

A Comparative Analysis On  
Solving University Departmental Course Allocation Problem  
Using AI Optimization Algorithms

by

Shafqat Hasan

20241046

Diby Fabian Dofadar

17101407

Riyo Hayat Khan

16201062

Towshik Anam Taj

20241045

A thesis submitted to the Department of Computer Science and Engineering  
in partial fulfillment of the requirements for the degree of  
B.Sc. in Computer Science

Department of Computer Science and Engineering  
Brac University  
January 2021

© 2021. Brac University  
All rights reserved.

# Declaration

It is hereby declared that


1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

**Student's Full Name & Signature:**



---

Shafqat Hasan  
20241046



---

Dibyo Fabian Dofadar  
17101407



---

Riyo Hayat Khan  
16201062



---

Towshik Anam Taj  
20241045

# Approval

The thesis titled “A Comparative Analysis On Solving University Departmental Course Allocation Problem Using AI Optimization Algorithms” submitted by

1. Shafqat Hasan (20241046)
2. Dibyo Fabian Dofadar (17101407)
3. Riyo Hayat Khan (16201062)
4. Towshik Anam Taj (20241045)

Of Fall, 2020 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 15, 2021.

## Examining Committee:

Supervisor:  
(Member)

---

Mahbubul Alam Majumdar  
Professor and Dean, School of Data and Sciences  
Brac University

Program Coordinator:  
(Member)



---

Md. Golam Rabiul Alam  
Associate Professor  
Department of Computer Science and Engineering  
Brac University

Head of Department:  
(Chair)

---

Sadia Hamid Kazi  
Assistant Professor and Deputy Head  
Department of Computer Science and Engineering  
Brac University

## Abstract

This paper discusses about various types of constraints, regulations, difficulties and solutions to overcome the challenges regarding university departmental course allocation problem. A CSP solver algorithm, Genetic Algorithm, Simulated Annealing and a hybrid of Genetic Algorithm and Simulated Annealing has been used separately to generate the best course assignment and also to compare the results generated by these four algorithms. The Department of Computer Science and Engineering of BRAC University has been used as a case study to discover the scope of automation in this research. After analyzing the information gathered from the department itself, some constraints were formulated. These constraints manage to cover all the aspects needed to be kept in mind while preparing a class schedule for a faculty member without any clashes. The goal is to generate optimized solution(s) which will fulfill those constraints. At this point, the main focus is on the perspective of the faculty members but in the near future, there will be enough opportunities for expansions, like focusing on the lab change procedure of the students, assignment of student tutors and many more.

**Keywords:** Course Allocation; CSP Solver; Genetic Algorithm; Simulated Annealing; Hybrid Algorithm.

## Dedication

We dedicate this work to our respected parents, and also to the members of the routine making committee of our department, who spend a lot of time in the course scheduling task for the whole department. Our work seeks to take the burden off of their shoulders.

## **Acknowledgement**

First and foremost, all praise to the Almighty. We would like to thank our supervisor, Dr. Mahbubul Alam Majumdar sir, for agreeing to take us as his thesis students. Not to mention our co-supervisor Arif Shakil sir, who provided valuable suggestion whenever we were facing difficulties. We would also like to thank the DCO of our department, Golam Zilani sir, for helping us with all the data from the department necessary for our work.

# Table of Contents

Declaration	i
Approval	ii
Abstract	iii
Dedication	iv
Acknowledgment	v
Table of Contents	vi
List of Figures	viii
List of Tables	ix
Nomenclature	x
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem Statement . . . . .	2
1.3 Paper Outline . . . . .	3
<b>2 Related Work</b>	<b>4</b>
<b>3 Optimization Algorithms</b>	<b>7</b>
3.1 CSP Solver Algorithm . . . . .	7
3.2 Genetic Algorithm . . . . .	8
3.3 Simulated Annealing . . . . .	9
3.4 Hybrid Algorithm . . . . .	11
<b>4 Methodology</b>	<b>12</b>
4.1 Dataset . . . . .	12
4.1.1 Data Pre-processing . . . . .	12
4.1.2 Dataset Description . . . . .	13
4.2 Constraints . . . . .	15
4.3 Implementation Details . . . . .	17
4.3.1 Generate Initial Solution . . . . .	18
4.3.2 CSP Solver Algorithm . . . . .	19
4.3.3 Genetic Algorithm . . . . .	20

4.3.4	Simulated Annealing . . . . .	21
4.3.5	Hybrid Algorithm . . . . .	23
<b>5</b>	<b>Results</b>	<b>25</b>
5.1	Visualization of Output . . . . .	25
5.1.1	Generating Current Semester Routine . . . . .	25
5.1.2	Extracting Individual Course Section Information . . . . .	26
5.1.3	Finding All Faculty Members Information . . . . .	26
5.1.4	Extracting Individual Faculty Information . . . . .	26
5.2	Results on Different Datasets . . . . .	27
5.2.1	Spring 2020 Dataset . . . . .	27
5.2.2	Summer 2020 Dataset . . . . .	31
5.2.3	Fall 2020 Dataset . . . . .	35
<b>6</b>	<b>Analysis</b>	<b>39</b>
6.1	Overall Analysis . . . . .	39
6.1.1	Spring 2020 Dataset . . . . .	39
6.1.2	Summer 2020 Dataset . . . . .	41
6.1.3	Fall 2020 Dataset . . . . .	42
6.2	Comparison Between Different Algorithms . . . . .	44
<b>7</b>	<b>Conclusion</b>	<b>46</b>
7.1	Limitations . . . . .	46
7.2	Future Works . . . . .	47
	<b>Bibliography</b>	<b>51</b>



# List of Figures

4.1	Snapshot of the Course Dataset . . . . .	14
4.2	Snapshot of the Faculty Dataset . . . . .	15
5.1	Snapshot of Semester Routine for All Departmental Courses . . . . .	25
5.2	Snapshot of Individual Course Information . . . . .	26
5.3	Snapshot of All CSE Faculty Members Information . . . . .	26
5.4	Snapshot of Individual Faculty Member Information . . . . .	26
5.5	Results of CSP Solver on Spring 2020 Dataset . . . . .	27
5.6	Results of Genetic Algorithm on Spring 2020 Dataset . . . . .	28
5.7	Results of Simulated Annealing Algorithm on Spring 2020 Dataset . . . . .	29
5.8	Results of Hybrid Algorithm on Spring 2020 Dataset . . . . .	30
5.9	Results of CSP Solver on Summer 2020 Dataset . . . . .	31
5.10	Results of Genetic Algorithm on Summer 2020 Dataset . . . . .	32
5.11	Results of Simulated Annealing Algorithm on Summer 2020 Dataset . . . . .	33
5.12	Results of Hybrid Algorithm on Summer 2020 Dataset . . . . .	34
5.13	Results of CSP Solver on Fall 2020 Dataset . . . . .	35
5.14	Results of Genetic Algorithm on Fall 2020 Dataset . . . . .	36
5.15	Results of Simulated Annealing Algorithm on Fall 2020 Dataset . . . . .	37
5.16	Results of Hybrid Algorithm on Fall 2020 Dataset . . . . .	38
6.1	Time vs Score Comparison Between Different Algorithm Results on Spring 2020 Dataset . . . . .	40
6.2	Score Frequency Variations Between Different Algorithms on Spring 2020 Dataset . . . . .	40
6.3	Time vs Score Comparison Between Different Algorithm Results on Summer 2020 Dataset . . . . .	41
6.4	Score Frequency Variations Between Different Algorithms on Summer 2020 Dataset . . . . .	42
6.5	Time vs Score Comparison Between Different Algorithm Results on Fall 2020 Dataset . . . . .	43
6.6	Score Frequency Variations Between Different Algorithms on Fall 2020 Dataset . . . . .	43

# List of Tables

4.1	Index Values of Timeslots . . . . .	13
5.1	Different Test Results of CSP Solver on Spring 2020 Dataset . . . . .	27
5.2	Different Test Results of Genetic Algorithm on Spring 2020 Dataset . . . . .	28
5.3	Different Test Results of Simulated Annealing on Spring 2020 Dataset . . . . .	29
5.4	Different Test Results of Hybrid Algorithm on Spring 2020 Dataset . . . . .	30
5.5	Different Test Results of CSP Solver on Summer 2020 Dataset . . . . .	31
5.6	Different Test Results of Genetic Algorithm on Summer 2020 Dataset . . . . .	32
5.7	Different Test Results of Simulated Annealing on Summer 2020 Dataset . . . . .	33
5.8	Different Test Results of Hybrid Algorithm on Summer 2020 Dataset . . . . .	34
5.9	Different Test Results of CSP Solver on Fall 2020 Dataset . . . . .	35
5.10	Different Test Results of Genetic Algorithm on Fall 2020 Dataset . . . . .	36
5.11	Different Test Results of Simulated Annealing on Fall 2020 Dataset . . . . .	37
5.12	Different Test Results of Hybrid Algorithm on Fall 2020 Dataset . . . . .	38
6.1	Different Implementation Results on Spring 2020 Dataset . . . . .	39
6.2	Different Implementation Results on Summer 2020 Dataset . . . . .	41
6.3	Different Implementation Results on Fall 2020 Dataset . . . . .	42
6.4	Datasets Comparison . . . . .	44

# Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

*AI* Artificial Intelligence

*CSP* Constraint Satisfaction Problem

*DCAP* Departmental Course Allocation Problem

*EOI* Expression of Interest

*GA* Genetic Algorithm

*HA* Hybrid Algorithm

*LS* Local Search

*NP* Non-deterministic Polynomial time

*SA* Simulated Annealing

# Chapter 1

## Introduction

University departmental tasks are the most crucial parts of a university system as it is interconnected to institutional management, efficiency and effectiveness. It includes inventory management, generating class schedule, assigning Student Tutors, handling the process of changing labs of students' due to clashes in their schedule etc. The work shown in this paper focuses on solving the Departmental Course Allocation Problem (DCAP). This is known as an NP-hard problem [1][2] and it involves assigning of faculties to their respective courses and timeslots, while also satisfying a set of constraints. The task of course allocation is quite time consuming when it is done manually. The integration of Artificial Intelligence (AI) has been instrumental in tackling this problem. However, there is no universal solution to this problem since different institutions have different requirements and they can vary from one institution to another. Researchers have approached this problem by using various optimization algorithms to meet these requirements. In this paper, we attempted to solve this problem by using four different algorithms, a CSP Solver Algorithm, Genetic Algorithm (GA), Simulated Annealing (SA) and Hybrid Algorithm combining GA & SA. Using the above mentioned algorithms, an optimal solution can be found in the least possible effort.

### 1.1 Motivation

As already mentioned, course allocation is a task that requires a lot of time and effort. In BRAC University, this is usually handled by some faculty members in the beginning of every semester. They have to iterate through the same process of allocating the available timeslots before finding a solution that matches all their requirements. While going through this process, they try to avoid clashes. However, in the first few attempts, it is nearly impossible to find a solution that is clash-free. Once a clash is found, they have to backtrack and search for options that can resolve the clash. It becomes hectic for those faculty members in charge to go through the same process again and again. Students also get frustrated when the routine keeps changing frequently. The scope of this problem is very large and complicated, but some parts of it can be solved using AI optimization algorithms. This will help to find optimal solutions to the allocation problem and assist the faculty members in their work. We hope that this will reduce the work-hour loss of the department and the productivity in other departmental tasks will increase.

## 1.2 Problem Statement

The scope of our thesis is the automation of the course allocation problem of the Department of CSE of BRAC University. To implement our idea, we have chosen the routine making task of the department as our case study. It includes a lot of factors and constraints which need to be handled individually while manually preparing the routine for each semester. Opening sections of courses, providing a time slot for every section opened, allocating rooms to every sections, providing lab slots for the courses that require labs and finally assigning a faculty to each section of the offered courses along with the labs is not an easy task. Our target is to handle as much of these constraints or conditions and propose an efficient solution that will be able to allocate faculties to the respective sections of courses.

For the undergraduate program, in every semester, generally, there are a total of six theory slots that starts from 8:00 am and ends at 4:50 pm where a slot lasts 1 hour 20 minutes. There are three lab slots that starts from 8:00 am and the last slot ends at 4:50 pm where a slot lasts three hours. There is another theory slot at 5:00pm - 6:20 pm for exceptional cases i.e. it is used when there are no other options. In our case study, this slot is excluded. One lab slot is equivalent to two consecutive theory slots. So in total, for six working days, there are thirty six slots. Besides, four of the courses offered by the department have alternate labs, which means that the labs take place by skipping a consecutive week. Courses offered by the department are of two categories, Core and Elective. Core courses are the courses that must be completed by the students of the department to complete their degree. Elective courses are the courses that are to be completed by the students according to their choice. So every section of all the courses need to be allotted to the timeslots mentioned above.

The department has seven categories of faculty members, starting from the Head(s) of the department, followed by the Associate Professor, Assistant Professor, Lecturer III, Lecturer II, Lecturer I and Contractual Lecturers. The Head and the Deputy Head of the department can take a maximum of two theory courses or six credits per semester. The Associate Professors take a maximum of three theory courses or nine credits each semester. Faculty members of the above mentioned categories usually do not take labs, so they are only allotted to theory slots. The Assistant Professors, Lecturer III, Lecturer II and Lecturer I can take at most 12 credits per semester. The distribution of the 12 credits are generally done by assigning them to three theory slots and two lab slots. The final category of faculty members are the Contractual Lecturers. They are given a maximum of 10.5 credits. The number of faculty members in each category varies every semester. In Spring 2020, there was one Head of the department and one Deputy Head, five Associate Professors, four Assistant Professors, sixty two Lecturer III, II and I and fifty eight Contractual Lecturers. In Summer 2020, there was one Head of the department and one Deputy Head, five Associate Professors, five Assistant Professors, fifty seven Lecturer III, II and I and thirty five Contractual Lecturers. In Fall 2020, there was one Head of the department and one Deputy Head along with one Distinguished Professor, five Associate Professors, five Assistant Professors, forty six Lecturer III, II and I and thirty nine Contractual Lecturers. The number of new Contractual Lecturers intakes differ from semester to semester, so their total number is not constant.

The distribution of credits for theory, regular lab and alternate labs are 3, 1.5 and 0.75 respectively. When a faculty is assigned to either of the above three, then that amount of credit is updated to the amount of credits taken for that semester. Every semester, an Expression of Interest (EOI) form is provided to all the faculty members except for the Contractual Lecturers, which is used to collect the list of courses they would prefer to teach. The preference of courses for the Contractual Lecturers can be taken from their appointment form where they are asked to provide their preferences. This works as an alternative to the EOI form, but serves the same purpose. This helps to sort out which faculty member is to be assigned to a course and also to know if there is any special request by any faculty member, i.e. any time slot or day that they do not want to take classes due to valid reasons or any sort of pre-commitments elsewhere. Keeping these in mind, the faculty members are assigned to suitable courses.

There are some sides of this problem that are not handled in our present work. One of them is allocation of rooms. The faculty members of the department who are in-charge of the routine making committee, chart out an approximate number of sections of the courses. They address the Office of the Registrar with the total number of rooms that the department might need for a semester, then the Office puts forward a certain number of rooms that can be given to the department. They take the rooms given by the Office of the Registrar and start preparing the routine. They start allocating the sections to the provided rooms and check whether all sections have been allocated or not. If not, then the committee again asks for rooms from the Office of the Registrar and they provide the required rooms. As the room allocation of the courses is not directly linked to the faculty allocation problem, we have not included this in our work. Another perspective which we have excluded is the faculty allocation of the postgraduate program. The timeslot distribution of this program differs from the undergraduate program and different sets of constraints are required to handle this perspective.

### 1.3 Paper Outline

- Chapter 2: This chapter discusses the related works conducted by other researchers.
- Chapter 3: This chapter gives the general overview of the AI optimization algorithms used in this work.
- Chapter 4: This chapter explains how the datasets were prepared, the constraints taken into consideration and the implementation details of the used algorithms.
- Chapter 5: This chapter visualizes the results with appropriate images, graphs and tables.
- Chapter 6: This chapter compares the performance of the different algorithms
- Chapter 7: This chapter concludes the work by putting forward the limitations in our work and proposing the scope of future research.

# Chapter 2

## Related Work

The researches in course allocation have been carried out for a long time. Various algorithms have been proposed by researchers such as graph coloring methods, constraint-based methods, population-based methods, metaheuristic methods, variable neighborhood search, hybrid and hyperheuristic approaches and so on. Population-based methods consist of genetic algorithm, ant-colony optimization, memetic algorithm, etc. Metaheuristic methods include tabu search, simulated annealing and great deluge [1]. Graph coloring algorithm is one of the first and most popular algorithms that have been used in this field. One of the major drawbacks of this algorithm is that, non-academic constraints result in complex problem formulation. Eventually, this creates difficulties in implementation. Researches using linear programming were not feasible as the complexity of variables and constraints increased. Other researchers have integrated constraint logic programming with several algorithms to solve the same problem. But the performance was poor as the results were very reactive to small changes [3]. In recent years, both population based and local-area based methods are being used to solve the course allocation problem. Among them, genetic algorithm, which is a population based method, has gained popularity because of its high optimization efficiency. Also, simulated annealing has been preferred by many researchers since it can avoid getting stuck at the local minimum [4]. In the first international conference on the practice and theory of automated timetabling, more emphasis has been laid on general problem solving algorithm such as genetic algorithm, simulated annealing and tabu search [5]. In our presented work, we have chosen a CSP solver algorithm, genetic algorithm, simulated annealing algorithm and a hybrid algorithm combining GA & SA to solve the course allocation problem of The Department of Computer Science and Engineering of BRAC University. We went for four separate approaches to make a comparative analysis on which algorithm can give the better result in terms of optimization and constraint satisfaction.

The timetabling problem can be formulated as a CSP and it can be implemented by applying CSP search algorithms. Since all constraints of this problem is impossible to satisfy, the violations of soft constraints need to be minimized as much as possible [6]. In [7], Dynamic Constraint Matching (DCM) was implemented along with Vertex Graph Coloring (VGC) in order to come up with a solution for the timetabling problem. DCM is formed by constraints logical formulation, collision matrix generation and validation using the collision matrix. Applying VGC alone does not ensure the satisfaction of all constraints. Integrating DCM with VGC does

not violate any of the hard or soft constraints. The execution time of this technique was less than 1 minute. Combining backtracking-free construction and local search technique with look-ahead capabilities, [8] have constructed the solution of timetabling problem. Considering the CSP as a constraint graph, a graph search algorithm named A\*-algorithm was implemented by [9]. Representing the timetabling problem as a PCSP (Partial Constraint Satisfaction Problem), [6] have defined a finite domain solver based on CHR (Constraint Handling Rules). A model was built using Constraint Logic Programming (CLP) to solve the course timetabling problem as a CSP [10].

