**Issues in Information Systems**
Volume 25, Issue 4 pp. 392-408, 2024

DOI: https://doi.org/10.48009/1_iis_2024_131

# A comparative study on solving university timetabling problems with emphasis on Genetic Algorithms

**Achini Herath,** *University of North Alabama, aherath@una.edu*
**Dawn Wilkins,** *University of Mississippi, dwilkins@cs.olemiss.edu*

**Abstract**

One of the basic requirements for the proper functioning of an institute is to have a conflict free timetable for its staff. Although mathematical programming could provide a high level of optimization in the generation of complex timetables, its forbiddingly high computational time discourages such an approach making approximation algorithms as the best alternative. One such method which gives satisfactory solutions is genetic algorithm (GA). This work is directed at finding an optimum solution to a general university timetable problem employing three-parent GA using tournament selection method and a new Uni-One-Point crossover technique. The results generated were compared with those obtained with several other search methods and found to be satisfactory enough to extend studies with more constraints to obtain quality timetables in the future.

**Keywords:** Genetic Algorithm, timetabling, evolution, three-parent crossover, Uni-One-Point crossover.

## Introduction

The non-availability of promising algorithms capable of exploring and exploiting all the possible solutions for real world NP-hard problems in the search for global maxima, lead scientists to develop efficient optimization algorithms to solve such problems. Such developments involve probing new and unexplored areas in the search space and exploiting selected small areas which hopefully lead to global maxima. To get quality solutions to problems related to science, engineering economics and business within an acceptable time frame, scientists have been developing heuristic and meta-heuristic algorithms (Suh, 2011). One such method, capable of yielding good results, is Genetic algorithms. It is a type of evolutionary algorithm which takes a meta-heuristic approach. The method needs limited information and carry-out the task by sacrificing completeness for speed. This nature inspired searching and optimization method involves techniques such as crossover, mutation and selection. However, since it is population dependent it is handicapped with longer computational time. It can solve problems associated with optimization, scheduling, economics, bioinformatics etc. reasonably well (Ausiello, 2013, Desale, 2015).

A well composed timetable provides accurate information about the allocation of rooms to staff and students of a given Institute during specific time periods. Constructing an acceptable timetable for a complex institute such as a university is a difficult task. Mathematical programming can provide a high level of optimization (global convergence) in the construction of complex timetables. However, the high computational time associated with this approach discourages its use (Moin, 2015) making approximation algorithms (Jones, 1999) as the next best option. One such method is metaheuristics. It includes genetic algorithm (GA) (Yaqin, 2010), discrete artificial bee colony (DABC) (Yin, 2011), tabu search (TS) (Zhang, 2007), and simulated annealing (SA) (Song, 2012).

Among these, GA is considered to be an able optimization method used to search for global maxima (Wan, 2013). It selects a pair of parents (chromosome pairs) to produce two offspring. The parents become part of the population as well. GA utilizes exploration and exploitation techniques. The operators for exploration constitute mutations whilst those for exploitations include the crossovers. The latter helps to converge the population to better solutions. It is important to maintain a balance between the rates of mutation. Whilst, too high rates prevent the population driving towards an optimum solution, due to the increase of the search area, too low rates prevent it reaching a global maximum by entangling in a local optimum, prematurely. It has been observed that when more than two parents are used, the evolutionary process get enhanced leading to favourable changes in the subsequent generation (Patel, 2012). This was demonstrated by the experiments conducted by Eiben and van Kemenade (Eiben, 1995). Their work showed significant gain with more than two parents especially with computationally less expensive diagonal crossover technique. Further, unimodal distribution crossover (UNDX) method which involves multi-parents has also shown impressive results (Elsayed, 2011).

In an earlier study, we utilized GA with two-parent systems to solve timetabling problems in educational institutes (Herath, 2016). Subsequently, studies were extended to improve the outcome using three-parents with the tournament selection and the uniform crossover methods.

The uniform three-parent crossover technique (Manning, 2013) involves the random selection of three parents followed by the comparison of each bit of the first parent with the corresponding one of the second parent. If they are the same it is being taken for the offspring. If it is different, the bit from the third parent is taken.

The results obtained were satisfactory enough for further investigation (Herath, 2020). The aim of the present investigation is to extend the studies adding a new Uni-One-Point crossover technique using tournament as the selection technique. The change yielded satisfactory results.

## Methodology

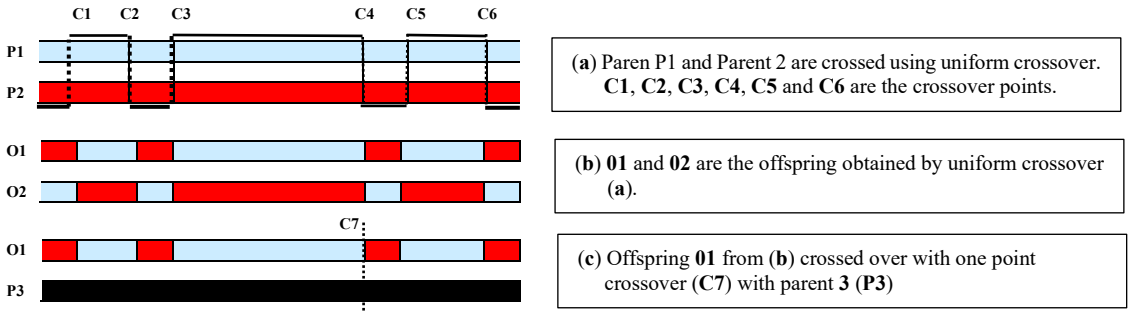
### Selection

The tournament selection method is used because it is efficient to code, work on parallel architectures and allows selection pressure to be changed with ease.

