

California State University, San Bernardino

CSUSB ScholarWorks

Theses Digitization Project

John M. Pfau Library

2010

Basic online scheduling system optimizer: A study in genetic algorithms [sic]

Norman Lee Langhorne

Follow this and additional works at: <https://scholarworks.lib.csusb.edu/etd-project>



Part of the [Theory and Algorithms Commons](#)

Recommended Citation

Langhorne, Norman Lee, "Basic online scheduling system optimizer: A study in genetic algorithms [sic]" (2010). *Theses Digitization Project*. 3850.

<https://scholarworks.lib.csusb.edu/etd-project/3850>

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

BASIC ONLINE SCHEDULING SYSTEM OPTIMIZER
A STUDY IN GENETIC ALOGRITHMS

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Norman Lee Langhorne, II

June 2010

BASIC ONLINE SCHEDULING SYSTEM OPTIMIZER

A STUDY IN GENETIC ALOGRITHMS

A Project

Presented to the

Faculty of

California State University,

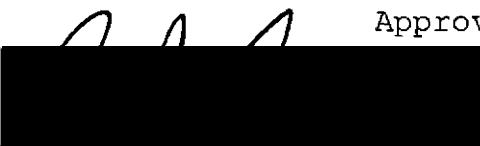
San Bernardino

by


Norman Lee Langhorne, II


June 2010

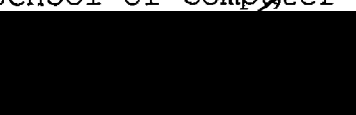
Approved by:


Dr. Arturo I. Concepcion, Chair,
School of Computer Science and Engineering

08 Jun 2010
Date


Dr. Anthony Metcalf,
Biology


Dr. Richard J. Botting,
School of Computer Science and Engineering


Dr. Ernesto Gomez,
School of Computer Science and Engineering

ABSTRACT

The scheduling of academic courses represents one of several difficult administrative tasks undertaken by departments in colleges and universities each academic session. This is primarily due to the many constraints imposed by the courses being offered, along with the restrictions imposed by faculty members, students and facility resources, each of which contributing to the overall complexity of the scheduling task. This paper addresses this NP-Hard problem domain by examining the use of Genetic Algorithms (GA's) in an attempt to create viable academic schedules. By employing the heuristic methods of GA's, we demonstrated how optimal solutions sets may be rendered through evolution. In addition, through experimentation we illustrated how the amount of effort required for solving such problems, increases as the number of constraints increase thus, increasing the overall problem complexity. Finally, we addressed the issue of locating the optimal solution set and how this task is beyond the scope of this examination and reserved for future research and exploration.

ACKNOWLEDGEMENTS

As I approach the end of my trek in seeking the Masters of Computer Science, I first want to sincerely thank the faculty and staff of the School of Computer Science and Engineering. Reaching this milestone would have been impossible without the help, guidance and commitment that they have shown me during the endeavor.

I especially want to thank and offer my sincerest gratitude to my committee chair, Dr. Arturo Concepcion, who has been the guiding light throughout this journey. I am most grateful for the encouragement he bestowed on me and his dedication to seeing me through this project. He has both challenged and enlightened me along the way and I am honored to have been one of his students.

I want to truly thank Dr. Anthony Metcalf from the Department of Biology for his guidance and advisement regarding my research. His involvement helped steer this project and validate my study and is most appreciated.

For all my years of study, I sincerely want thank Dr. Richard Booting. Dr. Botting was the first contact I had with the School of Computer Science and Engineering, and has been an inspiration to me throughout these many years.

I am so, very honored to have had his participation during this milestone quest.

I would also like to thank Dr. Ernesto Gomez for serving as committee member. I especially want to thank him for the patience and understanding extended by him during a very difficult and trying time. His thoughtfulness helped calm the waters, and will always be remembered.

In addition, I want to thank Monica Latimer for her time, effort and assistance towards keeping me on track. I am most appreciative for all that she has done on my behalf in helping me reach this goal.

Finally, but in no way last, I humbly thank Dr. Josephine Mendoza for all her all efforts as my academic advisor and in seeing me through this Master's program. I wish to thank her for all her patience and untiring support in helping me through this challenging process. Truly, it is due to her boundless commitment that I arrive at this point. I will always be in her debt.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS	iv
LIST OF FIGURES	xi
CHAPTER ONE: INTRODUCTION	
Background	1
Genetic Algorithms	3
Expected Results	5
CHAPTER TWO: SOFTWARE REQUIREMENTS SPECIFICATION	
Purpose	7
Scope	8
Overview	12
Product Perspective	12
System Interfaces	13
User Interfaces	16
Hardware Interfaces	17
Software Interfaces	17
Communication Interfaces	18
Memory Constraints	18
Operations	18
Site Adaption Requirements	18
Product Functions	19
Overview	19

Constraints	19
User Characteristics	21
Assumptions and Dependencies	22
Apportioning of Requirements	22
Specific Requirements	22
External Interface	22
Functions	22
Performance Requirements	22
Logical Database Requirements	22
Design Constraints	23
Standards Compliance	23
Software System Attributes	23
Reliability	23
Availability	23
Security	23
Maintainability	23
Portability	23
Organizing the Specific Requirements	23
System Mode	23
User Class	24
Feature	24
Stimulus	24

CHAPTER THREE: GENETIC ALGORITHM

Overview	25
Basic Genetic Algorithm	26
Basic Online Scheduling System	
Optimizer Algorithm	28
Initialization	28
First Generation Schedules	31
Scoring and Repair	31
Next Generation Schedules	32
Multi-Point Crossover	33
Uniform Crossover	35
Point Mutation	36
Solution Determinants	38
Summary	38
CHAPTER FOUR: SOFTWARE DESIGN	
System Implementation	40
Development Tools	40
Layered Architecture	40
Patterns	41
High-Level System Software Packages	42
Bossopt	42
BusinessModel	42
BMItem	42
BMPersistentSet	43

BMPersistentItem	44
BMDatabaseHelper	45
ResourceFactory	46
ResourceManager	46
CourseResourceMgr	47
FacultyResourceMgr	47
RoomResourceMgr	48
PreferenceMap	48
BMResourceItem	50
ScheduleFactory	51
ChromosomeItem	51
GeneItem	52
AcademicGene	52
AvailabilityMap	52
DatabaseModel	52
DBRecordSet	53
DBConnectionMgr	54
SystemModel	54
Data Item Subsystem	54
Database Access Subsystem	56
Resource Factory Subsystem	57
Schedule Factory Subsystem	58

CHAPTER FIVE: TESTING AND RESULTS

Introduction	60
Test Data Overview	60
Schedule Tests and Findings	64
Manual Case Test 1	64
Manual Case Test 2	65
Basic Online Scheduling System Case Test	67
Basic Online Scheduling System Optimizer Test	69
Findings	72
CHAPTER SIX: SYSTEM MAINTENANCE	
Introduction	74
Contents	76
Server Setup	76
Installation Instructions	78
Setup Basic Online Scheduling System Software	79
Getting Started	79
Basic Online Scheduling System Optimizer Installation	81
Basic Online Scheduling System Optimizer Project Build	82
CHAPTER SEVEN: CONCLUSION AND FUTURE DIRECTIONS	
Conclusion	83
Future Directions	84
REFERENCES	86

LIST OF FIGURES

Figure 2.1. List of Acronyms	11
Figure 2.2. Deployment Diagram	15
Figure 2.3. User Interface - Part 1	16
Figure 2.4. User Interface - Part 2	17
Figure 2.5. Use Case for Optimizer	19
Figure 3.1. Basic Genetic Algorithm	28
Figure 3.2. Single-Point Crossover	34
Figure 3.3. Multi-Point Crossover	35
Figure 3.4. Uniform Crossover	36
Figure 3.5. Point Mutation	37
Figure 4.1. Package Hierarchy	41
Figure 4.2. Data Item Subsystem	55
Figure 4.3. Database Subsystem	56
Figure 4.4. Resource Factory	58
Figure 4.5. Schedule Factory	59
Figure 5.1. Course Offerings	61
Figure 5.2. Courses Requiring Labs	62
Figure 5.3. Course Preferences	62
Figure 5.4. Non-Preferred Courses	63
Figure 5.5. Hour Preferences	63
Figure 5.6. Test Schedule 1	65
Figure 5.7. Test Schedule 2	67

Figure 5.8. Basic Online Scheduling System Schedule	69
Figure 5.9. Optimized Schedule	71

CHAPTER ONE

INTRODUCTION

Background

Started initially as an academic exercise in software development by the School of Computer Science and Engineering at California State University, San Bernardino, the Basic Online Scheduling System or (BOSS), has for the last five years served as a laboratory in the study of timetabling or schedule generation and in the development of algorithms and practical implementations used to solve this class of problems. As part of the study in software engineering, the BOSS project specifically addressed the timetabling problem by implementing systems that produced quarterly schedules for the School of computer Science & Engineering. Through its many incarnations, BOSS has approached the timetabling problem from many different angles with each iteration and implementation aiding in the advancement and overall maturity of the solution.

As is the case with most timetabling problems, studies have found these problems to be NP-Hard in time complexity, thus requiring very complex algorithms be employed to solve

them and that the solutions presented are often both varied and numerous. Such conditions are the result of problem spaces that contain a multitude of constraints or prerequisite conditions that must be met in order to achieve a valid solution set. In the case of quarterly scheduling, such constraints consist of instructor preferences which relate to which courses they want to teach. Also, the lecture rooms and labs required for the lecture session. In addition, the number of courses that each instructor is allowed to lecture daily must be addressed.

However, the most important characteristic of problems within this class of problem domains is the fact that more than one solution may be found to exist. This fact further complicates the overall problem in that it must be determined which of the candidate solutions is most optimum to the problem space. It is this nature that the many implementations of the BOSS project have had to contend with and overcome.

In an attempt to develop solutions for the timetabling problem, the most recent BOSS implementations have consisted of algorithms that render a single solution. In most cases, these single entity solutions represented a

"Good Fit" but, not necessarily the "Best Fit" solution. In addition, as newer constraints have been added to the problem space, ever more complex processing routines have been required to arrive at an acceptable but, not necessarily "Optimal" solution.

As the operational specifications of the BOSS have changed, so too have the implementation complexities. Bearing this in mind, the goal of the next system incarnation was to find an algorithm that would address the current constraints of the problem space and able to adapt to both changing and increasing limitations. This new algorithm would have to be scalable to the problem domain and allow for multiple tests criteria to be logically defined and processed. Also, the proposed algorithm should allow for multiple solution candidates to be generated and evaluated for correctness and overall solution worthiness with the most optimal candidate(s) presented to the solution space.

Genetic Algorithms

After researching various publicized optimizing algorithms related to timetabling problems, a class of algorithms was discovered that closely addressed the needs

and requirements of the future iteration of the BOSS. This class of solution providers is known as "Genetic Algorithms". Inherent within this class of algorithm are some noteworthy characteristics. First, Genetic Algorithms tend to thrive in environments in which several possible solutions are expected to exist. Secondly, their logical implementation is relatively simple and adaptable to most problem spaces. Finally, this class of algorithms relies on the use of candidate solution evaluation and generational filtering to eliminate inadequate solutions from the solution domain thereby propagating more worthy solutions to an optimal or "Best Fit" solution set.

Inspired both by nature and naturalist British Charles Darwin, Genetic Algorithms (GAs) make use of abstract genetic structures called chromosomes and genes to logically represent problem space. In addition, with the use of genetic functions such as selection, crossover and mutation, populations of solution candidates are created and evaluated for fitness and overall worthiness. Through multiple iterations of this mechanism, candidate solutions possessing the desired traits are created and allowed to pass from one generation into the next while, inadequate or less-desirable solutions ultimately are removed from future

populations. It is these behavioral traits that make GAs the appropriate tool for the BOSS optimization.

As such, the purpose of this study is to employ GA's in the next BOSS iteration. During the implementation process, metrics will be obtained and compiled for comparison with previous BOSS versions. It is hoped that by utilizing GA's that the overall scheduling efficiency and system flexibility will be enhanced and thereby further promoting the value and effectiveness of BOSS. [B2][B3]

Expected Results

As discussed in the aforementioned sections of this document, the main characteristic of the Genetic Algorithm (GA) is that it is an algorithm that renders solutions through the principle of evolution. Therefore, the main expectation will be that this type of processing will require an increase in both processing time and processing power to achieve desired results.

