FACULTY TIMETABLING USING GENETIC ALGORITHM

LIONG BOON YAUN

A thesis submitted in partially fulfillment of the requirements for the award of degree of Bachelor of Computer Science (Software Engineering)

Faculty of Computer Systems & Software Engineering
Universiti Malaysia Pahang

MAY 2011

ABSTRACT

Faculty Timetabling using Genetic Algorithm (FTGA) is an application that generate optimum timetable for faculty. The target user of this application is faculty staff who responsible in generate timetable. The problem statement of the project is many clashing exist in the timetable. Faculty staff needs to solve the clashing manually. This will waste time and it is a problem for staff to solve the clashing. By implement GA, clashing will be reduced. The objective of the project is to develop a prototype in scheduling application for generates an optimum timetable for a faculty. Genetic algorithm will be implemented. The scope of FTGA is Faculty of Computer Systems & Software Engineering (FCSSE). The methodology use in this project is prototype model. The testing result show 95 out of 100 test cases achieved the maximum fitness value which means there is no clashing in the timetable. The maximum generation is set to 15 generation. Population for each generation is 3 populations. Percentage of FTGA solve the problem is 95%.

ABSTRAK

Penjadualan Fakulti menggunakan Algoritma Genetik (FTGA) adalah sebuah aplikasi yang menghasilkan jadual yang optimum untuk fakulti. Target Pengguna untuk aplikasi ini adalah kakitangan fakulti yang bertanggung jawab dalam menghasilkan jadual waktu. Masalah bagi projek ini adalah berlakunya banyak bertembungan dalam jadual waktu. Kakitangan fakulti perlu menyelesaikan pertembungan ini secara manual. Hal ini membuang masa dan merupakan masalah bagi kakitangan falkuti untuk menyelesaikan pertembungan ini. Dengan menerapkan GA, pertembungan boleh dikurangkan. Tujuan projek ini adalah untuk membina prototaip dalam aplikasi penjadualan untuk menghasilkan jadual waktu yang optimum untuk fakulti. Algoritma genetik akan diimplikasikan. Skop FTGA adalah Fakulti Sistem Komputer & Kejuruteraan Perisian (FSKKP). Metodologi yang digunakan dalam projek ini adalah model prototaip. Keputusan ujian menunjukkan 95 daripada 100 kes uji mencapai nilai fitness maksimum dengan tiadanya pertembungan dalam jadual waktu. Generasi maksimum ditetapkan adalah 15 generasi. Populasi untuk setiap generasi adalah 3 populasi. Peratusan FTGA mengatasi masalah pertembungan jadual waktu adalah 95%.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

An introduction of Faculty Timetabling using Genetic Algorithm (FTGA) will be presented in this chapter. The purpose of this chapter is to discuss the objectives, problem statement and scope of the project.

1.1 Introduction

There are three typical of timetable clashing in a course timetable. They are student's timeslot clashing, lecturer's timeslot clashing and classroom clashing. These clashing happen are mostly because the timetable is scheduled manually. A scheduling application will be developed to replace manual scheduling. By implement Genetic Algorithm into the scheduling application, these clashing can be minimized. Thus, an optimum timetable can be produce.

1.2 Problem Statement

Producing an optimum timetable for each course in a University's faculty is not an easy task. Especially we need to concern about the constraints such as, limited classroom and laboratory, number of lecturer and students. Most of the timetable now is done by hand or with limited help of a simple administration system. The timetable generate by current existing application contains a lot of clashing. Faculty staff needs to solve the clashing manually. This will waste time and it is a problem for staff to solve the clashing. It is quite often involves taking the previous year's timetable and modifying it so it will work for the new semester [1]. Besides that, the number of student increase every year, this mean it is not suitable to use previous year timetable. The constraint for producing the timetable also must be reconsidered every semester. New timetable must be producing every semester.

1.3 Objectives

i. To develop a prototype in scheduling application for generates an optimum timetable for a faculty.

ii. To apply genetic algorithm to produce an optimum timetable.

1.4 Scope

i. The scope of this project is Faculty Computer System and Software Engineering.

ii. The applications that use to develop this system are Microsoft Visual Studio 2010 and Microsoft SQL 2005 for the database.

iii. Total of 18 Lecturers and approximate 60 class slots use in this project.

iv. Timetables will be generated based on the number of rooms.

v. This application is a standalone application.