### Operators

### Crossover

A new Uni-One-Point crossover method using three parents is used in this investigation. At the outset, a uniform crossover is carried out between two parents and the resulting offspring is subjected to a one-point crossover with the third parent to obtain an offspring for the next generation (**Figure 1**).



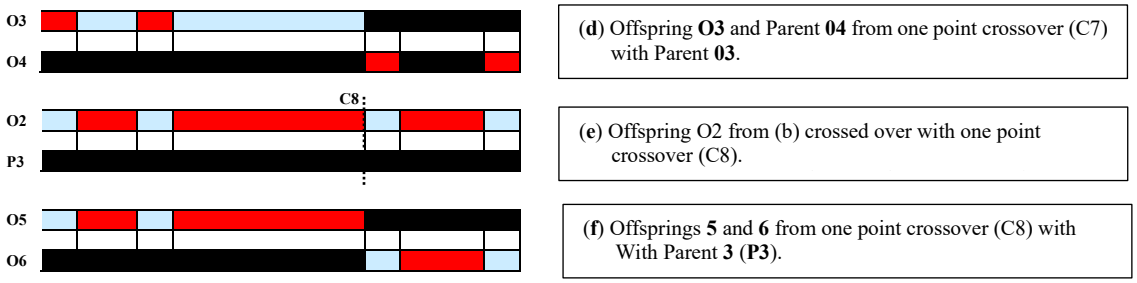



**Figure 1: Three parent Uni-One-Point Crossover (C8) with parent 3 (P3).**

**Mutation**

Mutation is carried out in a manner a new random valid individual is created. It is used to select genes to be copied into the individual who is to be mutated to maintain diversity. This is referred to as uniform mutation (Soni, 2014). It ensures that all the mutated individuals are valid.

## Testing Criteria:

The following criteria were tested using an API.
- tournament selection method.
- different crossover options (one-point, uniform, Uni-three-parent and Uni-One-Point crossovers).
- stopping condition (setting the number of generations).
- selecting the size of the population.
- fitness function.
- other parameters such as the mutation rate.

These criteria were tested using the following constraint satisfaction problems.
- graph coloring technique (Hindi, 2012).
- Travelling Salesman Problem (Abdoun, 2012).
- Sudoku (Douglas, 2014).

The tests were carried out to compare the different cross over methods, the number of generations, varying population size, and mutation rates.

**Chromosome class**.

It is an abstract class that describes the prototype functions set up by specific problems. It is composed of two fields. They are the encoded and the fitness fields. Whilst the former is used to encode the solution to the problem, the latter is used to cache the fitness of the solution. The fitness evaluation is horded as it will be called frequently during evaluation. This class constitutes four methods. One of them determines the fitness of the solution. Random initialization of the population is carried out by the second method. A new chromosome to wrap the encoded value is created by the third method. It is used in the crossover phase when offspring are formed from

parents' chromosomes. If the encoded value of the chromosome is modified, like in a mutation, the cached fitness value will not have any validity and will be cleared by the fourth method.

**Optimize GA class**

This class solves specific combination optimization problems with genetic algorithms. It has the fields to contain the population, selection type (Roulette, Tournament), the crossover type (Uniform, one Point, Uni-three parent, and Uni-one point), the mutation type (Mutate point, Switch point), the size of the elite portion of the population, size or the number of the chromosomes that are selected to enter the crossover phase, mutation rate, and a Boolean field that examines to see if some of values in chromosome are unique (for problems such as TSP).

The methods here are set to initialize the variables as well as to run the optimization to output the results stored in the population with their fitness value.

**Using the API to solve specific problems**

**Travelling Salesman problem**:

**Encode the solution:** Each solution is encoded with an array of integer having N elements, each element having value from 1 to N, the values of the elements need to be unique (no two elements have same value).

This array represents the indexes of the nodes through that order the salesman travels.

Example:
N = 8
Sample solutions:
(1, 2, 3, 8, 7, 6, 5, 4)
(4, 5, 6, 2, 3, 1, 7,8)

Note: The encoded value is actually a permutation of 1..N

Fitness evaluation

If D is the distance matrix of the graph (size of N+1 x N+1, since node 0 is also included), then fitness is computed by this formula:

$$(encoded) = [0,[0]] + D[encoded[0], encoded[1]] + \ldots + [[N-2], [N-1]] + [[N-1],0]$$

**Sudoku**

**Encode the solution:**

Each solution is encoded with an array of integer. There are 81 (9x9) squares in the matrix, however, some squares have been already filled with values at the beginning. Therefore, it only needs (81-M) number to encode the remaining blank squares, with M being the number of filled squares.

**Fitness evaluation**

The fitness of a solution is calculated as shown below.

- For each row, count the unique number in that row
- For each column, count the unique number in that column
- For each 3x3 block, count the unique number in that block

Sum all the counts from 9 rows, 9 columns and 9 blocks. This total sum is defined as the fitness of the solution. The best solution will need to have fitness of 243, which is (9*(9+9+9)).

**Graph Coloring Problem**

**Encode the solution**

A solution is encoded with an array of integer, each element being the color index of each node in the graph.