However, the trade-off for this increased processing effort should be a solution set that better reflects the desired user preferences and closely adheres to both the hard and soft constraints imposed upon the system. In addition, this new GA-based implementation should offer the

needed flexibility which will allow for future enhancements of the BOSS Optimizer.

Although, the intent of this study to find an implementation that will yield effective timetabling solutions, we do not guarantee the optimality of the presented solutions. The study of optimality as it relates to this study shall be left to future studies and endeavors.

CHAPTER TWO

SOFTWARE REQUIREMENTS SPECIFICATION

Purpose

This project is to provide the School of Computer Science at California State University, San Bernardino with an optimizing schedule module to enhance the latest version of the Basic Online Scheduling System. This product will be referred to as BOSS-Opt. The goal of this optimizer is to produce course schedules based upon historical course offerings, instructor preferences and lecture room availability.

Since BOSS-Opt is an enhancement to the core scheduling functionality of BOSS, the user of the system will have the option of either enabling or disabling the BOSS-Opt module during the generating of academic schedules. All other functions defined within the BOSS Software Requirements Specification (SRS), will not be affected by this modification.

Scope

The main goal of schedule optimizer is to provide several possible schedules for an academic period based upon proposed course offerings. Also, the optimizer will examine historical course data to determine the daytime or nighttime placement of courses within the proposed schedule. Finally, factors such instructor preferences and lecture room availability will be used to further drive the schedule creation process. During this process, it will be the role of BOSS-Opt to alleviate schedule conflicts, if possible, and report all unresolved scheduling conflicts to the user.

At the end of the schedule run, BOSS-Opt will internally produce a schedule offering in the SIS+ program format, which can be used by California State University staff to input the classes into the CSUSB schedule. Also, the schedule which is deemed as the final schedule for the Computer Science department will be used by BOSS-Opt during the creation of future schedules.

Apache	Open source web server that services data and applications within a Web environment.
BOSS	Basic Online Scheduling System
BOSS -OPT	Basic Online Scheduling System Optimizer
CSCI	Computer Science
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IE	Internet Explorer
Java	Objected oriented language developed by Sun Microsystems
JavaScript	<p>A scripting language developed by Netscape to enable Web authors to design interactive sites. Although it shares many of the features and structures of the full Java language, it was developed independently.</p> <p>JavaScript can interact with HTML source code, enabling Web authors to spice up their sites with dynamic</p>

	content. JavaScript is endorsed by a number of software companies and is an open language that anyone can use without purchasing a license.
Java Servlet	A java application that runs in a Web server environment providing server-sided processing and data access.
JDBC	Java Database Connectivity is and API that provides Java applications with access to various SQL data stores.
Linux	A multi user UNIX operating system.
MySQL	Open source relational database management system designed for speed, flexibility and ease of use in contemporary data centric systems.
OO	Object Oriented.
PHP	PHP Hypertext Protocol
Relational Database	A database system composed of two or more tables of data represented in column form that is inter-related by key attributes.
SRS	Software Requirements Specification.

TCP/IP	<p>Abbreviation of Transmission Control Protocol, and pronounced as separate letters. TCP is one of the main protocols in TCP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.</p> <p>IP is the internet protocol that defines the rules and transmission methods employed in the transfer of data across the network infrastructure.</p>
--------	---

Figure 2.1. List of Acronyms

Overview

The remainder of this chapter will have two major sections along with appendixes and will be ended with an index for the SRS documents [B1]. The second section will contain all the product outlines, perspective, function, interface, and user characteristics. The third section will include the specific requirements (e.g. hardware requirements).

Product Perspective

This project will provide a table of classes scheduled for a specific quarter in a specific year, for the computer science department.

The director of the School of Computer Science & Engineering will provide the BOSS scheduler with the classes for each quarter. In addition, historical schedule data will be presented to the scheduler system. The scheduler will generate a suggested schedule of classes for each quarter depending on:

- Historical daytime/nighttime offerings
- The classes that are offered in that quarter
- The classes that the teachers would like to teach.

- The time and location available for the classes.

The director of the school of computer science will be able to edit / change class locations day or time. The BOSS/BOSS-Opt scheduler then will notify the chair if there are any schedule conflicts.

System Interfaces

The BOSS Optimizer is composed of a multi-tiered, client-server architecture that utilizes a Web browser running either Microsoft Internet Explorer or Mozilla Firefox as the client front-end. The server or back-end component will consists of a Linux server running Apache Tomcat, PHP and MySQL relational database services. The objective of the server component is to act as the main repository of system data and logical functions. See Figure 2.2.

The client and server components communicate via HTTP over TCP/IP connections with user/client requests being relayed to the server for processing. Results in the form of HTML data will be returned to the client component to be decoded and ultimately displayed by the client. A typical session will comprise the following steps:

1. A user initiates the BOSS session via the client front-end.
2. During the creation of an academic schedule, the user requests that an optimized schedule be created.
3. An HTTP requests message is forwarded via TCP/IP to the Apache server.
4. The server invokes a Java process via Java Servlet controls.
5. The Java application in turn communicates via JDBC with MySQL data stores.
6. Schedules are created and persisted within the MySQL database.
7. A return is initiated by the server to the client using HTTP data to populate the user interface with the necessary graphic controls to aid the user in accessing the newly created schedules.

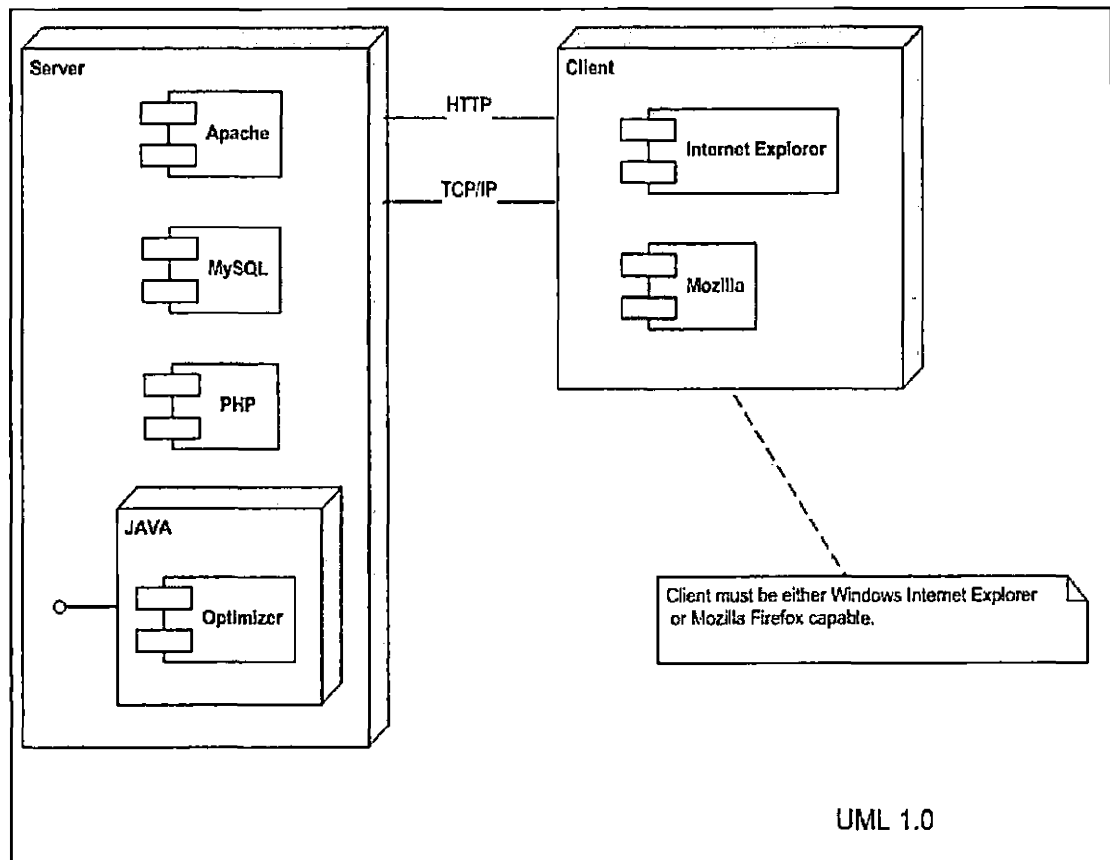


Figure 2.2. Deployment Diagram

User Interfaces

Initialization of the BOSS Optimizer will be controlled by an Optimize Schedule control contained within the current schedule view screen.

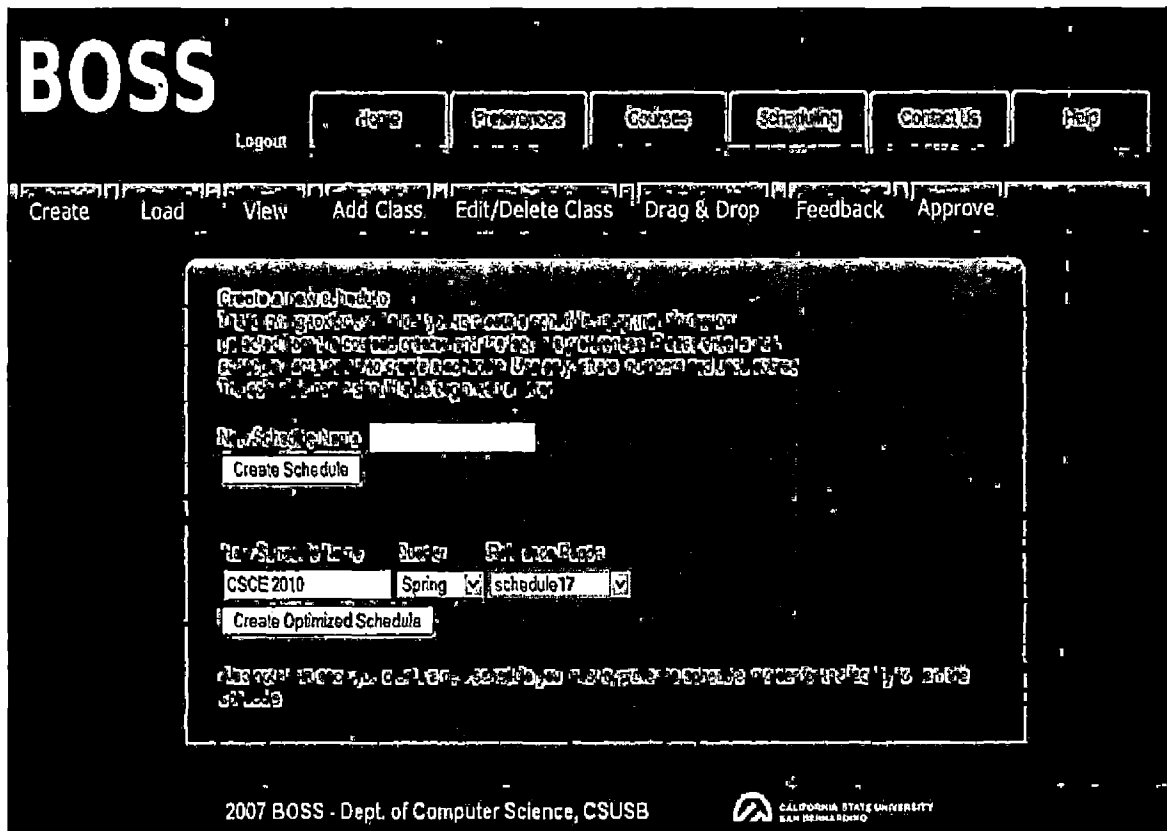


Figure 2.3. User Interface - Part 1



Figure 2.4. User Interface - Part 2

Hardware Interfaces

No hardware interface.

Software Interfaces

The BOSS-Opt will provide a Java interface that will allow the server component to interact with the optimizer. The BOSS-Opt will be used to pass database tables and optimizing options from the BOSS server to the BOSS-Opt.

Communication Interfaces

Microsoft Internet Explorer and Mozilla Firefox.

Memory Constraints

128 MB RAM min.

Operations

The scheduler will operate 24/7. Backups can be done of the database using the mySQL dump of the database once a month. Maintenance will be done on call, and mostly remote.

Site Adaption Requirements

The schedule will run from the Computer Science department Website.