GA has some certain advantages over other algorithms. It does not require complex mathematical formulation. Also, global search is performed smoothly because of the operators' ergodicity of evolution [11]. Although GA provides globally optimal solutions for complex search spaces [12], it needs more time to execute [13]. Several researchers have reduced this execution time by modifying genetic and heuristic operators and integrating LS techniques [1]. Since GA is a parallel random search optimization algorithm, it can achieve global optimization by improving the configuration of several resources [14] [15]. By using such a GA, time consumed to reach the optimal solution has been significantly lower compared to conventional GA [16] [17]. Another way of approaching this problem is to use Modified GA and Cooperative GA. Modified GA can produce results at a shorter time and Cooperative GA can reduce the cost value [18] [17]. Guided Search has been integrated with the classical GA to approach the University Course Timetabling Problem [1] [19]. Quality of generated timetable increases during the initial phase of GA according to [20]. The authors proposed to use the generated output of GA as input to another optimizing algorithm to enhance the efficiency. But it has also been found that normal GA does not provide optimal solutions compared to other meta-heuristic approaches [21]. More recent works on GA have proposed modifications to genetic operators such as the crossover operator [22]. Generally, two offsprings are produced in this operation, but the authors have suggested to produce only one offspring in each crossover by taking the best genes from the parents. Comparing the results of this approach with the conventional crossover techniques show that this method provides better results in lesser time than one-point crossover, but was not faster than the two-point crossover. Another variation of GA has been introduced by [23] that uses multiple levels of GA computation which they called MDGA (Multi-Depth GA). These levels were based on the depth of the objective function such as shallow, medium and deep. Their goal was to divide the problem to smaller ones to solve the timetabling problem while reducing its time consumption.

SA is a single-solution local search heuristic algorithm that works for both discrete and continuous problem [24]. It uses a temperature control parameter and a cooling schedule to escape the local minimum/maximum and attempt to reach a solution near global optimum [25]. A disadvantage of SA is that the convergence takes excessive amount of time when the search space is large [26]. To overcome this drawback, some researchers have controlled the temperature parameter of the algorithm which resulted in better performance. The global optimum can be found exponentially faster by tuning the rate of cooling the temperature. The performance of the algorithm is proportional to this rate. When the process of the cooling is slowed down, the computational cost is improved [27]. A similar approach of lowering the



temperature reduction value was proposed that also converges in lesser time, named Modified Simulated Annealing. Instead of using the linear exponential temperature decrease function, a parabolic exponential temperature decrease function has been implemented, that gives better solutions [4]. SA has been applied to another research [28] where the execution time was approximately 14 hours. To reduce this time, a parallel approach of running the SA has been implemented on a shared memory multiprocessor. This seemed to be promising at first, but the increasing number of competing processors created complications. A comparative study was conducted among SA, GA and a hybrid of SA-GA where SA produced the best result. Hybrid of SA-GA came in second and GA generated the worst result. The reason behind the under performance of GA was due to the smaller search space [29]. A related approach was taken by [30], where they used roulette wheel selection with a partially matched crossover (PMX) to improve the produced offsprings. To get more refined results, SA was hybridized with GA.

Taking the good properties of local and global area based algorithms, [31] have integrated GA with LS (Local Search) algorithms such as SA, TS (Tabu Search), RI (Randomized Iterative) Local Search. Their target was to take advantage of exploration ability of GA and exploitation ability of LS. The LS algorithms help GA to get out from the local optimum. Additionally, fuzzy logic is implemented to check soft constraint violation of the fitness function. Considering the fitness and execution time, hybrid of GA and TS have generated the best optimal solution even when the dataset was large. The hybrid of GA and SA have performed worse than the hybrid of GA and TS, but better than the hybrid of GA and RI. A two-stage approach of solving the exam scheduling problem has been taken in [32], where Constraint Programming (CP) was used to generate an initial solution that satisfies the hard constraints. Then SA is implemented to refine that generated solution by satisfying the soft constraints. Kempe chain neighborhood structure was applied with the SA to determine the starting temperature. The user can specify how long the algorithm will run depending on this structure. This increases the efficiency of the algorithm. Backtracking with forward checking has been used in the CP phase. Similar approach has taken by [33], where the CP was used to solve all the hard constraints and SA was used to improve the quality of solutions. For solving the teacher assignment and the course scheduling problem, a hybrid between an integer programming approach, a greedy heuristic and a modified SA has been proposed by [34]. According to them, this hybrid algorithm can resolve the issues of integer programming approach regarding large datasets. Another hybrid approach was proposed by [35], where a model was constructed based on the application of construction heuristics, TS, variable neighborhood descent and SA. The initial feasible solution is generated by solving the hard constraints. These constraints were satisfied using LS and TS. To improve the initially produced timetable, they minimized the violation of soft constraints with the use of variable neighbourhood descent and SA. Recently, [36] has presented a hybrid of Parallel Genetic Algorithm with LS. The Parallel GA was used to increase the convergence speed and to diversify the population. The solution provided by GA has been improved by minimizing soft constraint violation using LS and the elitism operator. These prevent the GA from getting stuck in the local optimum and leads to better performance.

# Chapter 3

## Optimization Algorithms

### 3.1 CSP Solver Algorithm

CSP solver algorithms are general purpose algorithms that are used to solve CSP. CSP is a special subset of search problems that uses factored representation for each state. Each of those states contains a set of variables with values. When the constraints on all those variables are satisfied by the values that they hold, then that problem is considered to be solved. Instead of using problem-specific heuristics, CSP solver algorithms use general-purpose heuristics to solve complex problems. The specialty of CSP solver algorithms is that they can reduce the size of the search space significantly by removing the variable/value pairs that do not satisfy the constraints. CSP is used to solve various real world problems such as Assignment Problems, Timetabling Problems, Production Scheduling, Transportation Scheduling and so on. Optimization Problems can be represented by a sequence of constraints. An objective constraint can be introduced by specifying a threshold value on the objective function. This threshold value is adjusted continuously to check if the values of the variables are satisfying the constraints or not. In this way the optimal value of the objective function is achieved [37].

CSP can be solved using various approaches such as backtracking search, filtering, learning and decomposition techniques, use of efficient representations and heuristics [38]. In these techniques, the constraints and the variables are considered to be constant. However, in real world problems, the environment does not remain static all the time. With each execution, the constraints and the variables keep changing. In order to deal with this sort of situation, Dynamic CSP was introduced. It is one of the many variants of CSP, in which the set of constraints evolves with the environment. This change of constraints can be in the form of addition or deletion. Due to these changes, the solutions that were generated previously may not be valid in the next problem of the sequence. This sequence is referred to static CSPs that are placed sequentially [39] [40]. One of the methods to solve Dynamic CSP is local repair method. The algorithm starts with previous consistent assignment (partial or complete), then fixes the inconsistency of changed constraints. It uses a sequence of local modifications that means modifying the assignment of only one variable [41] [42].

## 3.2 Genetic Algorithm

GA is an adaptive search based heuristic optimization technique. To produce the offspring for the next generation, the individuals of the population go through the process of natural selection. The offspring will inherit the characteristics of the individuals (parents) to be passed into the next generation. The chances of the survival of the offspring will be high if their parents have good fitness value, otherwise they will not be added to the next generation. This conforms to the rule of survival of the fittest. This procedure continues until it reaches the stopping criteria which can be maximum number of generations or a certain time limit [1]. GA is generally applied in search problems and optimization problems as it is capable of finding optimal solutions even in the most complex search spaces. It is mostly used in building Recurrent Neural Network, for Mutation Testing, Code Breaking, Filtering and Signal Processing, Image Processing, Learning Fuzzy Rule Base, Genetics Based Machine Learning and Scheduling Applications.

GA has gained popularity because of its various advantages over other optimization algorithms. One of them is that it can be easily parallelised. It means that multiple GAs can be used simultaneously to perform a single task [43]. Since each of them is independent from each other, the mutation and crossover are calculated separately. As a result, their individuals vary from one another. Among those individuals, the best one is selected as the solution to that task. Since GA is a population based algorithm, it searches from a population of points rather than a single point. That is why it works well with larger dataset compared to other algorithms. GA always gives a solution and it is improved over the time. It can be called an upgraded version of random local search since the optimal solutions are stored in every iteration. It can be used to optimize both continuous and discrete functions. Although GA is widely used for researches, it has some drawbacks. As previously mentioned, GA is capable of providing good results even when the search space is very large. However, it becomes computationally expensive with the increasing size of the search space - if population size is very large, algorithm performing speed will decrease intensely. On the other hand, if the population size is small, the problem will converge on the answer which is not necessarily optimal [17]. In GA, the fitness value is calculated over and over again, which can be time consuming in some cases.

The algorithm comprises of 5 phases.

- **Initial Population**

The initial population is defined by a set of individuals or chromosomes, that are considered to be the solution of the given problem. Chromosomes are formed by a set of variables known as Genes.

- **Fitness Function**

Fitness function is a function which measures how fit or good an individual is to advance to the next generation. Each individual of the population is given a fitness score. Higher the score is, better is the probability of being selected. Fitness function is computed repeatedly, and it can affect the computation speed of GA.

- **Selection**

Using the fitness function, the fittest individuals are selected from the initial population. Genes of these individuals are forwarded to the succeeding generation. Some methods of selecting these individuals are Roulette Wheel Selection, Stochastic Universal Sampling, Tournament Selection, Rank Selection and Random Selection.

- **Crossover**

The selected individuals acts as the parents that will produce offsprings by exchanging their genes. A crossover point is randomly selected within the genes of these parent chromosomes. The exchange of genes will be continued until this crossover point is reached. In this way the offsprings are created with the genes of their parents. The produced offsprings are included to the population. Crossover operators can be one-point, multi-point, uniform and so on.

- **Mutation**

The newly formed chromosomes are randomly modified in order to get a new solution that creates diversity in population. Mutation is operated on a low random probability. If this probability is set to high, the GA turns into a random search. Another crucial factor of this phase is that it prevents premature convergence. Mutation can be performed by randomly flipping, swapping, shuffling, resetting and inverting the genes.

These phases keep running until a termination condition is reached. These conditions can vary depending on the type of problems. One of the termination conditions can be stopping the algorithm when there is no improvement in the generated solution for a certain number of iterations. Moreover, if the possibility of getting better individuals in the next generations becomes low, then the execution of the algorithm can be stopped. Another way of termination is reaching a fixed number of generations. Also, when the fitness function reaches the predefined value, the GA can be terminated [44].

### 3.3 Simulated Annealing

SA is a probabilistic local search algorithm that is based on the process of physical annealing [45]. In this process a material is heated until it reaches an annealing temperature and then slowly cooled down to a stable structure. When the temperature is very high, the molecular structure of the material is weaker and can be changed more easily. When the temperature is low, the molecular structure becomes stronger and the structure is less likely to change. This idea of lowering the temperature to achieve stabilization is applied to solve the problem of Hill Climbing Algorithm, where the algorithm got stuck in local maxima. The reason is that, in Hill Climbing Algorithm, downward moves are not allowed. However, by using the concept of annealing, at high temperature, the probability of accepting a downward move is also high. This allows the algorithm to jump out of the local maxima. The probability becomes low as the temperature cools down with time [46].

SA is useful for generating optimal solutions to a large variety of problems such as Travelling Salesman Problem, Scheduling Problem, Task Allocation, Graph Coloring and Partitioning, Non-linear Function Optimization and so on. It can also deal with arbitrary systems and cost function. Like any other optimization algorithm, at first SA generates a random initial solution, and then explores the neighbor states. If the solution of these neighbor states are better than the current solution, then it shifts to the new solution. However, if the starting temperature is not high enough to move to neighbor states, the algorithm will behave like a Hill Climbing Algorithm and the final solution will be similar to the starting solution. One of the few drawbacks of SA is not being able to find the appropriate starting temperature for a whole range of problems. Another disadvantage of SA is its slow convergence as it is inherently sequential [26]. So, SA might take large amount of time when it comes to huge search spaces [4]. When there are few local minima in some problems, SA will not be able to take advantage of its core functioning principle i.e escaping from the local minima. In that case, simpler and faster methods like gradient descent will function better than SA.

The process of implementing SA algorithm is:

1. Generating a random initial solution  $s = s_0$  that checks all the boxes of an acceptable solution. The initial temperature will be  $t = t_0$ .
2. Temperature reduction function alpha ( $\alpha$ ) is defined. This function will be used in the temperature reduction rules such as  $t = t^*\alpha$ . The initial temperature is slowly cooled down using this function.
3. Select one of the neighbor solutions to check whether it is better than the current solution. This checking is done by calculating the difference between the new solution's cost  $c_{new}$  and current solution's cost  $c_{old}$ .

$$\Delta c = c_{new} - c_{old} \quad (3.1)$$

where  $\Delta c$  is the difference between new cost and old cost.

4. If the value of  $\Delta c$  is less than 0, it means the new solution is better than the old one and the algorithm is closer to an optimum. So, it will accept the new solution. If the value of  $\Delta c$  is greater than 0, it means the old solution is better and the algorithm is moving towards a worse solution. To avoid getting stuck in local maxima, acceptance probability is calculated to compare with a random number generated between 0 and 1. The acceptance probability is a function that tells whether we should accept the new solution or not.

$$P = \begin{cases} 1 & \text{if } \Delta c \leq 0 \\ e^{\frac{-\Delta c}{t}} & \text{if } \Delta c > 0 \end{cases} \quad (3.2)$$

Here,  $P$  represents the acceptance probability.  $P = 1$  denotes that the new solution will be accepted as the new cost is less than the old cost. Otherwise, the algorithm will move to the new solution based on the value given by  $e^{\frac{-\Delta c}{t}}$

5. The above two steps will be iterated until it reaches the termination conditions such as giving an acceptable solution for a certain set of parameters, reaching a specified end temperature or a fixed number of iterations. The algorithm can also be terminated if the cost becomes zero or does not change for a certain amount of iterations [28]. After every iteration, the initial temperature will be reduced according to  $\alpha$ .

The acceptance probability changes with respect to the temperature. When the temperature is higher, the algorithm has greater probability of accepting a worse solution. Accepting a worse solution allows exploration of the search space to look for the global maximum. As mentioned before, the concept of annealing is that, higher the temperature, higher the possibility of changing the structure of the material. When the temperature decreases, the probability of accepting a worse solution becomes very low. This allows exploitation, which means that the algorithm will not look through other parts of the search spaces except the one it is in. Here, it will rather try to converge and reach the global maximum. SA can take a long time to converge if it has to go through many iterations, but it has the capability to get out of the local maxima by jumping to bad solutions and eventually give an optimal solution.

### 3.4 Hybrid Algorithm

When an algorithm is used to solve a problem, it might not be able to generate the expected results, or be inefficient in terms of cost and complexity. Every algorithm has advantages as well as drawbacks. To overcome the limitations, multiple algorithms are integrated with each other. The shortcomings of one algorithm is covered up by the other algorithm. By merging these algorithms, the features that are required to solve the specific problems can be combined together. HA utilizes the strengths of these features to solve the same problem more efficiently [34]. This enhances the overall performance and the generated results are improved compared to the results generated by the algorithms individually.

There are various combinations of algorithms when it comes to hybridization. The algorithms are chosen depending on the problem requirements. For example, to solve optimization problems like university course timetabling, hybrid of GA with LS is popular. LS techniques include SA, Tabu Search, Randomized Iterative local search and so on. GA has high probability of getting stuck at the local optimum. On the other hand, RI local search is likely to achieve the optimal solution although it converges very slowly. To account for these downsides, GA and RI are integrated to take advantage of the high convergence speed of GA and the exploitation ability of RI. SA can be used in place of RI since RI has the possibility of getting trapped at the local optimum [31]. Another hybridization can be done between CP and SA. To satisfy the hard constraints, CP algorithm named backtracking with forward checking (BC-FC) is used. An initial feasible solution is generated by using this algorithm. In the second stage, SA is applied to improve the generated timetable from the first stage by satisfying the soft constraints. The main focus of the second stage is to optimize an objective function [32]. Many more approaches are taken by researchers to generate robust hybrid algorithms.

# Chapter 4

## Methodology

### 4.1 Dataset

To implement the algorithms mentioned in the previous chapter, dataset of a particular size or proportion is necessary. The dataset works as the input to provide an initial solution, and the optimization algorithms work on that solution to improve it further. This increases the accuracy and a final optimal result is generated.

#### 4.1.1 Data Pre-processing

The dataset was collected from The Department of CSE of BRAC University. These were routines of all courses along with the faculty members assigned to each of those courses. The routines were from Spring 2013 to Fall 2020, a total of 24 semesters. After studying the pattern of course allocation to the faculty members over all the semesters, we decided to use the routines of most recent semesters. So the routines of Spring, Summer and Fall 2020 were selected as the final datasets for the implementation of the proposed algorithms. The routines collected from the department had information related to the class timings of every offered course, the rooms they were allocated to and the faculty members assigned to these courses. As mentioned before, our work does not include the allocation of rooms, since it has no relation to the faculty allocation problem and mostly handled by the Office of the Registrar. So the room numbers mentioned in the routines were not necessary for our work. The theory and lab of the courses in the original routine were placed into two columns because they had different faculty member allocations and timeslots. For the ease of our implementation, these two columns were merged into one, and they were separated from each other by adding ‘T’ or ‘L’ at the end of their course codes. However, for some courses, the labs are held on alternate weeks. This means that if the lab of one section is held in a week, then the lab of another section will take place in the following week at the same timeslot. Generally, these two sections are formed in pairs with a credit limit of 0.75 each. We considered this pair as a single lab section with 1.5 credits in total, making it similar to the regular labs. Furthermore, the theory and lab timings were distributed in three different columns in the collected routines. The theory timings were provided in two columns; Day and Time. The lab timings were given in one column where the day and time were listed together and separated by a slash (/). For our purposes, we decided to assign values from 0-35 to these Day/Time pairs and brought them under two columns in

our final dataset. The table 4.1 shows how these values were assigned.

