for the medium and large datasets, whilst at the same time satisfying all of the hard constraints specified. In order to determine the significance of the results obtained, the Anova Test was used as the test of significance for the data gathered from SOUTS and other approaches. The ANOVA Two Factor Replication Test was used to generate results with a set significance level of 1% (p<0.01).

7.1 Time taken to generate Timetable Using a Small dataset

The time taken by SOUTS to produce a feasible timetabling solution using the small dataset was similar to some of the existing approaches and longer than the others (see Figure 7.1). The reason for this can be attributed to the fact that the timetabling problem is NP hard, which means that the time taken to produce a feasible solution increases exponentially as the size of the dataset increases. Another reason can be attributed to the limited number of courses and room catered for by the small dataset.

![Figure 7.1: Results after using a small dataset](image-url)
7.2 Time taken to generate Timetable Using a Medium dataset

The times taken by SOUTS to produce feasible timetabling solutions using the medium (see Figure 7.2) and large datasets (see Figure 7.3) were significantly lower than the existing approaches.

![Figure 7.2: Results after using a medium dataset](image)

7.3 Time taken to generate Timetable Using a Large dataset

![Figure 7.3: Results after using a large dataset](image)

7.4 Details of Anova: Two-Factor Test

Results from the Anova: Two Factor without Replication test indicates that the average time taken by SOUTS to generate timetables using small, medium and large datasets is at least
100 percent less than any of the other approaches (see table 7.1). This figure shows that SOUTS is more efficient than the others in terms of time taken to produce a feasible timetable. This efficiency was statistically confirmed and proven following an F Test.

**Table 7.1**: Details of Anova: Two-Factor without Replication

<table>
<thead>
<tr>
<th>SUMMARY</th>
<th>Count</th>
<th>Sum</th>
<th>Average</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOUTS</td>
<td>3</td>
<td>71.8</td>
<td>23.93333</td>
<td>769.3333</td>
</tr>
<tr>
<td>M1</td>
<td>3</td>
<td>902.4</td>
<td>300.8</td>
<td>150328.8</td>
</tr>
<tr>
<td>M2</td>
<td>3</td>
<td>1109</td>
<td>369.6667</td>
<td>230296.3</td>
</tr>
<tr>
<td>M3</td>
<td>3</td>
<td>161</td>
<td>53.66667</td>
<td>8640.333</td>
</tr>
<tr>
<td>M4</td>
<td>3</td>
<td>1491</td>
<td>497</td>
<td>287587</td>
</tr>
<tr>
<td>M5</td>
<td>3</td>
<td>313</td>
<td>104.3333</td>
<td>32656.33</td>
</tr>
<tr>
<td>M6</td>
<td>3</td>
<td>676</td>
<td>225.3333</td>
<td>74562.33</td>
</tr>
<tr>
<td>M7</td>
<td>3</td>
<td>835</td>
<td>278.3333</td>
<td>155758.3</td>
</tr>
<tr>
<td>M8</td>
<td>3</td>
<td>716</td>
<td>238.6667</td>
<td>94705.33</td>
</tr>
</tbody>
</table>

Based on the f-test analysis carried out on the time taken to generate feasible timetabling solutions, the results indicated a notable level of significance (p=0.00085483) (see Table 7.2). This result indicates that SOUTS is able to produce feasible timetables faster than existing approaches.

**Table 7.2**: Details of Anova: Two-Factor without Replication showing P Value

<table>
<thead>
<tr>
<th>Source of Variation</th>
<th>SS</th>
<th>df</th>
<th>MS</th>
<th>F</th>
<th>P-value</th>
<th>F crit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Taken in Minutes</td>
<td>1214393.105</td>
<td>2</td>
<td>607196.5526</td>
<td>11.34661516</td>
<td>0.00085483</td>
<td>6.226235</td>
</tr>
</tbody>
</table>
CHAPTER 8. CONCLUSIONS

University timetabling is a NP-hard problem, which means that the amount of computation required increases exponentially with problem size. The timetabling process has to be done in such a way that hard constraints must be satisfied in order to produce feasible timetables and soft constraints are desirable, but not absolutely essential. This thesis has described a Self-Organizing University Timetabling System (SOUTS) developed for university course timetabling. The SOUTS program is based on a concept that is patterned after self-organizing principles of biological systems and more specifically, mate selection. The algorithm prohibits violation of hard constraints and caters as much as possible for soft constraints, which can be violated. The SOUTS program developed allows lecturers to specify times that they are not available to lecture. The algorithm was tested using guidelines specified by the Metaheuristics Network (Abdullah, Shaker, McCollum, & McMullan, 2010). The experimental results obtained from SOUTS showed it satisfied all hard constraints for small, medium and large problems. SOUTS was also compared with existing approaches to determine its efficiency and applicability in solving the timetabling problem.

SOUTS solved the university course timetabling problem with more than twice the efficiency of existing approaches as confirmed by F test results with a significance factor of approximately 99.999. The results proved that SOUTS is not only applicable in solving the university timetabling problem, but also showed that it out performed existing approaches whilst maintaining feasible timetabling solutions.

SOUTS utilized principles from natural systems (biomimetics) and was able to efficiently solve a problem which, to date, has never been adequately solved although it has been subjected to decades of research. This proves than by patterning our approaches after self organizing,
natural systems, we may be able to effectively solve many such complex real world problems that have thus far evaded us in our quest for solutions.

8.1 Contributions

Pioneering contributions produced from this research are as follows:

- The development of a new approach that closely resembles the nature of the timetabling problem.
- The ability to produce high quality timetabling solutions, whilst satisfying all hard constraints and as much soft constraints as possible.
- The ability to work around constraints hence being able to solve and thus produce feasible timetabling solutions.
- The ability to generate feasible timetabling solutions in just a fraction of the time taken by existing approaches to produce timetabling solutions, given that they (existing solutions) have solved the timetabling problem.

8.2 Limitations of Implemented Solution

SOUTS has a few limitations that will have to be addressed in the future. If there are not enough rooms for all the courses offered by the University, then some courses will not be scheduled. Also, it does not cater for the dynamic updating and addition of constraints, both hard and soft.

8.3 Future Work

Future work in this research comprises further refining the current system to accommodate constraints entered into the system in human language. Furthermore, the user
interface needs to be enhanced to allow for the data entry process to be less tedious. Also, a method has to be devised to allow for the calculation of the proximity of one faculty or building to another. Finally, a more detailed future study is planned to assess the effectiveness of the refined system.
REFERENCES