1.5 Thesis Organization

This thesis is divided into 6 chapters and each chapter is devoted to discuss different issue in the project. Below is a summary of the content for each chapter:

i. Chapter 1
   - Introduction to the project is presented along with the project's problem statement, objectives of the project and the scopes of the project.

ii. Chapter 2
   - Some research and literature related to the project are reviewed and discussed in this chapter.

iii. Chapter 3
   - The methodology of the project development is discussed.

iv. Chapter 4
   - The implementation of the project is explained.

v. Chapter 5
   - The testing result of the system and discussion on the result are presented.

vi. Chapter 6
   - Summary of the project is presented.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter will discuss about the background of the genetic algorithm and the process in the algorithm. The schedule problem will also be discussed. Three applications that develop using GA will be explained. The purpose of this chapter is to increase the knowledge on how GA implements in scheduling.

2.2 Genetic Algorithm (GA)

The concept of Genetic Algorithm (GAs) is introduced by John Holland, a professor of psychology at the University of Michigan, in the early 1970s [2]. GA is inspired by Darwin's Theory of Evolution.

According to Darwin's Theory, each individual need to struggle for survive. For those with the "fitness" genetic traits will have greater chance to survive compared with others [3]. This is call natural selection. With natural selection, great genetic traits will be passed to the next generation. After a few generations, the genetic traits will become dominant among the population. The population is evolved based on "survival of the fittest" [3]. A top level description of this process is given in Figure 2.1 [4].

```
Create a population of creatures.
Evaluate the fitness of each creature.
While the population is not fit enough:
{
        Kill all relatively unfit creatures.
                        While population size < max:
        {
                Select two population members.
                Combine their genetic material to create a new creature.
                Cause a few random mutations on the new creature.
                Evaluate the new creature and place it in the population.
        }
}.
```

Figure 2.1: Top Level of GAs

GA is a powerful optimization method. It can use to solve many problems that are not easy to solve by other techniques. Examples of difficult problems, which cannot be solved by "traditional" way, are NP problems. NP stands for nondeterministic polynomial and it means that it is possible to "guess" the solution (by some nondeterministic algorithm) and then check it, both in polynomial time [5]. GA start with generate an initial population of chromosome. Define the fitness function and evaluate the chromosome with the fitness function. Chromosomes will be selected from the evaluation. Then, the GA's operator will be applied on the chromosomes that have been selected to create new chromosomes. Replace the initial population with the new population. Repeat the process until the fitness population is created. The following figure 2.2 is the basic outline of the Genetic Algorithms [6].

1. [Start] Generate random population of n chromosomes (suitable solutions for the problem)
2. [Fitness] Evaluate the fitness f(x) of each chromosome x in the population
3. [New population] Create a new population by repeating following steps until the new population is complete
    - [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected).
    - [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
    - [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
    - [Accepting] Place new offspring in a new population.
4. [Replace] Use new generated population for a further run of algorithm
5. [Test] If the end condition is satisfied, stop, and return the best solution in current population
6. [Loop] Go to step 2

Figure 2.2: Outlines of the Genetic Algorithms

The process is similar with Darwin's Theory of Evolution. GA purpose is to produce the "fitness" solution for the problem.

2.2.1 Encoding

GA starts with encoding the chromosome. Genetic algorithms code the candidate solutions of an optimization algorithm as a string of characters which are usually binary digits [7].

Table 2.1: Encoding

Chromosome A	1011 0011
Chromosome B	0100 1101
Chromosome C	0110 0101
Chromosome D	1001 0110

2.2.2 Fitness

Fitness can be define or compute by various way. The easiest way is by adding one point to each gene in the chromosome that corresponding to the ideal. The point that rated for each chromosome is better to transform into percentage. This will make it easier for selection stage. Another way is define a fitness function and evaluate each chromosome with the fitness function.

Table 2.2: Fitness

Chromosome	**Fitness**
Chromosome A	9
Chromosome B	7
Chromosome C	5
Chromosome D	4

2.2.3 Selection

This process guides the evolutionary algorithm to the optimal solution by preferring chromosomes with high fitness [8]. The most common techniques use in selection is Roulette wheel selection. Roulette selection is based on the fitness of the chromosome. Each chromosome represents a part of the pie. The chromosome with the highest fitness will have the largest area of the pie. The area of the pie is corresponding with the chromosome's fitness. The roulette wheel will spin and the chromosome that the arrow stops by will be selected. The arrow may stop by the chromosome that with the lowest fitness value. It is not necessarily that all the selected chromosomes are the fitness.

