Thesis No: CSER-M-19-04

UNIVERSITY COURSE SCHEDULING USING PROMINENT NATURE INSPIRED TECHNIQUES

by

Sk. Imran Hossain

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

Khulna 9203, Bangladesh

April, 2019

University Course Scheduling using Prominent Nature Inspired Techniques

by

Sk. Imran Hossain

Roll No: 1607506

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Khulna University of Engineering & Technology
Khulna 9203, Bangladesh

April, 2019

Declaration

This is to certify that the thesis work entitled "University Course Scheduling using Prominent Nature Inspired Techniques" has been carried out by Sk. Imran Hossain in the Department of Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna 9203, Bangladesh. The above thesis work or any part of this work has not been submitted anywhere for the award of any degree or diploma.

[illegible]

Signature of Supervisor

[illegible]

Signature of Candidate

Approval

This is to certify that the thesis work submitted by Sk. Imran Hossain entitled “University Course Scheduling using Prominent Nature Inspired Techniques” has been approved by the board of examiners for the partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering in the Department of Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh in April, 2019.

Acknowledgement

At first, I would like to thank Almighty for showering all his blessings on me whenever I needed. It is my immense pleasure to express my indebtedness and deep sense of gratitude to my supervisor Dr. Muhammad Aminul Haque Akhand, Professor & Head, Department of Computer Science and Engineering (CSE), Khulna University of Engineering & Technology (KUET) for his continuous encouragement, constant guidance and keen supervision throughout of this study. I am especially grateful to him for giving me his valuable time whenever I need and always providing continuous support in my effort.

Last but not least, I am grateful to my parents, family member and friends for their patience, support and encouragement during this period.

April, 2019 Author

Abstract

The University Course Scheduling Problem (UCSP) is a highly constrained real-world combinatorial optimization task. Solving UCSP means creating an optimal course schedule by assigning courses to specific rooms, instructors, students, and timeslots by taking into account the given constraints. Several studies have reported different metaheuristic approaches for solving UCSP including Genetic Algorithm (GA) and Harmony Search (HS) algorithm. Various Swarm Intelligence (SI) optimization methods have also been investigated for UCSP in recent times and a few Particle Swarm Optimization (PSO) based methods among them with different adaptations are shown to be effective. In this study, two novel PSO and Group Search Optimizer (GSO) based methods are investigated for solving highly constrained UCSP in which basic PSO and GSO operations are transformed to tackle combinatorial optimization task of UCSP and a few new operations are introduced to PSO and GSO to solve UCSP efficiently. In the proposed methods, swap sequence-based velocity and movement computation and its application are developed to transform individual particles and members in order to improve them. Selective search and forceful swap operation with repair mechanism are the additional new operations in the proposed methods for updating particles and members with calculated swap sequences. The proposed PSO with selective search (PSOSS) and GSO with selective search (GSOSS) methods have been tested on an instance of UCSP resembling the course structure of the Computer Science and Engineering Department of Khulna University of Engineering & Technology which has many hard and soft constraints. Experimental results revealed the effectiveness and the superiority of the proposed methods compared to other prominent metaheuristic methods (e.g., GA, HS).

Contents

LIST OF TABLES

LIST OF FIGURES

Nomenclature

UCSP	University Course Scheduling Problem
SI	Swarm Intelligence
PSO	Particle Swarm Optimization
GSO	Group Search Optimizer
PSOSS	PSO with Selective Search
GSOSS	GSO with Selective Search
GA	Genetic Algorithm
HS	Harmony Search
ILP	Integer Linear Programming
TS	Tabu Search
ACO	Ant Colony Optimization
SO	Swap Operator
SS	Swap Sequence

CHAPTER I

Introduction

Timetabling problems deal with event scheduling considering various constraints. These problems are NP-hard and can be either optimisation or feasibility problems. University Course Scheduling Problem (UCSP) is one of the most difficult timetabling optimization problems. This chapter provides an overview of timetabling problem, introduces UCSP as a timetabling problem, describes objectives of the thesis and also contains the thesis organization.

1.1 Overview of Timetabling Problem

Timetabling problems deal with scheduling of a fixed number of events using a fixed number of timeslots and resources in order to satisfy a set of constraints [1]–[3]. These problems are NP-hard [4], [5]. The set of the constraints of timetabling problems is usually divided into a set of hard constraints and a set of soft ones. Hard constraints are the conditions that must be satisfied for a working timetable whereas the soft constraints are conditions that may be violated but affect the solution's quality [6]. Some versions of the problem consider an objective representing for instance the cost (to be minimized) or the acceptability (to be maximized) of the schedule, while some other versions are pure feasibility problems. Timetabling problems have found many applications in different domains such as employee allotment, transport systems, educational organizations, sports activities and industrial applications [7]. In higher educational institutions, examination and course scheduling are two important common and challenging tasks for optimizing physical and human resources [8]. The University Course Scheduling problem (UCSP) typically is among the difficult timetabling optimization problems requiring a large number of soft and hard constraints to be satisfied.

1.2 University Course Scheduling Problem (UCSP)

The goal of UCSP is to assign all classes and laboratory sessions to instructors, rooms, and timeslots considering the hard and soft constraints in such a way that no dispute arises in these assignments [9]. Mathematically, the UCSP is defined as a triple $\langle E, T, C\rangle$, where $E = \{c_i, s_j, i_k\}$ contains three sets: set of classes c_i, set of students s_j, and set of instructors i_k; $T = \{t_1, \dots, t_n\}$ is a set of time slots and $C = \{C_1, \dots, C_n\}$ is the set of hard and soft constraints. The task is to assign E_i to the time slot T_i satisfying constraints C_i where $C_i \in C$. The challenges of the UCSP are the constraints, for example, instructors' dispositions, educational policies of the school, availability of teaching staffs and other physical resources. In UCSP, each instructor can teach one class at a timeslot and students can just go to one class at any given time. Other similar kinds of constraints are treated as hard constraints that must be satisfied. Common soft constraints of the UCSP are instructors' preferences for favored days and timeslots, expected to be satisfied to the extent possible. In the UCSP the main issue is to handle room allocation for lectures considering the maximum capacity of each room, the number of enrolled students in a course and other related facilities[10], [11]. Both hard and soft constraints may vary from institution to institution based on their resources and facilities. Any resource modification or update (including capacity alteration in resources) requires rescheduling of classes, which is very common at the beginning of a term.

1.3 Thesis Objectives

Various Swarm Intelligence (SI) optimization methods have been investigated for UCSP in recent times and a few Particle Swarm Optimization (PSO) based methods among them with different adaptations are shown to be effective. In this study, two novel PSO and Group Search Optimizer (GSO) based methods are investigated for solving highly constrained UCSP in which basic PSO and GSO operations are transformed to tackle combinatorial optimization task of UCSP and a few new operations are introduced to PSO and GSO to solve UCSP efficiently. In the proposed methods, swap sequence-based velocity and movement computation and its application are developed to transform individual particles and members in order to improve them. Selective search and forceful swap operation with repair mechanism are the additional new operations in the proposed methods for updating particles and members with calculated swap sequences. Objectives of this thesis work are:

- To solve UCSP using PSO with Selective Search (PSOSS).
- To solve UCSP using GSO with Selective Search (GSOSS).

- To evaluate the significance of using selective search and forceful swap application with repair mechanism in PSOSS and GSOSS.
- To compare the performance of the proposed PSOSS and GSOSS methods with other traditional methods like Genetic Algorithm (GA) and Harmony Search (HS).

1.4 Thesis Organization

The thesis is organized in five chapters.

Chapter I provides introductory discussion on timetabling problems, UCSP, thesis objectives and thesis organization.

Chapter II provides overview of prominent nature inspired optimization algorithms, brief discussion on existing methods to solve UCSP, observation on existing methods and scope of the research.

Chapter III describes the proposed PSOSS and GSOSS methods in detail.

Chapter IV contains experimental studies. In this chapter experimental setup, environment, input data preparation, experimental results and analysis are discussed.

Chapter V provides concluding remarks and possible future research directions.

CHAPTER II

Literature Review

Nature acts a source of inspiration for researchers and most of the new optimization algorithms are nature inspired. Several nature inspired optimization algorithms have been applied on University Course Scheduling Problem (UCSP) and they are found to be effective but there remains a lot of scope of research in this field. This chapter provides overview of a few nature inspired algorithms like Particle Swarm Optimization (PSO), Group Search Optimizer (GSO), Genetic Algorithm (GA), Harmony Search (HS). It also includes brief discussion on existing methods to solve UCSP, observation on existing methods and scope of research.

2.1 Prominent Nature Inspired Optimization Algorithms

Nature is the primary source of inspiration for researchers and most of the new optimization algorithms are nature inspired. Most of the nature inspired algorithms are biology inspired and among them some draws inspiration from Swarm Intelligence (SI). Many algorithms also draw inspiration from physical or chemical systems. Some algorithms like HS draw inspiration even from music. Most of the nature inspired algorithms starts with an initial population and tries to improve or evolve this population based on natural phenomenon. There is a vast number of nature inspired optimization algorithms and for this study we have considered PSO, GSO, GA and HS. Following subsections provide brief overview of these algorithms to make the thesis paper self-contained.

2.1.1 Particle Swarm Optimization (PSO)

PSO is an optimization algorithm based on the social behavior of swarms mimicking the movement of organisms (e.g., bird, fish, bat and firefly) in a swarm [12]–[15]. It works with a population of particles and every particle indicates a candidate solution of the optimization problem in multidimensional search space [16]. The fitness of each particle is calculated using a fitness function which is associated with the problem at hand. PSO starts with a population of particles which are randomly assigned to the search space. At every iteration, each particle of PSO adjusts its position based on velocity calculated considering (i) its present position, (ii) personal best position and (iii) global best position of the population. The personal best position of a particle is the best position found by the particle so far and the global best position is the best position found by all the particles. This process continues until it reaches a stopping criterion [17] and then the global best position is considered as the final outcome. PSO has found many applications in many domains [11], [18], [19].

Consider a search space of d dimensions consisting of z number of particles. If a particle's current position is X_p, personal best position is B_p and global best position among all the particles is G then, the velocity of a particle V_p is calculated using the following equation:

$$V_p^{(t)} = wV_p^{(t-1)} + c_1 r_1 \left(B_p - X_p^{(t-1)}\right) + c_2 r_2 \left(G - X_p^{(t-1)}\right) \qquad (2.1)$$

where, w is the inertia factor, c_1 is the cognitive coefficient, c_2 is the social coefficient, and $\{r_1, r_2\} \in [0,1]$ are random values. Inertia factor w scales the influence of the previous velocity, c_1 limits the size of the step the particle takes towards its personal best and c_2 limits the size of the step the particle takes towards its global best [20]. The position of the particle is updated using the following equation:

$$X_p^{(t)} = X_p^{(t-1)} + V_p^{(t)} \times T \qquad (2.2)$$

where, T represents time to convert velocity into distance and its value is assumed 1.

The basic steps of PSO are as follows:

Step 1. Initialization:

a. Create a population of z particles by randomly positioning them in the search space.
b. Calculate fitness of each particle and assign its current position (X_p) as personal best position (B_p).
c. Find the global best position (G) among all the particles.

Step 2. Position Update of Each Particle:

d. Calculate the velocity using Eq. (2.1).
e. Move to the new position according to Eq. (2.2).
f. Calculate fitness.
g. Update personal best position (B_p) and global best position (G) considering fitness of new position.

Step 3. Termination and Outcome:

Go to Step 2 if termination criterion is not met; otherwise, stop and consider global best position (G) as the final outcome.

2.1.2 Group Search Optimizer (GSO)

GSO is inspired by animal searching behaviour and social foraging [21], [22]. It adopts the scrounging strategies of house sparrows and employs especially animal scanning mechanism. General social foraging model mainly follows the producer-scrounger (P-S) model, in which the group members search either for finding producer or for joining scrounger opportunities. There are three kinds of member in the group: (i) producer that searches for food; (ii) scrounger that performs area copying behaviour in order to keep searching for opportunities to join the resources found by the producer; (iii) ranger (or dispersed members) that employs searching strategies of random walks for randomly distributed resources and performs random walk motion. At each iteration, the member located at the most promising resource is the producer, a number of members except producer in the group are selected as scroungers, and the remaining members are rangers. In an n-dimensional search space, the i-th member of the group at iteration t located in a position is denoted as $x_i^t \in R^n$ with a head angle of $\phi_i^t = [\phi_{i_1}^t, \cdots, \phi_{i_{n-1}}^t]$. The search direction of the i-th member is a unit vector denoted as $D_i^t(\phi_i^t) = [d_{i_1}^t, \cdots, d_{i_n}^t] \in R^n$. The individual components of the direction vector can be obtained from a Polar to Cartesian coordinate transformation as follows

$$d_{i_1}^t = \prod_{q=1}^{n-1} cos(\phi_{iq}^t) \tag{2.3}$$

$$d_{i_j}^t = sin(\phi_{i_{j-1}}^t) \times \prod_{q=j}^{n-1} cos(\phi_{i_q}^t),\ j \in \{2,3,\cdots,n-1\} \tag{2.4}$$

$$d_{i_n}^t = sin(\phi_{i_{n-1}}^t) \tag{2.5}$$

When a member of the group goes outside the search space, it is brought back to its previous position inside the search space. In GSO, the member located at the most favourable area of the search space at iteration t, i.e. the member with the best fitness, is considered as the producer x_p^t. The producer making a stopover at that location and searches the space for optimal resources. Animals do use high resolution vision mechanism that enable them to encode large field of view, which is far too complex for GSO to implement. In GSO, a simple scanning mechanism used by white crappie is utilised. The scanning mechanism of the white crappie starts at zero degree then scan in the lateral direction by sampling three points in a random manner in the vision field: one point at zero-degree x_z, one point on the hand side

$x_r$and one point at the left-hand side hypercube x_l. The three points for the producer $x_p^t$are described by

$$x_z = x_p^t + r_1 l_{max} D_p^t(\emptyset^t) \tag{2.6}$$

$$x_r = x_p^t + r_1 l_{max} D_p^t\left(\frac{\emptyset^t + r_2\theta_{max}}{2}\right) \tag{2.7}$$

$$x_l = x_p^t + r_1 l_{max} D_p^t\left(\frac{\emptyset^t - r_2\theta_{max}}{2}\right) \tag{2.8}$$

where $r_1 \in [0,1]$ is a random number, $r_2 \in (0,1)$ is a uniformly distributed random sequence, $\theta_{max} \in R$ is the maximum pursuit angle and $l_{max} \in R$ is the maximum pursuit distance. Producer will then find the best point with the highest fitness value. If this new point has better fitness value than the current point, then it will update the current point with new point. Otherwise it will remain in the current point and change the head angle to a randomly generated head angle defined by

$$\phi_i^{t+1} = \phi_i^t + r_2\alpha_{max} \tag{2.9}$$

where $\alpha_{max}$is the maximum turning angle. If the producer cannot reach a better position after some iterations τ, it will orient the heading angle back to zero degree. A number of members become scroungers in the group. They mainly search for opportunities and join the resources found by the producer. Three scrounging behaviours are observed in house sparrows in [23], which led them to model producer-scrounger (PS) behaviours, i.e. the search strategies. The three strategies used in PS model are:

(i) Area copying - searching in the immediate area around the producer;

(ii) Following - following another animal around;

(iii) Snatching - taking a resource directly from the producer.