Day	Time					
	08:00 - 09:20	09:30 - 10:50	11:00 - 12:20	12:30 - 1:50	02:00 - 03:20	03:30 - 04:50
Sunday	0	1	2	3	4	5
Monday	6	7	8	9	10	11
Tuesday	12	13	14	15	16	17
Wednesday	18	19	20	21	22	23
Thursday	24	25	26	27	28	29
Saturday	30	31	32	33	34	35

Table 4.1: Index Values of Timeslots

This table represents a total of 36 values equivalent to each of the Day/Time pairs of the collected routine. In most of the cases, the theory classes are allocated to the same time every two days of the week. Such as, a theory course on Sunday 08:00 am-09:20 am and Tuesday 08:00 am-09:20 am is denoted by 0 and 12 according to this table. The lab classes are held in two consecutive timeslots on the same day. For example, a lab class starting on Monday 11:00 am will end at 1:50 pm. In this case, the values to be assigned will be 8 and 9. In this way, the Day/Time pairs for each section of the original routines were kept unchanged, but brought under two columns in the final dataset. This concludes the necessary adjustments to create the course information dataset of the three individual semesters.

The preferred courses of the faculty members were prepared separately. This was not mentioned directly in the collected routines. From the analysis of the previous routines ranging from Spring 2013 to Fall 2019, the pattern of courses allocated to each of the faculty members were considered. In this way, the list of courses that can be given to the faculty members were brought under one column, and separated by a hyphen (-). In the cases of new faculty members, where there was no previous record of course allocation, their EOI forms were used. These forms are comprised of several courses in different columns that the faculty members want to take. For taking inputs easily, the course codes were added to the column of preferred courses using the hyphen separator in the final dataset. In this way, a total of six datasets were prepared and used for our implementation.

### 4.1.2 Dataset Description

As mentioned in the previous section, there are a total of six datasets, of which three are composed of course information and the rest have information of the faculty members. The first type of dataset (Course dataset) has five columns. They are:

- **Name** - This column includes all the course codes of that particular semester along with two special characters T or L where T denotes theory and L denotes lab respectively.
- **Section** - The Section column contains the section number of a particular course.



Name	Section	Credit	Slot 1	Slot 2
CSE320T	8	3	1	13
CSE320T	9	3	4	16
CSE320T	10	3	7	19
CSE320T	11	3	6	18
CSE320T	12	3	29	35
CSE321T	1	3	2	14
CSE321T	2	3	2	14
CSE321T	3	3	10	22
CSE321T	4	3	27	33
CSE321T	5	3	4	16
CSE321T	6	3	5	17
CSE321T	7	3	24	30
CSE321T	8	3	0	12
CSE321T	9	3	4	16
CSE321T	10	3	0	12

Figure 4.1: Snapshot of the Course Dataset

- **Credit** - This column holds the number of credits of a course. For the Department of CSE, the theory part of the course has 3.0 credits, and the lab part has 1.5 credits.
- **Slot 1** - The slot 1 column denotes the first slot of a week for theory or the first part of the lab slot.
- **Slot 2** - The slot 2 column denotes the second slot of a week for theory or the second part of the lab slot.

The second type of dataset (Faculty Dataset) has four columns. They are:

- **Name** - This column contains the names of all faculty members.
- **Initial** - The Initial column holds the unique initials of those faculty members which is provided by the university.
- **Maximum Credit Limit** - This column has the maximum amount of credits a faculty member can take in a particular semester. This value varies from one faculty member to another. Generally, the Professors get 6 credits, the Associate Professors get 9, the Assistant Professors and the Lecturers get 12 credits. The Contractual Lecturers are given 10.5 credits.
- **Preferred Courses** - The Preferred Courses column shows a list of courses that might be assigned to the faculty members. This list was formed based on the courses they have taken before and the EOI forms.

Name	Initial	Maximum Credit Limit	Preferred Courses
Dr Zahidur Rahman	ZAR	6	CSE422T-CSE470T-CSE471T
Warida Rashid	WAR	12	CSE110L-CSE111L-CSE220L-CSE221L-CSE331T-CSE420T-CSE420L-CSE422L-CSE471L
Wasif Ahmed	WAA	12	CSE330T-CSE330L-CSE490:1T-CSE490:1L
Md. Tanzim Reza	TRZ	12	CSE101T-CSE101L-CSE220L-CSE221L-CSE321T-CSE422T-CSE422L-CSE471L-CSE420L-CSE423L
Trisha Das	TRD	12	STA201T-CSE110L-CSE111L-CSE162L-CSE220T-CSE220L-CSE221T-CSE221L-CSE260T-CSE341L-CSE360L-CSE461L-CSE471L
Tanvir Rahman	TNR	12	STA201T-CSE110L-CSE162L-CSE230T-CSE331T-CSE370L-CSE390T-CSE420T-CSE420L-CSE421L-CSE422L-CSE423L
Md Tawhid Anwar	THD	12	CSE110L-CSE111T-CSE111L-CSE162L-CSE220L-CSE260L
Tanvir Ahmed	TAA	12	CSE250L-CSE251L-CSE350L-CSE460T-CSE460L
Shakila Zaman	SZN	12	CSE110L-CSE162L-CSE320T-CSE321T-CSE321L-CSE370L-CSE421L-CSE422L-CSE423L-CSE471L
Sabrina Zaman Ishita	SZI	12	CSE111L-CSE220L-CSE221L-CSE331T-CSE341T-CSE341L-CSE360L-CSE370L-CSE421L-CSE420L-CSE422L-CSE423L-CSE461T-CSE461L-CSE471L
Syed Zamil Hasan Shoumo	SZH	12	CSE220L-CSE221T-CSE221L-CSE260L-CSE341T-CSE422T-CSE422L-CSE423L-CSE341L-CSE360L-CSE461L-CSE471L
Mirza Md Tausif Shorif Snigdho	SSN	12	CSE230T
Salman Sayeed Khan	SSK	12	CSE101L-CSE111T-CSE111L-CSE230T-CSE260L-CSE320T-CSE330L-CSE360L-CSE370L-CSE420L-CSE421L-CSE461L

Figure 4.2: Snapshot of the Faculty Dataset

## 4.2 Constraints

Constraints play a vital role while designing and implementing a system. All of the constraints of a system can be classified into various sub-categories based on the type of the problem. In our system, we classified all the constraints into two categories, hard and soft constraints. While developing the system, all the hard constraints must be satisfied. On the other hand, the soft constraints may be violated, but it will be penalized with each violation based on the significance of that soft constraint. Each faculty member is initially given a score of 1.0. With each violation of soft constraints for a particular faculty member, the score decreases by a certain percentage. This percentage differs from one soft constraint to another. For our work, we have defined five hard constraints and six soft constraints. These are mentioned below:

### Hard Constraints

- For each theory section, one faculty member must be assigned to that particular section. No theory section can exist without having a faculty member.
- For each lab section, one to three faculty members must be assigned to that section based on the number of students. As we did not deal with student information data, we have assigned two faculty members to each lab section to make things simpler. No lab section can exist without having at least two faculty members.
- No faculty member should have more than one class in each slot. In other words, there cannot be any slot clash for a faculty member.

- A particular faculty member can not take courses exceeding his or her maximum credit limit.
- Each faculty member has a preferred list of courses of his or her choice. No faculty member should be given courses outside of their own preferred list.

### Soft Constraints

- A faculty member should work a maximum of 5 out of 6 working days in a week. If this is violated, then a 40% deduction will happen from the total score of that faculty member.
- If a faculty member is given more than 4 slots in a day, then there will be a 35% deduction from the score of that faculty member.
- If a faculty member is given 4 consecutive slots in a day, then the score will be penalized by 40% for that particular faculty member.
- This constraint deals with idle time minimization for each faculty member. A detailed overview of how the penalization works is given below:
  1. For 4 slots assigned in a day:
    - i) If a faculty member has a gap of 2 consecutive slots (3 hours) in a day, there will be a penalty of 15% from the score of that faculty member.
    - ii) If a faculty member has a gap of 1 slot (1.5 hours) twice in a day, there will be a penalty of 10% from the score of that faculty member.
  2. For 3 slots assigned in a day:
    - i) If a faculty member has a gap of 3 consecutive slots (4.5 hours) in a day, there will be a penalty of 15% from the score of that faculty member.
    - ii) If a faculty member has a gap of 1 slot (1.5 hours) along with a gap of 2 consecutive slots (3 hours) in a day, there will be penalty of 10% from the score of that faculty member.
    - iii) If a faculty member has a gap of 1 slot (1.5 hours) twice in a day, there will be a penalty of 5% from the score of that faculty member.
  3. For 2 slots assigned in a day:
    - i) If a faculty member has a gap of 4 consecutive slots (6 hours) in a day, there will be a penalty of 25% from the score of that faculty member.
    - ii) If a faculty member has a gap of 3 consecutive slots (4.5 hours) in a day, there will be a penalty of 20% from the score of that faculty member.
- Faculty members with a maximum credit limit of 6 or 9 credits, should not be given any lab classes. If this constraint is violated, then a 30% deduction from the total score will happen for that faculty member.

- Faculty members with a maximum credit limit of 6 or 9 credits, should not be given any slot before 11:00 am in a day. The penalty in this case will be 30% if an 08:00 am class is given and for a 09:30 am class, it will be 20% from the score of that faculty member.

### 4.3 Implementation Details

We proposed an idea regarding university course allocation, prepared our dataset which was used in implementing the idea and identified the constraints that need to be satisfied during the implementation. In this part we are going to discuss how the idea was implemented. Three classes were designed for our work, ‘Slot’, ‘Course’ and ‘Faculty’. In Slot class, there are three attributes named day, start\_time and end\_time. Day denotes the day of the week for a particular slot. Start\_time denotes the starting time of the slot and end\_time denotes the ending time of that slot. The Course class has three class variables - theory\_faculty, lab\_faculty1, lab\_faculty2. The theory\_faculty holds the faculty member initial for a particular theory class, lab\_faculty1 and lab\_faculty2 holds the two faculty member initials for a particular lab section. This class also has five attributes - name, section, credit, slot1, slot2. The name attribute denotes the course code of a particular course, section denotes the section number of that course, credit denotes the credit hours for the course, slot1 and slot2 denotes the two slots in a week for that particular course. The Faculty class has eight attributes named name, initial, preferred\_courses, maximum\_credit\_limit, initial\_credit, initial\_slots, current\_credit\_given and current\_routine. The name attribute denotes the name of a particular faculty member, initial denotes the initial of that faculty member, preferred\_courses denotes the preferred course list for every faculty member, maximum\_credit\_limit denotes the maximum credits that a faculty member can take, initial\_credit and initial\_slot were used as temporary variables to check hard constraints while generating the initial solution, current\_credit\_given denotes the total number of credits that has been given to a faculty member and current\_routine denotes the routine of a faculty member after being assigned to courses.

In constraint based problems, researchers generally use some conventional fitness functions for evaluating the generated result of a particular iteration. This is used to identify whether the result is improving from previous iterations or not. These fitness functions are used for cost calculation in these constraint based problems. Unlike other researches, we did not use any of the conventional fitness functions to evaluate our solutions. After analysing the case study, we decided it would be much more feasible to use a score-based evaluation approach. We designed a function where we checked each and every hard and soft constraints. Initially we set the score to 1.0 for each faculty member and after assigning course to them we checked for any hard constraint violation at first. We have a total of five hard constraints as we mentioned in the previous section, if any of the hard constraints were violated for a faculty member, the score 0.0 was returned for the whole function. If none of the hard constraints were violated, the six soft constraints were checked for violation in case of each faculty member. We penalized a certain percentage of score from the total score of that faculty member which was set to 1.0 initially. If the score became less than 0.0, it was assigned as 0.0. Otherwise the function calculated the

score for an iteration which was remaining for a faculty member after checking the soft constraints. After generating individual scores of all the faculty members, the average of all these scores was calculated and the final rounded score was returned, which worked as the evaluating factor. Thus, all the constraints were checked using this evaluation function replacing the conventional approaches. This function has been represented in equation (4.1).

$$S_E = \begin{cases} 0 & \text{if } \exists(P_{hc}) : P_{hc} = 0, \text{ where } 1 \leq hc \leq a \\ \frac{\sum_{i=1}^{n_f} s_i}{n_f} & \text{otherwise} \end{cases} \quad (4.1)$$

Here,  $S_E$  denotes the calculated score for the evaluation function.  $P_{hc}$  denotes penalty for  $hc^{\text{th}}$  hard constraint violation. The range of  $hc$  is from 1 to maximum number of hard constraints (a).  $s_i$  denotes the individual assignment score for  $i^{\text{th}}$  faculty member. Finally,  $n_f$  denotes the total number of faculty members for a particular semester. The individual assignment score for the  $i^{\text{th}}$  faculty member can be calculated using equation (4.2).

$$s_i = \begin{cases} 1 - \sum_{sc=1}^b P_{sc} & \text{if } 0 \leq \sum_{sc=1}^b P_{sc} \leq 1 \\ 0 & \text{if } \sum_{sc=1}^b P_{sc} < 0 \end{cases} \quad (4.2)$$

Here  $P_{sc}$  denotes penalty for  $sc^{\text{th}}$  soft constraint violation. The range of  $sc$  is from 1 to total number of soft constraints (b).

In our work, four optimization algorithms were used to solve the DCAP and one additional algorithm to determine the initial solution that was used for further optimization in each algorithm. The optimization algorithms are CSP Solver Algorithm, Genetic Algorithm, Simulated Annealing and Hybrid Algorithm. All these algorithms were written in Python 3, using Google Colab environment. Several iterations were tested for each algorithm to check when the score was not significantly updating. The values for maximum iterations were fixed near to that point. The value for maximum iterations varied from algorithm to algorithm.

### 4.3.1 Generate Initial Solution

As it was mentioned before, an additional algorithm was used for determining the initial solution for each optimization algorithm. It was designed based on the requirements of our problem. Initially, the score corresponding to the solution was set to 0. A loop was used for determining a solution until the score was greater than 0. Then a dictionary was generated where all the course codes were set as the keys and the initials of all the faculty members taking those courses were arranged into a list. Each list was set as the values corresponding to the particular keys. Then in each iteration of the loop, the solution was set empty and it was gradually filled with course-faculty elements while looping through the course list. A single course-faculty element includes 3 elements separated by dash (\_). The first element is the index of a course from the course list. The second element is denoted by 0, 1 or 2,

where 0 means theory faculty, 1 means the first lab faculty and 2 means the second lab faculty. The third element denotes the faculty index from the faculty list. This list was then assigned to the initial solution once finished populating. While assigning the elements all the hard constraints were checked for the faculty member with respect to the current course. The maximum credit limit for that faculty member was checked at first, whether it exceeded the limit or not. Also, the slot clash in that faculty member's routine was checked. The same process was executed until a faculty member was assigned for the current course-faculty element. After obtaining a successful element, it was added to the initial solution list. When the loop ended, the initial solution list was populated with every possible course-faculty elements. The score of this list was found using the evaluation function that was mentioned in equation (4.1). Then the initial solution list was returned, which was used as the initial solution for the four optimization algorithms of each dataset. Algorithm 1 gives an overview of what has been described above.

---

**Algorithm 1** Generate Initial Solution

---

```

function GENERATE_INITIAL_SOLUTION( )
    score  $\leftarrow$  0
    while score == 0 do
        initial_solution  $\leftarrow$  {}
        for c in course_list do
            elem  $\leftarrow$  allocate a randomized faculty in c satisfying hard constraints
            initial_solution  $\leftarrow$  initial_solution  $\cup$  elem
        end for
        score  $\leftarrow$  evaluate(initial_solution)
    end while
    return initial_solution
end function

```

---

### 4.3.2 CSP Solver Algorithm

The initial solution obtained from Algorithm 1 was assigned to the solution variable at the beginning of this algorithm. Then, the faculties were allocated in the courses based on the initial solution elements. The evaluation score of the assignment was initialized in the *max\_score* variable. The dictionary mentioned in the previous section was used in this algorithm as well. This was used to find out the suitable faculties for a particular course. In the while loop, the optimization process starts where a course-faculty element is generated randomly. A suitable faculty member would later replace the current assigned faculty for that element. The assignment was done based on the second portion of the element, which decided whether the faculty member was going to be the theory faculty or the first or second lab faculty for that course. This process was done in another loop inside the current loop where the random element that was selected, was removed from the current solution. Then a new course-faculty element was created using the currently split course from the removed element and the current faculty in the loop. This new element was then added to the current solution list. The current solution list including all course-faculty elements was assigned similarly as mentioned before. The evaluation function (4.1) was used to generate the score for current assignment.

---

**Algorithm 2** CSP Solver Algorithm

---

```
function CSP_SOLVER(initial_solution)
  solution  $\leftarrow$  initial_solution
  assign(solution)
  max_score  $\leftarrow$  evaluate(solution)
  while iteration < max_iteration do
    selected  $\leftarrow$  pick a random course_faculty element from solution
    unselected  $\leftarrow$  solution - selected
    for f in all possible faculties of course_name of selected do
      selected  $\leftarrow$  replace faculty of selected with f
      if evaluate(solution) < evaluate(unselected  $\cup$  selected) then
        solution  $\leftarrow$  unselected  $\cup$  selected
        max_score  $\leftarrow$  evaluate(unselected  $\cup$  selected)
      end if
    end for
    iteration  $\leftarrow$  iteration + 1
  end while
  return solution
end function
```

---

Then a comparison of the current score and the previous score was done to check if the current score was better or not. If it was better, then the current score was considered as the maximum score and the current solution was considered as the best solution. Otherwise, the previous score and solution were kept unchanged. After the inner loop ended, the maximum score and the best solution were generated. The iteration variable was incremented by 1 after each iteration was completed. This was done for all the course-faculty elements, until all the courses were assigned to the suitable faculty members. Each time this assignment process took place, all the previous assignments were reset before a new assignment occurred. Using the evaluation function, a score was generated for the current assignment of courses. The outer loop was terminated when the maximum number of iterations was reached. Thus, the maximum score and the best solution for this algorithm was found. Algorithm 2 gives an overview of what has been described so far.

### 4.3.3 Genetic Algorithm

The initial solution obtained from Algorithm 1 was assigned to a variable at the beginning of this algorithm as well. After the initial assignment process, the evaluation function was used to generate the score for current assignment. A loop was then started with the condition of stopping the algorithm when the maximum number of generations was reached. The selection function returned two variables named *selected* and *unselected* where *selected* refers to some of the elements that were selected from the population and the rest was stored in the *unselected* variable. The course-faculty elements in the *unselected* variable were then assigned as mentioned previously. The crossover operation was then performed on the selected course-faculty elements. In this operation, a few course-faculty elements were taken and the faculty index was then interchanged between them. After this operation ended, a new list was found which replaced the previous selected list.