**Fitness evaluation**

The fitness of a solution is calculated as shown below (Hindi, 2012).

Count the number of illegal edges: Edges that have two nodes painted with same color, and define this value as *m*

- Count the number of unique colors in the graph and define this number as *c*
- Fitness is defined as follow:

$$fitness = 1/(m + c/(c + 1))$$

We choose the above formula because for the following reasons:

- In a legal solution, *m* will be 0, in an illegal solution *m* is bigger than or equal to 1, therefore any legal solution will have fitness larger than 1, while any illegal solution will have fitness smaller than 1.
- Among legal solutions, those with fewer colors will have better fitness.

**University time table Problem**

A university needs to create a time table for a semester. The university has a set of rooms; each room has a capacity for a maximum number of students that can participate, a set of instructors, a set of classes (which have been assigned to each instructor beforehand). Each class has been populated with a number of students and can take place during a specific time slot in a set of time slot during the week.

To do: Assign each class to a time slot and room so that no instructor has conflict schedules (having to teach two class in the same time slot)

Here are the inputs of the problem:

- Instructors: Set of instructors, each with
    - id: Staff Id
        - name: Name
- Rooms: Set of rooms, each with
    - name: Room number
        - capacity: Number of seat
- TimeSlots: Set of TimeSlots, each with
    - id: Identity of each slot
        - startHour: start hour
        - startMinute: start minute
        - endHour: end hour
        - endMinute: end minute
        - dayOfWeek: day of week
- Class: Set of classes, each with
    - name: Name of the class
        - instructorId: id of the instructor that will teach this class ○ numberOfStudent: Number of registered student

The output of the problem:

- TimeSlotId for each class
- Room number of each class

Problem Definition:

The problem definition for the university timetable is as follows.

The hard constraints for the timetable are:

H1 - a student can be in only one class at any given time.

H2 – the size of the classroom can accommodate all the students.

H3 – an instructor can teach only one class at any given time.

The soft constraints for the timetable are as follows. These violations are penalized according to the number of violations.

S1 - an instructor's preference for a specific time slot.

S2 - an instructor's preference for a specific classroom.

An event (course, instructor) is assigned to a timeslot as well as a number of resources (students, classrooms). This is done in such way as to avoid any clashes between the classrooms, timeslots and events.

As stated by Rossi-Doria et al. (Rossi-Doria, 2003, Blum, 2002), the university timetable comprises of a set of $n$ events (classes, courses ) $E = \{e_1,e_2,...,e_n\}$ to be organized into a set of $p$ time slots $T = \{t_1,t_2,...,t_p\}$, a set of $m$ available rooms $R = \{r_1,r_2,...,r_m\}$ in which events can take place, a set of $k$ students $S = \{s_1,s_2,...,s_k\}$ who are available for the events, and a set of $l$ available features $F = \{f_1,f_2,...,f_l\}$ that are fulfilled by rooms and made essential by events.

The solution to the timetable can be represented in the form of an ordered list of pairs $(r_i,t_i)$, of which the index of each set is the identification number of an event $e_i \in E$ ($i = 1,2,\cdots,n$). For example, the time slots and rooms are assigned to events in an ordered list of sets:

$$(1,5),\ (4,10),\ (2,12),\ \cdots,\ (3,7)$$

where room 1 and time slot 5 are assigned to event 1, room 4 and time slot 10 are assigned to event 2, and so on.

The object of the timetable is to decrease the soft constraint violations of a possible solution (a possible solution is a timetable that has no hard constraint violations). The objective function $f(s)$ for a timetable $s$ is the weighted total of the number of hard constraint violations $H_{CV}$ and soft constraint violations $S_{CV}$.

$$f(s) := H_{CV}(s) * C + S_{CV}(s)$$

where $C$ is a constant. The value of C is greater than the largest possible number of $S_{CV}$. As a result, when C is set to a value ($C = 10^6$) and the result of $f(s)$is $10^6$ or higher, then the solution is rejected.

Encode the solution:

To find the *timeSlotId* and *roomIndex* for each class, so there will be 2 arrays of integers. However, it is noted that two classes may have the same *timeSlotId* or may have same *roomIndex*, but they cannot have both the same *timeSlotId* and *roomIndex*. It is reasonable to convert the pair (*slotId, roomIndex*) to a unique integer by this formula:

$$mergeIndex = roomIndex * timeSlots.length + timeSlotId$$

So each class will have a unique *mergeIndex*. The solution will be an array of integer with *nClass* element, each element having value form 1 to *timeSlots.length*rooms.length*.

In other words, a solution is a combination *C(N,k)* with *N = timeSlots.length*rooms.length* and *k=nClass.*

Fitness evaluation

For each instructor, all of his/her classes are collected, that have been allocated with an adequate room (a room that can accommodate all registered students), and then collect the *timeSlotId* from all of these classes. Then count the number of unique *timeSlotId* from this set.

Then, sum the count of unique *timeSlotId* for all instructors, and define that this is the fitness.

It is easy to see that a legal time table will have fitness that is equal to the number of classes.

University time table with soft constraints.

In the University time table problem, it is likely that we can find several solutions that satisfy the hard constraint (no violation on room size & conflict time slot for each instructor). Among these feasible solutions, we can pick out the best preferable solution by adding a soft constraint to the problem. Here, the soft constraint selected was the preferability of the instructors over some specific time slots (i.e. each instructor may prefer to teach at specific hours of day and specific days of week), and prefer specific rooms to teach in (i.e. each instructor may prefer to teach at some specific location).

