A GENETIC APPROACH TO THE UNIVERSITY TIMETABLE PROBLEM

by

JOSHUA LETCHER

Advisors

LISA COULTER, JOSHUA ECKROTH

A senior research paper submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science
in the Department of Mathematics and Computer Science
in the College of Arts and Science
at Stetson University
DeLand, Florida

Spring Term
2016
ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. Coulter and Dr. Eckroth, for their support and guidance throughout the project.

I would also like to thank my girlfriend, Kathryn, for all of the love and support she gave me during the completion of the project.

Joshua Letcher
# TABLE OF CONTENTS

ACKNOWLEDGMENTS ........................................................................................................ ii

TABLE OF CONTENTS ...................................................................................................... iii

TABLE OF TABLES ............................................................................................................. iv

TABLE OF FIGURES ......................................................................................................... 1

ABSTRACT ......................................................................................................................... 2

1. Introduction .................................................................................................................. 3

2. Related Works ............................................................................................................. 7

3. MIP Approach ............................................................................................................. 8

   3.1. Model ..................................................................................................................... 8

   3.2. Results ................................................................................................................ 10

4. Genetic Approach ...................................................................................................... 12

   4.1. Genetic Model ................................................................................................... 12

   4.2. Results ............................................................................................................... 14

5. Comparing Results .................................................................................................... 15

6. Conclusion .................................................................................................................. 20

APPENDIX A. SOFTWARE ARCHITECTURE .................................................................. 21

References ....................................................................................................................... 23
TABLE OF TABLES

Table 1: Data Sets ..................................................................................................................................................... 15
# TABLE OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data Set 1 Results</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Data Set 2 Results</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>Data Set 3 Results</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>Data Set 4 Results</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Data Set 5 Results</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>Time Comparison</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>(Appendix A): UML Class Diagram</td>
<td>22</td>
</tr>
</tbody>
</table>
I propose two approaches to solving the University Timetabling problem. In the first approach, an optimal solution will be shown using Mixed Integer Programming (MIP). The second approach uses a genetic algorithm to ease the computational cost associated with the MIP approach. While the solutions given by the genetic approach may not be optimal, they are near optimal and much faster to find.
1. Introduction

Scheduling problems, such as the Job-Shop Scheduling Problem (JSSP), have been studied extensively by many researchers in the fields of both operations research and artificial intelligence [1]. As an example, the JSSP can briefly be described as follows:

Given a set of jobs and a set of machines, find a schedule of minimal time to complete all the jobs. Each job consists of a sequence of operations, each of which uses one of the machines for a fixed duration. Once started, the operation cannot be interrupted and each machine can process at most one operation at a time. A schedule is an assignment of operations to time intervals of the machines. [2]

Many different approaches have been used to solve scheduling problems in as optimal a way as possible, including genetic algorithms [3] [4] and a particle swarm optimization [5]. There are also many variations on the problem which add different constraints or adapt the problem to different situations.

One such adaptation of the problem is known as the University Timetabling Problem (UTP). This problem is a challenge that every university must face, scheduling course sections based on a limited number of rooms and professors, along with very tight constraints from different levels. This problem is especially interesting due to its sheer
size, which unlike similar problems such as the High School Scheduling Problem, makes an exhaustive search unfeasible [6].

There have been many attempts to develop an algorithm which can solve this problem for universities. The common goal among them seems to be, given the students’ requests for courses, schedule them in sections so that the students schedule has no conflicting classes and the sections are filled as evenly as possible [7]. The algorithms, such as the one by Stewart and Clark, do a fairly good job at meeting the objective, but require foreknowledge of the classes students wish to take, and don’t take into consideration the preferences of departments, students, or professors. Most of these algorithms don’t actually handle the schedule of the sections themselves, but rather assign students to a predefined schedule of the sections.