---

**Algorithm 3** Genetic Algorithm

---

```
function GENETIC(initial_solution)
  solution  $\leftarrow$  initial_solution
  assign(solution)
  max_score  $\leftarrow$  evaluate(solution)
  while generation < max_iteration do
    selected, unselected  $\leftarrow$  selection(solution)
    assign(unselected)
    selected  $\leftarrow$  crossover(selected)
    selected  $\leftarrow$  mutation(selected)
    assign(selected)
    score  $\leftarrow$  evaluate(selected  $\cup$  unselected)
    if score > max_score then
      max_score  $\leftarrow$  score
      solution  $\leftarrow$  selected  $\cup$  unselected
    end if
    generation  $\leftarrow$  generation + 1
  end while
  return solution
end function
```

---

The mutation operation was then performed on this new selected list. A random course-faculty element was selected from the list at the beginning of this operation. Then another index from the faculty list was chosen randomly and this index was used to replace the faculty index of the course-faculty element. This changed list was returned at the end of this operation, and it later functioned as the new selected list. This selected list was then assigned as mentioned before. The score was generated using the evaluation function afterwards. The new score and the maximum score were compared to check if the new score was better or not. If it was better, the maximum score was replaced with the new score and the best solution was replaced with the current solution. The current solution was then updated as the combination of both the selected and unselected course-faculty elements. The generation was incremented by 1 after each iteration was completed. Once the maximum iteration was reached, the best solution for this algorithm was returned. Algorithm 3 gives an overview of what has been described so far.

#### 4.3.4 Simulated Annealing

The temperature reduction function  $\alpha$  and the initial temperature  $T$  were initialized at the beginning of this algorithm. For the DCAP,  $\alpha$  was chosen to be 0.9 and  $T$  was chosen to be 1.0. The value of  $\alpha$  can range from 0.8-0.99. After testing with different values of  $\alpha$  such as 0.825, 0.85, 0.9, 0.95 and 0.98, the scores generated by 0.9 were comparatively better than the others. For this reason, the value of  $\alpha$  was chosen as 0.9. The initial solution obtained from Algorithm 1 was assigned to a variable at the beginning of this algorithm too. The evaluation function (4.1) was called to calculate the score of the initial solution. This was stored as the old cost. This old cost was assigned to the *max\_score* variable initially. A loop was started with the termination condition of ending the algorithm when



the number of iterations exceeded the maximum number of iterations. A random course-faculty element was selected from the initial solution, which was stored in the selected variable. The rest of the elements were put in a list named unselected. The change\_neighbor operation was then performed on this chosen element. The course-faculty element was selected from the variable at the beginning of this operation. Then another index from the faculty list was chosen randomly and this index was used to replace the faculty index of the course-faculty element. The assign operation was executed as before by combining both the selected and unselected parts. The score of this assignment was calculated using the evaluation function again. This score was considered as the new cost. If this new cost was found to be better than the old cost, then the new cost was set as the maximum score and the current solution was updated with the combination of both the selected and unselected parts. A function named acceptance\_probability was called with the parameters old\_cost, new\_cost and  $T$ . In this function, we checked whether the new cost was less than the old cost. If so, then the value of probability variable was set as 1.0. If not, then the value of the probability was calculated using  $e^{-\frac{\Delta c}{T}}$ . Here  $\Delta c$  denotes the difference between new cost and old cost and  $T$  denotes the temperature. This value was stored in the probability variable.

---

**Algorithm 4** Simulated Annealing

---

```

function SIMULATED(initial_solution)
  Initialize alpha & T
  solution  $\leftarrow$  initial_solution
  assign(solution)
  old_cost  $\leftarrow$  evaluate(solution)
  max_score  $\leftarrow$  old_cost
  while iteration < max_iteration do
    s  $\leftarrow$  pick a random element from solution
    selected  $\leftarrow$  change_neighbor(solution, s)
    unselected  $\leftarrow$  solution - selected
    assign(selected  $\cup$  unselected)
    new_cost  $\leftarrow$  evaluate(selected  $\cup$  unselected)
    if new_cost > old_cost then
      max_score  $\leftarrow$  new_cost
      solution  $\leftarrow$  selected  $\cup$  unselected
    end if
    probability  $\leftarrow$  acceptance_probability(old_cost, new_cost, T)
    if probability < random value between 0 to 1 then
      old_cost  $\leftarrow$  new_cost
    end if
    T  $\leftarrow$  decrease T by the rate of alpha
    iteration  $\leftarrow$  iteration + 1
  end while
  return solution
end function

```

---

A random value between 0 to 1 was chosen which was used to check whether it was greater than the value of the probability variable. If this was true, then the old

cost was replaced by the new cost that was calculated before. The value of  $T$  was reduced by multiplying  $alpha$  with  $T$ . The iteration variable was also incremented by 1 once an iteration was completed. Finally, the solution of this algorithm was returned after the loop ended. Algorithm 4 gives an overview of what has been described so far.

### 4.3.5 Hybrid Algorithm

The hybrid algorithm implemented in this work is a combination of GA and SA. The temperature reduction function  $alpha$  and the initial temperature  $T$  were initialized at the beginning of this algorithm, similar to SA.  $Alpha$  was chosen to be 0.9 and  $T$  was chosen to be 1.0. The initial solution obtained from Algorithm 1 was assigned to a variable at the beginning of this algorithm.

---

#### Algorithm 5 Hybrid Algorithm

---

```

function HYBRID(initial_solution)
  Initialize alpha & T
  solution  $\leftarrow$  initial_solution
  assign(solution)
  max_score  $\leftarrow$  evaluate(solution)
  old_cost  $\leftarrow$  max_score
  while iteration < max_iteration do
    selected, unselected  $\leftarrow$  selection(solution)
    assign(unselected)
    selected  $\leftarrow$  crossover(selected)
    selected  $\leftarrow$  mutation(selected)
    assign(selected)
    new_cost  $\leftarrow$  evaluate(selected  $\cup$  unselected)
    if new_cost > old_cost then
      max_score  $\leftarrow$  new_cost
      solution  $\leftarrow$  selected  $\cup$  unselected
      reset switching_factor to 0
    else
      increment switching_factor by 1
    end if
    if switching_factor > maximum_tolerance_value then
      probability  $\leftarrow$  acceptance_probability(old_cost, new_cost, T)
      if probability < random value between 0 to 1 then
        old_cost  $\leftarrow$  new_cost
      end if
      T  $\leftarrow$  decrease T by the rate of alpha
    end if
    iteration  $\leftarrow$  iteration + 1
  end while
  return solution
end function

```

---

The evaluation function (4.1) was called to calculate the score of the initial solution. This was stored as the maximum score. This maximum score was assigned to the `old_cost` variable initially. Similar to the GA & SA, a loop was started with the condition of ending the algorithm when the number of iterations exceeded the maximum number of iterations. The same selection function from GA was called with the current solution where the current solution was divided into two parts, selected and unselected. The partial assignment was done using the unselected part of the current solution. The crossover and mutation operations of GA were also used here on the selected part. The assign function was again called with the selected part of the current solution. The score, which was considered to be the new cost of this assignment was then calculated using the evaluation function. If this new cost was better than the maximum score, then the new cost was set as the maximum score and the current solution was updated with the combination of both the selected and unselected parts. The main difference of this algorithm with the GA is that an additional operation was performed, which was to reset the switching factor to zero. Switching factor was introduced in this algorithm to shift from GA to SA when the algorithm was stuck in the local maxima for a long time. This switching factor was incremented by 1 when the value of new cost was not updated. When the value of switching factor reached a maximum tolerance value (in our case it was 20), the algorithm was considered to be stuck at the local maxima for a certain amount of time. Then the algorithm was shifted from GA to SA and the value of acceptance probability was calculated as mentioned in SA. The rest of the operations such as checking the probability to be smaller than a random value between 0 to 1 was carried out like SA. If the condition was satisfied then the old cost was replaced by the new cost. Then  $T$  was decreased using *alpha*. The value of iterating variable was increased by 1 once an iteration was complete. After the loop ended, the solution of this algorithm was returned which was the combination of both the selected and unselected lists. Algorithm 5 gives an overview of what has been described so far.

# Chapter 5

## Results

In this chapter, the outputs of the implemented algorithms will be analyzed and discussed. After generating the solutions for each algorithms, courses will be assigned to the particular faculty members based on the solutions. We can visualize the output in four different ways. Besides, the quality of the results of different algorithms on different datasets can be further analyzed via tables and graphs.

### 5.1 Visualization of Output

#### 5.1.1 Generating Current Semester Routine

The class routines for every sections with the assigned theory and lab faculty members can be generated after the completion of each algorithm. Figure 5.1 shows a snapshot of the routine of Spring 2020 after the successful execution of an optimization algorithm.

```
CSE421 Sec 1 Theory, Time: Monday: 11:00 am - 12:20 pm & Wednesday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: ARF
CSE421 Sec 2 Theory, Time: Thursday: 08:00 am - 09:20 am & Saturday: 08:00 am - 09:20 am, Credit: 3.0, Theory Faculty: ARF
CSE421 Sec 3 Theory, Time: Thursday: 09:30 am - 10:50 am & Saturday: 09:30 am - 10:50 am, Credit: 3.0, Theory Faculty: SAF
CSE421 Sec 4 Theory, Time: Thursday: 12:30 pm - 01:50 pm & Saturday: 12:30 pm - 01:50 pm, Credit: 3.0, Theory Faculty: ARF
CSE421 Sec 5 Theory, Time: Thursday: 03:30 pm - 04:50 pm & Saturday: 03:30 pm - 04:50 pm, Credit: 3.0, Theory Faculty: SRJ
CSE421 Sec 6 Theory, Time: Thursday: 09:30 am - 10:50 am & Saturday: 09:30 am - 10:50 am, Credit: 3.0, Theory Faculty: FSI
CSE421 Sec 7 Theory, Time: Thursday: 12:30 pm - 01:50 pm & Saturday: 12:30 pm - 01:50 pm, Credit: 3.0, Theory Faculty: SKZ
CSE421 Sec 8 Theory, Time: Sunday: 11:00 am - 12:20 pm & Tuesday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: SRJ
CSE421 Sec 9 Theory, Time: Thursday: 11:00 am - 12:20 pm & Saturday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: SAF
CSE421 Sec 10 Theory, Time: Thursday: 02:00 pm - 03:20 pm & Saturday: 02:00 pm - 03:20 pm, Credit: 3.0, Theory Faculty: SAF
CSE422 Sec 1 Theory, Time: Thursday: 11:00 am - 12:20 pm & Saturday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: MGR
CSE422 Sec 2 Theory, Time: Thursday: 03:30 pm - 04:50 pm & Saturday: 03:30 pm - 04:50 pm, Credit: 3.0, Theory Faculty: C10
CSE422 Sec 3 Theory, Time: Sunday: 11:00 am - 12:20 pm & Tuesday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: C58
CSE422 Sec 4 Theory, Time: Sunday: 02:00 pm - 03:20 pm & Tuesday: 02:00 pm - 03:20 pm, Credit: 3.0, Theory Faculty: ZAR
CSE422 Sec 6 Theory, Time: Thursday: 08:00 am - 09:20 am & Saturday: 08:00 am - 09:20 am, Credit: 3.0, Theory Faculty: TRZ
CSE422 Sec 7 Theory, Time: Monday: 12:30 pm - 01:50 pm & Wednesday: 12:30 pm - 01:50 pm, Credit: 3.0, Theory Faculty: SZH
CSE422 Sec 8 Theory, Time: Sunday: 08:00 am - 09:20 am & Tuesday: 08:00 am - 09:20 am, Credit: 3.0, Theory Faculty: MZP
CSE423 Sec 1 Theory, Time: Thursday: 08:00 am - 09:20 am & Saturday: 08:00 am - 09:20 am, Credit: 3.0, Theory Faculty: C52
CSE423 Sec 2 Theory, Time: Thursday: 09:30 am - 10:50 am & Saturday: 09:30 am - 10:50 am, Credit: 3.0, Theory Faculty: FSH
CSE423 Sec 3 Theory, Time: Thursday: 11:00 am - 12:20 pm & Saturday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: FSH
CSE423 Sec 4 Theory, Time: Sunday: 08:00 am - 09:20 am & Tuesday: 08:00 am - 09:20 am, Credit: 3.0, Theory Faculty: C52
CSE423 Sec 5 Theory, Time: Thursday: 09:30 am - 10:50 am & Saturday: 09:30 am - 10:50 am, Credit: 3.0, Theory Faculty: DMH
CSE423 Sec 6 Theory, Time: Thursday: 12:30 pm - 01:50 pm & Saturday: 12:30 pm - 01:50 pm, Credit: 3.0, Theory Faculty: C52
CSE423 Sec 7 Theory, Time: Thursday: 02:00 pm - 03:20 pm & Saturday: 02:00 pm - 03:20 pm, Credit: 3.0, Theory Faculty: DMH
CSE423 Sec 8 Theory, Time: Monday: 08:00 am - 09:20 am & Wednesday: 08:00 am - 09:20 am, Credit: 3.0, Theory Faculty: DMH
CSE460 Sec 11 Theory, Time: Monday: 03:30 pm - 04:50 pm & Wednesday: 03:30 pm - 04:50 pm, Credit: 3.0, Theory Faculty: TAA
CSE460 Sec 12 Theory, Time: Thursday: 11:00 am - 12:20 pm & Saturday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: TAA
CSE460 Sec 13 Theory, Time: Thursday: 03:30 pm - 04:50 pm & Saturday: 03:30 pm - 04:50 pm, Credit: 3.0, Theory Faculty: TAA
```

Figure 5.1: Snapshot of Semester Routine for All Departmental Courses

## 5.1.2 Extracting Individual Course Section Information

To get the information of a particular section of a course, the inputs are course name and section. The output of the theory and lab information (if exists) will be displayed like Figure 5.2.

```
Enter Course: CSE221
Enter Section: 2
CSE221 Sec 2 Theory, Time: Sunday: 09:30 am - 10:50 am & Tuesday: 09:30 am - 10:50 am, Credit: 3.0, Theory Faculty: MMM
CSE221 Sec 2 Lab, Time: Wednesday: 08:00 am - 09:20 am & Wednesday: 09:30 am - 10:50 am, Credit: 1.5, Lab Faculties: AHR & SRF
```

Figure 5.2: Snapshot of Individual Course Information

## 5.1.3 Finding All Faculty Members Information

The algorithm results provide the updated information of existing departmental faculty members of a particular semester. Figure 5.3 refers to the snapshot of all faculty members information on Spring 2020 dataset after successful assignments.

```
Name: Arnisha Khondaker, Initial: ARK
Preferred Courses: CSE220T, CSE220L, CSE221L, CSE260L, CSE320T, CSE321L, CSE330L, CSE341L, CSE370L, CSE420L, CSE421L, CSE423L
Max Credit Limit: 12.0, Current Credit Given: 12.0
Current Routine:
CSE260 Sec 9 Lab, Time: Sunday: 02:00 pm - 03:20 pm & Sunday: 03:30 pm - 04:50 pm, Credit: 1.5, Lab Faculties: C55 & ARK
CSE471 Sec 7 Lab, Time: Saturday: 11:00 am - 12:20 pm & Saturday: 12:30 pm - 01:50 pm, Credit: 1.5, Lab Faculties: C32 & ARK
CSE220 Sec 9 Theory, Time: Monday: 11:00 am - 12:20 pm & Wednesday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: ARK
CSE220 Sec 11 Theory, Time: Monday: 03:30 pm - 04:50 pm & Wednesday: 03:30 pm - 04:50 pm, Credit: 3.0, Theory Faculty: ARK
CSE320 Sec 4 Theory, Time: Sunday: 09:30 am - 10:50 am & Tuesday: 09:30 am - 10:50 am, Credit: 3.0, Theory Faculty: ARK

Name: Arif Shakil, Initial: ARF
Preferred Courses: CSE421T, CSE421L
Max Credit Limit: 12.0, Current Credit Given: 12.0
Current Routine:
CSE421 Sec 4 Theory, Time: Thursday: 12:30 pm - 01:50 pm & Saturday: 12:30 pm - 01:50 pm, Credit: 3.0, Theory Faculty: ARF
CSE421 Sec 5 Lab, Time: Tuesday: 11:00 am - 12:20 pm & Tuesday: 12:30 pm - 01:50 pm, Credit: 1.5, Lab Faculties: AKH & ARF
CSE421 Sec 7 Lab, Time: Monday: 08:00 am - 09:20 am & Monday: 09:30 am - 10:50 am, Credit: 1.5, Lab Faculties: C43 & ARF
CSE421 Sec 2 Theory, Time: Thursday: 08:00 am - 09:20 am & Saturday: 08:00 am - 09:20 am, Credit: 3.0, Theory Faculty: ARF
CSE421 Sec 1 Theory, Time: Monday: 11:00 am - 12:20 pm & Wednesday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: ARF

Name: Dr Abu Mohammad Khan, Initial: AMK
Preferred Courses: CSE330T
Max Credit Limit: 9.0, Current Credit Given: 9
Current Routine:
CSE330 Sec 1 Theory, Time: Monday: 12:30 pm - 01:50 pm & Wednesday: 12:30 pm - 01:50 pm, Credit: 3.0, Theory Faculty: AMK
CSE330 Sec 2 Theory, Time: Monday: 03:30 pm - 04:50 pm & Wednesday: 03:30 pm - 04:50 pm, Credit: 3.0, Theory Faculty: AMK
CSE330 Sec 9 Theory, Time: Sunday: 03:30 pm - 04:50 pm & Tuesday: 03:30 pm - 04:50 pm, Credit: 3.0, Theory Faculty: AMK
```

Figure 5.3: Snapshot of All CSE Faculty Members Information

## 5.1.4 Extracting Individual Faculty Information

One can also get the individual faculty information of a particular semester by giving the faculty initial as the input (Figure 5.4). The individual faculty assignment score can also be obtained from the output. This score denotes how efficiently the courses have been allotted to that particular faculty member.

```
Enter Initial: FSH
Name: Farhana Shahid, Initial: FSH
Preferred Courses: CSE310L, CSE260T, CSE422L, CSE423T, CSE423L
Max Credit Limit: 12.0, Current Credit Given: 12.0
Current Routine:
CSE423 Sec 3 Theory, Time: Thursday: 11:00 am - 12:20 pm & Saturday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: FSH
CSE423 Sec 2 Theory, Time: Thursday: 09:30 am - 10:50 am & Saturday: 09:30 am - 10:50 am, Credit: 3.0, Theory Faculty: FSH
CSE260 Sec 17 Theory, Time: Monday: 11:00 am - 12:20 pm & Wednesday: 11:00 am - 12:20 pm, Credit: 3.0, Theory Faculty: FSH
CSE310 Sec 1 Lab, Time: Tuesday: 02:00 pm - 03:20 pm & Tuesday: 03:30 pm - 04:50 pm, Credit: 1.5, Lab Faculties: AAR & FSH
CSE422 Sec 1 Lab, Time: Monday: 02:00 pm - 03:20 pm & Monday: 03:30 pm - 04:50 pm, Credit: 1.5, Lab Faculties: FSH & C02
```

Figure 5.4: Snapshot of Individual Faculty Member Information

## 5.2 Results on Different Datasets

### 5.2.1 Spring 2020 Dataset

#### i) CSP Solver :

The CSP solver algorithm was run five times with 5000 iterations each. The output for those five tests are organized in Table 5.1. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.8383	2021	273.23
2	0.8341	4750	646.58
3	<b>0.8466</b>	3317	511.99
4	0.8182	2306	760.06
5	0.8167	2006	303.21

Table 5.1: Different Test Results of CSP Solver on Spring 2020 Dataset

From the table, it is seen that Test 3 produced the maximum score with 3317 iterations and in 511.99 seconds. Test 5 generated the final score with the least number of iterations and Test 1 finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.5 below.