Product Functions

Overview. The BOSS Optimizer will be implemented using a Genetic Algorithm (GA). It is hoped that by using this form of optimization, that a global optimum may be obtained within a problem scope that offers many solutions. Also, in using GA, solution evolution will allow for the survival of the more optimum solutions while inferior solutions will be rendered extinct from the solution space. See figure 2.5.

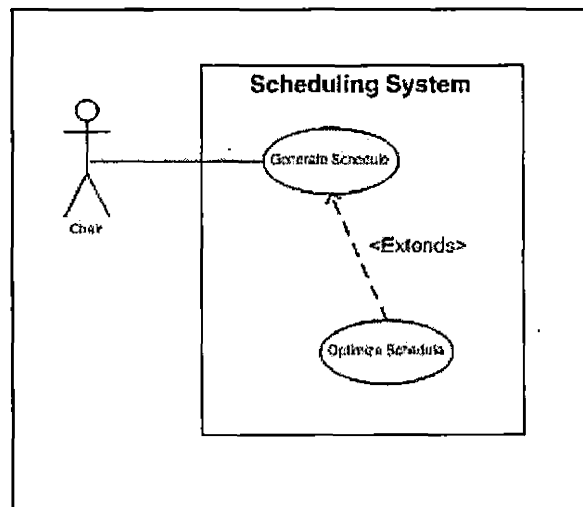


Figure 2.5. Use Case for Optimizer

Constraints. The BOSS Optimizer will adhere to two types of solution constraints. The first constraint, hard

constraints are rules that if violated will lead to invalid solutions. Soft constraints on the other hand are rules that if violated will lead to less than optimum solutions. Soft constraint violations will be permitted but will result in the solution fitness score being penalized.

Hard constraints

- Only one course can be held in a lecture room during any given time period.
- A professor/instructor can lecture only one course during any time period.
- Each course must alternate between morning and evening offerings based upon time last offered.
- Each assigned lecture room must accommodate all students enrolled in the course for which the room was assigned.
- Core related courses must be made available during both morning and evening periods.

Soft Constraints

- Professor/instructors must be assigned to courses of their first choice when possible.
- Professor/instructors must be assigned to lectures periods first choice when possible.
- No professor/instructor may lecture on more than two courses per academic calendar period.
- Each professor/instructor must be exempt from lecturing at least one academic period per calendar year.
- No professor/instructor shall lecture on two consecutive courses.

User Characteristics

The projected users of the BOSS-Opt will include CSCI faculty and staff, including the CSCI chair. These individuals should be familiar with computer operations in general and should not need additional basic instructions to use this product. All instructions and/or guidance can be obtained using the BOSS-Opt tool's help menu.

Assumptions and Dependencies

There were none.

Apportioning of Requirements

There were none.

Specific Requirements

External Interface. Schedule Creation webpage, see Figure 2.3.

Functions. Upon completion of the optimization process the user will be presented with a screen containing three schedule buttons. These buttons correspond to the optimized schedules that were created during optimization. By selecting either of these button, the corresponding schedule will be displayed in the Manage Schedule screen.

Performance Requirements. Code will be optimized for both functionality and speed of operation. Response time for the scheduling process should not exceed 35 wall time minutes.

Logical Database Requirements. MySQL will be the relational database manager; port 3306 must be open on the server to allow access for users.

Design Constraints

Standards Compliance. The BOSS-Opt module will comply with Java coding standards.

Software System Attributes

Reliability. The servers running the BOSS-OPT module will be completely functional.

Availability. The BOSS-OPT module will require a computer with 128 megabytes of RAM. This system should have a continuous operational uptime with system outages not to exceed more than 4 hours within a 30 day time period.

Security. The BOSS-OPT module will not have any special security features.

Maintainability. The code will be well documented and most modules will be reusable.

Portability. The code used in this program will be completely portable, allowing the code to go from one server to another as long they who comply with the hardware and software requirements detailed in this document.

Organizing the Specific Requirements

System Mode. Single mode of operation (normal mode).

User Class. There are three users of the CSCI

Scheduler tool: chair, faculty, and staff.

Feature. This software contains no external features.

Stimulus. This software supports no external stimuli.

CHAPTER THREE

GENETIC ALGORITHM

Overview

A Genetic Algorithm (GA) is a type of search technique used in computer science to locate solutions for problems that are by their nature NP-Hard or nondeterministic polynomial time ordered. Research has shown that NP-Hard problems have no known solution other than searching through each and every possible candidate solution in order to determine the optimal or best fit solution.

Genetic algorithms are implemented by abstracting individual solutions called chromosomes which in turn belong to a population or genome of candidate solutions or phenotypes. However, the most significant characteristic of GA's is that their solutions evolve. This evolution is achieved through the simulation of biological principles of natural genetics selection and evaluation methods. Such techniques over time tend to promote the survival of the "Fittest" by allowing solutions, having the desired traits, to be propagated from one generation to the next thereby, creating a solution set of the most optimal candidate solutions.

Basic Genetic Algorithm

For having such a sophisticated effect of finding optimal solutions belonging to NP-Hard problem spaces, the Genetic Algorithm is comprised of relatively simple processing steps. The basic steps include the following: initialization, evaluation, followed by replication and re-evaluation until the desired termination state is reached.

During the initialization phase, the GA produces a random set of chromosomes or solution individuals. Next, these candidates are evaluated and scored based upon how well they fit the desired solution.

Within the replication phase, chromosomes created during the initialization phase will next act as parents in the algorithm. During this phase, two parents will be probabilistically selected based upon their fitness score obtained during the previous phase. These two parents will then exchange genetic material or genes to create new individuals for the next solution generation. The newly created candidate solutions will then be scored for fitness as in the initialization phase.

Finally, at the end of each generation build phase, the system will evaluate the candidate solutions belonging to the new population for overall fitness. During this

step, an optimal solution is determined to exist or processing has proceeded to pre-defined limit of generations, the process will be terminated with the highest evaluated candidate being presented as the solution. Otherwise, the replication phase will be again be repeated thus creating the next, new solution population which will be scored and evaluated for possible solutions.

[B2] [B3] [B4]

BEGIN PROCESS

Generate a random population of n viable chromosomes (schedules).

Score and assign a fitness value to each chromosome in the current population.

DO UNTIL (*maximum number of generations' created or minimum scoring variance achieved*)

Create the next generation based upon genetic material of the current generation using various genetic functions such as selection, crossover, mutation and elitism.

Score and assign a fitness value to each chromosome in the current population.

Remove the parent generation and replace it with the new child generation.

END DO

Save the highest ranked 3 chromosomes from the final generation into the schedule database.

Display highest ranked schedule upon user interface screen.

END PROCES

Figure 3.1. Basic Genetic Algorithm

Basic Online Scheduling System Optimizer Algorithm

Initialization

In order to start the BOSS Optimizer (BOSS-Opt) process, a system call is made to the optimizer process specifying the schedule name, academic period and department values. These parameter values will be used for

queries against the database to obtain all pertinent data items.

Upon process activation, the optimizer will begin by first creating the data objects for both the task and resource elements. In order to establish proof of concept, the first spiral implementation will have hardcoded query parameters passed to data stores to retrieve related data items. However, later implementations will be parameter based and set by user definitions.

For this first implementation, task elements will correspond to the courses that are to be scheduled. Every course item will be represented by a data object comprised of a unique id, description, week period, day period and period length properties. In addition, each task object will contain a two-dimensional array. This array will have an x-axis representing the days of the week and a y-axis representing the hours of the day.

During construction of task element objects, each array element will be initialized with one of four values. For those array elements that correspond to periods that the course item is available to be scheduled, a value of '1' will be stored in said element. All remaining array

elements will be set to '4' to indicate invalid or forbidden period elements.

Next, the optimizer will create data objects for the resource elements. These resource elements will be used to define scheduling priorities and preferences to the system. During this proof of concept phase, resource element objects will be represented as instructor and classroom data objects and as stated earlier, the parameters required to access data stores that will be used in their creation, will be temporarily hardcoded. Like the task elements, resource elements will be comprised of unique id and description properties. Resource elements will also contain preference value property fields that will be used to help determine preference relationships between task and resource elements. In addition, each object will contain a two-dimensional array. This array will have an x-axis representing the days of the week and a y-axis representing the hours of the day.

As with the task elements, resource element objects will be initialized with one of four values. For those array elements that correspond to periods of first preference in scheduling will be set to a value of '1'. Elements determined to be optional or secondary in schedule

preference will be set to a value of '2'. Elements determined to be low in schedule preference or "Don't care" will be set to a value of '3'. All remaining array elements will be set to '4' to indicate invalid or forbidden period elements.

First Generation Schedules

During this phase of optimizer processing, the first generation of schedule objects will be created. The number of schedules created in this process will be parameter based but, should vary from 50 to 100 for this first phase of development. Schedule objects will be created by randomly pairing task and resource elements to obtain task solutions. In this process courses, instructors and rooms will be married by time slot and stored as genes within the chromosome or schedule object. In order to minimize the number of invalid schedule assignments, logic will be employed to aid in the prevention of hard constraint violations.

Scoring and Repair

After the initial generation of schedule objects has been created, the system will examine the population to locate errors such as overlapping room assignments or instructors assigned to more than one course having

conflicting times. During this process, corrections will be effected, if possible, to correct the invalid conditions. If error corrections cannot be made to the schedule, then the schedule will be allowed to go forward and will eventually be dropped from future generations due to extremely poor scoring.

After the repair process has been applied, each schedule will be scored to determine its solution fitness. It is during this phase that soft constraint violations will be determined based upon information contained within the resource element objects. Constraint violations will be tallied and constraint violation weights will be applied to determine a score reflective of how well the schedule fits the required system criterion. As is the purpose of genetic algorithms, schedules having lower fitness ratings will have less of chance of propagating their genes into the next generation than those schedules having higher fitness scores.

Next Generation Schedules

In order to create randomness within schedule populations, genetic operators will be applied during the generation of successive schedule generations. These operators will be applied to parent schedules of the most

recently generated population to create the *child* schedules of the next generation. This process of manipulation and replication will be continued for a specified number of generations as specified by system parameters. It is the intent of this process to ultimately render the most optimum schedule result by passing forward the best genetic results of the previous generation on to the next.

After the new generation of schedule objects has been created, the old or parent schedules will be removed from the system. This will result in the new child schedules becoming the new parent schedules that will be used in the creation of the next generation of child schedules.

During this phase genetic operators such as elitism, point mutation and crossover will be randomly applied to parent schedule objects to create new child schedules. After each new generation of schedules has been created, the system will once again apply both the repair and scoring processes. This entire generation process will continue forward until a viable set of schedules have been created.