First, in order to update the input of the problem, each instructor will have an extra field named

"preferedTimeSlotIds" to indicate the id of time slots on which he prefers to teach, and another field named "preferedRoomIds" to indicate the id of the rooms at which he prefers to teach in.

The next step is to define the fitness of the problem. This can be inspired from the Graph coloring problem (in which there are two constraints: The hard constraint is no violated edges, and the soft constraint is to minimize number of colors). To define the fitness for the Timetable problem with soft constraints, the following formula can be used.

$$fitness = 1/(m + c/(c + 1))$$

With *m* being the number of violated class (Classes that cannot be assigned with an appropriate room and time slot) and *c* being the number of class with unpreferable time slot/room (classes assigned with a time slot/room which is not preferred by the instructor).

By defining the fitness above, it is noticed that all feasible solutions will have fitness larger than 1 (as m=0), while all infeasible solutions will have fitness smaller than 1 (as m > 0).

In the ideal case, both *m* and *c* are equal to 0, the fitness will become infinity. To prevent this, a small adjustment can be made by making *c* equal to the number of unpreferable class plus 1. With this adjustment, the maximum fitness is 2.0 (when there is no class violation and all instructors' preferability is satisfied).

**Experiments and Results**

The under described experiments were conducted to test the efficiency of our new Uni-One-Point-Crossover (UOPC) procedure with other crossover operators [Uniform (UC), One-point (OPC) and Uni-Three-parent (U3PC)] using Tournament (TS) as the selection technique by comparing the results obtained for the Time Table Schedule (With Hard and Soft constraints) (TTS), the Traveling Salesman (TS) and the Graph Colouring (GC) problems. For each crossover method ten runs were executed. The data were averaged. The population size, crossover size, mutation rate and the total number of generations were kept constant in all the experiments.

**Time Table (Tables, graphs, 160, 200 classes)**

**With hard constraints**

A time schedule problem for a university was generated manually using the parameters, (1) number of time slots: 10 (5 days a week x 2 slots a day), (2) number of rooms: 8, (3) number of instructors: 10 and (4) Number of classes: 40 (each instructor has to teach 4 classes a week).

The number of classes that were effectively assigned with time slots and suitable rooms comprises the score to compare. The total number of classes is considered to be the best score.

The outcome of the crossover methods is shown in the following table (**Table 1**) and bar diagram (**Figure 2**).

**Table 1: Results of Timetable Schedule (Hard constraints).**

| | Crossover Type | Tournament selection |
|---|---|---|
| **1** | Uniform Crossover | 38.9 |
| **2** | Onepoint Crossover | **39.3** |
| **3** | Uni-OnePoint Crossover | 38.9 |
| **4** | Uni-three-parent Crossover | 38.3 |

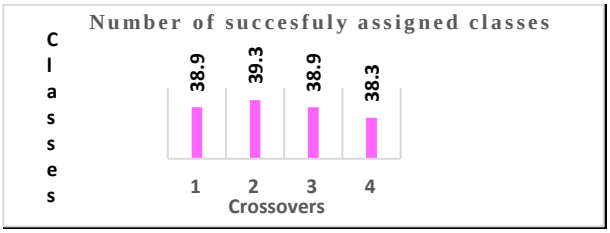



**Figure-2: Number of successfully assigned classes with different crossovers. (Higher is better)**

**Table 1** and **Figure 2** show that the number of successfully assigned classes is similar for all crossover techniques and the tournament selection method used in this experiment. The preferred highest value **39.3** is for the one-point crossover operator.

**With soft constraints**

Some soft constraints were introduced by allowing each instructor to pick 4 time slots and 4 rooms to his or her liking. The number of un-preferred classes is the score to compare. Thus, the best score is 0. **Table 2**and **Figure 3** give the results for all the crossover methods.

**Table 2: Results of Timetable Schedule (Soft constraints)**

| | Crossover Type | Tournament selection |
|---|---|---|
| **1** | Uniform Crossover | 18.7 |
| **2** | One-point Crossover | 19.3 |
| **3** | Uni-One-Point Crossover | 19.4 |
| **4** | Uni-three-parent Crossover | **17.7** |

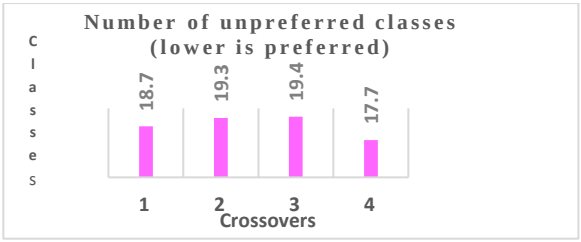



**Figure-3: Number of un-preferred classes with different Selection and Crossover methods (lower is better).**

As with hard constraints, the results for the soft constraints too are similar with all the crossover techniques used. The preferred lowest value for the Uni-three-parent crossover with the tournament selection procedure is **17.7**.

A program to further evaluate the performance of different crossover methods with the tournament selection procedure was initiated to create four bench mark problems. They were two hard-constraint problems and two soft-constraint problems. The hard-constraint problems were 1. A medium size problem with 160 classes and 2. A large size problem with 200 classes.