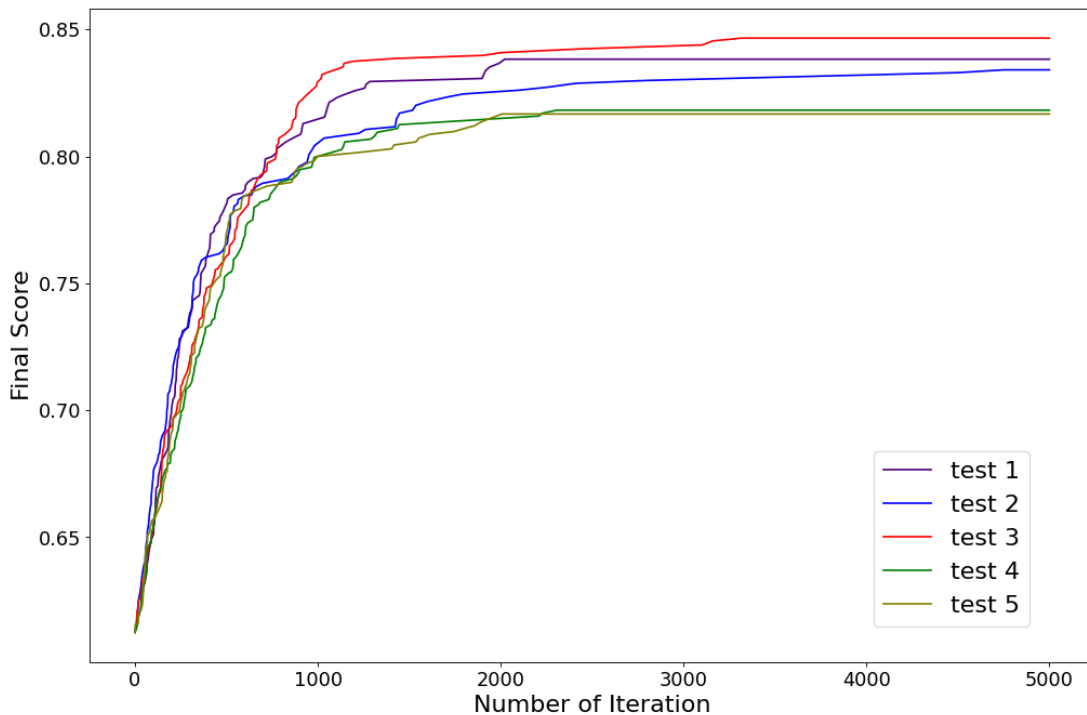


Figure 5.5: Results of CSP Solver on Spring 2020 Dataset

## ii) Genetic Algorithm :

The genetic algorithm was run five times with 2000000 iterations each. The output for those five tests are organized in Table 5.2. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	<b>0.8989</b>	1976297	6226.66
2	0.8886	1799938	5671.76
3	0.8780	1944908	6171.18
4	0.8856	1943705	6248.45
5	0.8920	1851671	5973.85

Table 5.2: Different Test Results of Genetic Algorithm on Spring 2020 Dataset

From the table, it is seen that Test 1 produced the maximum score with 1976297 iterations and in 6226.66 seconds. Test 2 generated the final score with the least number of iterations and finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.6 below.

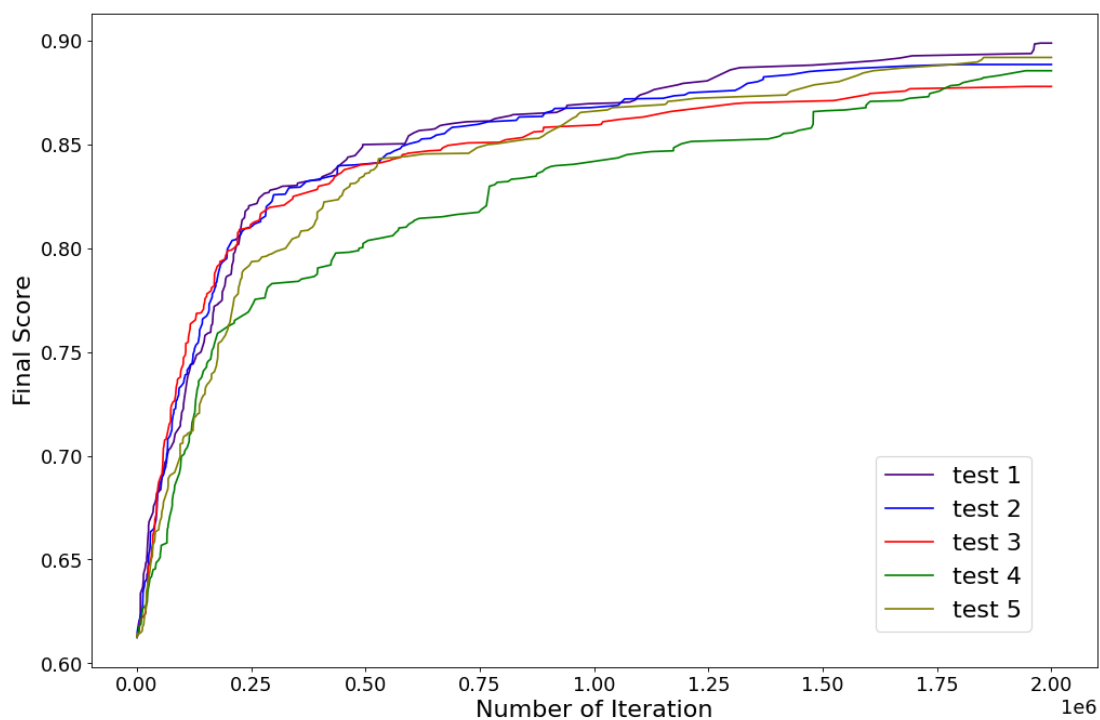


Figure 5.6: Results of Genetic Algorithm on Spring 2020 Dataset

### iii) Simulated Annealing :

The simulated annealing algorithm was run five times with 500000 iterations each. The output for those five tests are organized in Table 5.3. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.8424	309742	1230.69
2	0.8386	355115	1367.47
3	<b>0.8443</b>	465861	1801.88
4	0.8288	425074	1640.98
5	0.8405	432003	1669.25

Table 5.3: Different Test Results of Simulated Annealing on Spring 2020 Dataset

From the table, it is seen that Test 3 produced the maximum score with 465861 iterations and in 1801.88 seconds. Test 1 generated the final score with the least number of iterations and finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.7 below.

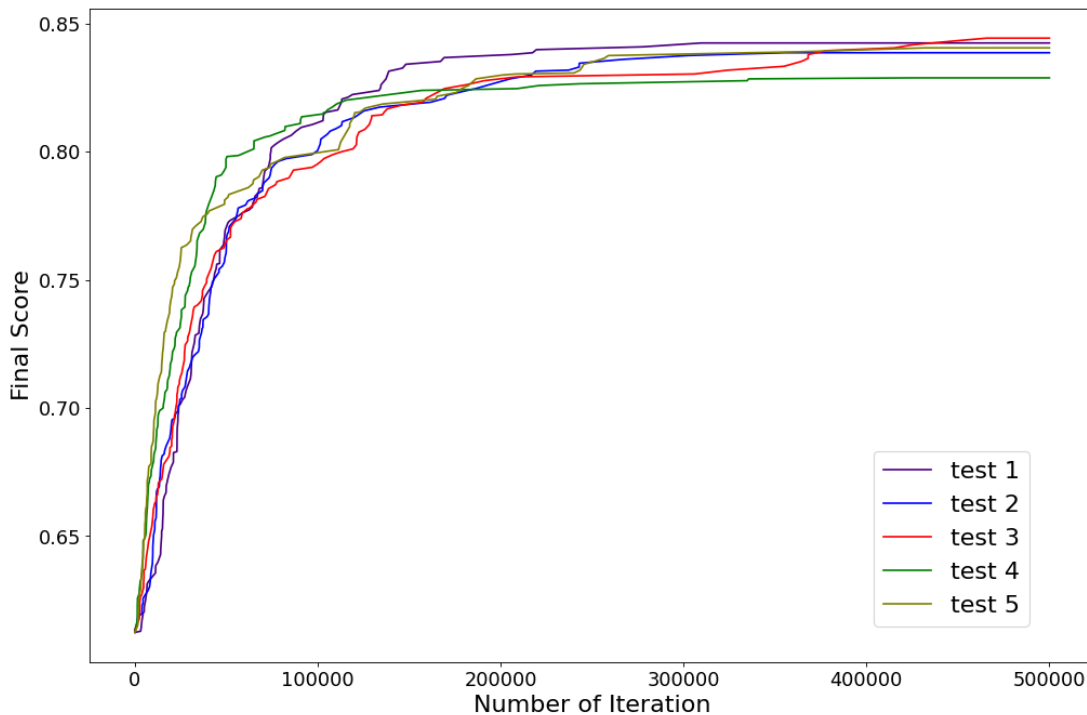


Figure 5.7: Results of Simulated Annealing Algorithm on Spring 2020 Dataset



#### iv) Hybrid Algorithm (GA + SA) :

The hybrid algorithm combining genetic algorithm and simulated annealing was run five times with 2000000 iterations each. The output for those five tests are organized in Table 5.4. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	<b>0.9114</b>	1966504	6197.17
2	0.8966	1922478	6029.10
3	0.8848	1995777	5963.78
4	0.8879	1963305	5736.98
5	0.8822	1927330	5686.91

Table 5.4: Different Test Results of Hybrid Algorithm on Spring 2020 Dataset

From the table, it is seen that Test 1 produced the maximum score with 1966504 iterations and in 6197.17 seconds. Test 2 generated the final score with the least number of iterations and Test 5 finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.8 below.

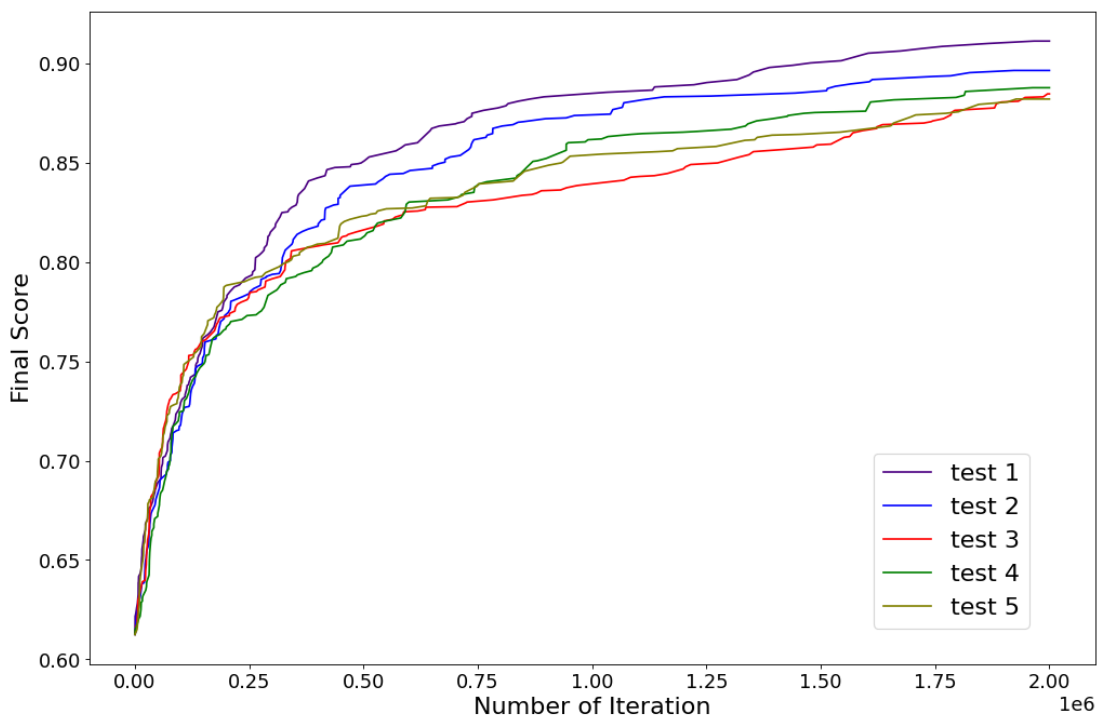


Figure 5.8: Results of Hybrid Algorithm on Spring 2020 Dataset

## 5.2.2 Summer 2020 Dataset

### i) CSP Solver :

The CSP solver algorithm was run five times with 5000 iterations each. The output for those five tests are organized in Table 5.5. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.8481	2165	171.01
2	0.8398	2435	193.68
3	0.8437	2744	218.98
4	0.8442	2249	176.30
5	<b>0.8689</b>	2394	192.24

Table 5.5: Different Test Results of CSP Solver on Summer 2020 Dataset

From the table, it is seen that Test 5 produced the maximum score with 2394 iterations and in 192.24 seconds. Test 1 generated the final score with the least number of iterations and finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.9 below.

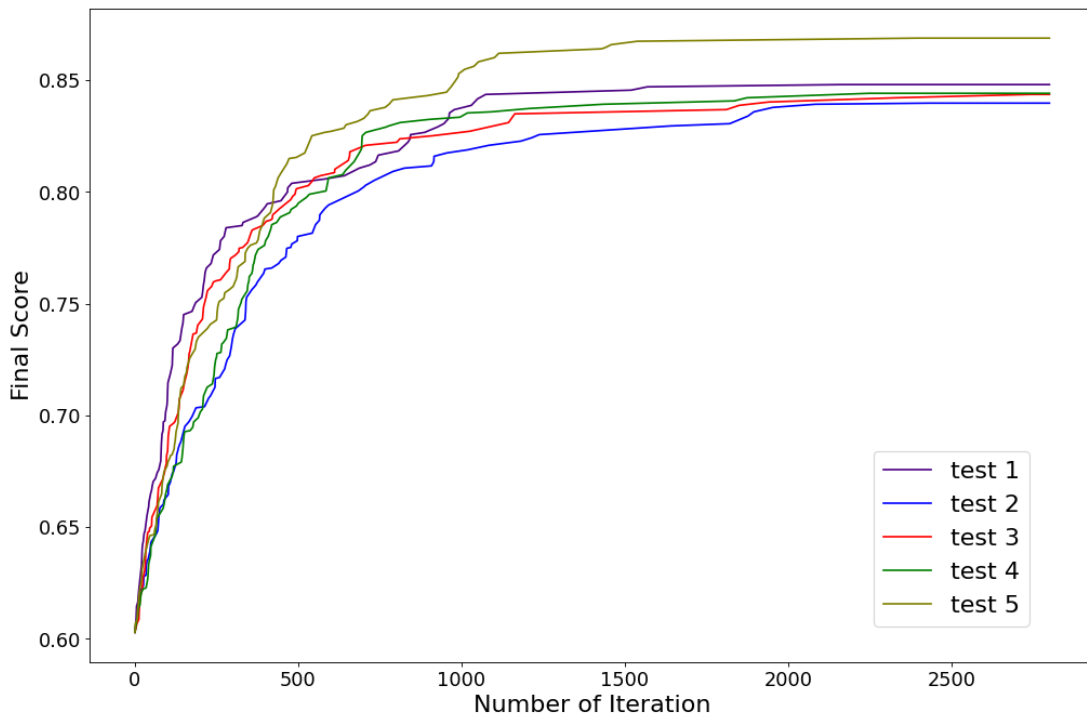


Figure 5.9: Results of CSP Solver on Summer 2020 Dataset

**ii) Genetic Algorithm :**

The genetic algorithm was run five times with 2000000 iterations each. The output for those five tests are organized in Table 5.6. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.8961	1813553	5007.55
2	<b>0.9175</b>	1772387	4171.75
3	0.9015	1972441	4645.52
4	0.8947	1789801	4231.74
5	0.8932	1935510	4592.8

Table 5.6: Different Test Results of Genetic Algorithm on Summer 2020 Dataset

From the table, it is seen that Test 2 produced the maximum score with 1772387 iterations and in 4171.75 seconds. Test 2 also generated the final score with the least number of iterations and finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.10 below.