Area copying is found to be the common scrounging behaviour in sparrows [23].Therefore, area copying is used in the GSO algorithm. The area copying behaviour of the *i*-th scrounger at iteration *t* is described as a random walk toward the producer defined by

$$x_i^{t+1} = x_i^t + r_3 \circ (x_p^t - x_i^t) \tag{2.10}$$

where $r_3 \in [0,1]$ is a uniform random sequence. The operator '∘' is a Hadamard product of the Schur product to compute the entry-wise product of the two vectors. While showing copying behaviour, the scroungers also keep looking for other opportunities. This behaviour is described by Eq. (2.9). The members in the group often have different searching and competitive abilities, which make the remaining members to disperse from their current locations according to their foraging efficiency. Different dispersal techniques are seen in animals, birds and insects such as ranging behaviour for habitat preferences. Ranging is a phase of search for resources without any cues. In GSO algorithm, dispersed members are called rangers. It is common for rangers to do random walks, which are thought to be the efficient searching method for randomly distributed resources. The i-th member, now a ranger, takes a random head angle using Eq. (2.9) and selects a random distance at iteration t defined by

$$l_i = r_1 \times l_{max} \tag{2.11}$$

where $r_1 \in [0,1]$ is a random number and l_{max} is the maximum pursuit distance a ranger can travel. The maximum pursuit l_{max} is calculated according to

$$l_{max} = \|U - L\| = \sqrt{\sum_{i=1}^{n} (U_i - L_i)^2} \tag{2.12}$$

where U_i and L_i are the upper and lower bounds for the ith dimension.

Using the random distance, i-th member jumps on to the new point defined by

$$x_i^{t+1} = x_i^t + l_i \times D_i^t(\phi^{t+1}) \tag{2.13}$$

Animals use simple strategy of maximizing resources and limiting the search to profitable patch. For example, when animals detect edge of the resource, they go back to the patch. In GSO, this simple strategy is used. When a member goes outside the boundary of the search space, it will head back to within the search space by resetting variables that caused crossing boundary.

The basic steps of GSO are as follows:

Step 1. Initialization

a. Create a group of w members by randomly positioning them in the search space.

Step 2. Member Categorization

a. Calculate fitness of each member of the group.

b. Sort members according to fitness value.

c. Select member having best fitness as producer, specified percentage of worst members as dispersed members and rest of the members as scroungers.

Step 3. Producer's Scanning

a. Producer scans search area for better position and moves to new better position.

Step 4. Scrounging

a. Each scrounger moves towards producer.

Step 5. Dispersed Members' Random Operation

a. Each dispersed member moves to new position using random walk.

Step 6. Termination and Outcome

a. Recalculate the group with updated producer, scroungers and dispersed members.

b. Go to Step 2 if termination criterion is not met; otherwise recalculate producer and consider its solution as final outcome.

2.1.3 Genetic Algorithm (GA)

GA is a population-based search and optimization algorithm inspired by Darwinian evolution and natural selection [24]. GA has three operators, namely, crossover, mutation, and selection. The GA works with a population initialized randomly that undergoes crossover with a crossover probability $p_c \in [0,1]$ and mutation with a mutation probability $p_m \in [0,1]$ [25]–[30]. The individuals in the population survive to next generation based on their fitness. GA obtains a solution with the highest fitness after several generations (i.e. iterations), which is considered the optimal solution [31]–[33]. GA has found successful applications in many domains [34], [35]. In GA an initial population of individuals is evolved for producing better solutions. Each individual is a complete solution to the problem at hand and is characterized by a set of parameters called Genes. These Genes are joined together to form a solution called Chromosome. Traditionally, solutions are encoded in binary as a string of 1s and 0s. A fitness function is used to calculated fitness of each individual which depends on the optimization problem. Individuals are selected, crossed and mutated in hope of creating individuals having better fitness value.

The steps of GA are as follows:

Step 1. Initialization

Create initial population by generating specified number of random individuals.

Step 2. Fitness Calculation

Calculate fitness of each individual of the population using a fitness function.

Step 3. Selection

Select individuals from the population so that they can breed a new generation. Selection probability generally depends on the fitness value so that, individuals having better fitness get a higher chance of reproducing.

Step 4. Crossover

Breed off-springs by crossing two random individuals from the pool of selected individuals at Step 3. Number of pairs depends on crossover rate. Most common crossover method is single point crossover. In single point crossover a random point called 'crossover point' is picked and between the two parent chromosomes bits to the right of that point are swapped. This process produces two off-springs, each having genetic information from both parents. These off-springs are incorporated in new population by replacing the parents.

Step 5. Mutation

Mutate a number of individuals in the population based on mutation probability. For binary encoding mutation is done by flipping some of the bits of a chromosome.

Step 6. Termination and Outcome

Go to Step 2 if termination criterion is not met; otherwise, stop and consider best solution from the population as the final outcome.

A variant of new population construction process is to retain a number of best individuals without alteration from current population to next population. This is done to ensure that solution of best fitness generated by GA is retained over iterations. This concept is known as elitism.

2.1.4 Harmony Search (HS)

HS is a population-based search and optimization algorithm inspired by an improvisation process in musical performance [36]. Search efficiency and exploratory power of HS has been reported in [37]. In HS, a solution is called a harmony, population is termed as Harmony Memory (HM) and population size is called Harmony Memory Size (HMS). An initial population of HM is randomly generated. A candidate harmony is improvised for each iteration using memory consideration, pitch adjustment, and random selection. The new harmony is compared with the worst harmony. The worst harmony is replaced by the new harmony if it is better than the worst harmony and the HM is updated. This process is repeated until a predetermined number of improvisations. The music improvisation is a process of searching for the better harmony by trying various combinations of pitches that should follow any of the following three rules [36]:

1. Playing any one pitch from the memory.

2. Playing an adjacent pitch of one pitch from the memory.
3. Playing a random pitch from the possible range.

HS algorithm mimics this process and follows any of the three rules below:

1. Choosing any value from the HS memory.
2. Choosing an adjacent value from the HS memory.
3. Choosing a random value from the possible value range.

The three rules in the HS algorithm are effectively directed using two essential parameters: Harmony Memory Considering Rate (*HMCR*) and Pitch Adjusting Rate (*PAR*).

The steps of HS algorithms are as follows:

Step 1. Initialization

Initialize the HS memory HM. The initial HM consists of a given number of randomly generated solutions to the optimization problems under consideration. For an n-dimension problem, an HM with the size of HMS can be represented as follows:

$$\mathrm{HM} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_n^1 \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{\mathrm{HMS}} & x_2^{\mathrm{HMS}} & \dots & x_n^{\mathrm{HMS}} \end{bmatrix}$$

where $[x_1^i \quad x_2^i \quad \dots \quad x_n^i]$ $(i = 1, 2, \dots, \mathrm{HMS})$ is a solution candidate.

Step 2. New Solution Improvisation

From HM improvise a new solution $[x_1' \quad x_2' \quad \dots \quad x_n']$ where each component x_j' is generated based on HMCR. HMCR is the probability of selecting a component from existing members of HM. So, 1-HMCR is the probability of generating x_j' component randomly. If x_j' comes from HM then it is selected from jth dimension of a randomly chosen HM member, and then it can be mutated based on mutation probability PAR.

Step 3. HM Update

If the fitness of the new solution obtained from Step 2 is better than the worst member of HM, then replace the worst member with new solution.

Step 4. Termination and Outcome

Go to Step 2 if termination criterion is not met; otherwise, stop and consider best solution from HM as the final outcome.

2.2 Existing Methods to Solve UCSP

A number of exact and meta-heuristic [38] approaches have been applied to different scheduling tasks including UCSP in the last few years. Integer Linear Programming (ILP) is an exact approach which optimizes an objective function of a mathematical model. ILP has been applied to scheduling problems including course timetabling problem [39], [40]. Some of the meta-heuristic approaches are Genetic Algorithm (GA) [6], [41]–[46], Tabu Search (TS) [8], Simulated Annealing (SA) [47], [48], Evolutionary Algorithm [49], [50], Hybrid Evolutionary Algorithm [51], Hybrid Evolutionary Approach with Nonlinear Great Deluge [52], Hybrid Electromagnetism-like Mechanism with Great Deluge [53], and Harmony Search (HS) algorithm [54], [55]. A TS based approach was proposed for UCSP with emphasis on the University of Dar-assalaam that generates schedule by heuristically minimizing penalties over infeasible solutions in [8]. A metaheuristic algorithm for UCSP combining Heuristics, SA, Variable Neighborhood Descent, and TS was proposed in [1].

2.2.1 GA based Methods to Solve UCSP

A GA based method is proposed to solve weekly course timetabling problem in [56]. This approach uses problem-specific chromosome representation and knowledge-augmented genetic operators for avoiding illegal timetables generation. A GA method for UCSP with multiple constraints is investigated in [42], which resulted in a timetable more acceptable to instructors. GA with a guided search strategy and local search techniques for the university course timetabling is proposed in [57]. This approach uses guided search strategy to create offspring and local search to improve the search efficiency. Course timetabling problem is modelled as a bi-criteria optimization problem and solved by a hybrid Multi-objective GA in [58]. This approach makes use of Hill Climbing and Simulated Annealing algorithms in addition to the standard GA approach.

2.2.2 HS based Methods to Solve UCSP

HS algorithm is used for university course timetabling problem in [54] which generates viable solution compared to previous works. A hybrid HS with hill climbing for local exploitation improvement and global best concept of PSO for improved convergence is proposed for university course timetabling problem in [55].

2.2.3 PSO based Methods to Solve UCSP

Recently, SI based optimization methods such as Ant Colony Optimization (ACO) [59], [60], Honey-Bee Mating Optimization [61] have been investigated for UCSP and other timetabling problems. In SI based optimization methods, Particle Swarm Optimization (PSO) is very popular [62] and various PSO based strategies have been examined for UCSP. A PSO based method considering a bunch of constraints and a repair mechanism for all infeasible solutions is proposed for UCSP in [11]. This method was applied for class scheduling of the department of information management of Kun-Shan University in Taiwan. An adaptive PSO based technique in which teachers are allowed to set their own weights to each hard and/or soft constraint is investigated to solve UCSP in [63]. A hybrid PSO method incorporating local search is proposed for generating timetable for Greece high school in [64]. Two different versions of PSO for UCSP, the inertia weight version and the constriction version is investigated in [16]. This study also utilizes interchange heuristic to explore the neighbouring solution space for improving solution quality.

2.2.4 Previous Works from the Department of CSE, KUET

ACO with GA is used in [65] for solving UCSP where ACO is used to provide UCSP's solution and GA operations such as selection and mutation is employed to improve UCSP’s solution. Modified Hybrid PSO is implemented in [66] considering various hard and soft constraints to solve a real-world UCSP. Grammatical Evolution is used in [67] for solving UCSP.

2.3 Observation on Existing Methods

PSO is popular due to its computational simplicity and adaptation ability [62]. PSO works with a population of particles; and, at every iteration, it calculates velocities for individual particles and adjusts those positions based on the velocities. PSO was proposed for continuous optimization (also called function optimization) in floating point numeric domain where interaction among particles (i.e., velocity calculation) are maintained through mathematical operations (e.g., addition, subtraction, and multiplication). On the other hand, any timetabling, especially UCSP, is a combinatorial optimization task and discrete in nature. To deal with such optimization tasks using PSO, high school timetabling [63], [64] and university course scheduling [11], [16] problems are first transformed into floating point numeric domain and then PSO is applied to obtain a workable solution. In these methods, a particle is encoded through floating point numeric representation of courses, instructors, rooms; and PSO operations (i.e., floating point numeric operations of velocity calculation and position update) are applied to improve it. Instead of transforming the UCSP to floating point numeric domain, a practical approach would be the modification of the algorithm. Performance of GSO on 23 unimodal and low dimension multimodal benchmark functions is verified in [21], [22], which shows competitive performance of GSO over GA and PSO. GSO is also applied for training of weights of a three-layer feedforward neural network, which is considered a continuous hard optimization problem due to the high-dimensional multimodal search space [21], [22]. GSO also shows competitive performance in this case compared to scaled conjugate gradient, GA, evolutionary programming, evolution strategies, and PSO. To the best of our knowledge, GSO has not been used to solve UCSP.

2.4 Scope of Research

From the observation of existing methods, it is clear that there is a scope of transforming the PSO for UCSP without transforming the UCSP to floating point numeric domain. Also, GSO seems to be promising for solving UCSP. As GSO has not been used to solve UCSP there is a scope of utilizing GSO for optimizing UCSP. In this study, swap sequence has been adapted for velocity/movement calculation; and selective search as well as forceful swap operation with repair mechanism has been introduced to PSO and GSO for dealing with highly constrained nature of UCSP.

CHAPTER III

Methodology

This chapter presents the proposed Particle Swarm Optimization with Selective Search (PSOSS) and Group Search Optimizer with Selective Search (GSOSS) methods in detail in different subsections explaining the transformation of PSO and GSO operations to tackle the combinatorial optimization task of University Course Scheduling Problem (UCSP) and the additional new operations introduced in PSOSS and GSOSS. Section 3.1 introduces some terminologies used in this study. Section 3.1.1 describes solution encoding for the UCSP in consideration. Section 3.1.2 describes the fitness calculation process for a solution. Section 3.1.3 introduces swap operator and swap sequences for UCSP. Forceful swap operation with repair mechanism and selective search, the added significant features of PSOSS and GSOSS, are described in section 3.1.4 and section 3.1.5, respectively. Section 3.2 presents the PSOSS method for solving UCSP. Sections 3.2.1 and section 3.2.2 explain the transformation of PSO operations to handle UCSP. The algorithm and illustration of the working procedure of PSOSS are presented in section 3.2.3 and section 3.2.4, respectively. Section 3.3 presents the GSOSS method for solving UCSP. Section 3.3.1 to section 3.3.5 explain the transformation of GSO operations to handle UCSP. The algorithm and illustration of working procedure of GSOSS are presented in section 3.3.6 and section 3.3.7, respectively.

3.1 Terminologies

Some terminologies including solution encoding for UCSP, solution’s fitness calculation process, Swap Operator and Swap Sequence for UCSP, Forceful swap operation with repair mechanism, selective search, population initialization are described in following subsections.