Multi-Point Crossover. Multi-Point Crossover will randomly select two chromosomes (*schedules*), from the current generation. Next, one or more pivot points will be

randomly generated to establish crossover point for the two chromosomes. Finally, the upper section of genes from one chromosome will be merged with the lower genes of the other parent to create a new child chromosome that will become part of the next schedule generation. See Figures 3.2 and 3.3. [B3][B4]

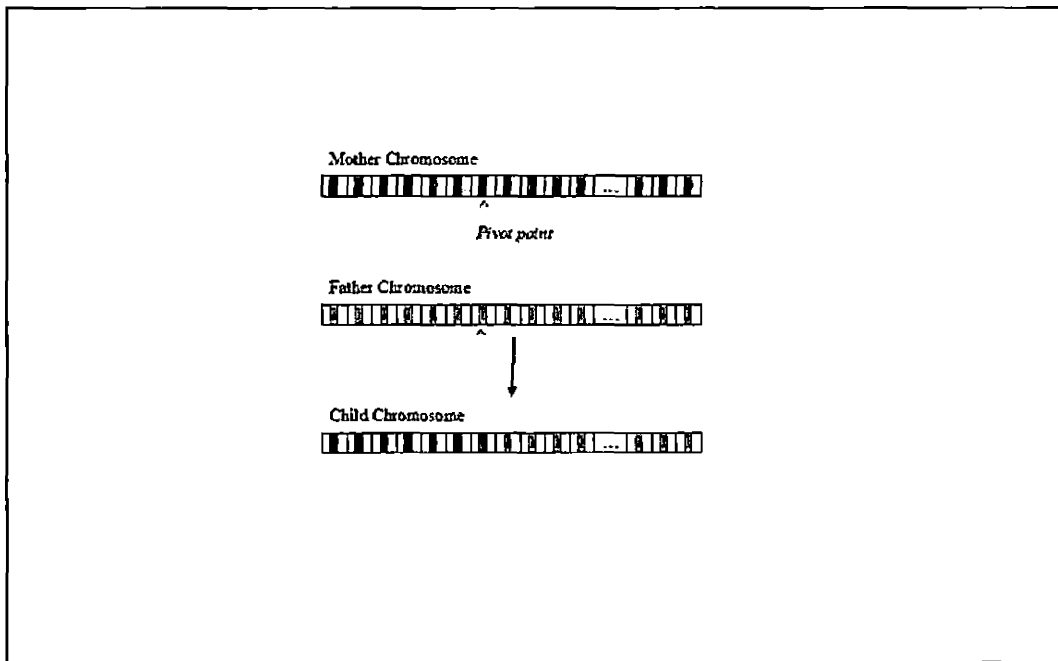


Figure 3.2. Single-Point Crossover

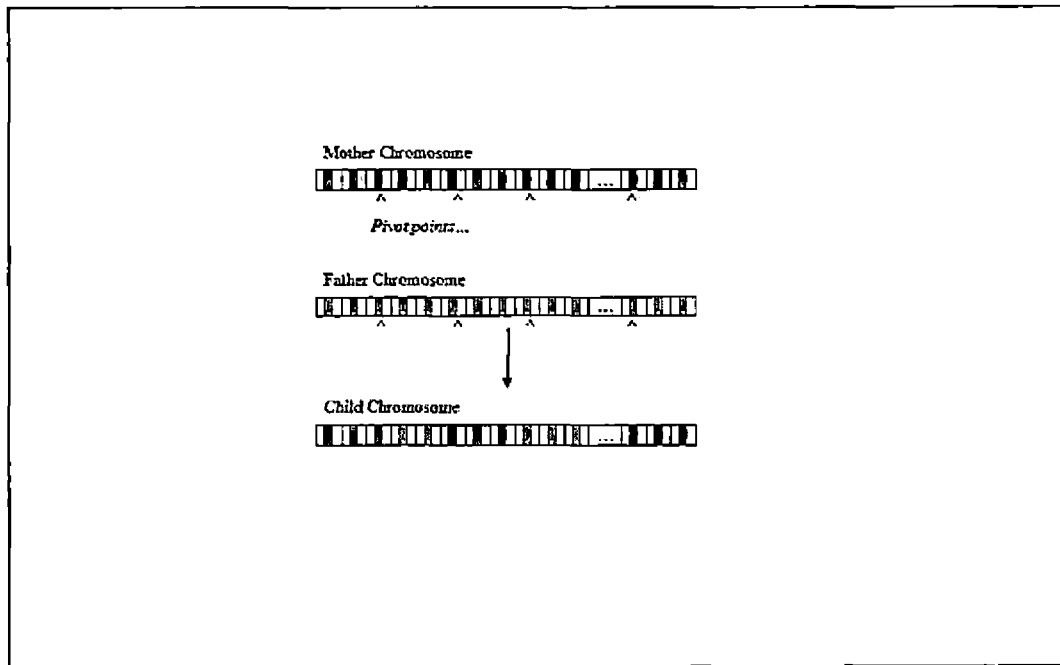


Figure 3.3. Multi-Point Crossover

Uniform Crossover. Uniform Crossover will randomly select two chromosomes (*schedules*), from the current generation. Next, random genes from each of the two parents will be selected to create a new child chromosome for the next generation. See Figure 3.4. [B3][B4]

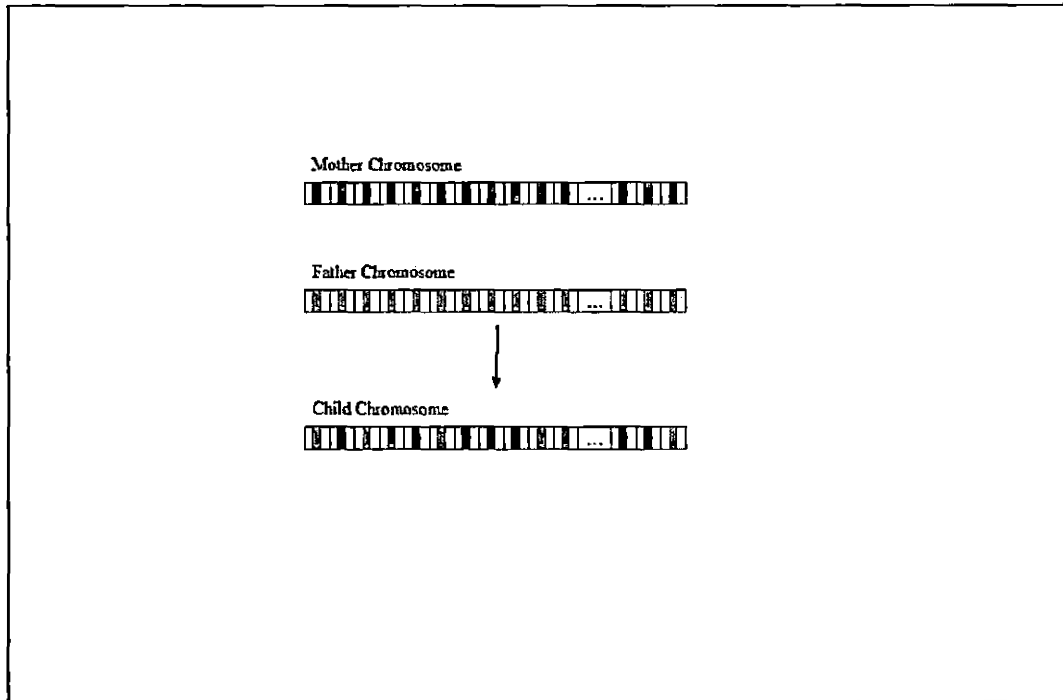


Figure 3.4. Uniform Crossover

Point Mutation. The Point Mutation function will first randomly select a parent chromosome (*schedule*), from the current schedule generation. Next, this function will randomly select a gene from the selected chromosome and changes one or more property values. See Figure 3.5.

[B2] [B3] [B4]

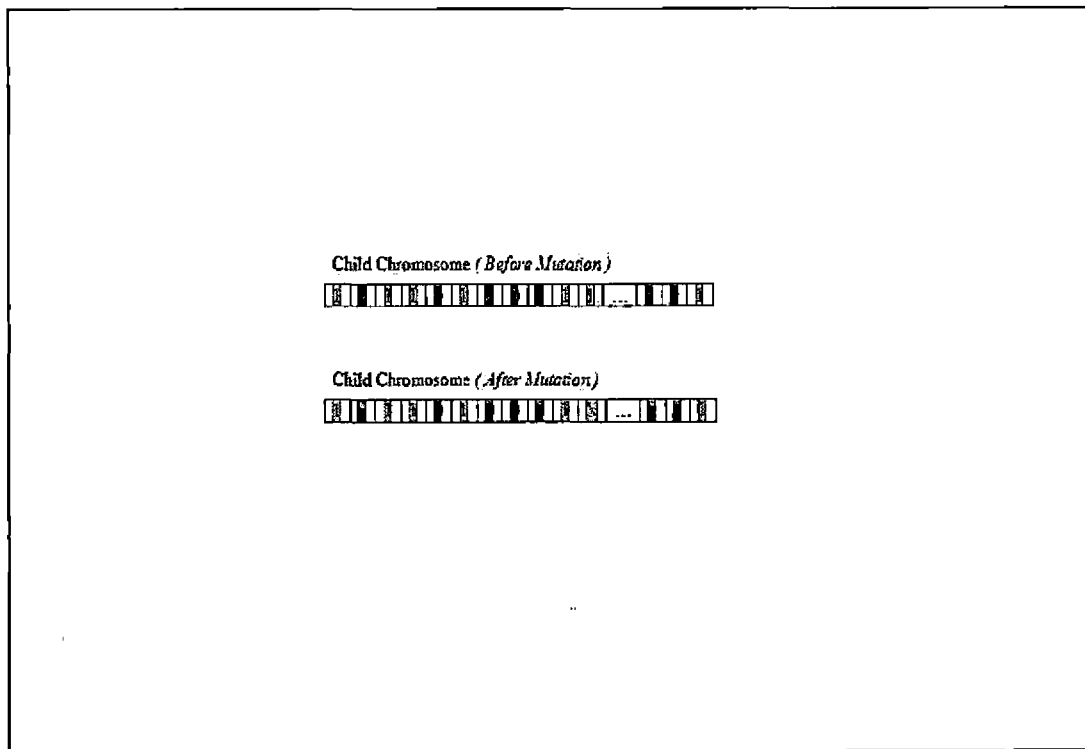


Figure 3.5. Point Mutation

Solution Determinants

At this point, two cut-off methods have been proposed. The first method will employ the optimizer to run for a prescribed number of generations and will halt when that number has been obtained. The other method will utilize an examination of scores per generation looking at the rate of change per generation. At the point the system determines that the change in fitness per generation has achieved a certain delta, the system will halt. Upon system halt, the final best three schedules will be determined and return to the BOSS as the optimized solution. [B2][B8]

Summary

By utilizing the Genetic Algorithm as the foundation for the schedule generation process, it is our expectation that this search method will provide both an effective and flexible foundation that will allow for future enhancements and modifications to BOSS-Opt system with minimal impact to overall system. As the BOSS-Opt matures and additional or different constraints are presented to the scheduling process, the GA should only be affected by new scoring and evaluation routines thus leaving the majority of generation

process unaffected by the new functionality. This will in theory allow for the BOSS-Opt to easily adapt to future demands.

CHAPTER FOUR

SOFTWARE DESIGN

System Implementation

Development Tools

Implementation of the BOSS-Opt was done using Java 5 and Sun's NetBeans Interactive Development Environment (IDE) version 6.5.1. In addition, a third party software library, Apache Commons Collections from The Apache Software Foundation was utilized to provide data access to MySQL databases.

Layered Architecture

The BOSS-Opt was built utilizing three functional software layers: Database Services, Business Model Services and System Model services specified in ranking order from lowest to highest thus creating a communications hierarchy. This implementation feature restricts communication between the service layers. As such, lower level layers are not allowed access to higher level services (i.e. Database Model layer cannot directly access the Business Model layer).

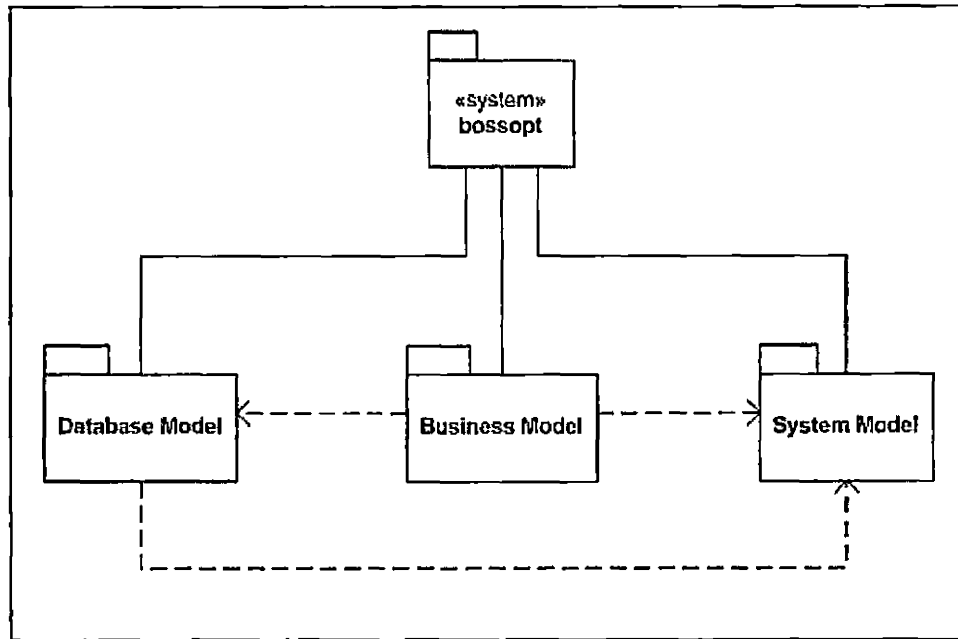


Figure 4.1. Package Hierarchy

Patterns

The BOSS-Opt is implemented using the following design patterns: Abstract Factory, Builder and Singleton implementation patterns. [B7]

High-Level System Software Packages

Bossopt

Bossopt is the top level package that contains the system applet class. This applet serves as the entry point for the scheduling system.