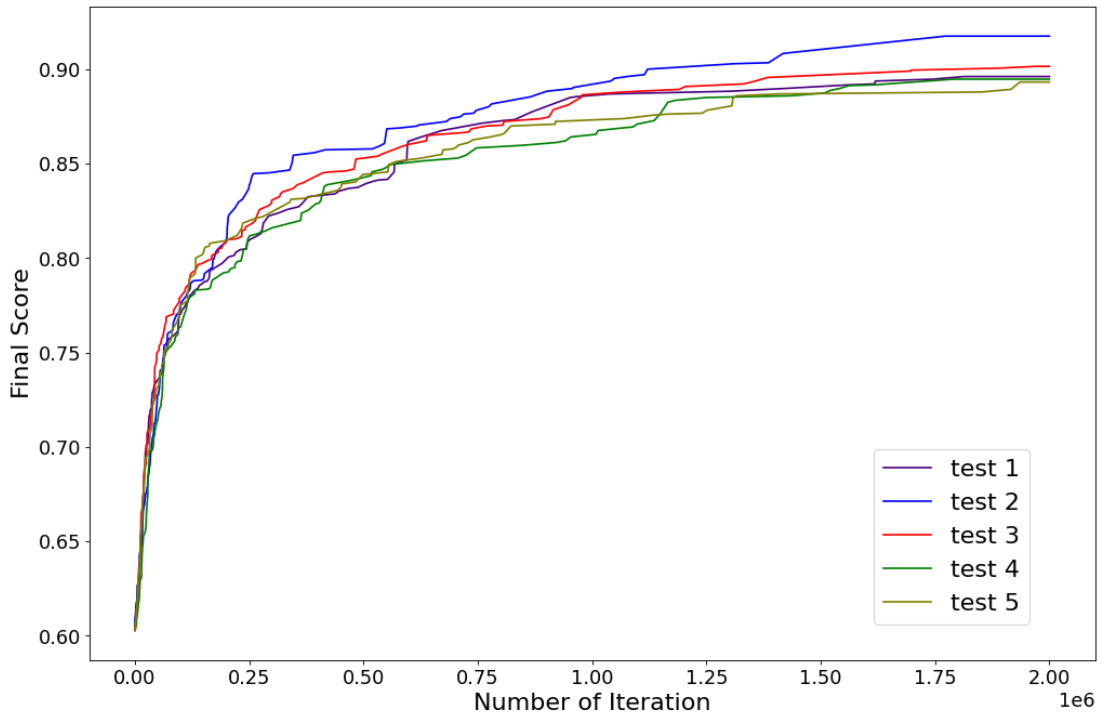


Figure 5.10: Results of Genetic Algorithm on Summer 2020 Dataset

### iii) Simulated Annealing :

The simulated annealing algorithm was run five times with 500000 iterations each. The output for those five tests are organized in Table 5.7. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.8461	283470	847.24
2	0.8291	207708	615.39
3	0.8388	172610	519.57
4	<b>0.8524</b>	440758	1309.18
5	0.8369	123680	365.8

Table 5.7: Different Test Results of Simulated Annealing on Summer 2020 Dataset

From the table, it is seen that Test 4 produced the maximum score with 440758 iterations and in 1309.18 seconds. Test 5 generated the final score with the least number of iterations and finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.11 below.

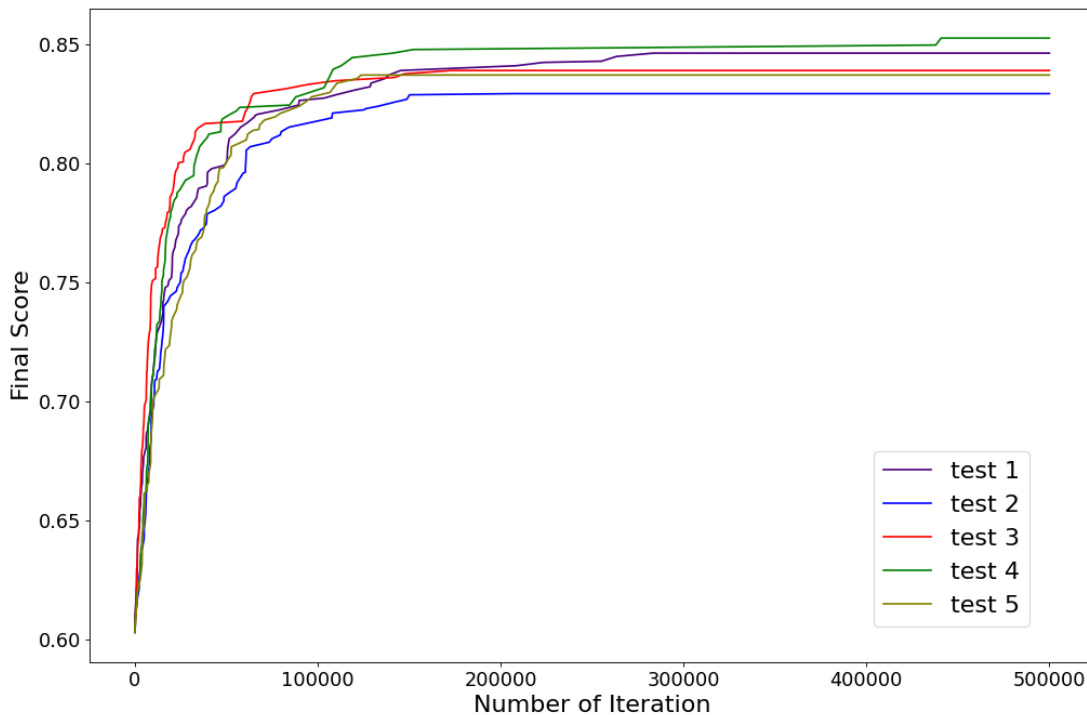


Figure 5.11: Results of Simulated Annealing Algorithm on Summer 2020 Dataset

#### iv) Hybrid Algorithm (GA + SA) :

The hybrid algorithm combining genetic algorithm and simulated annealing was run five times with 2000000 iterations each. The output for those five tests are organized in Table 5.8. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.9068	1893114	4633.96
2	0.9199	1926830	4270.59
3	<b>0.9238</b>	1975527	4378.62
4	0.9112	1789973	3996.59
5	0.9146	1958317	4406.36

Table 5.8: Different Test Results of Hybrid Algorithm on Summer 2020 Dataset

From the table, it is seen that Test 3 produced the maximum score with 1975527 iterations and in 4378.62 seconds. Test 4 generated the final score with the least number of iterations and finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.12 below.

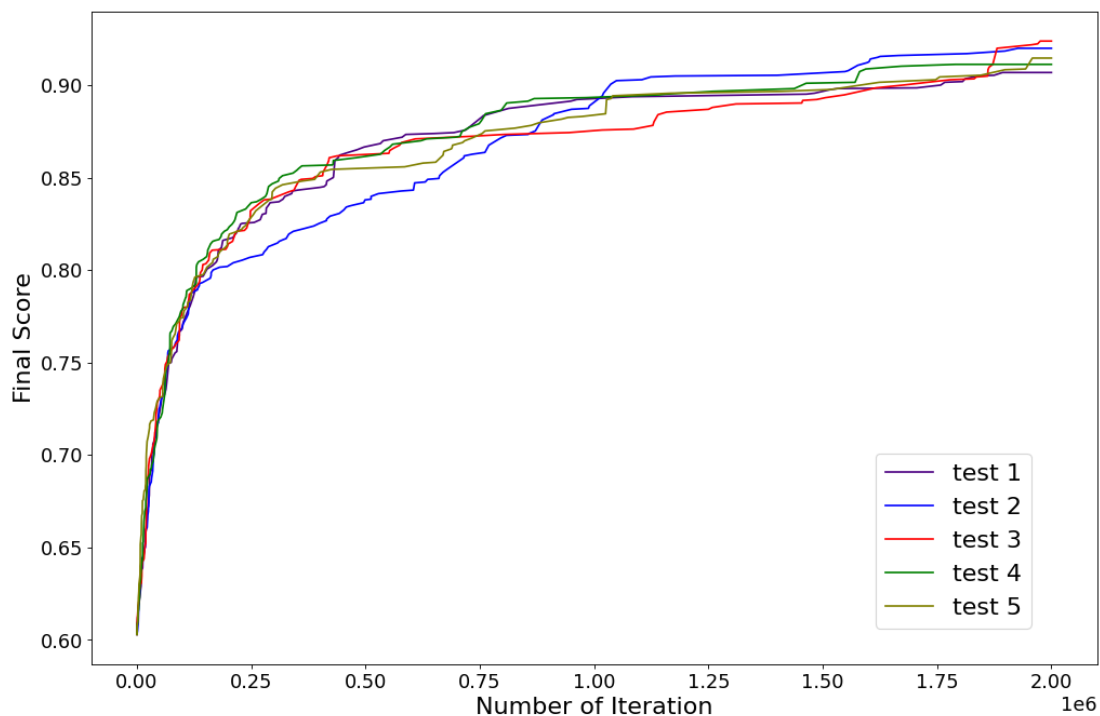


Figure 5.12: Results of Hybrid Algorithm on Summer 2020 Dataset

### 5.2.3 Fall 2020 Dataset

#### i) CSP Solver :

The CSP solver algorithm was run five times with 5000 iterations each. The output for those five tests are organized in Table 5.9. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.9436	1642	124.68
2	0.9465	1691	132.84
3	0.9411	1823	145.41
4	0.9475	2666	206.88
5	<b>0.9480</b>	1786	141.24

Table 5.9: Different Test Results of CSP Solver on Fall 2020 Dataset

From the table, it is seen that Test 5 produced the maximum score with 1786 iterations and in 141.24 seconds. Test 1 generated the final score with the least number of iterations and finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.13 below.

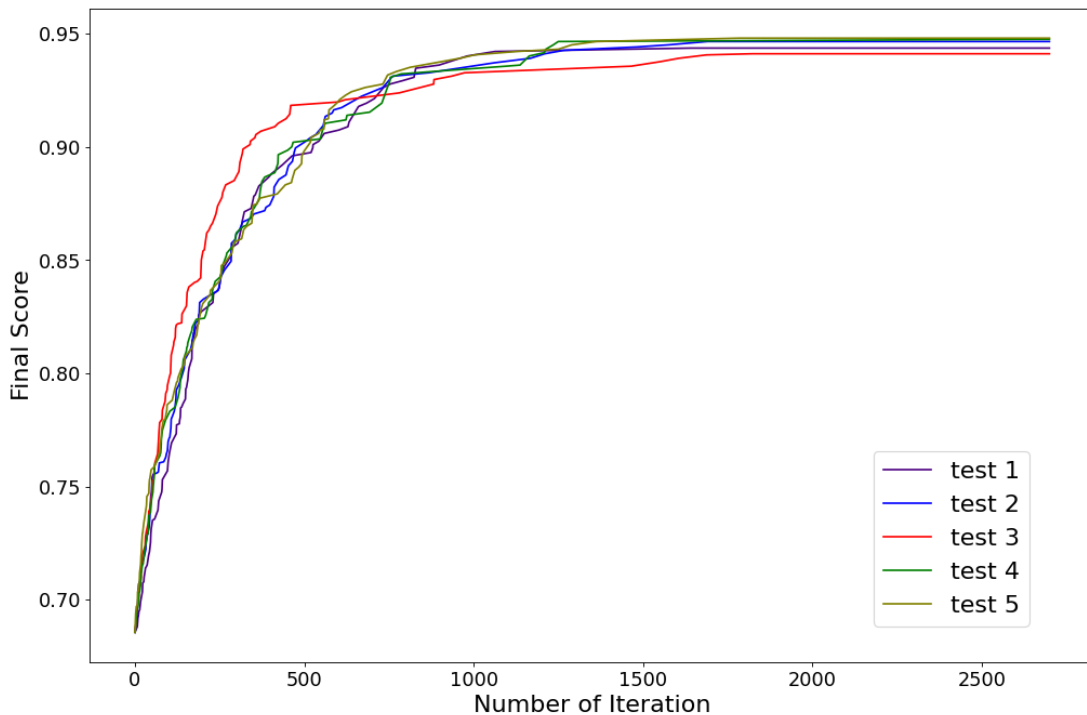


Figure 5.13: Results of CSP Solver on Fall 2020 Dataset

## ii) Genetic Algorithm :

The genetic algorithm was run five times with 2000000 iterations each. The output for those five tests are organized in Table 5.10. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.9584	1971371	4455.78
2	0.9579	1779945	3851.80
3	0.9634	1955789	4220.41
4	<b>0.9693</b>	1848219	3925.83
5	0.9639	1903073	4547.18

Table 5.10: Different Test Results of Genetic Algorithm on Fall 2020 Dataset

From the table, it is seen that Test 4 produced the maximum score with 1848219 iterations and in 3925.83 seconds. Test 2 generated the final score with the least number of iterations and finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.14 below.

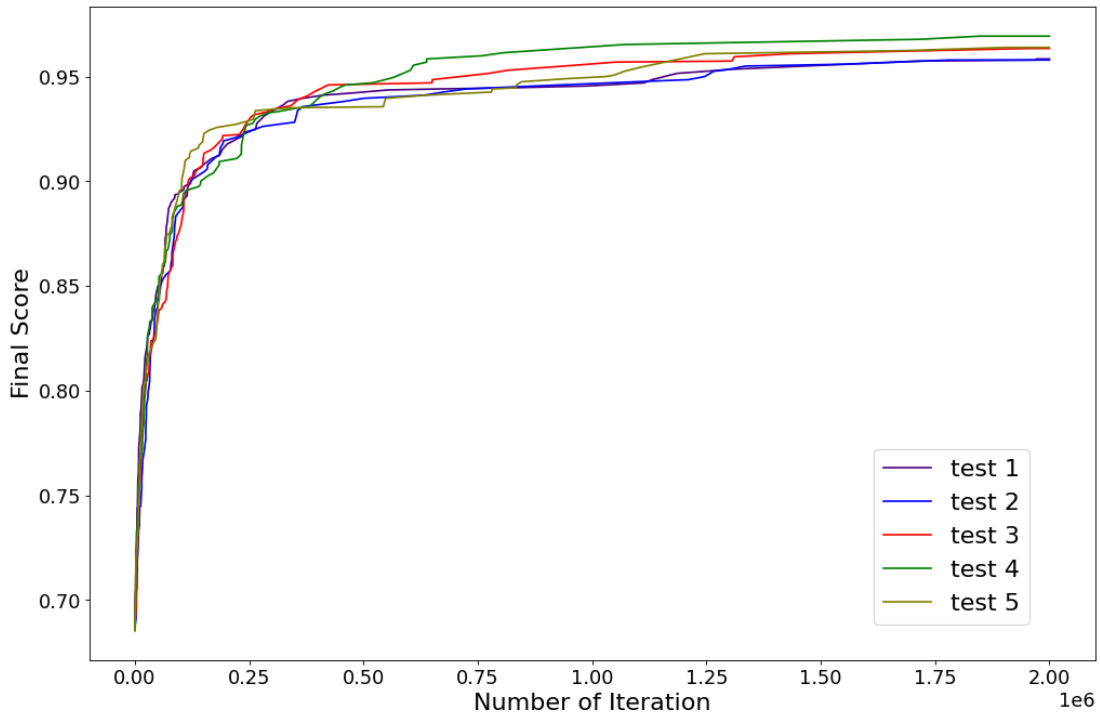


Figure 5.14: Results of Genetic Algorithm on Fall 2020 Dataset

### iii) Simulated Annealing :

The simulated annealing algorithm was run five times with 500000 iterations each. The output for those five tests are organized in Table 5.11. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.9193	106837	286.37
2	<b>0.9490</b>	245312	686.06
3	0.9307	72506	200.01
4	0.9277	163086	452.14
5	0.9228	69155	206.42

Table 5.11: Different Test Results of Simulated Annealing on Fall 2020 Dataset

From the table, it is seen that Test 2 produced the maximum score with 245312 iterations and in 686.06 seconds. Test 5 generated the final score with the least number of iterations and Test 3 finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.15 below.

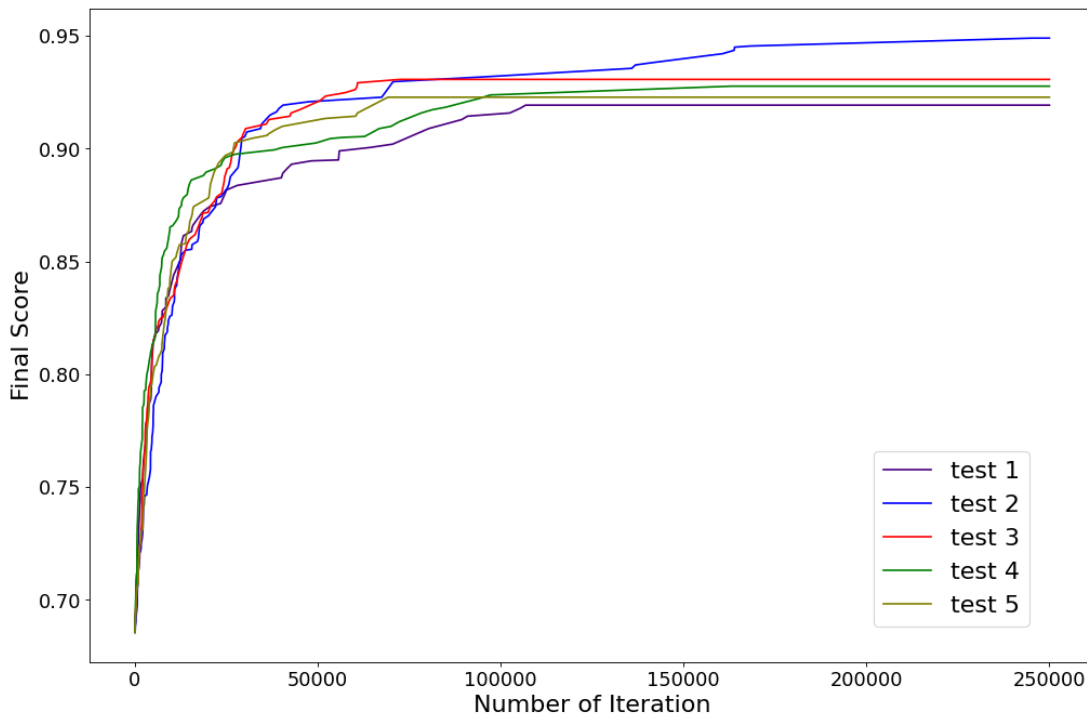


Figure 5.15: Results of Simulated Annealing Algorithm on Fall 2020 Dataset



#### iv) Hybrid Algorithm (GA + SA) :

The hybrid algorithm combining genetic algorithm and simulated annealing was run five times with 2000000 iterations each. The output for those five tests are organized in Table 5.12. The table includes the final score, iteration taken to reach that score and total time consumed to produce the final score.

Test No.	Final Score	Last Iteration	Total Time (Seconds)
1	0.9599	1489865	3310.09
2	0.9574	1705342	3792.93
3	0.9530	1047164	2078.64
4	<b>0.9738</b>	1261267	2601.99
5	0.9594	1919098	3819.14

Table 5.12: Different Test Results of Hybrid Algorithm on Fall 2020 Dataset

From the table, it is seen that Test 4 produced the maximum score with 1261267 iterations and in 2601.99 seconds. Test 3 generated the final score with the least number of iterations and finished execution earlier than the other tests. A score vs iteration graph can be generated with the help of the test results like Figure 5.16 below.