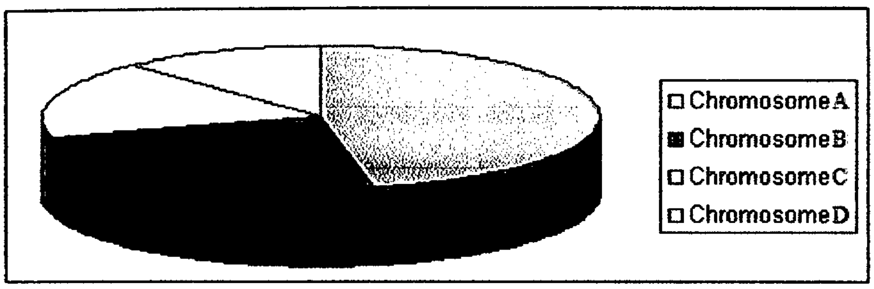


Figure 2.3: Roulette Wheel Selection

2.2.4 Crossover

Crossover is a process combining two selected chromosome genes to produce new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents [9]. The gene exchange based on the crossover point which is randomly chooses.

i. One Point Crossover

One Crossover point is randomly selected within the chromosome and interchanges the two parent chromosomes at this point to produce two new offspring.

Table 2.3: One Point Crossover

Chromosome A	100|**1 0110**
Chromosome B	110|*1 1001*
Offspring A	100|*1 1001*
Offspring B	*110*|**1 0110**

ii. Two Point Crossover

Two Crossover points are randomly selected within the chromosome and interchanges the two parent chromosomes at these points to produce two new offspring.

Table 2.4: Two Point Crossover

Chromosome A	100|**1 0**|110
Chromosome B	110|1 1|*001*
Offspring A	100|1 1|110
Offspring B	*110*|**1 0**|*001*

Crossover occurs during evolution according to a user-definable crossover probability [10].

2.2.5 *Mutation*

This is a background operator which allowing a large search space to be explored and produces some random changes in various chromosomes [8]. Mutation is a process where randomly choose one of the gene in the chromosome and change it. This process happens on the offspring's gene after the crossover process with a small probability which define by user. This probability should usually be set fairly low. If it is set to high, the search will turn into a primitive random search [11]. The purpose of the mutation is to ensure that the search algorithm is not trapped on a local optimum [12].

Table 2.5: Mutation

Offspring A	10*0*1 1110
Mutated Offspring A	*10**1**1 1110*
Offspring B	1101 0*0*01
Mutated Offspring B	1101 0*1*01

2.3 Scheduling Problem

Two Scheduling Problems will be briefly explained in this part which is Nurse Scheduling Problem and Job-Shop Scheduling Problem.

2.3.1 Nurse Scheduling Problem

Nurse Scheduling Problem (NSP) is a scheduling task consists of assignment of shifts and holidays to nurses for each day on the time horizon, taking into consideration a variety of conflicting interests or objectives between the hospitals and individual nurses [13]. NSP appear as a NP-hard problem. This is because the high number of constraints.

There are three shifts for a nurse which is morning shift, late shift and night shift. The time period is depend on the hospital. Different hospital maybe varies.

Table 2.6: Type of Shift

	Shift	From	Till
M	Morning Shift	06:45	14:45
L	Late Shift	14:30	22:00
N	Night Shift	22:00	7:00

No nurse will work all three shifts on the same day. A nurse doesn't do a night shift followed by a morning shift the next day [13]. This is because nurses who work rotating shifts had complaints concerning fatigue and this was highest in the night shift [14]. When the nurse is not in good condition, it maybe affects their efficiency. These are the basic constraint of the schedule.

Beside hard constraints and soft constraints need to be fulfill, there are other constraint such as hospital constraints, work regulation constraints, and personnel constraints.

There is some personnel nurse or night nurses that work in the hospitals. They are different from the fulltime nurses that work in hospital. They sign up contract or work agreement with the hospital. This work agreement leads to the formulation of the work regulation constraints. Following are some example of the work regulation constraints [15]:

- Minimum/Maximum number of consecutive days
- Minimum/Maximum number of hours worked
- Minimum/Maximum number of consecutive free days
- Maximum number of assignments per day of the week
- Maximum number of assignments for each shift type
- Maximum number of a shift type per week
- Number of consecutive shift types

- *Assign 2 free days after night shifts*
- Assign complete weekends
- Assign identical shift types during the weekend
- Maximum number of consecutive working weekends
- Maximum number of working weekends in a 4-week period
- Maximum number of assignments on bank holidays
- Restriction on the succession of shift types
- Patterns enabling specific cyclic constraints
- Balancing the workload among personnel

All of these constraints make the scheduling problem become complicated and hard to solve. Many research paper that study NSP trying to solve the problem using GA. Ahmad Jan, Masahito Yamamoto, and Azuma Ohuchi proposed a research paper to solve NSP. The objective of the research is to investigate difficulties that occur during solution of NSP using Evolutionary Algorithms, in particular Genetic Algorithms [13].