BusinessModel

This package contains the classes responsible for basic scheduling processes. These processes include resource and schedule factories along with helper utilities required to assist with data transfer between lower system layers.

BMItem. This class serves as the base class for all business model system data objects. All BMItems objects have an ID and description value. This allows for BMItems to be referenced by key from container objects such as map, lists and database tables.

Known subclasses of BMItem include: BMPersistentItem, BMResourceItem, ChromosomeItem and GeneItem.

The main function(s) of the class are as follows:

- BMItem() default constructor.
- BMItem(key) accepts unique key value for class instance.

- `BMItem(key, description)` accepts both a unique key value and item description for class instance.
- `GetID()` returns the key value for the item instance.
- `GetDesc()` returns a string description value for the item instance.
- `SetID(key)` accepts a key value which is used to set the key property for the item instance.
- `SetDesc(description)` accepts a string description value that is used to set the item instance description property.
- `clone()` inherited method that enables `BMItem` instances to be copied throughout the system.

`BMPersistentSet`. This class represents a container class that holds like `BMItems`. Persistent set allows for items to be loaded from and saved to external storage. In addition, this class allows for contained items to be sorted by different orders. Also, persistent sets may be replicated across the system.

The main function(s) of the class are as follows:

- `Erase()` deletes all items contained within the persistent set.

- `Erase(key)` accepts a key value which is used to remove the set item that corresponds to the key value.
- `GetItem(key)` accepts a key value which is used to retrieve the set item that corresponds to the key value. If the specified key is not found, this function will return a null reference.
- `GetRandomNext()` returns a reference to an item instance contained within the set. If the set is empty, this function will return a null reference.
- `Load(item)` accepts a `BMItem` instance and loads item to persistent set.
- `Load(hint, dbhelper)` accepts a `BMItem` type as a hint and database helper class to load one or more `BMItem` instances.

`BMPersistentItem`. Abstract class that inherits from `BMItem` and represents item data that may be retrieved from or stored on external data stores. Persistent items communicate with external data stores via database helper classes which implement the required logic to facilitate data transfer operations.

Known subclasses of BMPersistentItem include:

CourseItem, DBInfoItem, FacultyItem, RoomItem, ScheduleItem and UserItem.

The main function(s) of the class are as follows:

- GetDBHelper() returns a reference to the persistent item's database helper class.

BMDatabaseHelper. Abstract helper class used as a logical bridge between the business model and database layers which aids in the retrieving and storing of system items.

Known subclasses of BMDatabaseHelper include:

CourseDBHelper, DBInfoDBHelper, FacultyDBHelper, RoomDBHelper, ScheduleDBHelper and UserDBHelper.

The main function(s) of the class are as follows:

- ExecuteQuery(string) executes a SQL command string.
- GetDefaultSQL() returns a string value that represents the default SQL script used to define dataset table name.
- GetFilter() returns a string value that represents the SQL resultset filter script.

- `SetDefaultDb(enum)` accepts an enumeration which defines the default database unique identification number.
- `SetDefaultSQL(string)` sets the property used to define the dataset table name.
- `SetFilter(string)` sets the property used as resultset subquery filter.

ResourceFactory. Class used to express user and resource preferences to the scheduling system. Preferences are indicated by state values referenced by weekday-daily time periods.

ResourceManager. Abstract class used as the base class for all resource item factories. This class contains a persistent set member which acts as the container class for created resource items and exposes a function that returns a reference to the set member.

The main function(s) of the class are as follows:

- `ResourceManager()` constructor for resource manager object. Function initializes the protected member `BMPersistentSet` which act as the container class for resource items.

- `GetResourceSet()` returns a reference to the class member `BMPersistentSet` of resource items.

CourseResourceMgr. Subclass of `ResourceManager` used to create course resource items. Course resource items act as task elements for which scheduling activities are centered around.

The main function(s) of the class are as follows:

- `CourseResourceMgr(quarter, table)` constructor for course resource manager object. This constructor accepts a quarterly period identifier which is used to define the academic period for the derived schedule. Also, passed is an optional schedule name for a previous academic schedule that may be referenced during the course resource creation process.

FacultyResourceMgr. Subclass of `ResourceManager` used to create faculty resource items. Faculty resource items act as resource elements which are used in the process of course resources.

The main function(s) of the class are as follows:

- `SetHourRangePreferences(resource, bhour, ahour, day)`
private function used to set faculty scheduling preferences which are referenced during the academic scheduling process.

RoomResourceMgr. Subclass of `ResourceManager` used to create room resource items. Room resource items act as resource elements defining the available lecture rooms and labs which are used in the process of course resources.

PreferenceMap. Class used to express user and resource preferences to the scheduling system. Preferences are indicated by state values referenced by weekday-daily time periods.

The main function(s) of the class are as follows:

- `GetDayFromTimeSlot(timeSlot)` static function which accepts a timeslot value and returns the day of the week that the corresponds to the specified timeslot.
- `GetHourFromTimeSlot(timeSlot)` static function which accepts a timeslot identifier value and returns the hour of the day that the corresponds to the specified timeslot.

- `GetTimeSlot(day, hour)` static function that accepts a day and hour parameter to obtain preference map timeslot value.
- `GetTimeSlotValue(timeslot)` function that accepts a timeslot identifier value and returns a preference map timeslot value.
- `SetTimeSlotValue(timeslot, value)` function that accepts a timeslot identifier and value to set the preference map slot value. Function returns true if assignment completes successfully; false otherwise.
- `SetTimeSlotValue(timeslot, value, minutes)` function that accepts a timeslot identifier, value and minutes used to set the one or more preference map slot values. Function returns true if assignment completes successfully; false otherwise.
- `SetTimeSlotValue(day, hour, value)` function that accepts a day, hour and slot value to set the preference map slot value. Function returns true if assignment completes successfully; false otherwise.
- `SetTimeSlotValue(day, hour, value, minutes)` function that accepts a day, hour, value and minutes to set one or more preference map slot values. Function returns

true if assignment completes successfully; false otherwise.

BMResourceItem. Abstract class that inherits from BMItem and represents item data that may be retrieved from or stored on external data stores. Persistent items communicate with external data stores via database helper classes which implement the required logic to facilitate data transfer operations.

Known subclasses of BMResourceItem include: CourseResource, FacultyResource and RoomResource.

The main function(s) of the class are as follows:

- GetPreferenceMap() returns a reference to the resource item's preference map object.
- GetResourceType() returns an enumeration value which describes the resource data type, (e.g. course, faculty, room, etc.).
- SetPreferenceMap(map) accepts a preference map object which is used to set the map property of the resource item.
- SetResourceType(enum) accepts an enumeration value to set the resource type property.

ScheduleFactory. Class used to express user and resource preferences to the scheduling system. Preferences are indicated by state values referenced by weekday-daily time periods.

ChromosomeItem. Class used to express user and resource preferences to the scheduling system. Preferences are indicated by state values referenced by weekday-daily time periods. Cost, fitness

The main function(s) of the class are as follows:

- MultiCrossover(parent, pivots) replication function which accepts a parent chromosome item and an identifier which stipulates the number of pivot points to be applied to the replication process. This function will randomly select genetic elements from both the instance and parent chromosomes based upon the pivot points referenced and return a new child chromosome.
- UniformCrossover(parent) replication function which accepts a parent chromosome item. This function will randomly select genetic elements from both the instance and parent chromosomes and return a new child chromosome.

GeneItem. Abstract subclass of BMItem used to define base functionality of gene elements. This class exposes one abstract function PointMutation() which is to be implemented by subsequent subclasses.

AcademicGene. Subclass of GeneItem used to define the problem domain properties for the academic scheduling system. Academic gene items consist of the course identifier, faculty identifier and room identifier for each scheduled course item. In addition, each gene contains the assigned week day and start time for the course item.

The main function(s) of the class are as follows:

- PointMutation() public function used to randomly alter gene properties.

AvailabilityMap. Class used to express user and resource preferences to the scheduling system.

Preferences are indicated by state values referenced by weekday-daily time periods.

DatabaseModel

The databasemodel package includes all classes that abstract the external datastore elements. Currently, these

data items are maintained within a relational database management system.

DBRecordSet. Abstract helper class used as a logical bridge between the business model and database layers which aids in the retrieving and storing of system items.

Known subclasses of DBRecordSet include: DBCourseSet, DBInfoSet, DBFacultySet, DBRoomSet and DBUserSet.

The main function(s) of the class are as follows:

- ExecuteQuery(string) executes a SQL command string.
- GetDefaultSQL() returns a string value that represents the default SQL script used to define dataset table name.
- GetFilter() returns a string value that represents the SQL resultset filter script.
- SetDefaultDb(enum) accepts an enumeration which defines the default database unique identification number.
- SetDefaultSQL(string) sets the property used to define the dataset table name.
- SetFilter(string) sets the property used as resultset subquery filter.

DBConnectionMgr. Abstract helper class used as a logical bridge between the business model and database layers which aids in the retrieving and storing of system items.

The main function(s) of the class are as follows:

- `GetConnection(enum)` static function that accepts a enumeration as the unique database identifier and returns a logical connection to an external datastore.

SystemModel

System utility classes are contained within this package. These utility classes include the system logger, file dump and global static data definitions.

Data Item Subsystem

This system manages the various types of business level objects used by the scheduling system. In addition, this system defines the base class for all data related objects. These items include: persistent items, resource items and chromosome items. See Figure 4.2.

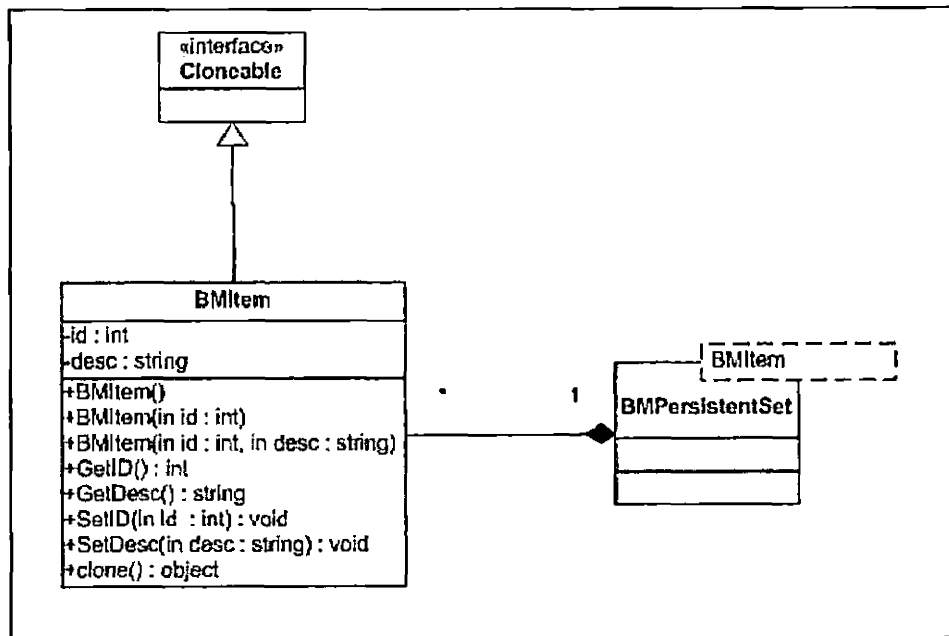


Figure 4.2. Data Item Subsystem

Database Access Subsystem

The database access system is the primary means for transferring data elements to and from external storage. Although this implementation effects communication with a Relational Database Management System(RDBMS), other external data stores may be accessed by extending this basic system.

In addition, in keeping with the separation of system services - Business Model to Database, helper classes are employed to act as mediator for the two distinct service layers. See Figure 4.3.