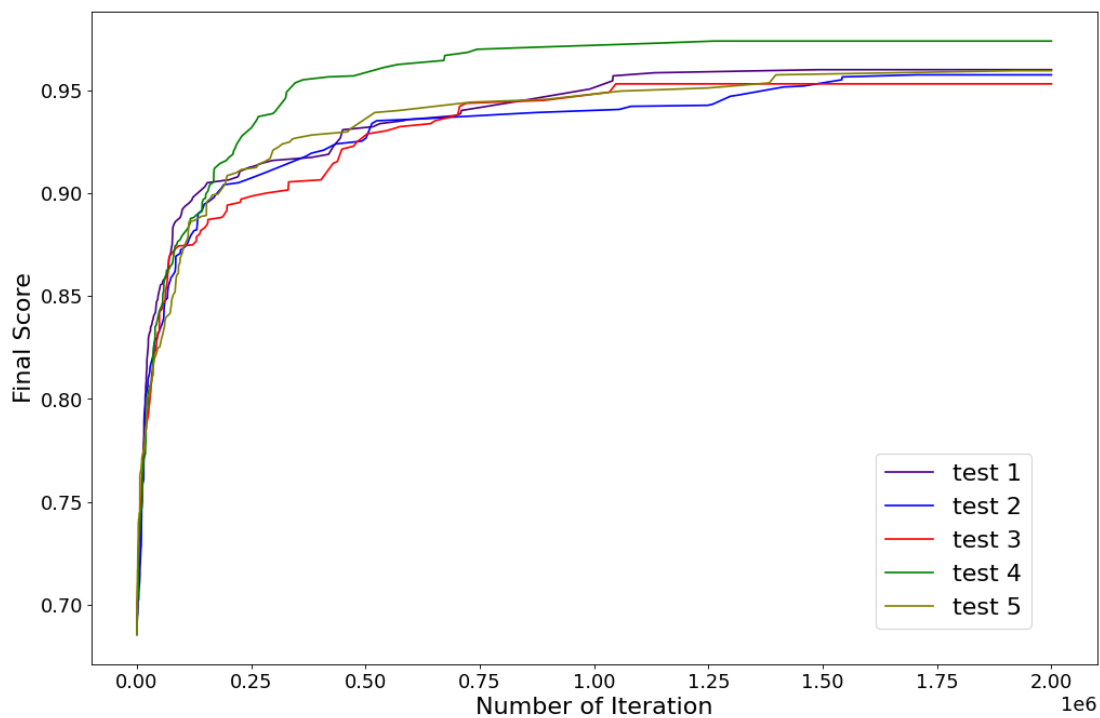


Figure 5.16: Results of Hybrid Algorithm on Fall 2020 Dataset

# Chapter 6

## Analysis

This chapter shows different algorithm performances on each dataset and discusses a comparative description in the later part.

### 6.1 Overall Analysis

In this section, the best score from each algorithm is analyzed and visualized using graphs and column charts. This score shows the efficiency of the algorithms in terms of all the available faculty member assignments.

#### 6.1.1 Spring 2020 Dataset

After running the four different algorithms on Spring 2020 Dataset, the best results from each algorithm are referred in Table 6.1 below.

Algorithm	Best Score	Total Time (Seconds)
CSP Solver	0.8466	511.99
GA	0.8989	6226.66
SA	0.8443	1801.88
HA	0.9114	6197.17

Table 6.1: Different Implementation Results on Spring 2020 Dataset

From the table, it is clearly seen that HA produced the best output among all four. GA also performed pretty well but both these algorithms were very much time consuming. It took more than 1.5 hours to complete the 2000000 iterations. SA finished execution in around 30 minutes and produced the lowest score. The least time consuming algorithm in this dataset was CSP Solver but it generated the second lowest score among all the algorithms.

Figure 6.1 shows the score vs time graph of these four algorithms. It is seen that the iterations of CSP Solver and SA finished much earlier than the other two and produced the final score. But in GA and HA the score gradually increased over time and outperformed the other two algorithms in terms of the final score.

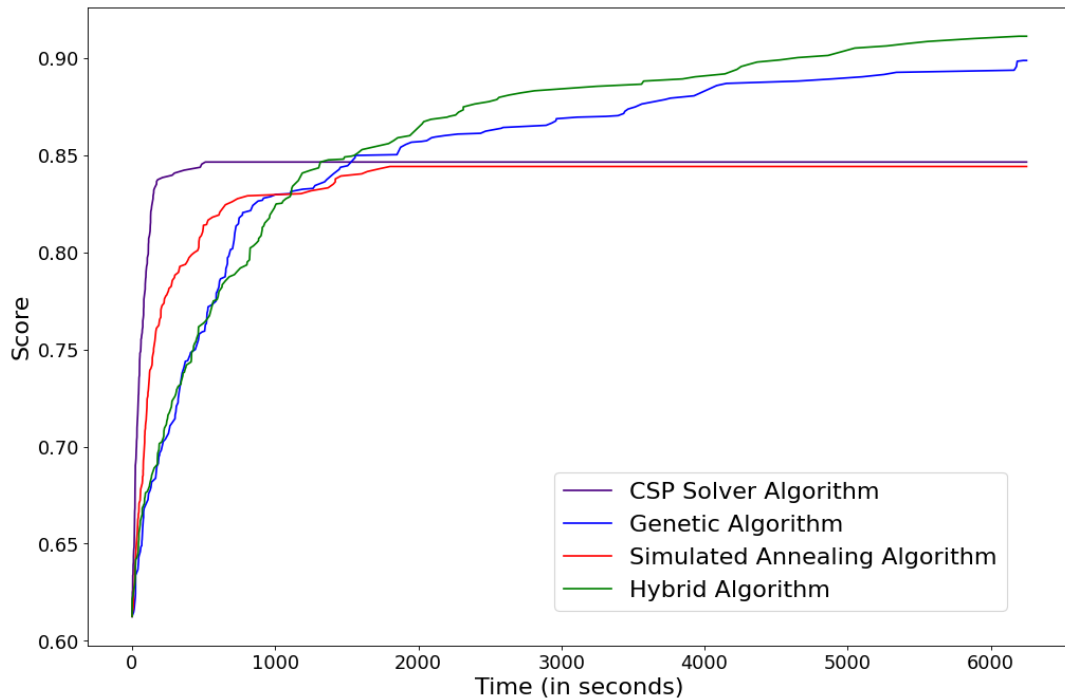


Figure 6.1: Time vs Score Comparison Between Different Algorithm Results on Spring 2020 Dataset

As it was previously mentioned, the evaluation function calculates the average score of all the individual faculty member assignment scores. For each algorithm, these assignment scores should be different. From Figure 6.2 we can clearly visualize the variations of the assignment score frequencies for all four algorithms.

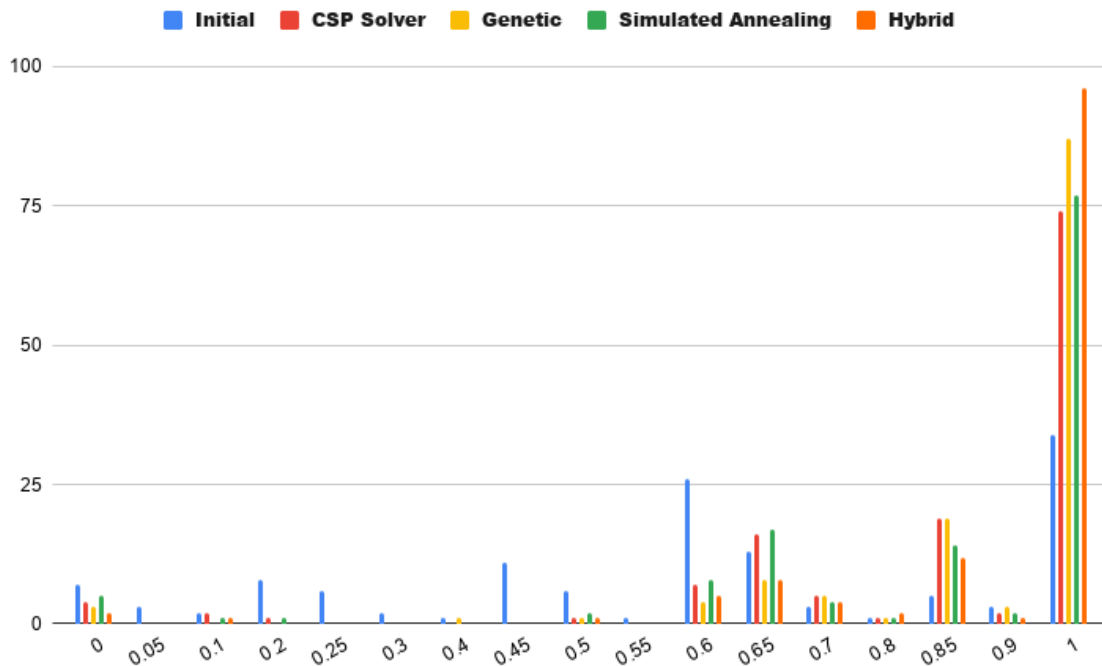


Figure 6.2: Score Frequency Variations Between Different Algorithms on Spring 2020 Dataset

## 6.1.2 Summer 2020 Dataset

The best results produced from each algorithm for Summer 2020 Dataset are referred in Table 6.2 below.

Algorithm	Best Score	Total Time (Seconds)
CSP Solver	0.8689	192.24
GA	0.9175	4171.75
SA	0.8524	1309.18
HA	0.9238	4378.62

Table 6.2: Different Implementation Results on Summer 2020 Dataset

It is clearly seen that HA produced the best output among all four algorithms. GA also performed pretty well but both these algorithms were very much time consuming like the previous dataset. It took almost 1.25 hours to complete the 2000000 iterations. SA finished execution in around 20 minutes but produced the lowest score. The least time consuming algorithm in this dataset was CSP Solver but it generated the second lowest score among all the algorithms.

Figure 6.3 shows the score vs time graph of these four algorithms. It is seen that the iterations of CSP Solver and SA finished much earlier than the other two and produced the final score. But in GA and HA the score gradually increased over time and outperformed the other two algorithms in terms of the final score.

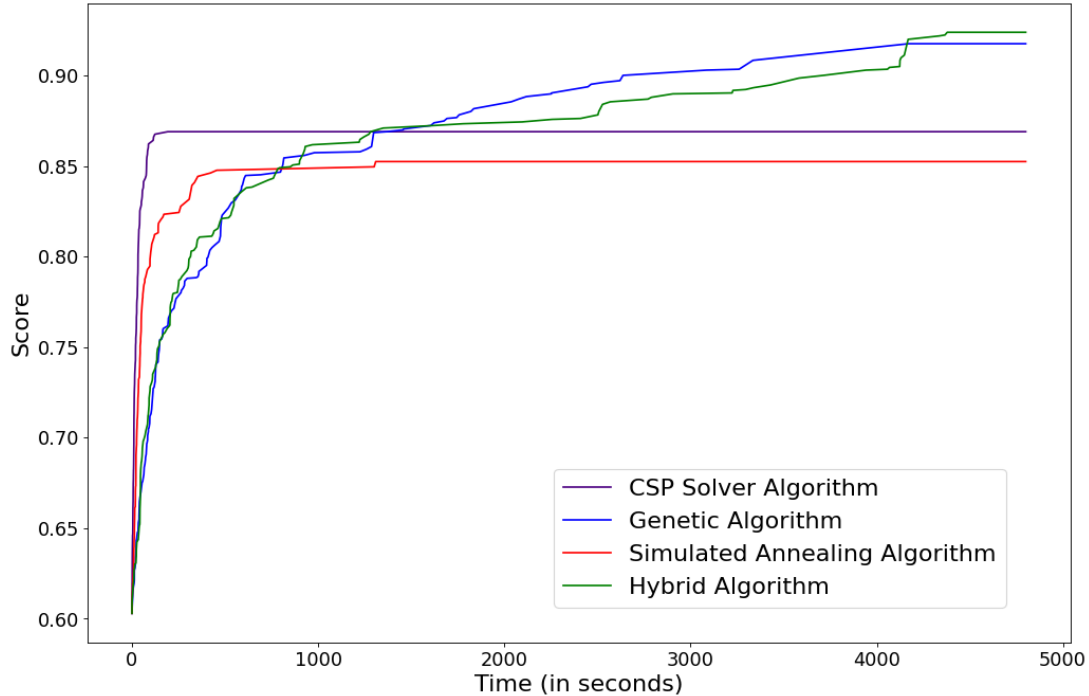


Figure 6.3: Time vs Score Comparison Between Different Algorithm Results on Summer 2020 Dataset

Like the previous score frequency graph, from Figure 6.4 we can also visualize the variations of the different assignment score frequencies for all four algorithms.

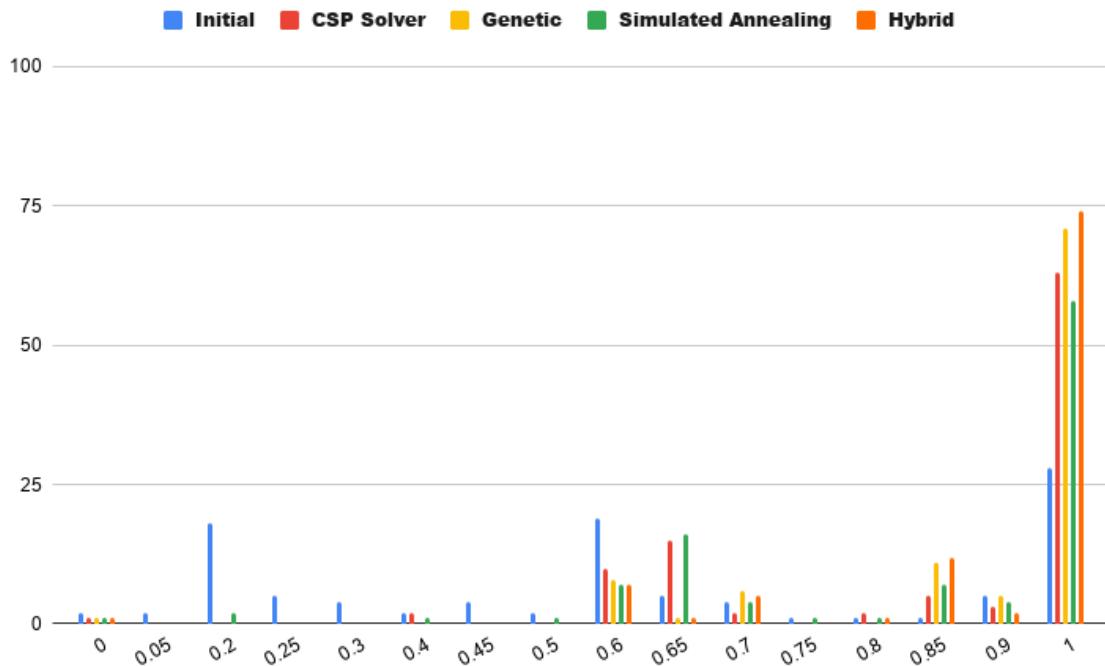


Figure 6.4: Score Frequency Variations Between Different Algorithms on Summer 2020 Dataset

### 6.1.3 Fall 2020 Dataset

The best results produced from each algorithm for Fall 2020 Dataset are referred in Table 6.3 below.

Algorithm	Best Score	Total Time (Seconds)
CSP Solver	0.9475	206.88
GA	0.9693	3925.83
SA	0.9490	686.06
HA	0.9738	2601.99

Table 6.3: Different Implementation Results on Fall 2020 Dataset

It is visible that HA produced the best output among all four algorithms here as well. GA performed pretty well too but both these algorithms were very much time consuming like the previous two datasets. It took more than 1 hour to complete the 2000000 iterations for GA and for HA it took around 45 minutes to produce the best output. SA and CSP Solver finished execution earlier than the other two algorithms but generated lower scores than the others.

Figure 6.5 shows the score vs time graph of these four algorithms. The characteristics of the graph are pretty much similar to the other two score vs time graphs.

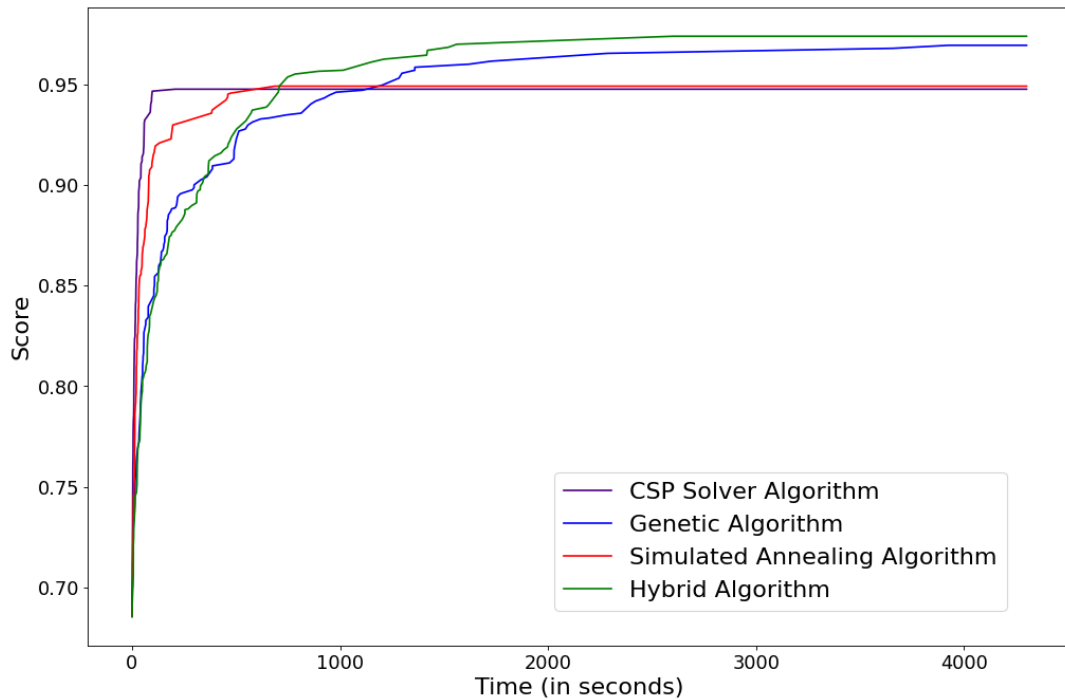


Figure 6.5: Time vs Score Comparison Between Different Algorithm Results on Fall 2020 Dataset

Figure 6.6 shows the variations of the different assignment score frequencies for all algorithms. A major noticeable thing here is that in all four algorithms, the frequency of score 1 is much superior compared to the other score frequencies.