These problems are associated with 40 and 50 instructors, respectively. Each instructor takes 4 classes a week and each class is held during one the 10 available time slots for the week (2 time slots a day: 5 days a week). The number of rooms needed to conduct classes is determined by dividing the available number of rooms by the number of time slots. Since, the available number of rooms are limited, getting at a feasible solution without violating any constraints becomes a tedious task. In here, the number of classes assigned (higher the better) indicates how successful the performance was with regards to hard constraints.

The soft-constraint problems also included, (1) A medium size problem with 160 classes and (2) A large size problem with 200 classes. To find an acceptable solution with ease, the number of available rooms was doubled compared to the hard constraint problems. However, the performance was evaluated by the soft-constraint violations (the lower the better). It is necessary that the resulting solution has to satisfy all the hard constraints before it could be subjected to a performance evaluation.

It was also decided at this point to compare the performance of GA with other combinatorial optimization methods which may be of help to find better solutions to timetabling problems. The methods included, (1) Particle Swarm Optimization (PSO) (Thangaraj, 2011), (2) Simulated Annealing (SA). (Russel, 1995, Abramson, 1991, Fredrikson, 2016, Rosocha, 2015, Varty, 2017. Brownlee, 2011, Kirkpatrick, 1983, Chmait, 2013, Patrick, 2012) and Tabu Search (TS) (Burke. 1997). The results of GA were compared with each other as well as with the above mentioned optimization techniques. The performance was tested using 10 runs and the results were averaged.

Hard constraint problem results: Case 1: Timetable for 160 classes.

**Table-3: Results of successfully assigned classes for a medium sized classes with hard constraints, Particle Swarm Optimization, Simulated Annealing and Tabu.**

| | Crossover Type | Tournament selection | Results |
|---|---|---|---|
| **1** | Particle swarm | | 112.2 |
| **2** | Simulated annealing | | 111.1 |
| **3** | Tabu search | | **153.0** |
| **4** | Uniform Crossover | 120.5 | |
| **5** | One-Point Crossover | **147.6** | |
| **6** | Uni-One-Point Crossover | 146.6 | |
| **7** | Uni-three-parent Crossover | 141.6 | |
| | | | |

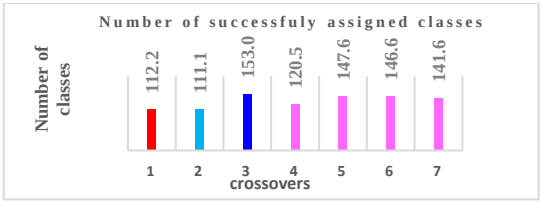



**Figure-4: Number of successfully assigned classes for a medium size class with hard constraints, Particle Swarm Optimization and Simulated Annealing and Tabu (higher the better).**

Here, the highest value of **147.6** is obtained with one-point crossover method. The values obtained for PSO and SA were 112.2 and 111.1 respectively. **Figure 4** shows the results when the number of classes is increased to 160.

**Case 2: Timetable for 200 classes**.
The table (**Table 4**) and the corresponding bar diagram (**Figure 5**) for **Case 2** are given below.

**Table-4: Results of successfully assigned classes for a large sized classes with hard constraints, Particle Swarm Optimization, Simulated Annealing and Tabu.**

| | Crossover Type | Tournament selection | Results |
|---|---|---|---|
| **1** | Particle swarm | | 132.2 |
| **2** | Simulated annealing | | 129.2 |
| **3** | Tabu search | | **181.6** |
| **4** | Uniform Crossover | 142.125 | |
| **5** | One-Point Crossover | 174.25 | |
| **6** | Uni-One-Point Crossover | **176.125** | |
| **7** | Uni-three-parent Crossover | 163.375 | |

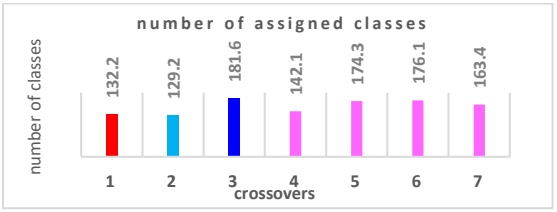



**Figure-5: Number of successfully assigned classes for large size classes using hard constraints, Particle Swarm Optimization, and Simulated Annealing, and Tabu (higher the better).**

The preferred highest value of **176.125** in here was shown for the Uni-One-Point crossover and tournament selection combination. The results for PSO and SA were 132.2 and 129.2 respectively.

**Soft constraint problem results**

**Case 1: Timetable for 160 classes.**

The results when the number of classes was 160 are shown in the table (**Table 5**) and the bar diagram (**Figure 6**).

**Table-5: Number of un-preferred classes assigned for a medium size class with soft constraints, Particle Swarm Optimization Simulated Annealing and Tabu.**

| | Crossover Type | Tournament selection | Results |
|---|---|---|---|
| **1** | Particle swarm | | No acceptable results |
| **2** | Simulated annealing | | No acceptable results |
| **3** | Tabu search | | 13.7 |
| **4** | Uniform Crossover | No acceptable results | |
| **5** | One-Point Crossover | **12.75** | |
| **6** | Uni-One-Point Crossover | 23 | |
| **7** | Uni-three-parent Crossover | 33.375 | |

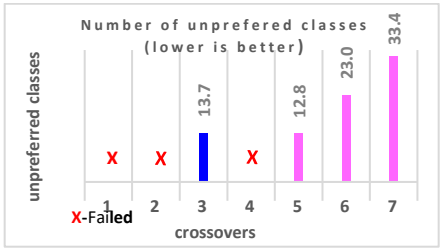



**Figure-6: Number of un-preferred classes with tournament Selection and Crossover methods using soft constraints, Particle Swarm Optimization and Simulated Annealing and Tabu (lower is better).**

The preferred lowest value here is **12.75** with the one-point crossover, tournament selection combination. Both PSO and SA showed a result of 0 (no acceptable results).