3.1.1 Solution Encoding for UCSP

The UCSP in consideration consists of u number of instructors, q number of courses, s number of rooms, and h number of timeslots. Each yearly student intakes are grouped together in a batch and the total number of batches is o. Each batch may be further subdivided into k number of subgroups. The proposed PSOSS uses z number of particles to solve the UCSP. Sets of instructors, courses, rooms, timeslots, batches, subgroups are represented as follows:

$-Set\ of\ Instructors, I = \{e_i \mid i = 1, \dots, u\}$

$-Set\ of\ Courses, C = \{c_i \mid i = 1, \dots, q\}$

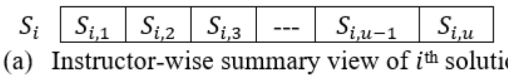
$-Set\ of\ Rooms, R = \{r_i \mid i = 1, \dots, s\}$

$-Set\ of\ Timeslots, T = \{t_i \mid i = 1, \dots, h\}$

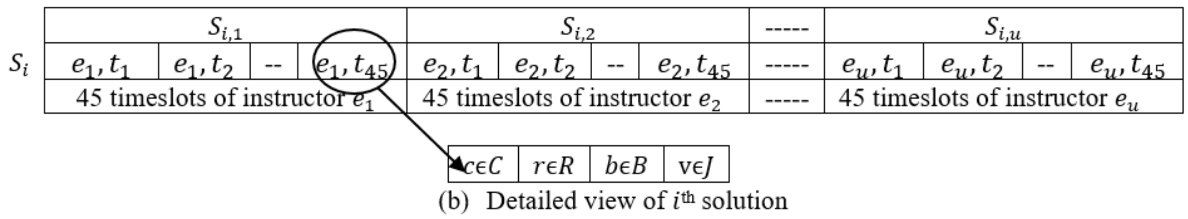
$-Set\ of\ Batches, B = \{b_i \mid i = 1, \dots, o\}$

$-Set\ of\ Subgroups, J = \{v_i \mid i = 1, \dots, k\}$

Each instructor object contains information about assigned courses and preference for conducting a class in a particular timeslot. Each course object contains information about the number of timeslots required for a class, number of classes per week, course type (theory/laboratory) and number of students per class. Each room object contains information about allowed courses and seat capacity. Each timeslot represents a teaching period and the total number of timeslots (h) is 45 (9 periods in each day for 5 working days of a week). Each batch object contains subgroups and information about number of assigned students to that batch. Each subgroup object contains information about the number of assigned students. A Solution of UCSP in consideration, S_i where i is the solution number is represented by instructor-wise solutions in a one-dimensional matrix as shown in Fig. 3.1. Fig. 3.1 (a) is the instructor-wise summary view of a solution where $S_{i,1}$, $S_{i,2}$, ... , $S_{i,u}$ denote the 1st, 2nd,…, uth instructor's solution, respectively. Fig. 3.1 (b) shows a solution's detailed



(a) Instructor-wise summary view of i^{th} solution.



(b) Detailed view of i^{th} solution

Figure 3.1: Solution representation for UCSP.

		Sun	Mon	Tue	Wed	Thu	------	Sun	Mon	Tue	Wed	Thu
Timeslot	1	e_1, t_1	e_1, t_{10}	e_1, t_{19}	e_1, t_{28}	e_1, t_{37}	------	e_u, t_1	------	------	------	------
	2	e_1, t_2	e_1, t_{11}	e_1, t_{20}	e_1, t_{29}	e_1, t_{38}	------	e_u, t_2	------	------	------	------
	3	e_1, t_3	e_1, t_{12}	e_1, t_{21}	e_1, t_{30}	e_1, t_{39}	------	e_u, t_3	------	------	------	------
	4	e_1, t_4	e_1, t_{13}	e_1, t_{22}	e_1, t_{31}	e_1, t_{40}	------	e_u, t_4	------	------	------	------
	5	e_1, t_5	e_1, t_{14}	e_1, t_{23}	e_1, t_{32}	e_1, t_{41}	------	------	------	------	------	------
	6	e_1, t_6	e_1, t_{15}	e_1, t_{24}	e_1, t_{33}	e_1, t_{42}	------	------	------	------	------	------
	7	e_1, t_7	e_1, t_{16}	e_1, t_{25}	e_1, t_{34}	e_1, t_{43}	------	------	------	------	------	------
	8	e_1, t_8	e_1, t_{17}	e_1, t_{26}	e_1, t_{35}	e_1, t_{44}	------	------	------	------	------	e_u, t_{44}
	9	e_1, t_9	e_1, t_{18}	e_1, t_{27}	e_1, t_{36}	e_1, t_{45}	------	------	------	------	------	e_u, t_{45}

Figure 3.2: Mapping of timeslots of a solution in days and periods.

view of 45 timeslots for each instructor and each timeslot identifies one of the nine teaching periods in a day for five working days in a week. The total number of timeslots is $45 \times u$ for u instructors; timeslots $e_1, t_1 - e_1, t_{45}$ for the first instructor, $e_2, t_1 - e_2, t_{45}$ for the second instructor, and so on. Each timeslot comprises assigned course, room, batch, and subgroup information as shown in Fig. 3.1(b). Fig. 3.2 represents the mapping of timeslots of a solution in days and periods. As an example, timeslot e_1, t_{44} represents 5th working day's 8th period of the first instructor.

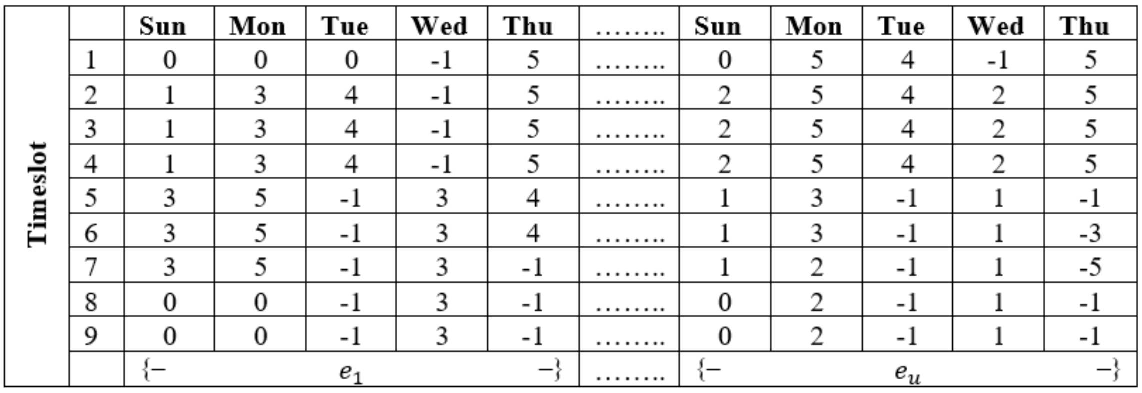
3.1.2 Fitness Calculation

Each instructor's preference for conducting a class in a particular timeslot is represented by an integer value as shown in Fig. 3.3. A higher value corresponds to a higher preference of an instructor to conduct the class in that particular timeslot. Whereas, a negative value shows the instructor's non-preference. The fitness of ith solution S_i is calculated by considering fitness of each of the instructor's solution which belongs to that solution using the following equation:

$$F(S_i) = \sum_{j=0}^{u} F(S_{i,j}), \tag{3.1}$$

where, $F(S_i)$ is the fitness of the solution, and $F(S_{i,j})$ is the fitness of the ith solution's jth instructor's solution. Now, fitness of each instructor's solution is calculated by considering quality and violation of the instructor's solution using the following equation:

$$F(S_{i,j}) = Q(S_{i,j}) - V(S_{i,j}), \tag{3.2}$$



Timeslot		Sun	Mon	Tue	Wed	Thu	……..	Sun	Mon	Tue	Wed	Thu
	1	0	0	0	-1	5	……..	0	5	4	-1	5
	2	1	3	4	-1	5	……..	2	5	4	2	5
	3	1	3	4	-1	5	……..	2	5	4	2	5
	4	1	3	4	-1	5	……..	2	5	4	2	5
	5	3	5	-1	3	4	……..	1	3	-1	1	-1
	6	3	5	-1	3	4	……..	1	3	-1	1	-3
	7	3	5	-1	3	-1	……..	1	2	-1	1	-5
	8	0	0	-1	3	-1	……..	0	2	-1	1	-1
	9	0	0	-1	3	-1	……..	0	2	-1	1	-1
		{–		e_1		–}	……..	{–		e_u		–}

Figure 3.3: Sample preference values for instructors.

where, $Q(S_{i,j})$ is the quality of the instructors' solution, and $V(S_{i,j})$ is the violation of the instructor's solution. Preference values of corresponding positions where courses are assigned to an instructor are summed up to calculate the quality of an instructor's solution. Violation of the instructors' solution is calculated using the following equation:

$$V(S_{i,j}) = \sum_{a=1}^{tc} 2^{l_a}, \tag{3.3}$$

where, tc is the total number of blocks of consecutive classes in an instructor's solution and l_a is the number of classes in ath block. The exponential in Eq. (3.3) is used to mimic the human nature. If the number of consecutive classes increases, then the dissatisfaction of an instructor increases rapidly. For example, three consecutive classes are much more difficult (practically almost impossible) to manage for an instructor than two consecutive classes.

3.1.3 Swap Operator and Swap Sequence for UCSP

A Swap Operator (SO) denotes the index of items to be swapped in a list [68]–[70]. Consider the list $A = \{a, b, c, d\}$ with indices $\{0, 1, 2, 3\}$:

A	a	b	c	d
Index	0	1	2	3

A $\mathrm{SO}(1,3)$ produces a new list $B = A + \mathrm{SO}(1,3)$ as follows:

$$\overset{\longleftrightarrow}{}$$

B	a	**d**	c	**b**
Index	0	1	2	3

here, '+' does not mean any arithmetic operation rather it means the swap operation $\mathrm{SO}(i, j)$ on A.

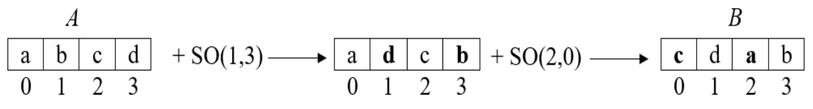
A Swap Sequence (SS) is a group of SOs defined as [68]–[70]:

$$SS = \{ SO_1, SO_2, SO_3, \cdots, SO_n\} \tag{3.4}$$

Deployment of an SS means application of all the SOs in an SS in that very particular order. Moreover, if applying SS on a list A yields a list B (i.e., $B = A + SS$), then it can be written as

$$SS = B - A \tag{3.5}$$

For example, if $SS = \{(1,3), (2,0)\}$ then, $B = A + SS$. This operation is illustrated below:



The SS to convert one solution of a UCSP to another one is a collection of swap sequences which is measured in an instructor-to-instructor basis. Consider a UCSP consisting of two instructors e_1 and e_2 each having two courses $\{c_1, c_2\}$ and $\{c_3, c_4\}$, respectively. Fig. 3.4 shows two different solutions A and B for the UCSP in consideration. In solution A, instructor e_1 has a course c_1 in timeslot 1 and another course c_2 in timeslot 3 whereas in solution B, course c_1 is in timeslot 0 and c_2 is in timeslot 2. So, the required SS for converting the schedule of e_1 in solution A to schedule of e_1 in solution B is $SS_1 = \{(1,0), (3,2)\}$. Similarly, $SS_2 = \{(0,3), (2,1)\}$. So, the complete swap sequence for converting solution A to solution B is $SS = \{SS_{i_1}, SS_{i_2}\} = \{\{(1,0), (3,2)\}, \{(0,3), (2,1)\}\}$.

3.1.4 Forceful Swap Operation with Repair Mechanism

Forceful Swap Operation with Repair Mechanism is an added feature of PSOSS and GSOSS methods. UCSP is highly constrained in nature and most of the constraints are interrelated. Consequently, if a class needs to be shifted to a new timeslot then all the involved members

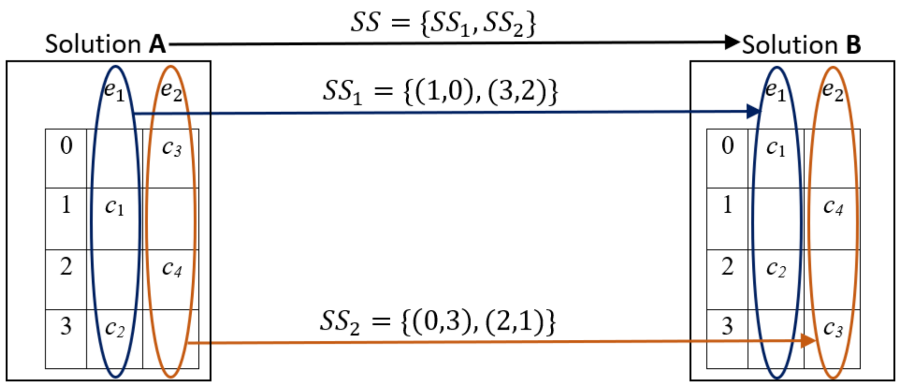


Figure 3.4: Instructor wise Swap Sequences (SSs) of complete SS.