2.3.2 Job Shop Scheduling Problem

Job-shop is a system that process n number of tasks on m number of machines [16]. In job-shop scheduling problem (JSSP) environment, there are j jobs to be processed on m machines with a certain objective function to be minimized. JSSP with j jobs to be processed on more than two machines have been classified as a combinatorial problem. They cannot be formulated as a linear programming and no simple rules or algorithms yield to optimal solutions in a short time [16].

The job-shop scheduling problem (JSSP) can be described as a set of n jobs denoted by J j where j =1, 2,..., n which have to be processed on a set of m machines denoted by Mk where k =1, 2,..., m. Operation of j th job on the k th machine will be denoted by Ojk with the processing time P jk [16]. Makespan is the time that needed to complete all the process. The objective is to determine the schedule which minimized the makespan.

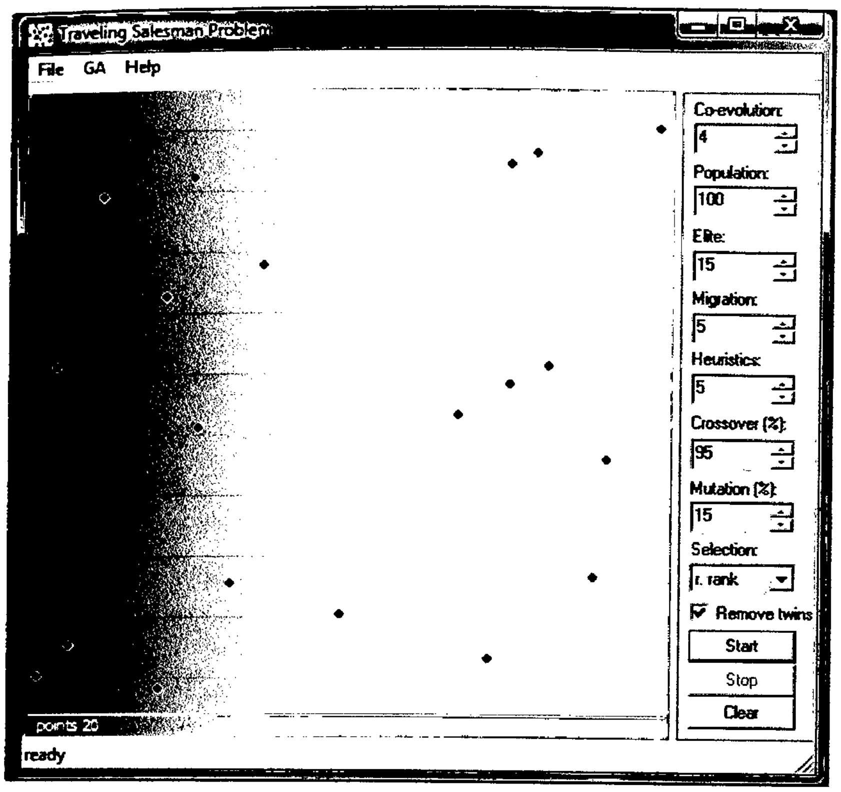
In 1976 Garey provided a proof that this problem is NP-complete for m>2, that is, no optimal solution can be computed in polynomial time for three or more machines [17].

2.4 Existing Genetic Algorithm Application

Three existing application using genetic algorithm will be briefly explain. These applications are Traveling Salesman Problem application, GA Class Schedule, and Chess Tournament Management System.

2.4.1 Travelling Salesman Problem Application

Travelling Salesman Problem (TSP) is one of the NP-hard problems. The problem is to find the shortest possible path, given N vertices so that each vertex is visited exactly once [18]. This TSP application is developed using Genetic Algorithm. Figure 2.4 shows the interface of the application.



Co-evolution: 4
Population: 100
Elite: 15
Migration: 5
Heuristics: 5
Crossover (%): 95
Mutation (%): 15
Selection: r. rank
☑ Remove twins
points 20

Figure 2.4: Interface of TSP Application

First, click on the gray part of the application to produce red dots. The information such as population, migration, percentage of crossover and percentage of mutation are initially set. User can change these information be they start to solve the problem. The result is shown below.

The shortest path to travel all the red dots and return to the starting point is shown by the blue line. Figure 2.5 shows the result if the shortest path.