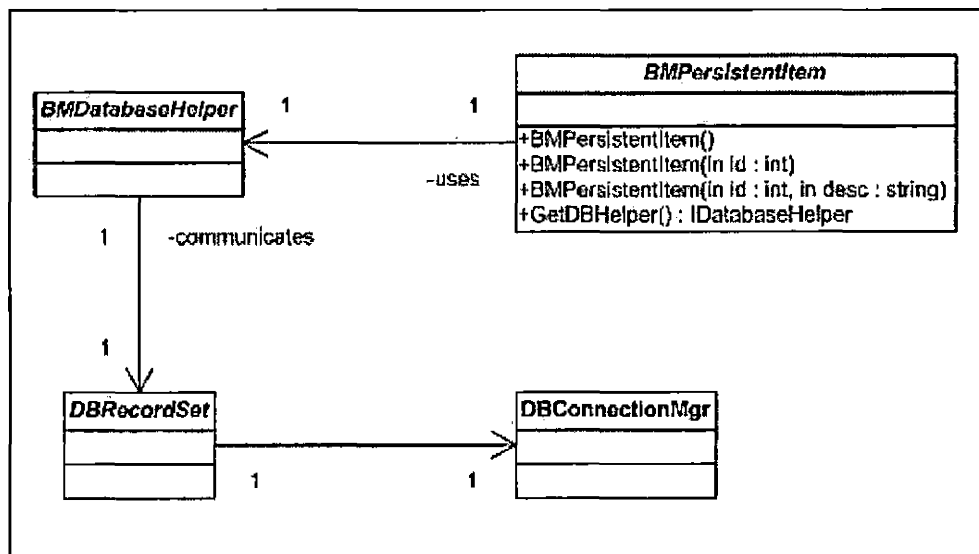


Figure 4.3. Database Subsystem

Resource Factory Subsystem

In order to present resource data items to the scheduling system, the Resource Factory system is tasked with obtaining data elements from external storage and reformatting these items into resource objects that may be referenced and/or manipulated by the scheduling process.

Although the resource items are primarily used to transmit persistent data properties to the scheduling process, each resource instance also contains a data matrix that abstracts the user and system preferences which will be evaluated during the creation of schedule items. See Figure 4.4.

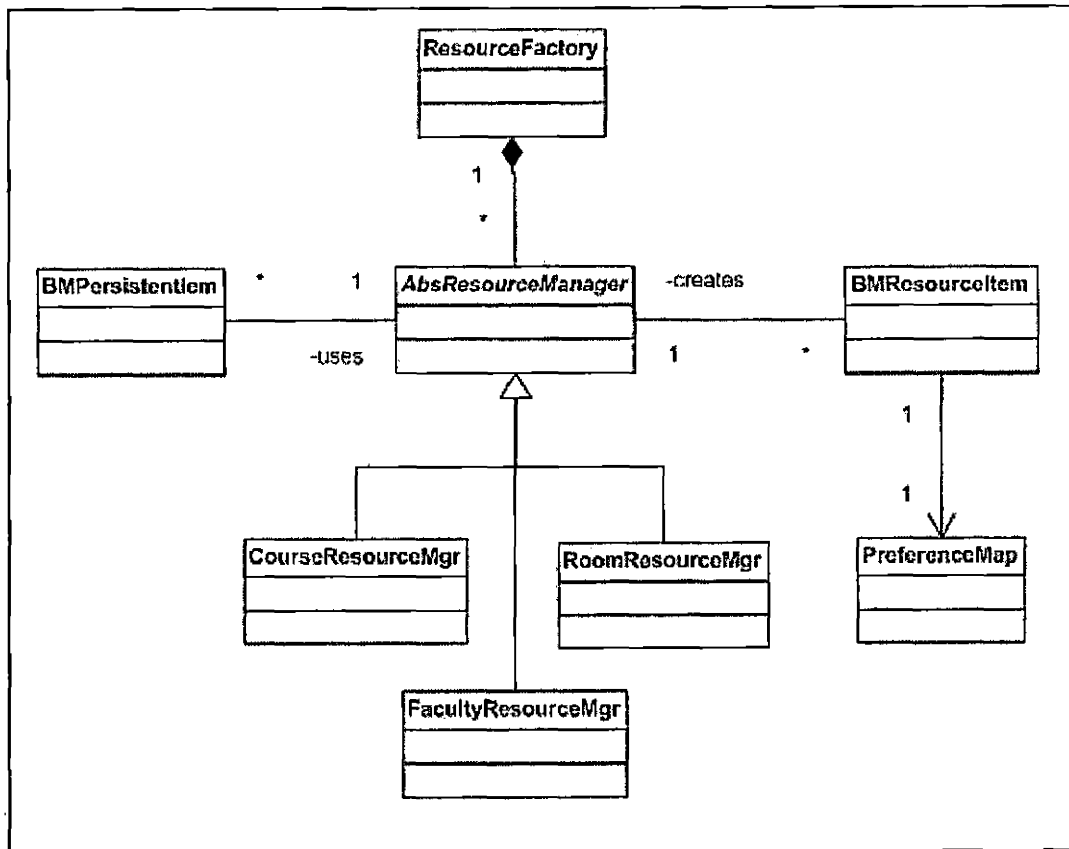


Figure 4.4. Resource Factory

Schedule Factory Subsystem

The Schedule Factory is the subsystem that manages the schedule creation process. This system retrieves resource items from the Resource Factory and processes these items by utilization of Genetic Algorithm (GA) methods. See Figure 4.5.

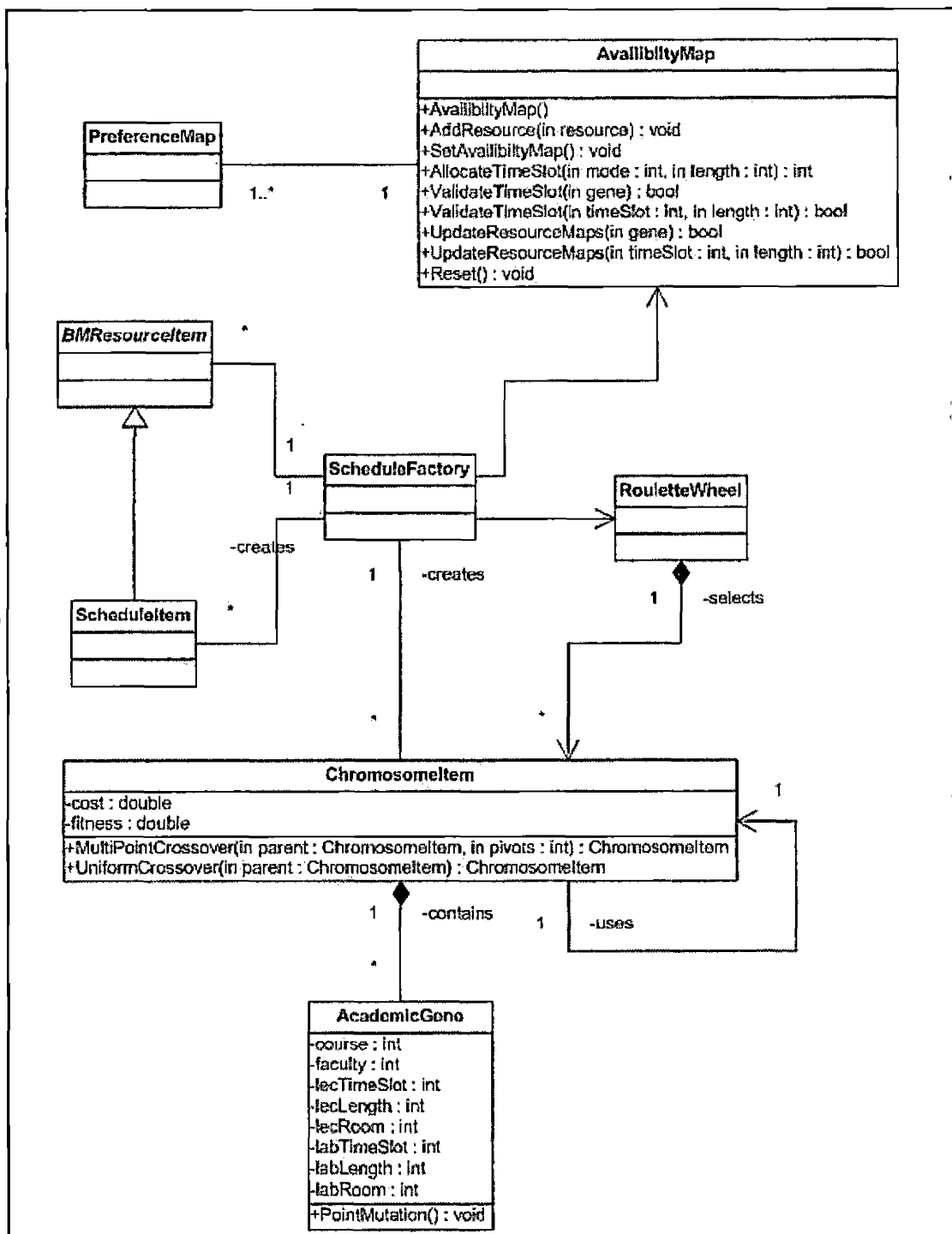


Figure 4.5. Schedule Factory

CHAPTER FIVE

TESTING AND RESULTS

Introduction

In an attempt prove the viability of the BOSS Optimizer system, four separate scheduling techniques were employed with the results from each scheduling method measured and evaluated for correctness.

It should be noted that in none of the four techniques was the main focus on finding the most optimal solution. However, each resulting schedule was examined to see how close it came to representing a solution that met the system constraints and adhered to system preferences.

Test Data Overview

Each of the four schedule builds relied on the same schedule data elements. These elements included courses, faculty, faculty course preferences and faculty availability preferences.

Course offerings observed during testing are included in Figure 5.1. This lists the course number, course name

along with the lecture length and the number of required sections.

<i>Course ID.</i>	<i>Course Name</i>	<i>Length (mins.)</i>	<i>Sections</i>
CSCI_201	Computer Science I	75	2
CSCI_202	Computer Science II	75	2
CSCI_292	Object Oriented Programming	75	2
CSCI_310	Digital Logic	110	2
CSCI_313	Machine Organization	110	2
CSCI_320	Programming Languages	75	2
CSCI_330	Data Structures	75	2
CSCI_431	Algorithm Analysis	110	2
CSCI_455	Software Engineering	75	2
CSCI_500	Formal Languages	110	2
CSCI_512	Artificial Intelligence	110	2
CSCI_535	Numeric Computation	110	2
CSCI_610	Modern Computer Architecture	110	2
CSCI_635	Numeric Simulation	110	2

Figure 5.1. Course Offerings

Courses having one or more laboratory periods are defined in Figure 5.2. This lists the course number, course name, lab length, preferred room number and required number of sections.

<i>Course ID.</i>	<i>Course Name</i>	<i>Length (mins.)</i>	<i>Room No.</i>	<i>Sections</i>
CSCI_201	Computer Science I	75	JB358	2
CSCI_202	Computer Science II	75	JB358	2
CSCI_310	Digital Logic	110	JB357	2
CSCI_320	Programming Languages	110	JB359	2
CSCI_610	Modern Computer Architecture	75	JB358	2

Figure 5.2. Courses Requiring Labs

In Figure 5.3., are list the courses that each professor desires to lecture. The desired courses are recorded in order of preference with the first entry having a greater ranking of preference than the next or successor.

<i>Professor</i>	<i>Preferred Courses</i>
Georgio	CSCI_401, CSCI_620
Botting	CSCI_320, CSCI_620, CSCI_646
Gomez	CSCI_610, CSCI_431, CSCI_401
Karant	CSCI_525, CSCI_634, CSCI_689
Mendoza	CSCI_572, CSCI_689
Murphy	CSCI_500, CSCI_350, CSCI_646
Schubert	CSCI_535, CSCI_635, CSCI_310
Turner	CSCI_292, CSCI_488, CSCI_572
Grad. Student	CSCI_201, CSCI_202

Figure 5.3. Course Preferences

In contrast to the professor list of desired courses, Figure 5.4., lists courses that each professor chooses not

to lecture. As with the desired courses, these courses are recorded in order of preference with the first entry having a higher ranking of dislike than the next or successor.

<i>Professor</i>	<i>Non-Preferred Courses</i>
Georgio	CSCI_201, CSCI_202
Grad. Student	CSCI_689, CSCI_660, CSCI_656, CSCI_646, CSCI_634

Figure 5.4. Non-Preferred Courses

Hour preferences are defined by Figure 5.5., which lists the faculty preference regarding which hours he/she desires to lecture.