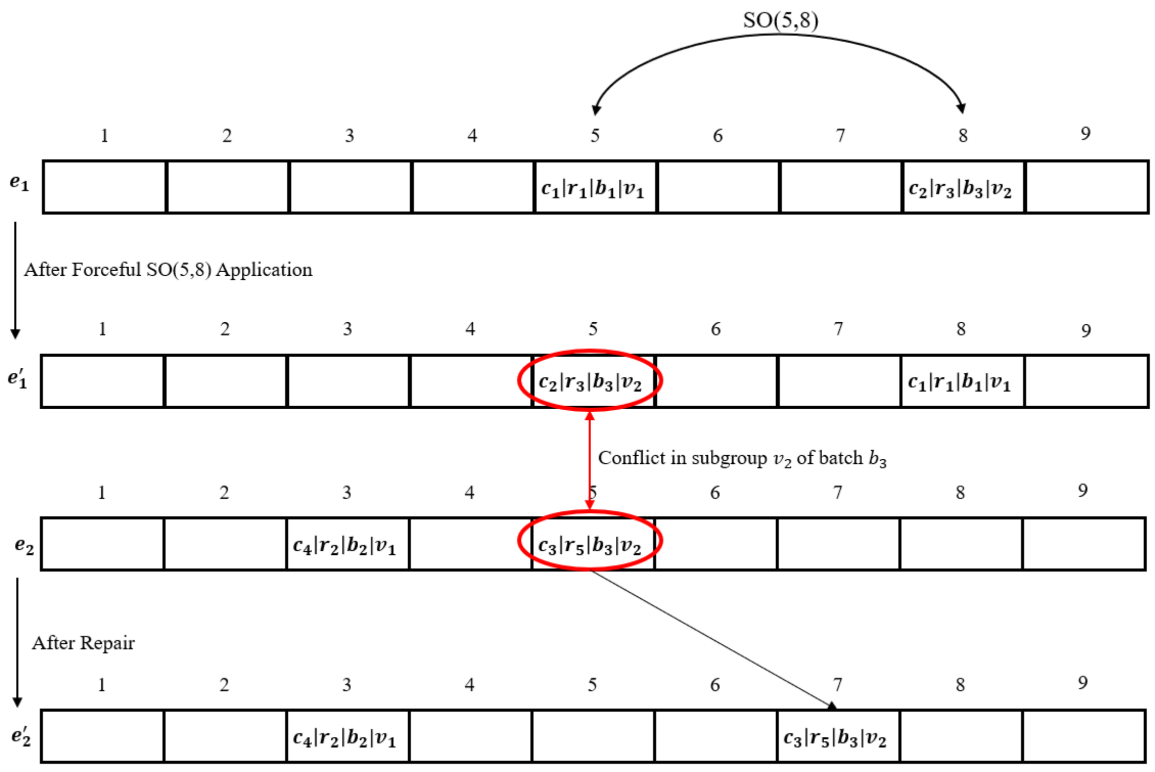


Figure 3.5: Illustration of the repair mechanism.

such as instructor, students, and room need to be free in that timeslot. As a result, most of the swaps cannot be applied because of violation of constraints. Forcefully applying an SO can cause conflicts. Therefore, a repair mechanism is involved in forceful SO operation to make sure that no invalid solution results in that process. The repair mechanism works by randomly moving conflicting courses to non-conflicting positions. The repair mechanism is illustrated in Fig. 3.5. Consider a UCSP instance consisting of two instructors e_1 and e_2. Instructor e_1 has course c_1 of batch b_1, subgroup $v_1$in room r_1 and course c_2 of batch b_3, subgroup v_2 in room r_3 at timeslots 5 and 8, respectively. Instructor e_2 has course c_4 of batch b_2, subgroup v_1 in room r_2 and course c_3 of batch b_3, subgroup v_2 in room r_5 at timeslots 3 and 5, respectively. Now, if an SO(5,8) is applied forcefully on the solution of e_1 then, course c_2 of e_1 comes at timeslot 5 which results in a conflict with e_2 because subgroup v_2 of batch b_3 is already engaged in a class with e_2 (shown with circle in Fig. 3.5). This conflict is resolved by moving the conflicting course of e_2 to a randomly chosen non-conflicting timeslot 7.

3.1.5 Selective Search

Selective search is one of the most important features of PSOSS and GSOSS. In selective search, each solution generated by applying an SO of SS is considered as an intermediate solution. Suppose, $SS = \{SO_1, SO_2, SO_3, \cdots, SO_n\}$ then the selective search can be written as

$$\begin{gathered} SC^1 = SC + SO_1 \\ SC^2 = SC^1 + SO_2 \\ \vdots \\ SC^n = SC^{n-1} + SO_n \end{gathered}$$

In the above cases, $SC^1, SC^2, \cdots, SC^n$ are the intermediate solutions and the intermediate solution having the highest fitness becomes the final solution SC in selective search as defined by the following equation:

$$SC = \max\{SC^f\}, \qquad f = 1,2,\cdots,n \tag{3.6}$$

SS generating the final solution is $SS = \{SO_1, SO_2, SO_3, \cdots, SO_f\}$, $1 \leq f \leq n$.

The ultimate solution SC in selective search is the intermediate solution possessing the highest fitness value. Thus, the selective search technique explores the opportunity of keeping better solution from the intermediate solutions.

3.1.6 Population Initialization

Like other population-based algorithms, the proposed PSOSS and GSOSS methods also starts with an initial population. Each individual (particle in PSOSS and member in GSOSS) in the initial population is assigned a random solution. A random solution is generated by randomly assigning timeslots to the assigned courses of all instructors. Timeslots are allocated by maintaining all the hard constraints and it is checked that courses requiring multiple consecutive slots do not include any break periods. Weighted random distribution is used for assigning rooms to courses so that, a room having the highest load has the lowest probability of getting selected.

3.2 PSO with Selective Search (PSOSS) for Solving UCSP

Proposed PSOSS method works with a population of particles in which individual particle contains a feasible solution, calculates velocity of each individual particle using swap sequence and updates each particle with the computed velocity through selective search and forceful swap operation with repair mechanism. The particle encoding, velocity computation and other operations are described in the following subsections.

3.2.1 Particle Encoding

The proposed PSOSS uses z number of particles to solve the UCSP. Set of particles is represented as follows:

$-Set\ of\ Particles, P = \{p_i \mid i = 1, \dots, z\}$

Each particle object contains a feasible solution to the UCSP i.e. the complete schedule for instructors, batches, and rooms.

3.2.2 Velocity Computation using Swap Operator and Swap Sequence

SO and SS have been used in the proposed PSOSS method for velocity calculation. In the proposed method, swap sequence SS is treated as velocity to update a particle's solution at each iteration which is calculated using the following equation

$$SS = \alpha(SGB - SC) + \beta(SPB - SC) \otimes \gamma SSPA \tag{3.7}$$

where, SGB is the global best solution of the swarm, SC is the current so lution of the particle, SPB is the personal best solution of the particle, $SSPA$ is the previously velocity applied, $\otimes$ is merge operation and $\{\alpha, \beta, \gamma\}$ are selection probabilities for selecting a bunch of SOs from the corresponding SSs. $SGB - SC$ represents instructor-wise SSs to reach SGB from SC and $SPB - SC$ is the instructor-wise SSs to reach SPB from SC.. If $SSGB = SGB - SC$ and $SSPB = SPB - SC$ then, Eq. (3.7) can be rewritten as:

$$SS = \alpha SSGB + \beta SSPB \otimes \gamma SSPA \tag{3.8}$$

After selection of SOs with $\{\alpha, \beta, \gamma\}$, SS becomes

$$SS = SSSGB + SSSPB \otimes SSSPA = SSSGB + SSM \tag{3.9}$$

where, $SSSGB, SSSPB, SSSPA$ are the selected SS from $SSGB, SSPB$ and $SSPA$, respectively and SSM is the swap sequence resulting from merging $SSSPB$ with $SSSPA$. As $SSSGB$ and SSM may contain redundant SOs, redundant swaps are removed from them and they become $SSSGBM$ and $SSMM$, respectively. Finally, SS becomes:

$$SS = SSSGBM + SSMM \tag{3.10}$$

A portion of $SSSGBM$ is forcefully applied to current solution SC to ensure that SC moves a little towards SGB. Selective search is used to retain best intermediate solution while applying SS. The sequence of SOs generating the best intermediate solution is considered as the final velocity which becomes the previously applied velocity for the next iteration.

3.2.3 PSOSS Algorithm for Solving UCSP

The proposed PSOSS method for solving UCSP is shown in Algorithm 3.1.1. The notations and inputs of the proposed algorithm are listed at the beginning of the Algorithm 3.1.1.

Algorithm 3.1.1: PSOSS-UCSP

Input:

Instructors' Information, Batches' Information,

Courses' Information, Classrooms' Information, Break Times' information,

n	-	total number of iterations
z	-	total number of particles
u	-	total number of instructors
α	-	selection probability of a swap from swap sequence to global best solution
β	-	selection probability of a swap from swap sequence to personal best solution
γ	-	selection probability of a swap from previously applied swap sequence
f	-	percentage of swaps to be forced towards global best solution
l	-	length of random swap sequence

Output:

An optimal solution of UCSP

Variables:

t	-	iteration counter
SC_i	-	ith particle's current solution
SR	-	A random solution
SPB_i	-	ith particle's personal best solution
$SSPA_i$	-	ith particle's previously applied swap sequence
SGB	-	global best solution
SSH	-	swap sequence holder for selective search
SH	-	solution holder for selective search
SSR	-	random swap sequence
$SSGB$	-	swap sequence to global best solution
$SSPB$	-	swap sequence to personal best solution
$SSSGB$	-	selected swap sequence to global best solution
$SSSPB$	-	selected swap sequence to personal best solution

$SSSPA$	-	selected swap from previously applied swap sequence
SSM	-	swap sequence by merging $SSSPB$ and $SSSPA$
$SSSGBM$	-	minimized swap sequence from $SSSGB$
$SSSGBM_j$	-	j^{th} instructor's swap sequence in $SSGBM$
$SSMM$	-	minimized swap sequence from SSM
$SSMM_j$	-	j^{th} instructor's swap sequence in $SSMM$
NSF	-	number of swaps to be forced towards global best solution
$SSCA$	-	currently applied swap
P	-	set of particles
CC	-	set of conflicting classes

Step 1: Initialization

```
1.  $t \leftarrow 1$
2.  create $z$ number of particles and append them to $P$
3.  **for** $i \leftarrow 1$ to $z$ **do** //*for each particle*
4.      $SC_i \leftarrow SR$
5.      calculate fitness of $SC_i$ as described in section 3.1.2
6.      $SPB_i \leftarrow SC_i$
7.      $SSPA_i \leftarrow \emptyset$
8.  **end for**
9.  $SGB \leftarrow$ *solution*(max $P$)
```

Step 2: Computation and application of velocity

```
1.  **for** $i \leftarrow 1$ to $z$ **do** //*for each particle*
2.      $SSH \leftarrow \emptyset$
3.      $SH \leftarrow \emptyset$
4.      $SSCA \leftarrow \emptyset$
5.      **Step 2.1: Calculate velocity using Eq. (3.7), Eq. (3.8), Eq. (3.9) and Eq. (3.10)**
6.          **if** $SSPA_i = \emptyset$ **then**
7.              $SSPA_i \leftarrow SSR$ of length $l$
8.          **end if**
9.          $SSGB \leftarrow SGB - SC_i$
10.         $SSPB \leftarrow SPB_i - SC_i$
```

```
11.         $SSSGB \leftarrow \alpha * SSGB$
12.         $SSSPB \leftarrow \beta * SSPB$
13.         $SSSPA \leftarrow \gamma\ * SSPA_i$
14.         $SSM \leftarrow SSSPB \otimes\ SSSPA$ //merge $SSSPB$ with $SSSPA$
15.         $SSSGBM \leftarrow$ swapMinimizer($SSSGB$) //remove redundant swaps
16.         $SSMM \leftarrow$ swapMinimizer( $SSM$)
17.     Step 2.2: Apply $f$ percent of swaps from $SSSGBM$ using forceful swap
        operation with repair mechanism and selective search
18.         for $j \leftarrow 1$ to $u$ do //for each instructor
19.             $NSF \leftarrow f * |SSSGBM_j|$
20.             for $a \leftarrow 1$ to $NSF$ do
21.                 $SC_i \leftarrow SC_i + SSSGBM_j[a]$ forcefully
22.                 Step 2.2.1: Repair mechanism
23.                     $CC \leftarrow$list of conflicting classes in $SC_i$ resulting from $SSSGBM_j[a]$
                            application
24.                     if $CC \neq \emptyset$ then
25.                         for all $cc \in CC$
26.                             move $cc$ to a randomly selected non-conflicting position
27.                         end for
28.                     end if
29.                 $SSCA \leftarrow SSCA \cup \{SSSGBM_j[a]\}$
30.                 selectiveSearch($SC_i$, $SSCA$, $SH$, $SSH$)
31.             end for
32.         end for
33.     Step 2.3: Apply remaining swaps of $SSSGBM$ using selective search
34.         for $j \leftarrow 1$ to $m$ do //for each instructor
35.             $NSF \leftarrow f * |SSSGBM_j|$
36.             for $a \leftarrow NSF + 1$ to $|SSSGBM_j|$ do
37.                 if $SSSGBM_j[a]$ is applicable then
38.                     $SC_i \leftarrow SC_i + SSSGBM_j[a]$
39.                     $SSCA \leftarrow SSCA \cup \{SSSGBM_j[a]\}$
40.                     selectiveSearch($SC_i$, $SSCA$, $SH$, $SSH$)
```

```
41.                 end if
42.             end for
43.         end for
44.     Step 2.4: Apply $SSMM$ using selective search
45.         for $j \leftarrow 1$ to $u$ do //for each instructor
46.             for $a \leftarrow 1$ to $|SSMM_j|$ do
47.                 if $SSMM_j[a]$ is applicable then
48.                     $SC_i \leftarrow SC_i + SSMM_j[a]$
49.                     $SSCA \leftarrow SSCA \cup \{SSMM_j[a]\}$
50.                     selectiveSearch($SC_i, SSCA, SH, SSH$)
51.                 end if
52.             end for
53.         end for
54.     Step 2.5: Update $SC_i$, $SSPA_i$ and $SPB_i$
55.         $SC_i \leftarrow SH$
56.         $SSPA_i \leftarrow SSH$
57.         calculate fitness of $SC_i$ as described in section 3.1.2
58.         if fitness($SC_i$) > fitness($SPB_i$) then
59.             $SPB_i \leftarrow SC_i$
60.         end if
61. end for
```

Step 3: Update SGB

```
1.  $SGBT \leftarrow$ solution(max $P$)
2.  if fitness($SGBT$) > fitness($SGB$) then
3.      $SGB \leftarrow SGBT$
4.  end if
```

Step 4: Check the stopping criterion

```
1.  while $t \neq n$ do
2.      $t \leftarrow t + 1$
3.      repeat Step 2 and Step 3
```

```
4.  end while
```

Step 5: Return SGB as result

Algorithm 3.1.2 selectiveSearch

```
Input: $SC_i, SSCA, SH, SSH$
1.  if $SH = \emptyset \vee fitness(SH) < fitness(SC_i)$ then
2.      $SH \leftarrow SC_i$
3.      $SSH \leftarrow SSCA$
4.  end if
```

In the proposed method, the initial population of particles is generated by creating a specified number of particles. Each particle's current solution SC_i gets initialized by a random solution SR as described in section 3.1.6. The fitness of each particle's current solution is calculated as described in section 3.1.2. SC_i also becomes the personal best solution SPB_i initially. The previously applied swap sequence for a particle $SSPA_i$ is initially empty. The solution having the highest fitness among all the particles is selected as the global best solution SGB. In each iteration, velocity for each particle is calculated using (3.7), Eq. (3.8), Eq. (3.9) and Eq. (3.10). *SH* is used to hold best intermediate solution and *SSH* holds the swap sequence that produces *SH*. For each particle, *SH* and *SSH* are emptied to be used for selective search. $SSCA$ is used to hold the applied swap sequence on a particle's solution in each iteration and it is also emptied initially. If the previously applied swap sequence $SSPA_i$ of a particle is empty, then a randomly generated swap sequence SSR of length l is assigned to $SSPA_i$. SSR is generated by creating l number of unique SOs for each instructor. For example, if a UCSP instance consists of two instructors namely e_1 and e_2 then a possible random swap sequence of length 2 is {(2,10),(5,8)} for e_1 and {(3,6),(1,7)} for e_2. Then, instructor-wise swap sequences to reach SGB and SPB_i from SC_i are calculated which are represented by $SSGB(= SGB - SC_i)$ and $SSPB(= SPB_i - SC_i)$, respectively. Some swaps are selected for each instructor from $SSGB$ based on the selection probability α denoted by $SSSGB(= \alpha * SSGB)$. Similarly, $SSSPB(= \beta * SSPB)$ and $SSSPA(= \gamma * SSPA_i)$ are the selected swaps from $SSPB$ and $SSPA_i$, respectively. $SSSPB$ and $SSSPA$ are merged together to $SSM(= SSSPB \otimes SSSPA)$. Redundant swaps are removed from $SSSGB$ and SSM using *swapMinimizer*() function, results of which are denoted by $SSSGBM$ and $SSMM$, respectively. After that, for each instructor a portion, $f * |SSSGBM_j|$ of $SSSGBM$ is selected where, $SSSGBM_j$ is the swap sequence corresponding to jth instructor and f is the percentage of swaps to be forced towards global best solution. Swaps of this selected portion are forcefully applied to SC_i and resulting conflicts are resolved by randomly moving the conflicting classes to non-conflicting positions as described in section 3.1.4. Rest of the swaps are applied to SC_i if they do not create any conflicts. Similarly, the swaps from $SSMM$ are applied only if they are applicable. Each applied swap gets added to $SSCA$ which holds currently applied swap sequence for a particle and the selective search technique is used after applying each swap to ensure that the best intermediate solution is retained. Algorithm 3.1.2 shows the required steps of selective search. It simply updates SH and SSH with SC_i and $SSCA$, respectively only if SC_i is found to be better than SH. Finally, after the application of all the swaps the best intermediate solution SH becomes particle's solution SC_i and the swap sequence SSH that produces SH becomes $SSPA_i$ for next iteration. Then, SPB_i is updated if SC_i is found better than SPB_i. Finally, SGB is recalculated and the algorithm goes to the next iteration. The algorithm uses a predefined number of iterations n as the termination criterion. After termination SGB is considered as the final outcome.