While these algorithms work decently for larger universities which can offer multiple sections of every course, they are less effective for smaller universities such as Stetson where many courses can only have one section offered per semester. In the past, if a Stetson student had a conflict in their schedule, they would need to contact the professor individually and request the section be offered at a different time. This would leave the professor with the task of either determining a new time that would work for them and the rest of the students, or in the hard spot of not being able to let the student with the schedule conflict take the course.
My goal is to create an algorithm which will use the graduation requirements of students in order to schedule sections in a way which minimizes conflicts in the students’ schedules. After scheduling the course sections, the algorithm will be able to provide each student with a “recommended schedule” to aid them in graduating on time. This will ease the stress of course registration time on both students and the professors.

For the purposes of this paper, the problem will be defined as follows:

Given a set of professors, a set of students, a set of course sections, a set of rooms, and a set of times, a list of requirements for each student, a list of courses that each professor can teach. Find a schedule for the course sections that maximizes the number of required courses being taken by students.

The following constraints will also be used while attempting to find a schedule:

1. No two courses can be in the same room at the same time.
2. Each professor can only be in one room at any given time.
3. Each student can only be in one room at any given time.
4. Each room can only hold its capacity of students at any time.
5. Each course section must be in one and only one room at one time.
6. Each room can only have one course in it at any given time.
7. Each course must have a professor.
8. A student can only be assigned a room if there is a course there at the same time.

9. Each professor can only teach classes in their department.

These constraints were chosen in order to provide a close to realistic model without overcomplicating the system.

Using the above problem statement and constraint, two approaches were devised to attempt to find a solution to the UTP. The first approach involved defining a mathematical model to represent the problem and constraints, then solving this model using the Gurobi solver. While this model works, due to the NP-Hardness of the problem the amount of time necessary to get an optimal solution using this method grows exponentially as larger data sets are applied to the problem. The second approach was a genetic algorithm in which schedules “breed” to produce a “child” schedule for a specified amount of generations in hopes of finding a good solution.
2. Related Works

Many approaches have been used to study the university timetabling problem. The MIP approach presented in this paper is not a new approach, but because each institution has its own individual requirements the model differs greatly from different works. Other integer programming solutions are typically broken into sub problems due to the size of the institution they were developed for [8].

More modern approaches to solving this problem involve the use of artificial intelligence. Mihaela Opera defined an architecture for multi-agent systems and evaluated the communication process that agents in such a system may have [9]. To my knowledge, the system was never implemented.

Genetic algorithms have also been used to solve similar problems to this one, however like the MIP approach, the genetic approach displayed in this paper differs from the other genetic approaches because they used a different problem definition.
3. MIP Approach

3.1. Model

The mathematical model to represent the problem can be defined as follows. Let:

- \( R \) be the set of rooms available
- \( S \) be the set of students
- \( T \) be the set of times available
- \( P \) be the set of professors
- \( C \) be the set of course sections

We will also define the following functions for the model:

\[ \text{S}(s_i, r_j, t_k) = 1 \text{ if student } i \text{ is in room } j \text{ at time } k, 0 \text{ otherwise} \]

\[ \text{C}(c_i, r_j, t_k) = 1 \text{ if course section } i \text{ is in room } j \text{ at time } k, 0 \text{ otherwise} \]

\[ \text{P}(p_i, r_j, t_k) = 1 \text{ if professor } i \text{ is in room } j \text{ at time } k, 0 \text{ otherwise} \]

\[ \text{Req}(s_i, c_j) = 1 \text{ if student } i \text{ needs to take course } j, 0 \text{ otherwise} \]

\[ Q(p_i, c_j) = 1 \text{ if professor } i \text{ can teach course } j \]

Using these variables and functions, the constraints for the problem can be defined as follows,
No two courses can be in the same room at the same time

\[ \forall r \in R, t \in T \sum_{c \in C} C(c, r, t) \leq 1 \]

Each professor can only be in one room at any given time

\[ \forall p \in P, t \in T \sum_{r \in R} P(p, r, t) \leq 1 \]

Each student can only be in one room at any given time

\[ \forall s \in S, t \in T \sum_{r \in R} S(s, r, t) \leq 1 \]

Each course section must be in one and only one room

\[ \forall c \in C \sum_{t \in T} \sum_{r \in R} C(c, r, t) = 1 \]