<i>Professor</i>	<i>Preferred Lecture Times</i>	<i>Days</i>
Botting	before 5PM	M - F
Gomez	after 3PM	M - F
Karant	after 2PM	M - F

Figure 5.5. Hour Preferences

Schedule Tests and Findings

Manual Case Test 1

For this test, a schedule was created using 10 randomly selected courses. These courses were randomly placed in morning and evening periods in an effort to balance the academic load. During this scheduling process, neither faculty nor course preferences were recognized. As such, both professor and room assignments were fairly random with the exception of lab offerings which were allocated to resource rooms.

The intent of this test was to establish a baseline for the creation of a very simple case test and compare these findings to the generation of more elaborate schedules. In this test, the total time required to complete this schedule was 17 minutes. Results of this test can be seen in Figure 5.6.

<i>Class</i>	<i>Description</i>	<i>Days</i>	<i>Time</i>	<i>Faculty</i>	<i>Room</i>
CSCI_201	Comp Sci I	MW	8:00 A.M. - 9:15 A.M.	Student	JB358
CSCI_201	Comp Sci I_Lab	W	9:30 A.M. - 10:45 A.M.	Student	JB358
CSCI_201	Comp Sci I_Lab	M	12:00 P.M. - 1:15 P.M.	Student	JB358
CSCI_292	OO Programming	TR	8:00 A.M. - 9:15 A.M.	Turner	JB324
CSCI_320	Prog. Lang	TR	10:00 A.M. - 11:15 A.M.	Mendoza	JB359
CSCI_320	Prog. Lang_Lab	T	11:30 A.M. - 1:15 P.M.	Mendoza	JB359
CSCI_320	Prog. Lang_Lab	W	2:00 P.M. - 3:50 P.M.	Mendoza	JB359
CSCI_330	Data Structures	TR	10:00 A.M. - 11:15 A.M.	Georgio	JB322
CSCI_431	Algorithms	TR	6:00 P.M. - 7:50 P.M.	Gomez	JB357
CSCI_455	Software Eng.	MW	10:00 A.M. - 11:50 A.M.	Concepcion	JB359
CSCI_500	Formal Lang.	TR	9:00 A.M. - 10:15 A.M.	Botting	JB113
CSCI_512	A.I.	MW	10:00 A.M. - 11:50 A.M.	Turner	JB319
CSCI_535	Numeric Comp.	MW	12:00 P.M. - 1:50 P.M.	Karant	JB356
CSCI_635	Prog. Lang	TR	6:00 P.M. - 7:50 P.M.	Georgio	JB357

Figure 5.6. Test Schedule 1

Manual Case Test 2

In this test, another course schedule was created strictly by manual effort. This schedule consisted of 14 courses, 3 of which comprised both lecture and supplemental laboratory periods. As in the previous test, courses were randomly placed between morning and evening periods in an effort to evenly distribute the academic offerings.

For each course and laboratory offering, a professor was assigned. However, unlike the previous test, in the event that a faculty member registered a desired to lecture a particular subject, their preference was taken into

consideration where possible. In addition, any faculty preferences stipulating which courses that he/she chose not to lecture were also observed. In the event that faculty preferences did not exist for a particular course, a professor was randomly selected and assigned to the course offering.

Along with professor course preferences, professor time period preferences were also honored where possible. These preferences were used to convey the weekly and hourly period's professors desired or were available to lecture.

In utilizing this manual method of scheduling, a total effort of 127 minutes was recorded. Although this method did render a viable academic schedule, it was not necessarily the most optimal. Issues such as the number gaps between lecture periods, room distribution and professor workloads were not proven to be best possible solution. However, the manual method did by all accounts produce a workable course schedule as can be seen in Figure 5.7.

<i>Class</i>	<i>Description</i>	<i>Days</i>	<i>Time</i>	<i>Faculty</i>	<i>Room</i>
CSCI_201	Comp Sci I	MW	8:00 A.M. - 9:15 A.M.	Student	JB358
CSCI_201	Comp Sci I_Lab	W	9:30 A.M. - 10:45 A.M.	Student	JB358
CSCI_201	Comp Sci I_Lab	M	12:00 P.M. - 1:15 P.M.	Student	JB358
CSCI_202	Comp Sci II_Lab	TR	4:00 P.M. - 5:15 P.M.	Student	JB358
CSCI_202	Comp Sci II_Lab	T	5:30 P.M. - 6:45 P.M.	Student	JB358
CSCI_202	Comp Sci II_Lab	W	6:00 P.M. - 7:15 P.M.	Student	JB358
CSCI_292	OO Programming	MW	2:00 P.M. - 3:15 P.M. 10:00 A.M. - 11:50	Turner	JB324
CSCI_310	Digital Logic	MW	A.M.	Karant	JB357
CSCI_310	Digital Logic_Lab	W	12:00 P.M. - 1:45 P.M.	Karant	JB357
CSCI_310	Digital Logic_Lab	R	2:00 P.M. - 3:50 P.M. 10:00 A.M. - 11:15	Karant	JB357
CSCI_320	Prog. Lang	TR	A.M.	Botting	JB359
CSCI_320	Prog. Lang_Lab	T	11:30 A.M. - 1:15 P.M.	Botting	JB359
CSCI_320	Prog. Lang_Lab	M	4:00 P.M. - 5:50 P.M.	Botting	JB359
CSCI_330	Data Structures	TR	12:00 P.M. - 1:15 P.M.	Georgio	JB124
CSCI_431	Algorithms	TR	4:00 P.M. - 5:50 P.M.	Gomez	JB357
CSCI_455	Software Eng.	MW	8:00 A.M. - 9:50 A.M.	Concepcion	JB359
CSCI_500	Formal Lang.	TR	9:00 A.M. - 10:15 A.M. 10:00 A.M. - 11:50	Murphy	JB324
CSCI_512	A.I.	TR	A.M.	Mendoza	JB110
CSCI_535	Numeric Comp.	TR	2:00 P.M. - 3:50 P.M.	Schubert	JB322
CSCI_610	Modern Comp	TR	6:00 P.M. - 7:50 P.M.	Gomez	JB356
CSCI_610	Comp_Lab Modern	R	8:00 P.M. - 9:15 P.M.	Gomez	JB356
CSCI_610	Comp_Lab	W	6:00 P.M. - 7:15 P.M.	Gomez	JB356
CSCI_635	Prog. Lang	TR	2:00 P.M. - 3:50 P.M.	Schubert	JB322

Figure 5.7. Test Schedule 2

Basic Online Scheduling System Case Test

As our first automated test, the aforementioned data elements were presented to BOSS. Total processing time

utilizing this method was less than 5 seconds and produced the schedule depicted in Figure 5.8.

Although BOSS produced a schedule that was at best viable, upon closer examination, both incompleteness of course assignments along with course assignment errors become apparent.

First, with this schedule failed to assign rooms for all course sections. For those sections that do possess room assignments, it is apparent that these rooms were simply scheduled based upon room preference entries. As a result of this, course assignments have resulted in room conflicts due two or more course sections being assigned to the same room with overlapping time frames.

Second, of the twenty-four sections scheduled during this test, only seven sections were scheduled for weekly periods Tuesday and/or Thursday. This has resulted in the scheduled being biased for Monday, Wednesday periods thus resulting in an uneven distribution.

From this test we can see that BOSS, for the most part, does a reasonable job at creating schedules that are potentially viable with the aid of additional editing. However, we conclude that BOSS in many cases will fail to produce schedules that are optimal in their assignments.

Schedule - Fall 02						
Fall Classes						
Class	Description	Days	Time	Faculty	Room	
Winter Classes						
Class	Description	Days	Time	Faculty	Room	
CS6 120	Comp Sci	MW	9:20AM - 10:35AM	Grad Student		
CS6 201 Lab 1	Comp Sci Lab 1	MW	10:50AM - 12:05PM	Grad Student	JB336	
CS6 201 Lab 2	Comp Sci Lab 2	W	10:50AM - 12:05PM	Grad Student	JB336	
CS6 202	Comp Sci	MW	12:20PM - 1:35PM	Grad Student		
CS6 202 Lab 1	Comp Sci Lab 1	M	2:50PM - 4:05PM	Grad Student	JB336	
CS6 202 Lab 2	Comp Sci Lab 2	W	2:50PM - 4:05PM	Grad Student	JB336	
CS6 202	Comp Sci	MW	2:20PM - 3:35PM	David Toner		
CS6 300	Data Log Lab	W	4:00PM - 5:15PM	Josefine Mendoza		
CS6 300 Lab 1	Data Log Lab 1	M	6:05PM - 7:20PM	Josefine Mendoza	JB337	
CS6 300 Lab 2	Data Log Lab 2	W	6:05PM - 7:20PM	Josefine Mendoza	JB337	
CS6 316	Maths Com	TR	8:00AM - 9:15AM	Yasir Khan		
CS6 320	Program Lang	TR	10:00AM - 11:15AM	Richard Eddins		
CS6 320 Lab 1	Program Lang Lab 1	T	11:35AM - 12:50PM	Richard Eddins	JB339	
CS6 320 Lab 2	Program Lang Lab 2	R	11:00AM - 12:15PM	Richard Eddins	JB339	
CS6 330	Data Structures	TR	2:30PM - 3:45PM	Karen Wapner		
CS6 331	Algorithms	TR	3:00PM - 4:15PM	Ernesto Gomez		
CS6 405	Sci Eng	TR	5:00PM - 6:15PM	Georgina Escobar		
CS6 500	Formal Lang	MW	9:20AM - 10:35AM	Chris Murphy		
CS6 512	AI	MW	12:20PM - 1:35PM	Staff		
CS6 535	Net Comp	MW	2:20PM - 3:35PM	Karl Scharf	JB336	
CS6 610	Modern Comp	MW	4:00PM - 5:15PM	Ernesto Gomez		
CS6 610 Lab 1	Modern Comp Lab 1	M	6:05PM - 7:20PM	Ernesto Gomez	JB337	
CS6 610 Lab 2	Modern Comp Lab 2	W	6:05PM - 7:20PM	Ernesto Gomez	JB337	
CS6 600	Alm Sys	TR	8:00AM - 9:15AM	Karl Scharf		

Figure 5.8. Basic Online Scheduling System Schedule

Basic Online Scheduling System Optimizer Test

Our final test was conducted with the enhanced version of BOSS or BOSS Optimizer (BOSS-Opt). As in the previous test, the aforementioned data elements were also presented to BOSS-Opt. However, unlike the previous BOSS test, BOSS-

Opt required a processing time of 25 minutes. The results of this test can be seen in Figure 5.9.

Although, the amount of processing effort for BOSS-Opt was much greater in comparison to the previous BOSS, upon inspection, it's apparent that BOSS-Opt has produced a more viable scheduling solution.

First, all course sections have complete assignments. These assignments consist of each section having a lecturer, room and weekly period time slot assigned. Second, faculty course preferences and hour preferences were upheld in most course assignments. Finally, rooming conflicts have been reduced or eliminated.