3.2.4 Illustration of Solution Update Mechanism in PSOSS

A schematic representation of the solution update mechanism of a particle in the proposed method is shown in Fig. 3.6. Suppose, a system consists of three instructors e_1, e_2 and e_3, each having five weekly timeslots. In Fig. 3.6, SC is a particle's current solution which consists of individual solution of all three instructors, SGB is the global best solution and SPB is the particle's personal best solution. $SSGB(= SGB - SC)$ represents instructor-wise swap sequences to reach SGB from SC , and $SSPB(= SPB - SC)$ is the instructor-wise swap sequences to reach SPB from SC. $SSPA$ is the previously applied swap sequence. The circle (○) symbol inside the swap sequences represents a swap operator. In the first step, some swaps are selected for each instructor from $SSGB$ based on the selection probability α denoted by $SSSGB(= \alpha * SSGB)$. Similarly, $SSSPB(= \beta * SSPB)$ and $SSSPA(= \gamma * SSPA)$ are the selected swaps from $SSPB$ and $SSPA$, respectively. The redundant swaps are removed from $SSSGB$ using *swapMinimizer*() function, result of which is denoted by $SSSGBM$ (swaps numbered 1, 2, 3, 4, 5 and 6 in Fig. 3.6). Then, $SSSPB$ and $SSSPA$ are merged together to $SSM(= SSSPB \otimes SSSPA)$ before removing the redundant swaps from

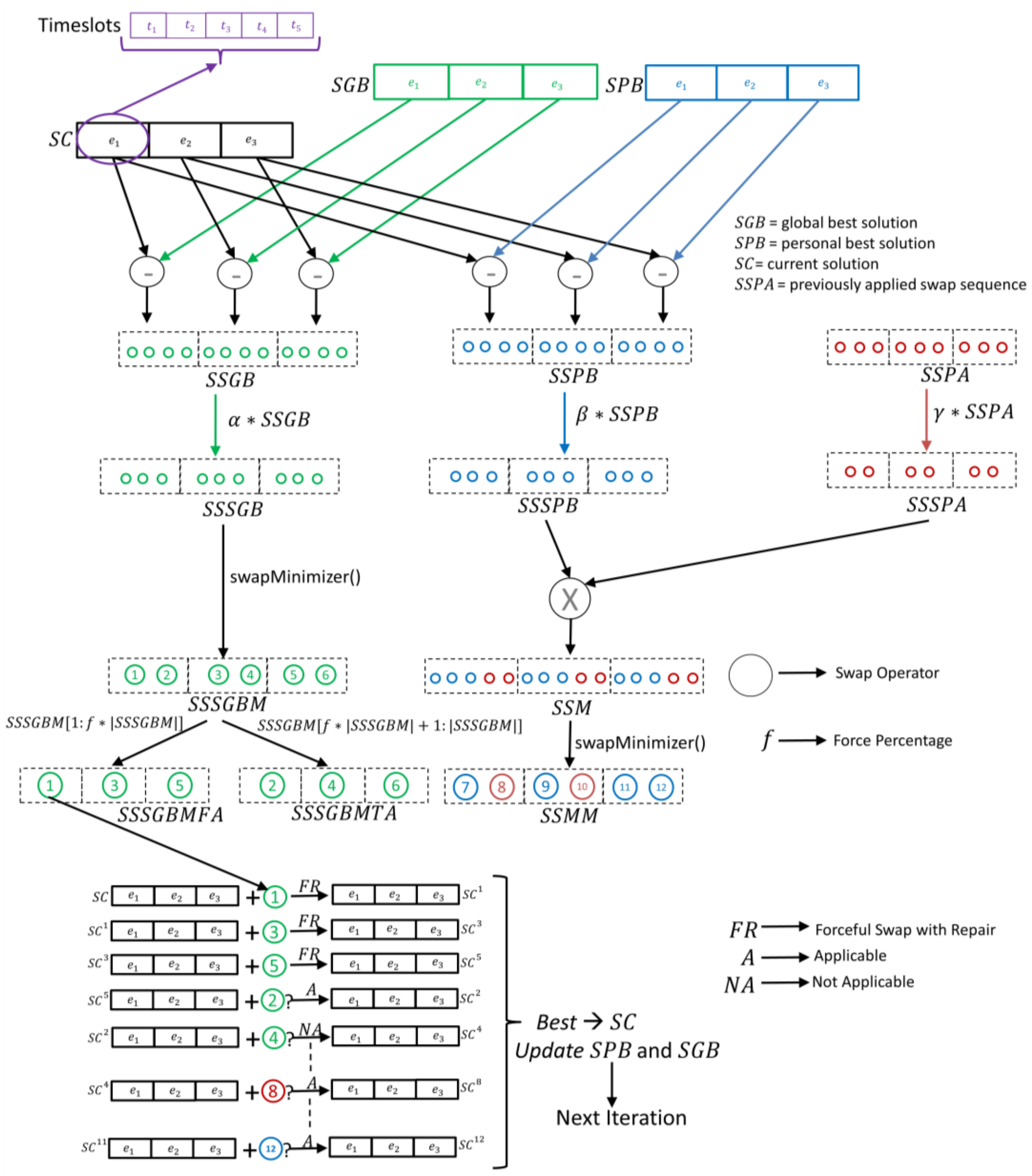


Figure 3.6: Illustration of the solution update mechanism of a particle in PSOSS.

them. The redundant swaps are removed from SSM using *swapMinimizer*() function, result of which is denoted by $SSMM$ (swaps numbered 7, 8, 9, 10, 11 and 12 in Fig. 3.6). After that, a portion of $SSSGBM$ is selected using the equation $* |SSSGBM|$, where f is the force percentage. This selected portion of $SSSGBM$ is denoted by $SSSGBMFA$ (swaps numbered 1, 3 and 5 in Fig. 3.6) and the rest of the swaps are denoted by $SSSGBMTA$ (swaps numbered 2, 4 and 6 in Fig. 3.6). Swaps of $SSSGBMFA$ are forcefully applied to SC and then the swaps of $SSSGBMTA$ are applied to SC if they do not create any conflicts. Any conflict resulting from forceful swap operation is handled by repair mechanism as described in section 3.1.4. Similarly, the swaps from $SSMM$ are applied only if they are applicable. In Fig. 3.6, a solution resulting from application of a swap is represented by assigning that swap number above the solution. For example, if a swap say 1 is applied on solution SC then it becomes SC^1 and similarly applying swap 3 on SC^1 makes it SC^3. In the example shown in Fig. 3.6, swaps numbered 1, 3 and 5 are forcefully applied on initial solution SC making it SC^5. Then, swap numbered 2 gets applied on SC^5 resulting in SC^2 as there is no conflicts. Solution stays at SC^2 because swaps numbered 4, 6 and 7 are not applied because of conflicts. Then, rest of the swaps 8, 9, 10, 11 and 12 are applied because they do not cause any conflicts. The best one among these solutions is then picked as particle's current solution SC. Accordingly, SPB and SGB are updated for the next iteration.

3.3 GSO with Selective Search (GSOSS) for Solving UCSP

Proposed GSOSS method works with a group of members in which individual member represents a feasible solution, calculates movement of scroungers and dispersed members using swap sequence and updates each member with the computed movement through selective search and forceful swap operation with repair mechanism. The member encoding, categorization, producer's scanning, scrounging, dispersed members' random operation and other operations are described in the following subsections.

3.3.1 Member Encoding

The proposed GSOSS uses w number of members, x number scroungers and y number of dispersed members to solve the UCSP. Sets of members, scroungers and dispersed members are represented as follows:

$-Set\ of\ Members, M = \{m_i \mid i = 1, \dots, w\}$

$-Set\ of\ Scroungers, SC = \{sc_i \mid i = 1, \dots, x\}$

$-Set\ of\ Dispersed\ Members, D = \{d_i \mid i = 1, \dots, y\}$

Each member object contains a feasible solution to the UCSP. SC and D are subsets of M.

3.3.2 Member Categorization

GSOSS works with a population of members. The fitness function of Eq. (3.1) described in section 3.1.2 is used to calculate fitness of solutions of all the members of a population. The member having best fitness solution is selected as producer, worst 20% members become dispersed members and rest of the members become scroungers.

3.3.3 Producer's Scanning

In each iteration of GSOSS algorithm producer tries to improve fitness of its solution. This is done by shifting courses of producer's solution to slots having higher preference values from respective instructors.

3.3.4 Scrounging

In the proposed GSOSS method swap sequence is used for calculating the movement of scroungers. For each scrounger sc a swap sequence $SSPR$ to reach producer is first calculated using following equation:

$$SSPR = SPR - Ssc \tag{3.11}$$

where $SSPR$ is the producer's solution and Ssc is the scrounger's solution. Then a portion of $SSPR$ is selected with swap selection probability $\alpha \epsilon [0,1]$:

$$SSSPR = \alpha * SSPR \tag{3.12}$$

Finally, redundant swaps are removed from $SSSPR$ and it becomes $SSSPRM$ which is used to move scrounger towards producer. $SSSPRM$ is applied on a scrounger solution using forceful swap operation with repair mechanism selective search. A portion of $SSSPRM$ is forcefully applied to a scrounger solution to ensure that the scrounger moves a little towards the producer.

3.3.5 Dispersed Members' Random Operation

In the proposed GSOSS method movement of a dispersed member is also calculated using swap sequence. In case of a disperse member, a randomly generated swap sequence SSR of length l is used for its random movement. SSR is generated by creating l number of unique SOs for each instructor. Before applying SSR on a dispersed member redundant swaps are removed from SSR and it becomes $SSRM$. $SSRM$ is applied on a dispersed member using forceful swap operation with repair mechanism and selective search.

3.3.6 GSO with Selective Search (GSOSS) for Solving UCSP

The proposed GSOSS method for solving UCSP is shown in Algorithm 3.2.1. The notations and inputs of the proposed algorithm are listed at the beginning of the Algorithm 3.2.1.

Algorithm 3.2.1: GSOSS-UCSP

Input:

Instructors' Information, Batches' Information, Courses' Information, Classrooms' Information, Break Times' information,

n	-	total number of iterations
w	-	total number of members
u	-	total number of instructors
α	-	selection probability of a swap from scrounger's swap sequence to producer
f	-	percentage of swaps to be forced towards producer
l	-	length of random swap sequence

Output:

An optimal solution of UCSP

Variables:

t	-	iteration counter
S_i	-	ith member's solution
PR	-	producer
SPR	-	producer's solution
SR	-	a random solution
SH	-	solution holder for selective search
SSR	-	random swap sequence
SSR_j	-	jth instructor's swap sequence in SSR_j
$SSPR$	-	swap sequence to producer's solution
$SSSPR$	-	selected swap sequence to producer's solution
$SSSPRM$	-	minimized swap sequence from $SSSPR$
$SSSPRM_j$	-	jth instructor's swap sequence in $SSGPRM$
NSF	-	number of swaps to be forced
M	-	set of members
SC	-	set of scroungers
x	-	total number of scroungers
D	-	set of dispersed members
y	-	total number of dispersed members
CC	-	set of conflicting classes

Step 1: Initialization

```
10. $t \leftarrow 1$
11. create $w$ number of members and append them to $M$
```

```
12. for $i \leftarrow 1$ to $w$ do //for each member
13.     $S_i \leftarrow SR$
14.     calculate fitness of $S_i$ as described in section 3.1.2
15. end for
```

Step 2: Member Categorization

```
1. descendingSort($M$) //descending sort members according to fitness value
2. $PR \leftarrow M[0]$ //select member having highest fitness as producer
3. $D \leftarrow$ worst 20% of $M$
4. $SC \leftarrow M - PR - D$
```

Step 3: Producer's Scanning

```
1. $PR \leftarrow$ improveProducer($PR$)
```

Step 4: Scrounging

```
62. for $i \leftarrow 1$ to $x$ do //for each scrounger
63.     $SH \leftarrow \emptyset$
64.     $SSPR \leftarrow SPR - S_i$
65.     $SSSPR \leftarrow \alpha * SSPR$
66.     $SSSPRM \leftarrow$ swapMinimizer($SSSPR$)
67.     Step 4.1: Apply $f$ percent of swaps from $SSSPRM$ using forceful swap
        operation with repair mechanism and selective search
68.         for $j \leftarrow 1$ to $u$ do //for each instructor
69.             $NSF \leftarrow f * |SSSPRM_j|$
70.             for $a \leftarrow 1$ to $NSF$ do
71.                 $S_i \leftarrow S_i + SSSGBM_j[a]$ forcefully
72.                 repairSolution($S_i$)
73.                 selectiveSearch($S_i, SH$)
74.             end for
75.         end for
76.     Step 4.2: Apply remaining swaps of $SSSPRM$ using selective search
77.         for $j \leftarrow 1$ to $u$ do //for each instructor
```

```
78.              $NSF \leftarrow f * |SSSPRM_j|$
79.              **for** $a \leftarrow NSF + 1$ to $|SSSPRM_j|$ **do**
80.                 **if** $SSSPRM_j[a]$ is applicable **then**
81.                    $S_i \leftarrow S_i + SSSGBM_j[a]$
82.                    *selectiveSearch*($S_{i,} SH$)
83.                 **end if**
84.              **end for**
85.           **end for**
86.        $S_i \leftarrow SH$
87.        calculate fitness of $S_i$ as described in section 3.1.2
88.     **end for**
```