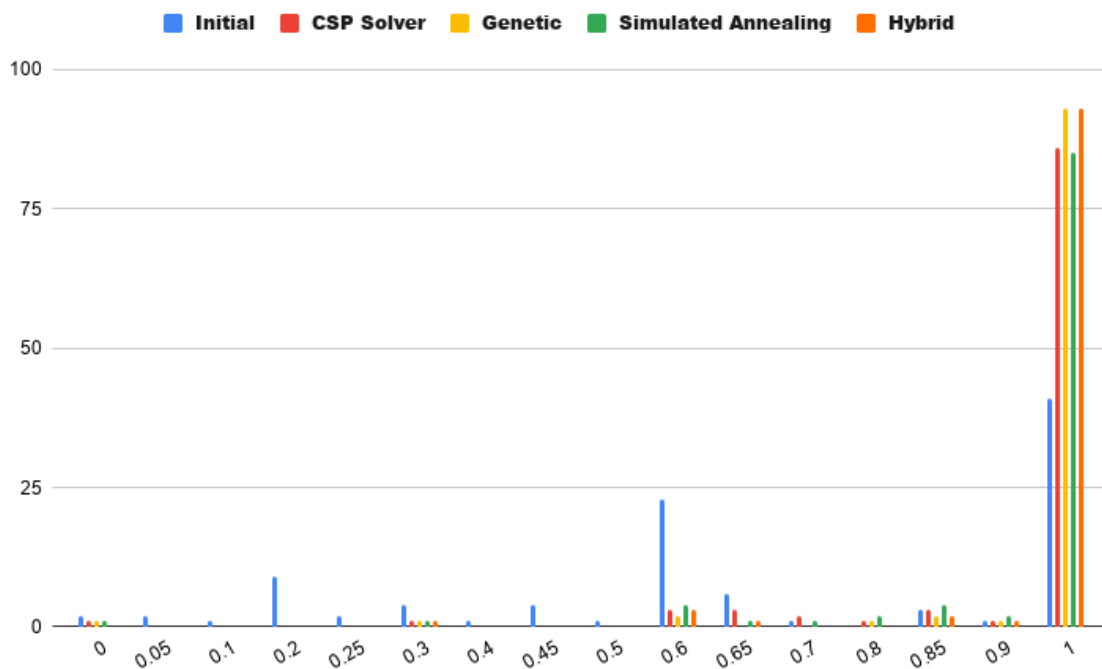


Figure 6.6: Score Frequency Variations Between Different Algorithms on Fall 2020 Dataset

## 6.2 Comparison Between Different Algorithms

To compare between the characteristics of different algorithms, it is necessary to study the patterns of each dataset first. Table 6.4 shows the properties of the three datasets used in this research.

Dataset Name	Type	Number of Sections	Number of Faculty Members
Spring 2020	Large	458	132
Summer 2020	Medium	355	103
Fall 2020	Small	318	101

Table 6.4: Datasets Comparison

In Spring 2020 dataset pair, both the number of sections and the number of faculty members were higher than the other two. We labeled these as the largest dataset pair in this research. Due to the Covid-19 pandemic, Summer 2020 and Fall 2020 were held online. That is why the number of sections and faculty members decreased significantly. As in Summer 2020, there were more sections compared to Fall 2020, we considered these dataset as medium and thus Fall 2020 dataset pair was labeled as the small dataset in this research. The implementation results for each algorithm differed based on these type of datasets.

The less time consuming algorithm in our research was CSP Solver algorithm. In all the three dataset pairs, this algorithm gave the final score faster than the others. But the scores were not the best compared to others. As previously mentioned, at each iteration the algorithm assigns only one faculty to each course keeping rest of the elements fixed. As this algorithm did not have any evolutionary steps like crossover and mutation in GA, there was lesser chance to assign the fittest element to the solution at each iteration. It only assigned the faculty in the element which was considered best according to the previous updated solution.

The Genetic Algorithm gave the second best result in our research. It gave higher score than CSP Solver and SA. But it took a lot of time to produce the high score compared to others. If we look at the previous section, we will see in the implementation result tables, the total time consumed for GA was between 1 to 1.5 hours based on the size of the dataset pair. The main reason behind this was that the score updates were not frequent. The score updated gradually after each successful crossover and mutation, thus it took a lot of iterations to reach the final score. Another major drawback of this algorithm was it got stuck in the local maxima for a long time. This means that after a certain period of time the iteration difference between the previous score and updated score was lot higher. This problem was solved in the Hybrid Algorithm where a tolerance value was fixed so that the algorithm can switch to another approach.

Simulated Annealing is one of the most effective AI optimization algorithms which is a type of local search technique. In our research, this algorithm completed execution within 20 to 40 minutes depending on the size of the dataset pair. But the final score was not efficient enough compared to GA and HA. In Spring and Summer dataset pair, this algorithm produced lesser score than CSP Solver. But in the Fall 2020 dataset which was considered as small dataset, it performed slightly better than

CSP Solver. One notable thing for this algorithm is it does not get stuck in the local maxima because it chooses a slightly bad move in each iteration. This process is done mathematically while calculating the acceptance probability function. This technique is used in Hybrid Algorithm where it helped the GA to update the score more frequently.

The last and the most effective optimization algorithm was Hybrid Algorithm which combined the basic approaches of GA and SA. As previously mentioned, after a certain time period GA used to get stuck in local maxima for a long time. To reduce this time, the acceptance probability calculation of SA was added in this algorithm. This little addition helped the algorithm significantly. If the normal genetic functionalities failed to update the score after a certain amount of iterations (tolerance value), the algorithm switches to the acceptance probability part so that it can get out of the local maxima soon. Since this algorithm works almost similar to the normal GA, it takes a lot of time too. But as this algorithm takes lesser iteration difference to update the value of current score, the final result after completing 20000000 iterations was higher than GA. This is the main reason Hybrid Algorithm proved to be the most efficient approach to solve DCAP in our research.



# Chapter 7

## Conclusion

In this report, a comparative study was conducted between four AI optimization algorithms to solve the course allocation problem of The Department of CSE, BRAC University. The integration of HA in this problem has surpassed the performance of other algorithms (CSP Solver, GA and SA) in terms of the score. Considering the execution time, GA and HA took more time to complete execution compared to CSP Solver and SA. These algorithms were implemented on three scales of datasets (Small, Medium and Large), and it has been seen that they perform the best on small datasets. With the increasing size of the datasets, the scores decreased, but, HA still outperformed the other three algorithms. So the hybridization of algorithms should be explored more frequently as it can enhance the effectiveness of the problem solving techniques compared to when the algorithms are used individually.

### 7.1 Limitations

The work done in this paper does not include all the aspects of the course allocation problem of the department. This is due to the fact that the inclusion of some of these aspects increases the complexity of the problem, and the idea becomes harder to implement. Again, there are some aspects that are not directly related to the problem at hand. All these aspects can be defined as the limitations of our work. Firstly, our work only handles the Undergraduate Program of the department. The datasets created using the course information and the faculty members belong to this Program only. Our department offers Postgraduate degrees as well, but it was not within the scope of our research. As mentioned in the previous chapters, our work also does not include the allocation of rooms to the courses, primarily because they are not related directly to the course allocation problem. The allocation of rooms is mostly dependant on the Office of the Registrar, and it is hard to bring something under an automated process that is outside the scope of the department. The implemented idea in this work does not handle classes of courses taken at the 05:00 pm slot. This slot is usually assigned to courses if no other slot is available throughout the day. Since this is an exceptional case, it was not included in our work. Sometimes there are cases when some faculty members are assigned credits beyond their actual credit limit. Overload in terms of working hours, other departmental tasks such as the course allocation problem itself can be some of these cases. These cases are usually rare and they are handled by the department according to necessity. For this reason, it was not included in our work. Finally, in previous semesters, some

of the lab sections were allocated to only one faculty member. But in our case, we have fixed two faculty members to each lab for reducing complexity. CSE419 and CSE474 are two such courses whose lab classes are taken by only one faculty member. Our work does not handle the allocation of labs like these two courses. These are some of the notable shortcomings of the work presented in this paper.

## 7.2 Future Works

Every research work has room for more improvement. Our work can also be improved further and more areas of the problem can be covered as well. Analysis of related works and our own work has shown that the results generated by the implementation of the base algorithms are not always satisfactory. These algorithms can be improved a lot with small modifications in their parameters. The base algorithms thus become more advanced and eventually give better results. So the slight modifications of the algorithms like GA and SA should be done more and more by the researchers. Some other optimization algorithms like PSO, Tabu Search, Ant-Colony Optimization and Memetic algorithm can also be used for works similar to ours. A comparative analysis of these algorithms and the algorithms used in our work will give a better understanding of the nature of these algorithms and the results that these produce. The prepared preferred course list of our dataset does not include any type of priority. A priority for which course is preferred over the other can be introduced in further modifications of our work. Another solution to this problem can be in terms of setting a maximum course limit for each faculty member for a particular course. This limit will determine how many sections of a particular course can one faculty member take in a semester. Strategies similar to our implemented idea can also be used to solve other departmental tasks such as Student Tutor appointment and Lab change procedure of the students. We hope that our work will open new doors of opportunities for the researchers who are working with similar topics and through further research and innovations they will be able to take this idea to a more advanced level where the management will be simpler and easier.

# Bibliography

- [1] S. Yang and S. N. Jat, “Genetic algorithms with guided and local search strategies for university course timetabling,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, no. 1, pp. 93–106, 2010.
- [2] E. Aycaan and T. Ayav, “Solving the course scheduling problem using simulated annealing,” in *2009 IEEE International Advance Computing Conference*, IEEE, 2009, pp. 462–466.
- [3] E. Yu and K. Sung, “A genetic algorithm for a university weekly courses timetabling problem,” *International Transactions in Operational Research*, vol. 9, pp. 703–717, Nov. 2002. DOI: 10.1111/1475-3995.00383.
- [4] O. Odeniyi, E. Omidiora, S. Olabiyisi, and A. Aluko, “Development of a modified simulated annealing to school timetabling problem,” *International Journal of Applied Information Systems*, vol. 8, pp. 16–24, Jan. 2015. DOI: 10.5120/ijais14-451277.
- [5] E. Burke, K. Jackson, J. H. Kingston, and R. F. Weare, “Automated university timetabling: The state of the art,” *Comput. J.*, vol. 40, pp. 565–571, 1997.
- [6] S. Abdennadher and M. Marte, “University timetabling using constraint handling rules,” in *JFPLC*, 1998, pp. 39–50.
- [7] L. Hiryanto, “Incorporating dynamic constraint matching into vertex-based graph coloring approach for university course timetabling problem,” Jun. 2013, pp. 68–72, ISBN: 978-1-4673-5784-5. DOI: 10.1109/QiR.2013.6632539.
- [8] A. Schaerf, “Combining local search and look-ahead for scheduling and constraint satisfaction problems,” in *IJCAI*, 1997, pp. 1254–1259.
- [9] K. Pokorny and R. Vincent, “Multiple constraint satisfaction problems using the a-star (a\*) search algorithm: Classroom scheduling with preferences,” *Journal of Computing Sciences in Colleges*, vol. 28, pp. 152–159, May 2013.
- [10] T. El-Sakka, “University course timetable using constraint satisfaction and optimization,” *International Journal of Computing Academic Research*, vol. 4, pp. 83–95, Jun. 2015.
- [11] M. Gen and R. Cheng, *Genetic Algorithms Engineering Design*. John Wiley Sons, Ltd, Jan. 1997, pp. 380–405, ISBN: 9780471127413. DOI: 10.1002/9780470172254.ch7.
- [12] E. Burke, D. Elliman, and R. Weare, “A genetic algorithm based university timetabling system,” vol. 1, Dec. 1994.

- [13] A. Jain, “Formulation of genetic algorithm to generate good quality course timetable,” *International Journal of Innovation, Management and Technology*, vol. 1, pp. 248–251, Aug. 2010.
- [14] W. Wen-jing, “Improved adaptive genetic algorithm for course scheduling in colleges and universities,” *International Journal of Emerging Technologies in Learning (iJET)*, vol. 13, no. 06, pp. 29–42, 2018.
- [15] S. Alves, S. Oliveira, and A. Rocha Neto, “A novel educational timetabling solution through recursive genetic algorithms,” Oct. 2015, pp. 1–6. DOI: 10.1109/LA-CCI.2015.7435955.
- [16] D. Abramson and J. Abela, “A parallel genetic algorithm for solving the school timetabling problem,” 1992.
- [17] M. Abbaszadeh and S. Saeedvand, “A fast genetic algorithm for solving university scheduling problem,” *IAES International Journal of Artificial Intelligence*, vol. 3, no. 1, p. 7, 2014.
- [18] S. Ghaemi, M. Vakili, and A. Aghagolzadeh, “Using a genetic algorithm optimizer tool to solve university timetable scheduling problem,” Mar. 2007, pp. 1–4, ISBN: 978-1-4244-0778-1. DOI: 10.1109/ISSPA.2007.4555397.
- [19] S. Naseem, “A guided search genetic algorithm for the university course timetabling problem,” Oct. 2009.
- [20] H. V. Yamazaki and J. Pertoft, *Scalability of a genetic algorithm that solves a university course scheduling problem inspired by kth*, 2014.
- [21] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, *et al.*, “A comparison of the performance of different metaheuristics on the timetabling problem,” in *International conference on the practice and theory of automated timetabling*, Springer, 2002, pp. 329–351.
- [22] N. G. A. H. Saptarini, P. I. Ciptayani, and I. B. I. Purnama, “A custom-based crossover technique in genetic algorithm for course scheduling problem,” *TEM Journal*, vol. 9, no. 1, pp. 386–392, 2020.
- [23] A. A. Gozali and S. Fujimura, “Solving university course timetabling problem using multi-depth genetic algorithm,” in *SHS Web of Conferences*, EDP Sciences, vol. 77, 2020.
- [24] D. Bertsimas and J. Tsitsiklis, “Simulated annealing,” *Statistical Science*, vol. 8, no. 1, pp. 10–15, 1993, ISSN: 08834237. [Online]. Available: <http://www.jstor.org/stable/2246034>.
- [25] E. Aarts, J. Korst, and W. Michiels, “Simulated annealing,” English, in *Search Methodologies : Introductory Tutorials in Optimization and Decision Support Techniques*, E. Burke and G. Kendall, Eds. Germany: Springer, 2005, pp. 187–210, ISBN: 0-387-23460-8. DOI: 10.1007/0-387-28356-0-7.
- [26] D. J. Ram, T. Sreenivas, and K. G. Subramaniam, “Parallel simulated annealing algorithms,” *Journal of parallel and distributed computing*, vol. 37, no. 2, pp. 207–212, 1996.

- [27] J. M. Thompson and K. A. Dowsland, “A robust simulated annealing based examination timetabling system,” *Computers & Operations Research*, vol. 25, no. 7-8, pp. 637–648, 1998.
- [28] D. Abramson, “Constructing school timetables using simulated annealing: Sequential and parallel algorithms,” *Management Science*, vol. 37, pp. 98–113, 1991.
- [29] E. Norgren and J. Jonasson, *Investigating a genetic algorithm-simulated annealing hybrid applied to university course timetabling problem : A comparative study between simulated annealing initialized with genetic algorithm, genetic algorithm and simulated annealing*, 2016.
- [30] W. A. Algasm, “Hybrid algorithm to solve timetabling problem,” in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, vol. 928, 2020, p. 032053.
- [31] M. Kohshori and M. Saniee Abadeh, “Hybrid genetic algorithms for university course timetabling,” *International Journal of Computer Science Issues*, vol. 9, Mar. 2012.
- [32] D. T. Anh and K.-H. Lam, “Combining constraint programming and simulated annealing on university exam timetabling.,” Jan. 2004, pp. 205–210.
- [33] T. L. June, J. H. Obit, Y.-B. Leau, and J. Bolongkikit, “Implementation of constraint programming and simulated annealing for examination timetabling problem,” in *Computational Science and Technology*, Springer, 2019, pp. 175–184.
- [34] A. Gunawan, K. Ng, and K. Poh, “Solving the teacher assignment-course scheduling problem by a hybrid algorithm,” *International Journal of Computer, Information, and System Science, and Engineering*, vol. 1, Jan. 2007.
- [35] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria, “An effective hybrid algorithm for university course timetabling,” *Journal of Scheduling*, vol. 9, no. 5, pp. 403–432, 2006.
- [36] A. Rezaeipannah, S. S. Matoori, and G. Ahmadi, “A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search,” *Applied Intelligence*, pp. 1–26, 2020.
- [37] S. C. Brailsford, C. N. Potts, and B. M. Smith, “Constraint satisfaction problems: Algorithms and applications,” *European Journal of Operational Research*, vol. 119, no. 3, pp. 557–581, 1999.
- [38] G. Verfaillie and T. Schiex, “Solution reuse in dynamic constraint satisfaction problems,” in *AAAI*, vol. 94, 1994, pp. 307–312.
- [39] G. Verfaillie and N. Jussien, “Constraint solving in uncertain and dynamic environments: A survey,” *Constraints*, vol. 10, no. 3, pp. 253–281, 2005.
- [40] R. J. Wallace, D. Grimes, and E. C. Freuder, “Dynamic constraint satisfaction problems: Relations among search strategies, solution sets and algorithm performance,” in *International Workshop on Constraint Solving and Constraint Logic Programming*, Springer, 2009, pp. 105–121.

- [41] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, “Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems,” *Artificial intelligence*, vol. 58, no. 1-3, pp. 161–205, 1992.
- [42] B. Selman, H. Levesque, and D. Mitchell, “A new method for solving hard satisfiability problems,” Jul. 1992.
- [43] E. Cantú-Paz, “A survey of parallel genetic algorithms,” *Calculateurs paralleles, reseaux et systems repartis*, vol. 10, no. 2, pp. 141–171, 1998.
- [44] M. Safe, J. Carballido, I. Ponzoni, and N. Brignole, “On stopping criteria for genetic algorithms,” vol. 3171, Sep. 2004, pp. 405–413, ISBN: 978-3-540-23237-7. DOI: 10.1007/978-3-540-28645-5\_41.
- [45] M. Nandhini and S. Kanmani, “A survey of simulated annealing methodology for university course timetabling,” *International Journal of Recent Trends in Engineering*, vol. 1, no. 2, p. 255, 2009.
- [46] K. Patrick, “Comparison of simulated annealing and hill climbing in the course timetabling problem,” *African Journal of Mathematics and Computer Science Research*, vol. 5, no. 11, pp. 176–178, 2012.