Schedule - AlphaTest_Opt1

Fall Classes

Class	Description	Days	Time	Faculty	Room
-------	-------------	------	------	---------	------

Winter Classes

Class	Description	Days	Time	Faculty	Room
CSCI 201	Comp Sci I	TR	8:00 A.M. - 9:15 A.M.	Student	JB358
CSCI 201	Comp Sci I Lab	T	9:30 A.M. - 10:45 A.M.	Student	JB358
CSCI 201	Comp Sci I Lab	W	8:00 A.M. - 9:15 A.M.	Student	JB258
CSCI 202	Comp Sci II	TR	8:00 A.M. - 9:15 A.M.	Murphy	JB359
CSCI 202	Comp Sci II Lab	R	9:30 A.M. - 10:45 A.M.	Murphy	JB359
CSCI 202	Comp Sci II Lab	F	8:00 A.M. - 9:15 A.M.	Student	JB257
CSCI 202	OO Programming	MW	8:00 A.M. - 9:15 A.M.	Turner	JB108
CSCI 310	Digital Logic	MW	1:00 P.M. - 2:50 P.M.	Schubert	JB357
CSCI 311	Digital Logic Lab	W	3:00 P.M. - 4:50 P.M.	Schubert	JB357
CSCI 311	Digital Logic Lab	M	1:00 P.M. - 2:50 P.M.	Gomez	JB108
CSCI 313	Machine Org	MW	8:00 A.M. - 9:50 A.M.	Murphy	JB103
CSCI 320	Program Lang Lab	F	1:00 P.M. - 2:50 P.M.	Botting	JB358
CSCI 320	Program Lang Lab	W	2:30 P.M. - 4:20 P.M.	Botting	JB359
CSCI 320	Program Lang	MW	1:00 P.M. - 2:15 P.M.	Botting	JB359
CSCI 330	Data Structures	MW	8:00 A.M. - 9:15 A.M.	Botting	JB113
CSCI 431	Algorithms	MW	8:00 A.M. - 9:50 A.M.	Karam	JB359
CSCI 465	Soft Eng	MW	5:00 P.M. - 7:15 P.M.	Turner	JB108
CSCI 500	Formal Lang	MW	5:00 P.M. - 7:50 P.M.	Murphy	JB109
CSCI 512	AI	TR	8:00 A.M. - 9:50 A.M.	Turner	JB108
CSCI 535	Num Comp	TR	1:30 P.M. - 2:50 P.M.	Schubert	JB358
CSCI 510	Modern Comp	TR	9:00 A.M. - 9:50 A.M.	Gomez	JB357
CSCI 510	Modern Comp Lab	R	10:00 A.M. - 11:15 A.M.	Gomez	JB357
CSCI 510	Modern Comp Lab	W	8:00 A.M. - 9:15 A.M.	Gomez	JB357
CSCI 635	Num Sim	MW	1:00 P.M. - 2:50 P.M.	Turner	JB112

Figure 5.9. Optimized Schedule

Findings

After performing several schedule tests with like data, we have shown how by increasing the size of the scheduling problem or its complexity by means of added constraints or preferences, that the overall processing effort increases. This holds true for both manual and automated test cases. In the manual tests, we demonstrated how by adding a few more course sections and adhering to a set of predefined preferences increased the processing effort by a factor of 12. In the case of the automated tests, by adhering to faculty preferences, we increased the effort by a factor of 300.

However, it should be noted that with the increase in processing effort, there was verifiable increase in the viability of the schedule produced. This study by example tends to correlate with the characteristics of timetabling or scheduling problems being NP-Hard and polynomial in solution time complexity.

In conclusion, though we have demonstrated that by utilizing Genetic Algorithms (GA's) in our implementation of BOSS-Opt, we have been able to produce better scheduling solutions, we have yet prove that our use of GA's has lead

to creation of the most optimal solution. That hopefully,
will be for those whom choose to carry forward this study.

CHAPTER SIX

SYSTEM MAINTENANCE

Introduction

The Basic Online Scheduling System (BOSS) version 4.2 is a PHP program that assists a department at CSUSB to help create and manage a schedule for the entire school year. For reference on how to use the software, please see the BOSS help pages. What follows is a document that will aid in the installation of BOSS. The current version of BOSS has four user types:

- Admin- Users able to add and edit the users for a particular department.
- Faculty- Teachers who can setup their preferences on what courses they want to teach.
- Staff- Users who can view and print the schedule once it has been made.
- Chairs- Users who can create courses and schedules from the courses and faculty preferences.

The code is divided up into four folders, one folder for each user. Each folder has a header folder that

contains the header and footer files for each php file. The manner in which BOSS is shown is contained within these files.

The schedule is created through a near optimal algorithm. The idea is to lay out each class so they are spread out and classes with the same level are not overlapping. Then each teacher's preference is attached to a particular class. If the teacher is unavailable during a time the class is at then the algorithm will attempt to move the class. Finally rooms are assigned. The classes that are in the same room at the same time or do not have an assigned teacher will be flagged as a problem. The general flow of the program is as follows:

1. The admin creates the users.
2. The chair creates the courses for a schedule and sets his/her preferences.
3. The faculty each set their own preferences.
4. The chair creates a schedule.
5. The chair edits the schedule.
6. The chair approves a schedule draft.
7. The faculty responds through feedback forms.
8. The chair edits the schedule again.
9. The chair approves the final schedule.

10. The staff views the final schedule for printing.

Contents

This cd contains the following folders:

- Boss - contains all of the source code for BOSS.
- Boss-Opt - contains the schedule optimization java applet and supporting files.
- Documentation - contains the documentation for BOSS, IEEE documents and design documents
- Db - sample MySQL databases; the import command:

```
mysql -u 'username' -p 'dbname' < 'sql database file'
```

can be used to copy a database file into MySQL as long as the database 'dbname' already exists
- Presentation - presentation used to promote BOSS on the final day
- Setup_software- contains some software you may find helpful in designing BOSS

Server Setup

The following are instructions on how to setup a development server for BOSS.

BOSS system is an online application. It needs a web server, which has Apache 2, PHP 5, MySQL 5, and Sendmail. Our recommendations for hardware and software requirements are:

- CPU: 600 Mhz (Intel PIII) or 1.4 GHz (AMD Athlon)
- RAM: 512 MB RAM
- HDD: 20 GB IDE or SATA HDD - Ethernet network card.
- OS: CentOS.

<http://www.centos.org/>

- Apache(2.0.59): The Apache Software Foundation provides support for the Apache community of open-source software projects.

<http://www.apache.org>

- PHP(5.1.6): PHP is a widely used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

<http://www.php.net>

- MySQL(5.0.48): The MySQL® database has become the world's most popular open source database because of its consistent fast performance, high reliability and ease of use. It's used in more than 6 million installations ranging from large corporations to

specialized embedded applications on every continent in the world.

<http://www.mysql.com>

- Sendmail(8.13.5): sendmail X is a message transfer system that has been designed with these main topics in minds: 'security' 'reliability' 'efficiency' 'configurability' 'extendibility'.

<http://www.sendmail.org>

Installation Instructions

The following command can be used install all modules needed. If error occurs, please investigate and resolve using a web search engine, since debugging installation is beyond the scope of this manual.

```
# yum install php mysql httpd
```

For additional Installation help, Dr. Turner's website has a lot of good information that can help you with the initial setup of the Linux web server, CVS, etc. You can find his info at:

<http://csci.csusb.edu/turner/sysadmin>

Setup Basic Online Scheduling System Software

Within the setup software folder there are several executables that will help you develop PHP applications on your system (mostly for windows). A few notable ones are:

- Eclipse and phpEclipse - Eclipse is an IDE normally used for developing Java and C++ apps, but you will find a folder called `net.sourceforge.phpclipse_1.1.8.bin.dist` that contains plugins that allow eclipse to be used with php as well.
- Wamp - This is a wonderful program for windows that will install apache, php5, and MySQL all at once and in the right places so you don't have to do extra setup.

Getting Started

In order to start using BOSS, after you have copied the source code to the apache web directory, you need to setup Boss to connect to MySQL and then create the appropriate databases in MySQL. The structures of the databases can be found in the database design document in the design folder.

To make sure Boss properly connects you must change the MySQL variables at the top of boss/util/setup.php. These include the hostname of the MySQL server, as well as the username and password you use to connect. NOTE that you may have to change several of these variables throughout the setup.php, login.php, loginpost.php page.

Currently there are two utility pages that will help setup your databases in the boss/util folder. First use createMaster.php to create the master database that keeps a record for each department database. You can then use the createDept.php page to create new departments. When a department is created an admin user will automatically be created with the username "admin" and the password "password", note all in lowercase. With the admin user, you can now create any of the users you need. If these pages fail then either you don't have MySQL setup correctly or you have not changed the setup.php variables to your account.

For security purposes there is no page to delete a department. This can be done however directly through the MySQL console by dropping the department database and deleting the department record out of master.info.

Basic Online Scheduling System Optimizer Installation

The BOSS-Opt is a Java applet that is executed from the standard version of BOSS. The following instructions detail the BOSS-Opt setup:

1. From the project CD, copy the BOSSOpt\build\classes folder to the %Apache%\htdocs\boss\chair directory.
2. Next, copy from CD the BOSSOpt\dist folder to the %Apache%\htdocs\boss\chair directory.
3. Locate nameSchedule.php in the %Apache%\htdocs\boss\chair directory and rename this file to nameSchedule.old.
4. Copy from the project CD, BOSSOpt\forms\nameSchedule.new to the %Apache%\htdocs\boss\chair directory.
5. Next, rename %Apache%\htdocs\boss\chair\nameSchedule.new to nameSchedule.php.
6. Finally, copy from the project CD, BOSSOpt\forms\BOSSOptApplet.php to the %Apache%\htdocs\boss\chair directory.

This completes the setup of the BOSS Optimizer system.

Basic Online Scheduling System Optimizer Project Build

The version of BOSS Optimizer used during this project study was developed and compiled using Sun's NetBeans IDE. A copy of this IDE can be found on the project CD under the NetBeans directory.

To re-compiled the BOSS-Opt project, simply install the NetBeans IDE on the designated development PC. Then, using NetBeans, create a new Java project from existing source code by pointing to the BOSSOpt\src directory.

CHAPTER SEVEN

CONCLUSION AND FUTURE DIRECTIONS

Conclusion

After an examination of the end results for this project study, we have determined that by applying a Genetic Algorithm to the BOSS-Opt implementation, we have been able to produce a system that in comparison to the previous version of BOSS renders more optimum schedule solutions. One of the most noteworthy characteristics of this new implementation is that it all but eliminates the need for manual editing of resultant schedules. In addition, by applying the GA to this scheduling problem, we have been able to create a simple software framework that is easily adaptable to ever-changing system constraints thereby, satisfying our previously stated system requirements. However, we must emphasize that our solution has not been proved to generate the most optimum solution. At best, all that can be stated is that the solutions generated by this process are a "Good Fit". The pursuit of the "Best Fit" is an endeavor that we leave for those desiring to carry forth this research.

Future Directions

As previously stated, the optimizing routine implemented in this project does not allow for us to adequately identify all local optima of the solution set. As such, identification of the local minimum escapes our detection thus preventing our determination of the "Best Fit" schedule. Going forward, future research should center on routines and techniques that would help identify local optima within the solution space and ultimately detect the optimum solution.

One suggested avenue of research deals with the application of simulated annealing techniques within the Roulette Wheel selection function. Such techniques may lend to better selection of solution candidates in the crossover phase of the process thereby propagating solutions having desired traits and attributes forward with each generation.

Another area of research to consider is the use of Bayesian probability along with branch and bound algorithms in determining the fitness of solution candidates. Use of the aforementioned techniques may better enable the GA to eliminate less desirable candidates from each solution set.

In addition, several recent studies have been conducted in the area of population regression and deterministic population shrinkage routines to better locate optima. [B5] [B6]

It is hoped that by applying one or more of these determinate routines that the BOSS-Opt will be able identify optimum solution candidates thereby, greatly enhancing the efficiency and effectiveness of the BOSS Optimizer (BOSS-Opt).

REFERENCES

- [B1] The Institute of Electrical and Electronics Engineers, Inc., IEEE Std 830-1998, *"IEEE Recommended Practice for Software Requirements Specifications"*
- [B2] Burke, Edmund, Elliman, David, Weare, Rupert, Department of Computer Science, University of Nottingham, *"A Genetic Algorithm Based University Timetabling System"*, July 1995
- [B3] Fernandes, Carlos, Caldera, Joao Paulo, Melicio, Fernando, Rosa, Agostinho, ACM 1-58113-086-4/99/0001, *"High School Weekly Timetabling by Evolutionary Algorithms"*, February 1999
- [B4] Sigl, Branimir, Golub, Marin, Mornar, Vedran, Faculty of Electrical Engineering, University of Zagreb, *"Solving Timetable Scheduling Problem Using Genetic Algorithms"*, July 2008
- [B5] Laredo, Juan Luis J., Merelo, Juan Julian, Fernandes, Carlos, Gagne, Christian, Department of Architecture and Computer Technology, University of Granada. ETSIT. Spain, *"Improving Genetic Algorithms Performance via Deterministic Population Shrinkage"*, July 2009

[B6] Yu, Tian-Li, Lin, Wei-Kai, Taiwan Evolutionary Intelligence Laboratory, Department of Electrical Engineering, National Taiwan University, "Optimal Sampling of Genetic Algorithms on Polynomial Regression", July 2008

[B7] Gamma, Erich, Johnson, Ralph, Helm, Richard, Vlissides, John M., Booch, Grady, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994

[B8] Reeves, Collin R., Rowe, Jonathan E., "Genetic Algorithms: Principles and Perspectives: a Guide to GA Theory", Boston Springer Science & Business Media, 2003