Step 5: Dispersed Members' Random Operation

```
1.   **for** $i \leftarrow 1$ to $y$ **do** *//for each dispersed member*
2.      $SH \leftarrow \emptyset$
3.      $SSR \leftarrow$ random swap sequence of length $l$
4.      $SSRM \leftarrow$ *swapMinimizer*($SSR$)
5.      **Step 5.1: Apply *f* percent of swaps from $SSRM$ using forceful swap operation
        with repair mechanism and selective search**
6.         **for** $j \leftarrow 1$ to $u$ **do** *//for each instructor*
7.            $NSF \leftarrow f * |SSRM_j|$
8.            **for** $a \leftarrow 1$ **to** $NSF$ **do**
9.               $S_i \leftarrow S_i + SSRM_j[a]$ ***forcefully***
10.              *repairSolution*($S_i$)
11.              *selectiveSearch*($S_{i,} SH$)
12.           **end for**
13.        **end for**
14.     **Step 5.2: Apply remaining swaps of $SSRM$ using selective search**
15.        **for** $j \leftarrow 1$ to $u$ **do** *//for each instructor*
16.           $NSF \leftarrow f * |SSRM_j|$
17.           **for** $a \leftarrow NSF + 1$ to $|SSRM_j|$ **do**
18.              **if** $SSRM_j[a]$ is applicable **then**
```

```
19.                 $S_i \leftarrow S_i + SSRM_j[a]$
20.                 $selectiveSearch(S_{i,} SH)$
21.             **end if**
22.         **end for**
23.     **end for**
24.     $S_i \leftarrow SH$
25.     calculate fitness of $S_i$ as described in section 3.1.2
26. **end for**
```

Step 4: Termination and Outcome

```
5.  $M \leftarrow P \cup SC \cup D$
6.  **while** $t \neq n$ **do**
7.      $t \leftarrow t + 1$
8.      repeat **Step 2, Step 3** and **Step 4**
9.  **end while**
10. *descendingSort*($M$) *//descending sort members according to fitness value*
11. $PR \leftarrow M[0]$ *//select member having highest fitness as producer*
12. **return** $SPR$ *//consider producer's solution as outcome*
```

Algorithm 3.2.2 improveProducer

Input: PR *//Producer*

```
1.  $PR_{temp} \leftarrow copy(PR)$
2.  $C \leftarrow$ list of courses scheduled for $PR_{temp}$
3.  **for all** $c \in C$
4.      $SL \leftarrow$ list of slots having better preference values than $c$'s current position
5.      **for all** $sl \in SL$
6.          move $c$ to slot $sl$
7.          calculate fitness of $PR_{temp}$
8.          **if** *fitness*($PR$)<*fitness*($PR_{temp}$) **then**
9.              return $PR_{temp}$
10.         **end if**
11.     **end for**
```

```
12. end for
13. return $PR$
```

Algorithm 3.2.3 repairSolution

```
Input: $S_i$ //solution to repair
14. $CC$ ← list of conflicting classes in $S_i$
15. if $CC \neq \emptyset$ then
16.     for all $cc \in CC$
17.         move $cc$ to a randomly selected non-conflicting position
18.     end for
19. end if
```

Algorithm 3.2.4 selectiveSearch

```
Input: $S_i$ , $SH$
5. if $SH = \emptyset \vee fitness(SH) < fitness(S_i)$ then
6.     $SH \leftarrow S_i$
7. end if
```

In the proposed algorithm, initial group of members M is generated by creating specified w number of members. Each member's solution gets initialized by a random solution. The fitness of each member's solution is calculated as described in section 3.1.2. In each iteration, M gets sorted in descending order according to fitness value of members. Then, the member having the highest fitness is selected as the producer PR, worst 20% members become dispersed members D and rest of the members gets added to scroungers' list, SC. After that, producer PR's fitness is improved using *improveProducer*() function as shown in Algorithm 3.2.2. *improveProducer*() function tries to improve fitness of PR's solution by shifting courses to slots having higher preference values. After improving producer scroungers get processed. For each scrounger SH is emptied to be used for selective search. SH is used to hold best intermediate solution in selective search. Then, instructor-wise swap sequences to reach PR's solution SPR from scrounger *sc*'s solution $SSPR$ is calculated. Some swaps are selected for each instructor from $SSPR$ based on the selection probability α denoted by $SSSPR$. Redundant swaps are removed from $SSSPR$ using *swapMinimizer*() function, result of which are denoted by $SSSPRM$. After that, for each instructor a portion of $SSSPRM$ is selected using the equation $f * |SSSPRM_j|$, where f is the force percentage and $SSSPRM_j$ is the swap sequence corresponding to an instructor j. Swaps of this selected portion are forcefully applied to scrounger's solution and resulting conflicts are resolved using repairSolution function. Algorithm 3.2.3 shows the required steps of repairSolution function which works by randomly moving the conflicting classes to non-conflicting positions as described in section 3.1.4. Rest of the swaps are applied only if they do not create any conflicts. The selective search technique is used after applying each swap to ensure that best intermediate solution is retained. Algorithm 3.2.4 shows the required steps of selective search. It simply updates SH only if new solution is found better than SH. Finally, after the application of all the swaps the best intermediate solution SH becomes scrounger's solution. Operation for dispersed members are quite similar to operations on scroungers rather the fact that, dispersed members work with random swap sequence instead of swap sequence to producer. The algorithm uses a predefined number of iterations n as the termination criteria. After termination, producer PR's solution SPR is considered as the final solution.

3.3.7 Illustration of an iteration of GSOSS Algorithm

A schematic representation of an iteration of the proposed method in shown in Fig. 3.7. Suppose, a system consisting of three instructors e_1, e_2 and e_3, each having five weekly slots that is needed to be scheduled. In Fig. 3.7, M is the initial group of members. M is sorted in descending order according to fitness values of the members. Member having best fitness becomes the producer PR, worst 20% members become dispersed members D and rest of the members become scroungers, $SC(= G - P - D)$. In the first step, producer PR is improved using *improveProducer*() function. Then, for each scrounger sc, instructor-wise swap sequences to reach producer's solution SPR from scrounger sc's solution S_{SC} is calculated which is represented by $SSPR(= SPR - S_{sc})$. The circle (○) symbol inside the swap sequences represents a swap operator. Some swaps are selected for each instructor from $SSPR$ based on the selection probability α denoted by $SSSPR(= \alpha * SSPR)$. Redundant swaps are removed from $SSSPR$ using *swapMinimizer*() function, result of which are denoted by $SSSPRM$ (green coloured swaps numbered 1, 2, 3, 4, 5 and 6 in Fig. 3.7). Then, a portion of $SSSPRM$ is selected using the equation $RM\ [1{:}\ f * |\ SSSPRM\ |]$, where f is the force percentage. This selected portion of $SSSPRM$ is denoted by $SSSPRMFA$ (swaps numbered 1, 3 and 5 in Fig. 3.7) and the rest of the swaps are denoted by $SSSPRMTA$ (green coloured swaps numbered 2, 4 and 6 in Fig. 3.7). Swaps of $SSSPRMFA$ are forcefully applied to S_{sc} and then the swaps of $SSSPRMTA$ are applied to S_{sc} only if they do not create any

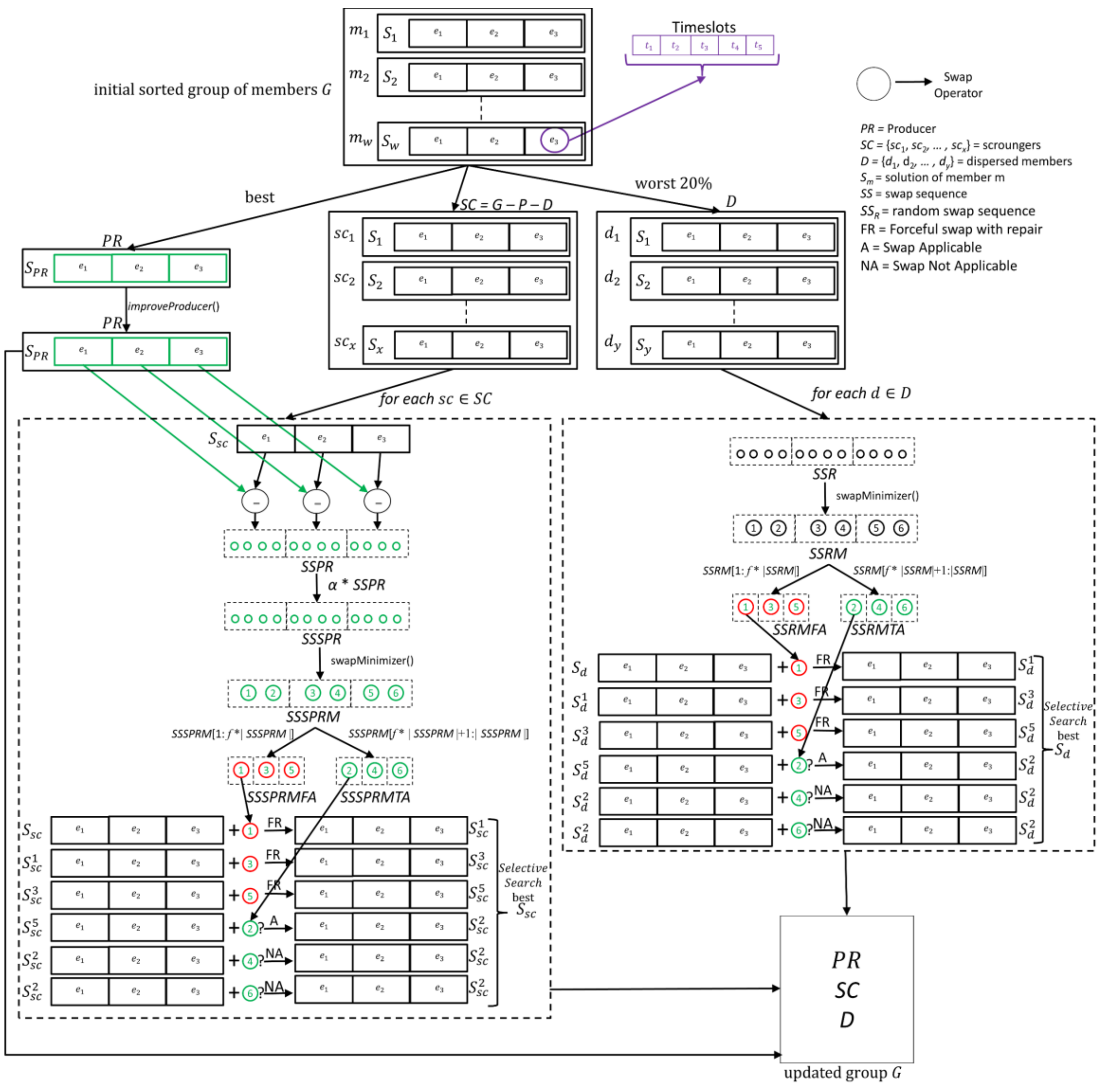


Figure 3.7: Illustration of an iteration of GSOSS.

conflicts. Any conflict resulting from forceful swap application is handled by repair mechanism as described in section 3.1.4. In the example shown in Fig. 3.7, swaps numbered 1, 3 and 5 are forcefully applied on initial solution S_{sc} making it S_{sc}^{5} . Then, swap numbered 2 gets applied on S_{sc}^{5} resulting in S_{sc}^{2} as there is no conflict. Solution stays at S_{sc}^{2} because swaps numbered 4 and 6 are not applied because of conflicts. The best one among these solutions is then picked as sc's solution. On the other hand, for each dispersed member $d \in D$ a random swap sequence is applied on its solution S_d. The process of applying random swap sequence to a dispersed member is same as in case of scrounger. Finally, M is updated using P, SC and D for the next iteration.

CHAPTER IV

Experimental Studies

This chapter investigates the effectiveness and performance of the proposed using Particle Swarm Optimization with Selective Search (PSOSS) and using Group Search Optimizer with Selective Search (GSOSS) methods for obtaining a workable timetable. The performance of the proposed methods has been compared with the performances of Genetic Algorithm (GA) and Harmony Search (HS) for the same problem set with comparable experimental and parameter settings. In short, the proposed PSOSS and GSOSS are transformations of PSO and GSO respectively for University Course Scheduling Problem (UCSP) plus selective search and forceful swap operation with repair mechanism. Therefore, the standard PSO and GSO are also brought into the comparison excluding the two additional operations incorporated into PSOSS and GSOSS to identify the significance of the operations. This chapter also provides an experimental analysis for a better understanding of the performance of the proposed method.

4.1 Experimental Setup

For solving UCSP with GA, single point crossover with a crossover rate of 0.70 is used. Repair mechanism ensures that valid solutions are generated after crossover. Mutation is performed by randomly changing the timeslot of a course for a randomly selected instructor with a mutation probability of 0.20. Elitism is also considered for implementation with an elite list of size 2.

In this study of UCSP, HS algorithm is implemented with a Harmony Memory Consideration Rate (HMCR) of 0.95 and a Pitch Adjustment Rate (PAR) of 0.1.

There is a common parameter alpha (α) in GSOSS and GSO because GSOSS is an extension of GSO. Value of α considered for implementation is 0.9. Also a force rate of 100% has been used for GSOSS.

There are several parameters common in PSOSS and PSO because PSOSS is an extension of PSO. The common parameters are alpha (α), beta (β) and gamma (γ); and the values considered for implementation are 0.3, 0.5 and 0.2, respectively. Also, a force rate of 100% has been used for PSOSS.

Boost C++ libraries [71] have been used for implementation of the methods. The methods have been implemented in Visual C++ of Microsoft's Visual Studio 2013 on Windows 10 platform on Intel$^{(R)}$ Core$^{(TM)}$ i7-7700 CPU @ 3.60 GHz processor, and 16 GB RAM.