**Case 2: Timetable for 200 classes.**

**Table-6: Results of un-preferred classes assigned for a large size class with soft constraints, Particle Swarm Optimization and Simulated Annealing.**

| | Crossover Type | Tournament selection | Results |
|---|---|---|---|
| 1 | Particle swarm (PSO) | | No acceptable results |
| 2 | Simulated annealing (SA) | | No acceptable results |
| 3 | Uniform Crossover | No acceptable results | |
| 4 | One-Point Crossover | 38.3 | |
| 5 | Uni-One-Point Crossover | 97.5 | |
| 6 | Uni-three-parent Crossover | 56.1 | |

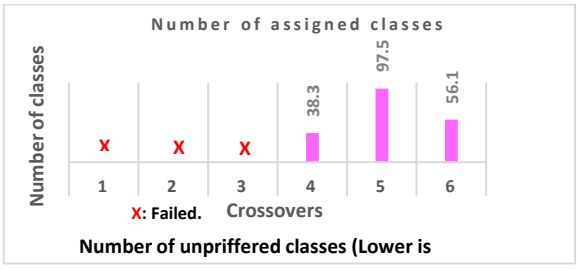



**Figure-7: Number of un-preferred classes with different Selection and Crossover methods, using soft constraints, Particle Swarm Optimization and Simulated Annealing.**

The uniform crossover/tournament combination along with PSO and SAGA were unable to find a feasible solution (one that satisfied all the hard constraints). As such, their soft constraints fitness was not considered.

### Traveling Salesman

The input data were selected from the sgb128_dist dataset. It contains a 128X128 matrix of city-city distance. The total distance the salesman has to travel is the score to compare. The best solution is the one with the lowest score. **Table-7** and **Figure-8** show the results of different crossover methods.

**Table 7: Results of Traveling Salesman Problem.**

| | Crossover Type | Tournament selection |
|---|---|---|
| 1 | Uniform Crossover | 90040.2 |
| 2 | One-Point Crossover | 75927.4 |
| 3 | Uni-One-Point Crossover (UOPC) | 63980 |
| 4 | Uni-three-parent Crossover (U3PC) | 104543.2 |

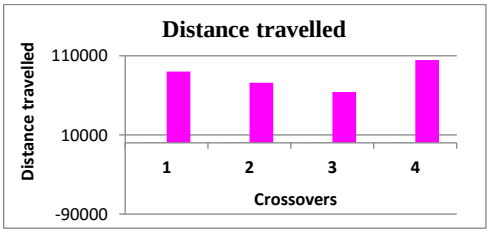



**Figure-8: Distance travelled with different Selection and Crossover methods.**

For economic reasons institutes are in search of methods that help to find the least distance a salesman has to travel to a given destination without repletion. This experiment was carried out to check the validity of a genetic operator (U3PC) / tournament selection combination to solve TS and few other problems prior to using it to solve the university timetabling problem. The investigation was extended to cover other frequently used crossover operators. The results tabulated in **Table-7** accompanied by the bar diagram (**Figure-8**) suggest that the UOPC /tournament combination gives lowest distance value (**63980**) compared to the rest of the selection methods.

## Graph Colouring

The required input data were taken from the queen 16_16 dataset. It contained a graph having 256 vertices and 12640 edges. The number of colors is the score to compare and the lowest value represents the best solution. The result of different crossover operators and the tournament selection method are shown in the table (Table-8) and the graph (**Figure-9**).

**Table 8: Results of Graph Colouring Method. (minimum number of colours-lower is better).**

| | Crossover Type | Tournament selection |
|---|---|---|
| 1 | Uniform Crossover (UC) | 92.7 |
| 2 | One-Point Crossover | 101.6 |
| 3 | Uni-One-Point Crossover (UOPC) | 93.6 |
| 4 | Uni-three-parent Crossover | 156.6 |

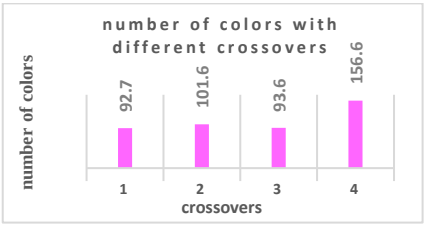



**Figure 9: Number of colours with crossover operators and the tourn selection method.**

The aim of this coloring method is to utilize the minimum number of colors to solve problems. The results of this experiment are tabulated in table (**Table-8**) and illustrated in **Figure 9**. The results indicate that the Uni-three-parent crossover /tournament combination gives the highest value (156.6) making the method unsuitable. The lowest number of colors (**92.7**) is given by the uniform crossover-tournament combination making it the best option among the rest.

## Sudoku

As input data, a sample sudoku quiz from https://sudoku.com/ was used. The sum of number of unique digits in 9 rows, 9 columns and 9 blocks was the score to compare. The best score to archive would be 9x9x3 = 243. Result of tournament selection/crossover methods are shown in **Table 9** and graph (**Figure 10**).