Each room can only have one course in it at any given time

\[ \forall r \in R, t \in T \sum_{c \in C} C(c, r, t) \leq 1 \]

Each course section must have a professor

\[ \forall r \in R, t \in T \sum_{c \in C} C(c, r, t) - \sum_{p \in P} P(p, r, t) = 0 \]

Each professor can only teach certain courses

\[ \forall p \in P, c \in C \sum_{t \in T} \sum_{r \in R} P(p, r, t)C(c, r, t) \leq Q(p, c) \]

A student can only be assigned a room if there is a course there at the same time

\[ \forall s \in S, t \in T, r \in R S(s, r, t) \leq \sum_{c \in C} C(c, r, t) \]

A professor must teach 3 courses

\[ \forall p \in P \sum_{r \in R} \sum_{t \in T} \sum_{c \in C} P(p, r, t)C(c, r, t) = 3 \]
A student must take 4 courses

\[ \forall s \in S \sum_{r \in R} \sum_{t \in T} \sum_{c \in C} S(s, r, t) C(c, r, t) = 4 \]

With the above definitions, the goal for the model can be represented by the following expression:

\[ \text{maximize} \sum_{r \in R} \sum_{t \in T} \sum_{s \in S} \sum_{c \in C} S(s, r, t) \times \text{Req}(s, c) \times C(c, r, t) \]

This tells the system to maximize the number of required courses being taken by students, which in turn minimizes the overlap of courses students need to take.

Using C#, this model was put into a format that the Gurobi solver can understand. Gurobi uses the branch and bound method in order to find an optimal solution for the given system.

### 3.2. Results

The MIP model was run with various data sets, and an optimal solution was found if and only if the constraints were able to be satisfied. In the case that the constraints could not be satisfied, the Gurobi solver output the constraints that were causing the problem. The data sets were kept fairly small in order to ensure enough time for a solution to be found. It is important to note that while Gurobi provides an optimal solution, there may be other equally optimal solutions.
Even at the small scale used for the purposes of this paper, the problem appears to be very computationally intensive. As seen in the tables below, even small changes in the data increase the time it takes to run exponentially. This is due, in part, to the NP-Hardness of the problem. While the actual amount of time will vary by computer, the growth seen will be consistent.
4. Genetic Approach

4.1. Genetic Model

One challenge of using a genetic approach is coming up with a model that works well for it. In many cases, a string of text (usually binary) is used to represent the model. This allows for easy manipulation to “evolve” the model from one state to the next in hopes of creating a better result. For this project, a couple different genetic models were attempted. The first try involved using a string to represent the data, the string was of the format.

First a header of the information:

{# of Rooms};{# of Times};{# of Courses};{# of Professors};{# of Students};

Then, for each course there was a block of the format, each course block was separated by a comma:

{# of Sections};{Section 1 Schedule}&…&{Section n Schedule}

Following the course blocks, there was a block for the professors:

{Professor 1 Schedule}&{Professor 2 Schedule}&…&{Professor n Schedule}

And finally, there was a block for the students:

{Student 1 Schedule}&{Student 2 Schedule}&…&{Student n Schedule}

For each block, a “Schedule” is a binary string where 1 indicates that the entity is in a room at a time, and 0 means that it is not.
This seemed like a good way to represent the data at first, but due to the complexity of the space many of the generated schedules were being dropped because they are invalid. Because so many schedules were being dropped in many runs the algorithm either failed to work, or took much longer than it should.

Due to the inefficiency of the string representation above, an object representation of the model was developed. In this representation, Schedules, Professors, Students, Courses, and Course Sections are all classes to represent their specific data. The Schedule class holds the most importance while the other classes are used mostly as data containers.

In both approaches, definitions also needed to be made for crossovers and mutations. A crossover is when parents are combined in a particular way in order to form a child. For this case, the crossover function takes two parents and returns a child where the schedules for each entity are randomly selected from one of the parents. This results in a fairly even combination of the schedules from both parents. A mutation is a genetic operator used to maintain diversity between generations. For the purpose of this project, a simple swap mutation was used. The mutation function first decides if it will mutate at all, and if it chooses to mutate it randomly selects a pair of schedules in the child and swaps them. This helps avoid local minima by preventing the population from becoming too similar.
4.2. Results

As in the MIP approach, the genetic approach was run several times on the same data sets. Even on the larger data sets tested, the genetic approach was usually able to find a good solution in under a second. In most cases, the good solution found by the genetic approach was actually an optimal solution.