4.2 Experimental Environment

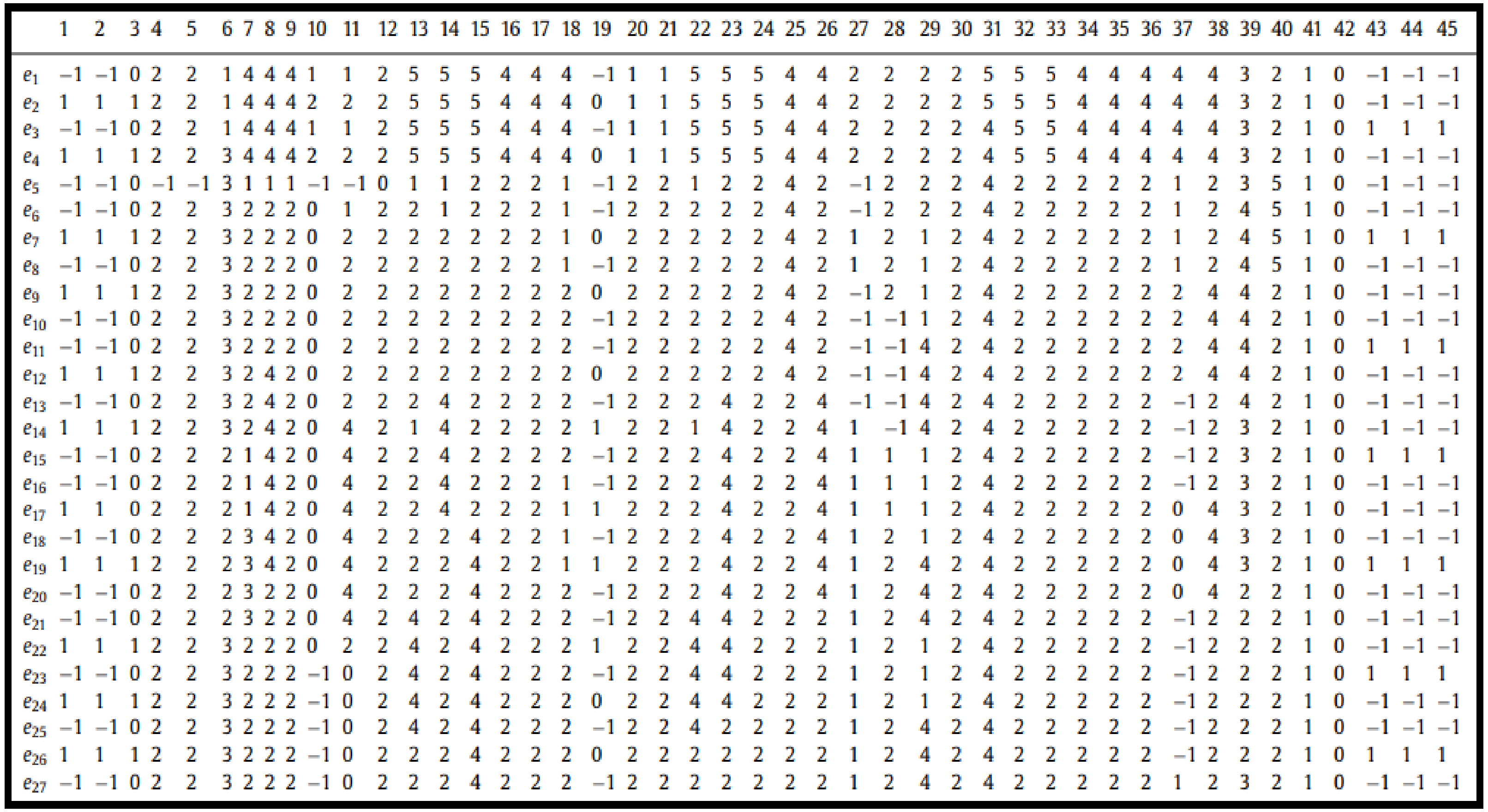
In the experimental environment, instructors' flexibility is considered. The weekly timeslots for instructors and their preferences are given in Fig 3.2 and Fig. 3.3, respectively. The preferences varied from -1 to 5, where -1 means the lowest preference and 5 means the highest preference. Experiments with input data resembling the course structure of the department of Computer Science and Engineering (CSE) of Khulna University of Engineering & Technology (KUET) have been conducted. In KUET, there are 5 days for teaching in a week and each teaching day is divided into 9 teaching timeslots of 50 minutes duration. The duration of each laboratory session of undergraduate level as well as a postgraduate class is three consecutive timeslots.

Considered hard constraints:

- A student can only go to a single class in a timeslot.
- An instructor cannot conduct multiple classes in a timeslot.
- Courses cannot be assigned to break periods.
- Courses requiring multiple slots such as laboratory courses cannot include break periods.
- Courses can be assigned to allowed rooms only.

Considered soft constraints:

- Maintain preference of instructor as much as possible.
- Keep the number of consecutive classes as few as possible for instructors.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
e_1	−1	−1	0	2	2	1	4	4	4	1	1	2	5	5	5	4	4	4	−1	1	1	5	5	5	4	4	2	2	2	2	5	5	5	4	4	4	4	4	3	2	1	0	−1	−1	−1
e_2	1	1	1	2	2	1	4	4	4	2	2	2	5	5	5	4	4	4	0	1	1	5	5	5	4	4	2	2	2	2	5	5	5	4	4	4	4	4	3	2	1	0	−1	−1	−1
e_3	−1	−1	0	2	2	1	4	4	4	1	1	2	5	5	5	4	4	4	−1	1	1	5	5	5	4	4	2	2	2	2	4	5	5	4	4	4	4	4	3	2	1	0	1	1	1
e_4	1	1	1	2	2	3	4	4	4	2	2	2	5	5	5	4	4	4	0	1	1	5	5	5	4	4	2	2	2	2	4	5	5	4	4	4	4	4	3	2	1	0	−1	−1	−1
e_5	−1	−1	0	−1	−1	3	1	1	1	−1	−1	0	1	1	2	2	2	1	−1	2	2	1	2	2	4	2	−1	2	2	2	4	2	2	2	2	2	1	2	3	5	1	0	−1	−1	−1
e_6	−1	−1	0	2	2	3	2	2	2	0	1	2	2	1	2	2	2	1	−1	2	2	2	2	2	4	2	−1	2	2	2	4	2	2	2	2	2	1	2	4	5	1	0	−1	−1	−1
e_7	1	1	1	2	2	3	2	2	2	0	2	2	2	2	2	2	2	1	0	2	2	2	2	2	4	2	1	2	1	2	4	2	2	2	2	2	1	2	4	5	1	0	1	1	1
e_8	−1	−1	0	2	2	3	2	2	2	0	2	2	2	2	2	2	2	1	−1	2	2	2	2	2	4	2	1	2	1	2	4	2	2	2	2	2	1	2	4	5	1	0	−1	−1	−1
e_9	1	1	1	2	2	3	2	2	2	0	2	2	2	2	2	2	2	2	0	2	2	2	2	2	4	2	−1	2	1	2	4	2	2	2	2	2	2	4	4	2	1	0	−1	−1	−1
e_{10}	−1	−1	0	2	2	3	2	2	2	0	2	2	2	2	2	2	2	2	−1	2	2	2	2	2	4	2	−1	−1	1	2	4	2	2	2	2	2	2	4	4	2	1	0	−1	−1	−1
e_{11}	−1	−1	0	2	2	3	2	2	2	0	2	2	2	2	2	2	2	2	−1	2	2	2	2	2	4	2	−1	−1	4	2	4	2	2	2	2	2	2	4	4	2	1	0	1	1	1
e_{12}	1	1	1	2	2	3	2	4	2	0	2	2	2	2	2	2	2	2	0	2	2	2	2	2	4	2	−1	−1	4	2	4	2	2	2	2	2	2	4	4	2	1	0	−1	−1	−1
e_{13}	−1	−1	0	2	2	3	2	4	2	0	2	2	2	4	2	2	2	2	−1	2	2	2	4	2	2	4	−1	−1	4	2	4	2	2	2	2	2	−1	2	4	2	1	0	−1	−1	−1
e_{14}	1	1	1	2	2	3	2	4	2	0	4	2	1	4	2	2	2	2	1	2	2	1	4	2	2	4	1	−1	4	2	4	2	2	2	2	2	−1	2	3	2	1	0	−1	−1	−1
e_{15}	−1	−1	0	2	2	2	1	4	2	0	4	2	2	4	2	2	2	2	−1	2	2	2	4	2	2	4	1	1	1	2	4	2	2	2	2	2	−1	2	3	2	1	0	1	1	1
e_{16}	−1	−1	0	2	2	2	1	4	2	0	4	2	2	4	2	2	2	1	−1	2	2	2	4	2	2	4	1	1	1	2	4	2	2	2	2	2	−1	2	3	2	1	0	−1	−1	−1
e_{17}	1	1	0	2	2	2	1	4	2	0	4	2	2	4	2	2	2	1	1	2	2	2	4	2	2	4	1	1	1	2	4	2	2	2	2	2	0	4	3	2	1	0	−1	−1	−1
e_{18}	−1	−1	0	2	2	2	3	4	2	0	4	2	2	2	4	2	2	1	−1	2	2	2	4	2	2	4	1	2	1	2	4	2	2	2	2	2	0	4	3	2	1	0	−1	−1	−1
e_{19}	1	1	1	2	2	2	3	4	2	0	4	2	2	2	4	2	2	1	1	2	2	2	4	2	2	4	1	2	4	2	4	2	2	2	2	2	0	4	3	2	1	0	1	1	1
e_{20}	−1	−1	0	2	2	2	3	2	2	0	4	2	2	2	4	2	2	2	−1	2	2	2	4	2	2	4	1	2	4	2	4	2	2	2	2	2	0	4	2	2	1	0	−1	−1	−1
e_{21}	−1	−1	0	2	2	2	3	2	2	0	4	2	4	2	4	2	2	2	−1	2	2	4	4	2	2	2	1	2	4	2	4	2	2	2	2	2	−1	2	2	2	1	0	−1	−1	−1
e_{22}	1	1	1	2	2	3	2	2	2	0	2	2	4	2	4	2	2	2	1	2	2	4	4	2	2	2	1	2	1	2	4	2	2	2	2	2	−1	2	2	2	1	0	−1	−1	−1
e_{23}	−1	−1	0	2	2	3	2	2	2	−1	0	2	4	2	4	2	2	2	−1	2	2	4	4	2	2	2	1	2	1	2	4	2	2	2	2	2	−1	2	2	2	1	0	1	1	1
e_{24}	1	1	1	2	2	3	2	2	2	−1	0	2	4	2	4	2	2	2	0	2	2	4	4	2	2	2	1	2	1	2	4	2	2	2	2	2	−1	2	2	2	1	0	−1	−1	−1
e_{25}	−1	−1	0	2	2	3	2	2	2	−1	0	2	4	2	4	2	2	2	−1	2	2	4	2	2	2	2	1	2	4	2	4	2	2	2	2	2	−1	2	2	2	1	0	−1	−1	−1
e_{26}	1	1	1	2	2	3	2	2	2	−1	0	2	2	2	4	2	2	2	0	2	2	2	2	2	2	2	1	2	4	2	4	2	2	2	2	2	−1	2	2	2	1	0	1	1	1
e_{27}	−1	−1	0	2	2	3	2	2	2	−1	0	2	2	2	4	2	2	2	−1	2	2	2	2	2	2	2	1	2	4	2	4	2	2	2	2	2	1	2	3	2	1	0	−1	−1	−1

Figure 4.1: Input preference values for instructors.

4.3 Input Data Preparation

Fig. 4.1 lists the used preference values of all the instructors. There are five batches of students in the considered dataset: four batches in the undergraduate level and one batch at the postgraduate level. Four batches of undergraduate level are represented by b_1, b_2, b_3 and b_4, respectively. Postgraduate batch is represented by b_5. Each of the batches in

Table 4.1: Batch and course information

Batch Name	**Course Code**	**Classes / Week**	**Timeslots / Class**	**Type of Course**	**Number of Students**
b_1/1st Year Undergraduate	CSE 1201	3	1	Theory	60
	CSE 1202	2	3	Laboratory	30
	CSE 1203	3	1	Theory	60
	CSE 1204	2	3	Laboratory	30
	EEE 1217	3	1	Theory	60
	EEE 1218	1	3	Laboratory	30
	CHEM 1207	3	1	Theory	60
	CHEM 1208	1	3	Laboratory	30
	MATH 1207	3	1	Theory	60
	ME 1270	1	3	Laboratory	30
b_2/2nd Year Undergraduate	CSE 2200	2	3	Laboratory	30
	CSE 2201	3	1	Theory	60
	CSE 2202	2	3	Laboratory	30
	CSE 2207	3	1	Theory	60
	CSE 2208	1	3	Laboratory	30
	CSE 2213	3	1	Theory	60
	EEE 2217	3	1	Theory	60
	EEE 2218	2	3	Laboratory	30
	MATH 2207	3	1	Theory	60
b_3/3rd Year Undergraduate	CSE 3200	2	3	Laboratory	30
	CSE 3201	3	1	Theory	60
	CSE 3202	2	3	Laboratory	30
	CSE 3203	3	1	Theory	60
	CSE 3204	1	3	Laboratory	30
	CSE 3207	3	1	Theory	60
	CSE 3211	3	1	Theory	60
	CSE 3212	1	3	Laboratory	30
	ECE 3215	3	1	Theory	30
b_4/4th Year Undergraduate	CSE 4207	3	1	Theory	60
	CSE 4208	1	3	Laboratory	30
	CSE 4211	3	1	Theory	60
	CSE 4212	1	3	Laboratory	30
	CSE 4239	3	1	Theory	60
	IEM 4227	3	1	Theory	60
	HUM 4207	3	1	Theory	60
b_5/ Postgraduate	CSE 6225	1	3	Theory	10
	CSE 6465	1	3	Theory	10
	CSE 6471	1	3	Theory	10

Table 4.2: Course information for each instructor

Instructor ID	Number of Courses	Course Code	Weekly Course Load (Timeslots/Week)
e_1	4	CSE 1203, CSE 1204, CSE 2201, CSE 2202	18
e_2	4	CSE 3211, CSE 3212, CSE 4239, CSE 6225	12
e_3	2	CSE 3201, CSE 3202	9
e_4	1	CSE 3200	6
e_5	3	CSE 4211, CSE 4212, CSE 6471	9
e_6	2	CSE 4207, CSE 6465	6
e_7	1	CSE 2207	3
e_8	1	CSE 1201	3
e_9	1	CSE 2208	3
e_{10}	2	CSE 2200, CSE3207	9
e_{11}	1	CSE 3203	3
e_{12}	1	CSE 1202	6
e_{13}	1	CSE 4208	3
e_{14}	1	CSE 2213	3
e_{15}	1	CSE 3204	6
e_{16}	1	EEE 1217	3
e_{17}	1	EEE 1218	3
e_{18}	1	EEE 2217	3
e_{19}	1	EEE 2218	6
e_{20}	1	MATH 1207	3
e_{21}	1	MATH 2207	3
e_{22}	1	ECE 3215	3
e_{23}	1	ME 1270	3
e_{24}	1	IEM 4227	3
e_{25}	1	CHEM 1207	3
e_{26}	1	CHEM 1208	3
e_{27}	1	HUM 4207	3