**Table 9: Results of Sudoku method.**

| | Crossover Type | Tournament selection |
|---|---|---|
| **1** | Uniform Crossover | **240.6** |
| **2** | One-point Crossover | 236.5 |
| **3** | Uni-One-Point Crossover | 239.4 |
| **4** | Uni-three-parent Crossover | 233.3 |

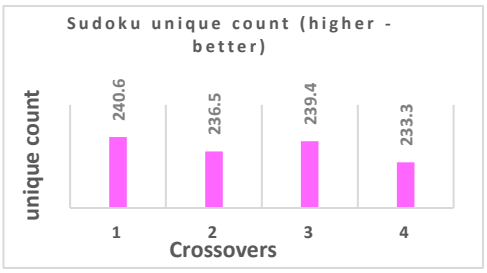



**Figure 10: Sudoku unique count with tournament selection and different crossover methods.**

In solving the Sodoku problem the uniform crossover with tournament selection combination gives the highest value (**240.6**) among others.

## Conclusion

The present work involves the development of optimization algorithms to obtain solutions to timetabling problems. A literature survey conducted by us indicated that genetic algorithms could yield promising results. We were able to repeat some of this work and decided to use varying crossover methods with the tournament selection technique. It was also decided to check the validity of this procedure by applying the procedure to other problems as well. A new Uni-One-Point crossover technique was also introduced into this work.

With the timetabling problem, which is our main topic in this investigation, the results obtained with all the genetic algorithm techniques (crossover/tournament selection) showed no drastic variations. They were comparable. The preferred highest value **39.3** was for the one-point crossover operator. With hard constraints some improvements were observed due to the increase of search area.

Some soft constraints were introduced by allowing each instructor to pick 4 time slots and 4 rooms to his or her liking. The number of un-preferred classes is the score to compare. Thus, the best score is 0.

As with hard constraints, the results for the soft constraints too were similar in all selected operators and the crossover techniques used. The preferred lowest values for the Uni-three-parent crossover and uniform crossover operators with tournament selection method were **17.7** and **18.7**.

With a reasonably large number of classes the Uni-One-Point crossover with tournament selection gave better results for hard constraints.

The PSO and SA methods showed no impressive results when hard constraints were considered. With soft constraints it is noted that PSO, SA and GA with uniform cross over failed to find a feasible solution (one that satisfied all the hard constraints), thus their soft-constraint fitness was not considered. As future work, it might be worthwhile to change the parameters to obtain the desired results.

It was also decided to check the validity of this procedure by applying it to other problems. One such application was to find the shortest possible distance travelled in the travelling salesman

problem. The results obtained in this investigation are shown in **Table-7** and **Figure-8**. Among them the lowest distance of 63980 was obtained with the Uni-One-Point crossover/tournament selection combination. The uniform crossover and Uni-three-parent crossover gave the forbidden higher values.

With respect to the graph colouring problem, acceptable minimum number of colors was obtained with all the crossover operators except for the Uni-One-Point method. The results are as tabulated in **Table-8** (**Figure-9**). The lowest score of **92.7** was reported with the uniform crossover/tournament combination.

With regards to Sudoku, comparable results showing the desired higher scores were obtained with all the techniques. However, uniform crossover yielded a slightly higher score of **240.6** (**Table-9**, **Figure-10**).

The results obtained show that it may be worthwhile pursuing additional work by changing factors like the population size, operators and combining evaluation methods. Hybrid methods with genetic algorithms combined with other various optimization approaches using the newly adapted crossover and evaluation methods used in this study would very likely produce desired solutions for the scheduling problems.

## References

Abdoun, O., Abouchabaka, J. & Tajani, C., (2012). Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem. *IJES Int. J. Emerg. Sci.*, 2, 61–77.

Abramson, D., (1991). Constructing school timetables using simulated annealing: sequential and parallel Algorithms, *Management Science*, 37(1), 98–113.

Ausiello, G., Petreschi, R. (ed.) (2013). The Power of Algorithms: Inspiration and Examples in Everyday Life, Allesandro Panconesi, *The One Million Dollars Problems*, p. 72, Springer Science & Business Media. ISBN 3642396526.

Blum, C., Correia, S., Dorigo, M., Paechter, B., Rossi-Doria, O., & Snoek, M. (2002). A GA Evolving Instructions for a Timetable Builder. In E. Burke, & P. De Causmaecker (Eds.), *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling* (PATAT 2002), 120-123). Gent, Belgium: KaHo SintLieven.

Brownlee, J., (2011). Clever Algorithms. Nature-Inspired Programming Recipes, 1st Ed., Lulu Enterprises, Australia, 436.

Burke, E. K., Jackson, S., Kingston, H., & Weare, F. (1997). Automated Timetabling: The State of the Art. *The Computer Journal*. 40(9) 565-571.

Chmait, N., Challita, K., (2013). Using simulated annealing and ant colony optimization algorithms to solve the scheduling problem. *Computer Science and Information Technology*, 1(3), 208–224.

Desale, S., Rasool, A., Andhale, S., & Rane, P. (2015). Heuristic and Meta-Heuristic Algorithms and Their Relevance to the Real World: A Survey, *International Journal of Computer Engineering in Research Trends*, 2(5), 296-304. ISSN: 2349-7084.