Due to the random nature of the algorithm, there are some outliers seen that take much longer than the average case, but for the most part the time taken by the algorithm seems to be fairly consistent.
5. Comparing Results

Table 1 show the data sets that were used for testing the two algorithms. Both approaches were tested with the exact same data sets and use the same constraints to define what it means to be a valid schedule.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Courses</th>
<th>Sections</th>
<th>Rooms</th>
<th>Times</th>
<th>Professors</th>
<th>Students</th>
<th>Majors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>2</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>33</td>
<td>4</td>
<td>10</td>
<td>11</td>
<td>40</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1: Data Sets

Unfortunately, the MIP Approach crashed after 58 hours on the 5th data set. Before crashing, Gurobi had output a predicted objective value of 91 but this is not usable because the predictions are often off. It’s interesting to keep this in mind, but because it did not finish running, the results from the 5th data set for the MIP Approach will be ignored.
As seen in Figure 1, for the first data set the MIP approach consistently gave the optimal objective, while in the Genetic approach managed to get the optimal objective twice out of the five runs. The average run time for the Genetic approach was 0.86 seconds, while for the MIP approach it was 4.05 seconds, and the average objective for the Genetic approach was 34 while the average objective for the MIP approach was 40.

The following figures show that similar results were found for the other data sets. For Data Sets 2 and 4, the Genetic approach found an optimal solution on every test run in considerably less time than the MIP approach.
Figure 2: Data Set 2 Results

Figure 3: Data Set 3 Results
Figure 4: Data Set 4 Results

Figure 5: Data Set 5 Results
As seen in Figures 1-5, the Genetic approach found the same objective value as the MIP approach in most cases, but usually did so in much less time. When comparing the two approaches, it is also important to note the rate of change for how long it takes the approaches to find a solution. Figures 6 shows the average time to find a solution for both approaches over the data sets.

![Figure 6: Time Comparison](image)

Note that the value used for the MIP approach for data set 5 is made large to help show the growth, but not too large so that the other points remained visible. In actuality, the value would be greater than 209607.9871 seconds, as that is how long the algorithm was allowed to run before it crashed. Further investigation is needed into why the MIP approach was able to solve Data Set 3 faster than Data Set 2, as Data Set 3 appears more complex. From Figure 6, it is clear to see that as the complexity of the data grows, the time taken by the MIP approach grows much faster than the time taken by the Genetic approach.
6. Conclusion

Both the MIP approach and the Genetic approach appear to be good ways to solve the University Timetabling Problem. The MIP approach however, is very slow and thus computationally costly therefore it may be more practical, depending on the situation, to use the Genetic approach to solve the problem. As it has been shown that the Genetic approach does have a good chance of finding the optimal solution in a reasonable amount of time, the approach could be used to conduct “What-if” scenarios by university staff to quickly see how the schedule and objective would change with little tweaks to the parameters. Perhaps a combination of the two algorithms should be used, the Genetic approach could be used to help the university tune parameters such as number of rooms, times, professors, sections, etc., then the MIP approach to find the optimal solution with those parameters.

I am very interested in developing other approaches to solving this problem and perhaps tuning the Genetic approach more to see if the probability of obtaining an optimal result with it can be increased.
APPENDIX

APPENDIX A. SOFTWARE ARCHITECTURE
The architecture used for this paper is fairly simple. It is a 2 layered system, the top layer is the “runners” layer and the bottom layer is the “model” layer. The Strategy Pattern is used for controlling switching between different algorithms, and an Adapter object is used to facilitate communication between the different algorithms and the runners. Figure 7 shows a simple UML Class Diagram for the system.

Figure 7 (Appendix A): UML Class Diagram
References