Douglas, R., (2014). Solving Sudoku puzzles with genetic", *Game Behaviour*,1(1) (2014). *@inproceedings{Douglas2014SolvingSP*

Eiben, A. E. & van Kemenade, C. H. M. (1995) Performance of multi-parent crossover operators on numerical function optimization problems. *Technical Report TR-95-33, Leiden University*; available from http://www.liacs.nl/TechRep/1995/.

Elsayed, S. M., Sarker, R. A. & Essam, D. L. (2011). GA with a new multi-parent crossover for solving IEEE-CEC2011 competition problems. *In proceedings of IEEE Congress on Evolutionary Computation*, 1034-1040.

Fredrikson R., & Dahl, J. (2016). A comparative study between a simulated annealing and a genetic algorithm for solving a university timetabling problem, Tech. rep. Stockholm, Sweden: KTH, School of Computer Science and Communication (CSC), 45.

Herath, A. K. (2016). Genetic algorithm for university," MSc. Thesis, Dept. Comp. Inform, Sc., Univ. Mississippi, USA.

Herath, A. K. (2020). Genetic algorithm for university, PhD. thesis, *Dept. Comp. Inform, Sc., Univ*. Mississippi, USA.

Hindi, M., & Yampolskiy, R. V., (2012). Genetic Algorithm Applied to the Graph Coloring Problem. Paper presented at the *Twenty-third Midwest Artificial Intelligence and Cognitive Science Conference* (MAICS); April 21-22, Cincinnati, OH.

Jones, A. Rabelo, L. C., & Sharawi, A. T. (1999). Survey of job shop scheduling techniques, in *Wiley Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, New York, NY, USA.

Kirkpatrick S., Gelatt, C. D., & Vecchi, M. P., (1983) Optimization by simulated annealing. *Science*, 220, 671–680.

Manning, T., Sleator, R. D. & Walsh, P. (2013) Naturally selecting solutions: the use of genetic algorithms in bioinformatics. *Bioengineered*, *International Journal of Computer Science and Information Technologies (IJCSIT)*, 4(5), 266–278.

Moin, N. H. Chung Sin O., & Omar, M (2015). Hybrid Genetic Algorithm with Multiparents Crossover for Job Shop Scheduling Problems, *Math. Probl. Eng*., 12, 01-12. https://doi.org/10.1155/2015/210680.

Patel, R., & Raghuwanshi, M. M. (2012). Multi-objective Optimization Using Multi Parent Crossover Operators, J. *Journal of Emerging Trends in Computing and Information Sciences*, 2(2), 33-39.

Patrick, K., (2012). Comparison of simulated annealing and hill climbing in the course timetabling problem, *African Journal of Mathematics and Computer Science Research,* 5(11), 176-178.

Rosocha, L., Vernerova, S. & Verner, R., (2015). Medical Staff Scheduling Using Simulated Annealing. Quality Innovation Prosperity, 19(1), 1–11. Available at: http://www.qip-journal.eu/index.php/QIP/article/view/405.

Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L. & St¨utzle, T. (2003). A comparison of the performance of different metaheuristics on the timetabling problem," in *Proc. 4th Int. Conf. Pract.Theory Automated Timetabling* (Lecture Notes in Computer Science), 2740, 329–351.

Russell, S. J. & Norvig, P. (1995). Artificial Intelligence: A Modern Approach, Pearson Education, 2 edition, *Prentice Hall, Englewood Cliffs, New Jersey 07632*. ISBN D-IH-IQBSOS-E.

Song, S. Z., Ren, J. J. & Fan, J. X. (2012). Improved simulated annealing algorithm used for job shop scheduling problems in *Advances in Electrical Engineering and Automation*, vol. 139, pp.17–25, Springer, Berlin, Germany.

Soni, N. & Kumar, T. (2014). Study of Various Mutation Operators in Genetic Algorithms. *Int. j. comput. sci. inf. technol*., 5(3), 4519–4521.

Suh, W. J. Park, C. S. & Kim, D. W. (2011). Heuristic vs. meta-heuristic optimization for energy performance of a post office building. *Proceedings of the 12th conference of international building performance simulation association*, Sydney, Australia: IBPSA., 704-711. ; Zabidee, F. & Adnan, M. H. M. (2024). Optimization in university student timetables: A comprehensive literature review, *Journal of Advanced Research in Applied Sciences and Engineering Technology*, 41(1), 14-43.

Thangaraj, R., Pant, M., Abraham, A., & Bouvry, P. (2011). Particle swarm optimization: hybridization perspectives and experimental illustrations, *Applied Mathematics and Computation*, 217, 5208–5226.

Varty, Z., (2017). Simulated Annealing Overview. Retrieved from: http://www.lancaster.ac.uk/~varty/RTOne.pdf.

Wan, W., & Birch, J. B. (2013). An Improved Hybrid Genetic Algorithm with a New Local Search Procedure, *J. Appl. Math.*, 2013, 1-10. http://dx.doi.org/10.1155/2013/103591

Yaqin, Z., Beizhi, L., & Lv, W. (2010). Study on job-shop scheduling with multi-objectives based on genetic algorithms. *International Conference on Computer Application and System Modeling (ICCASM '10)*, 10294–v10298, IEEE.

Yin, M., Li, X., & Zhou, J. (2011). An efficient job shop scheduling algorithm based on artificial bee colony. *Scientific Research and Essays*, 6(12), 2578–2596.

Zhang, C., Li, P., Guan, Z., & Rao, Y., (2007). A tabu search algorithm with a new neighbourhood structure for the job with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11), 3229–3242.