Table 4.3: Information for classrooms and laboratories

Room ID	Room Type	Room capacity	Allowable Courses
r_1	Lecture	60	Any theory course
r_2	Lecture	60	
r_3	Lecture	60	
r_4	Lecture	60	
r_5	Lecture	60	
r_6	Laboratory	30	CSE 2200, CSE 3202, CSE 4208, CSE 4212
r_7	Laboratory	30	CSE2200, CSE4212, CSE3212, CSE2208, CSE3202
r_8	Laboratory	30	CSE 1202, CSE 2202
r_9	Laboratory	30	CSE1204
r_{10}	Laboratory	30	CSE3204
r_{11}	Laboratory	30	ME1270
r_{12}	Laboratory	30	EEE1218, EEE2218
r_{13}	Laboratory	30	CHEM1208

undergraduate level is divided into two subgroups namely v_1 and v_2. In total, 38 courses are

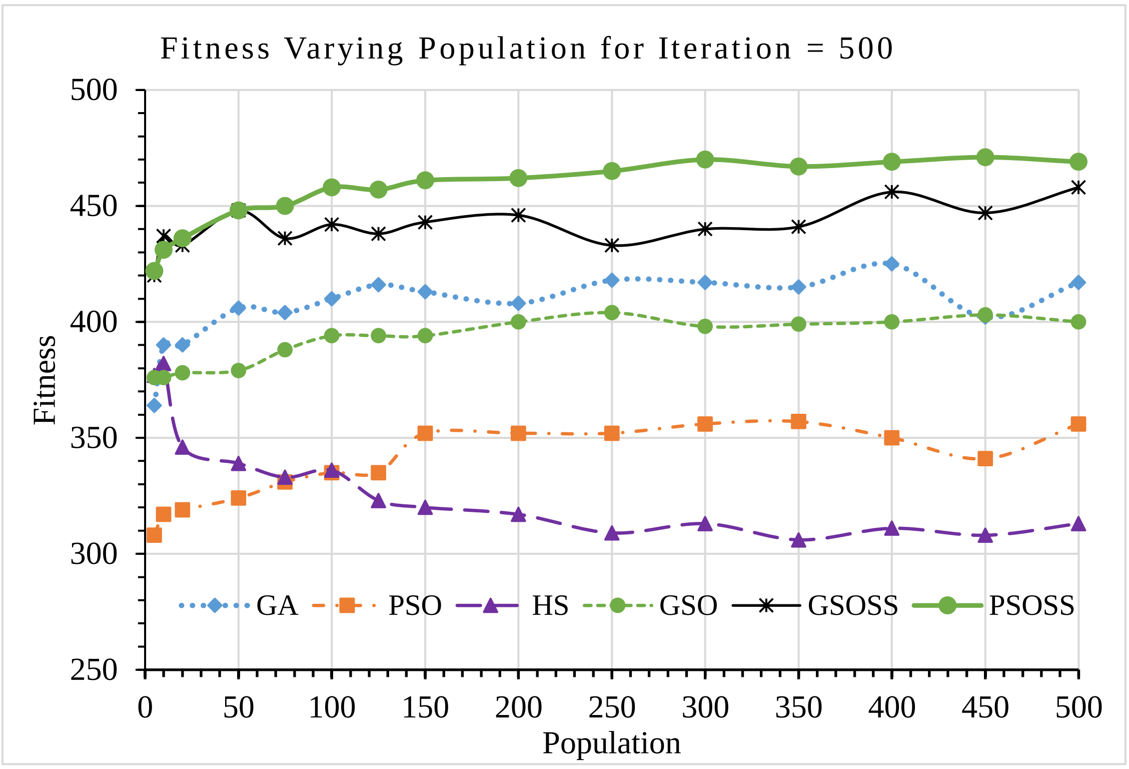


Figure 4.2: Performance analysis of fitness for different population sizes.

taught by 27 instructors. Odd-numbered courses represent theory courses and even-numbered courses represent laboratory courses. Table 4.1 shows which courses belong to which batch, number of weekly classes required for a course, time duration of a class, course type and the number of registered students of a course. Table 4.2 shows the number of courses assigned to an instructor, the courses allocated to each instructor and the weekly course load of each instructor. Table 4.3 shows the room id, room type, maximum seating capacity and allowable courses that can be taught in that room. There are two types of rooms: lecture room and laboratory room. As the laboratory rooms support a maximum of 30 students, a batch of 60 students needs to be divided into two subgroups of 30 students.

4.4 Experimental Results and Analysis

Proposed PSOSS and GSOSS are population-based methods and; therefore, population size is one of the important parameters similar to other population-based algorithms including GA and HS to which the performance of the proposed methods will be compared to. The population size is a parameter of meta-heuristic algorithms having an impact on the computational cost of the algorithm that increases with growing population size. On the other hand, convergence speed is a performance measure for metaheuristic algorithms where the

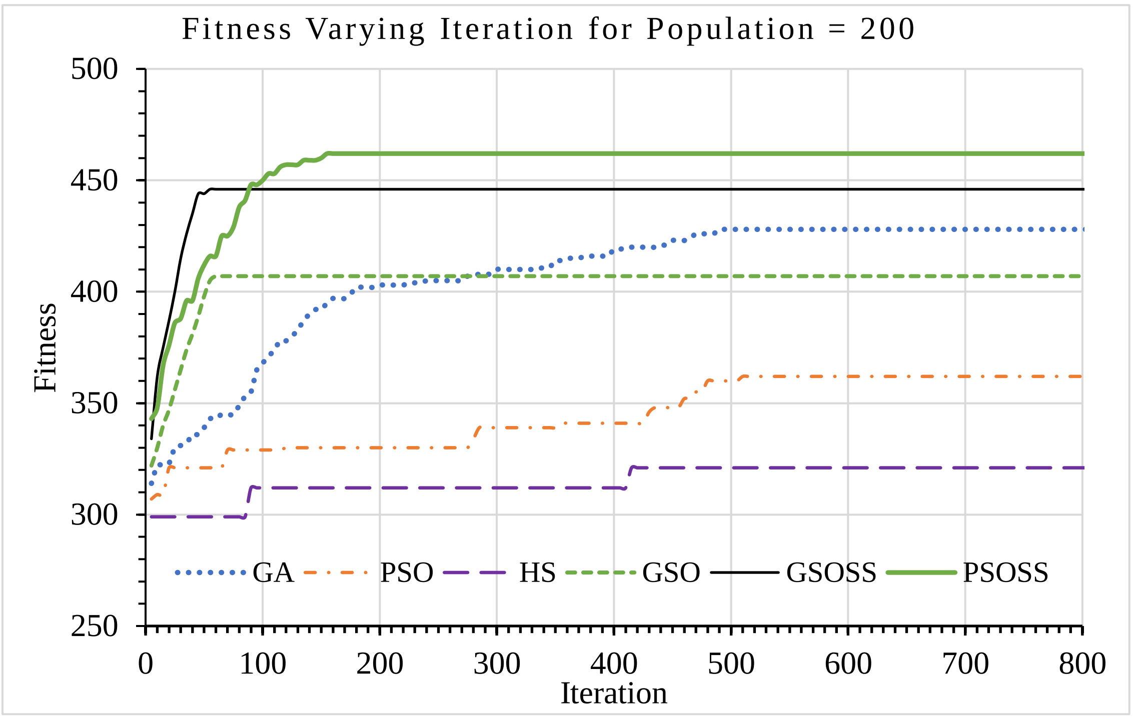


Figure 4.3: Performance analysis of fitness in different iterations.

maximum number of iterations required to reach an optimal solution is an important parameter. The first set of performance measures for the methods are the population size and the number of iterations. The next set of performance measures for the methods are the fitness values of solutions, instructors' satisfaction followed by the sample timetables generated for an instructor. The two sets of performance measures will be carried out in this section such that the methods can be compared to each other.

Fitness values of GA, PSO, HS, GSO, GSOSS and PSOSS are measured by varying the population size from 5 to 500 and the best fitness values are plotted over the population sizes in Fig. 4.2. It is observed from the figure that fitness of all the methods improves more or less with population size except for HS algorithm. The performance of HS algorithm decreases continuously after population size of 20. If the HMS (i.e., population) in HS algorithm is high, the probability of selecting solution for different instructors from different entities of HM also increases while improvising a new harmony (complete solution) resulting in a conflict which causes the HS algorithm to fail to create a new harmony. This might be the reason for choosing small HMS in existing research on HS algorithm [54], [55]. However, the performances of the proposed GSOSS and PSOSS are better than the HS algorithm with any population size. The HS algorithm has shown the best fitness value 382 for population size of 10. On the other hand, the fitness of PSOSS with a population size of

Table 4.4: Average and best fitness comparison among GA, PSO, HS, GSO, GSOSS and PSOSS of 25 trials for different population sizes

Method	Population	Average Fitness	Std. Deviation of Fitness	Best Fitness
GA	200	421.08	6.26	**428**
	300	416.76	7.82	427
	400	420.08	7.52	428
PSO	200	343.64	12.97	**369**
	300	346.17	10.59	356
	400	345.05	7.96	357
HS	5	371.52	9.11	**394**
	10	365.80	9.29	382
	20	347.64	9.36	364
GSO	200	394.00	8.02	**414**
	300	391.45	5.34	398
	400	394.69	6.66	404
GSOSS	200	448.84	6.11	446
	300	449.73	5.67	456
	400	456.67	5.32	**458**
PSOSS	200	460.44	5.12	471
	300	460.83	4.76	470
	400	462.87	5.41	**471**

10 was 431, which is 437 in case of GSOSS. The best achieved fitness by PSOSS was 471 with a population size of 450 and the best achieved fitness by GSOSS was 458 with a population size of 500. On the other hand, the best fitness values achieved for GA, PSO and GSO are 425 (for population size of 400), 357 (for population size of 350) and 404 (for population size of 250), respectively. At a glance, PSOSS is much better than the rest of the methods and GSOSS is also found superior to other compared methods.

Fig 4.3 compares the achieved fitness of the methods from iteration 5 to 800 for fixed population size of 200 on a sample run. The figure reflects the convergence nature of the methods with iteration. It is remarkable from the figure that the fitness value of any method is very poor at the beginning (e.g., 5, 10) and improves over iterations. As an example, for iteration 10, the achieved fitness values are 322, 309, 299, 330, 362 and 348 for GA, PSO, HS, GSO, GSOSS and PSOSS, respectively. On the other hand, the methods achieved the fitness values 368, 329, 312, 407, 446, 450 at iteration 100. From the figure it is observed that after 500 iterations only PSO is shown to have improved the performance and it achieved the best fitness value 362 at iteration 510 and onwards. On the other hand, the best fitness values achieved by GA, HS, GSO, GSOSS and PSOSS are 428 (at iteration 490 and onwards), 321 (at iteration 415 and onwards), 407 (at iteration 60 and onwards), 446 (at iteration 55 and onwards) and 462 (at iteration 155 and onwards), respectively. The figure revealed the faster convergence of the proposed GSOSS and PSOSS as well as the outperformance in comparison with other methods in terms of iteration variations. GSOSS achieves fast convergence due to its producer scanning feature.

Fig. 4.2 and Fig 4.3 also clearly reveal the significance of selective search and application of forceful swap operation in the proposed GSOSS and PSOSS methods. The performance variation in terms of fitness between PSO and PSOSS, GSO and GSOSS as seen in Fig. 4.2 and Fig 4.3 reflects the effects of these operations. The best achieved fitness value of PSO was 357 in Fig. 4.2 for population size of 350 in terms of population variation. At the same population size of 350, PSOSS achieved fitness value of 467 which is much better than that of PSO. Outperformance of PSOSS over PSO is also clearly visible in Fig 4.3 in terms of iteration variation. The best achieved fitness value of GSO was 404 in Fig. 9 for population size of 250 in terms of population variation. At the same population size of 350, GSOSS achieved fitness value of 433 which is much better than that of GSO. Outperformance of GSOSS over GSO is also clearly visible in Fig 4.3 in terms of iteration variation. Due to the use of the forceful swap operation in PSOSS, all the particles move a little towards the global best solutions and selective search ensures that the best intermediate solution is retained while applying an SS. These are the reasons why PSOSS performs much better than PSO and the rest of the investigated methods.

Table 4.4 shows the average fitness with standard deviation and the best fitness comparison among all the investigated methods for different population and iteration sizes. Fig. 4.2 shows that HS algorithm works well only for small population sizes and the fitness obtained for all other methods have proved to be better for population size larger than 100. Therefore, in the experiments, population sizes used for GA, PSO, GSO, GSOSS and PSOSS are 200, 300 and 400 whereas population sizes used for HS algorithm are 5,10 and 20. According to Fig 4.3, GA, HS, GSO, GSOSS and PSOSS do not show any significant improvement of fitness after 500 iterations but PSO keeps showing better results up to 510 iterations. Due to this fact, the number of iterations used for all the experiments is 600 for all methods. Each method was run for 25 times for particular population size and the presented results in the table are the outcome are the outcome of 25 trials for each setting. The best results for each method are highlighted in boldface type in the table. From the table, it is observed that best fitness acheived by GA, PSO, HS, GSO and GSOSS are 428, 369, 394, 414, and 458 respectively. Whereas, the proposed PSOSS is shown to achieve 471 which is much better

Table 4.5: Instructors' satisfaction values achieved by implemented methods

Instructor	**Course Load**	**Achieved Satisfaction in %**					
		GA	**PSO**	**HS**	**GSO**	**GSOSS**	**PSOSS**
e_1	18	74.07	75.31	85.19	91.36	**97.53**	96.30
e_2	12	91.23	82.46	78.95	90.98	**92.98**	**92.98**
e_3	9	65.91	77.27	70.45	81.82	86.36	**88.64**
e_4	6	90.00	43.33	**100**	90.00	96.67	90.00
e_5	9	88.89	**100**	70.37	66.67	88.89	88.89
e_6	6	**77.27**	54.55	68.18	54.55	**77.27**	**77.27**
e_7	3	69.23	30.77	46.15	**100**	69.23	69.23
e_8	3	84.62	53.85	46.15	53.85	69.23	**92.31**
e_9	3	50.00	33.33	50.00	41.67	66.67	**83.33**
e_{10}	9	**88.89**	55.56	74.07	59.26	85.19	85.19
e_{11}	3	66.67	58.33	58.33	58.33	91.67	**100**
e_{12}	6	**66.67**	37.50	**66.67**	54.17	58.33	58.33
e_{13}	3	**66.67**	**66.67**	50.00	**66.67**	**66.67**	**66.67**
e_{14}	3	75.00	50.00	66.67	91.67	**100**	**100**
e_{15}	6	62.50	58.33	58.33	29.17	58.33	**66.67**
e_{16}	3	83.33	66.67	66.67	**100**	91.67	75.00
e_{17}	3	**66.67**	50.00	50.00	50.00	58.33	**66.67**
e_{18}	3	75.00	66.67	66.67	**83.33**	50	66.67
e_{19}	6	62.50	**70.83**	54.17	66.67	58.33	62.50
e_{20}	3	83.33	66.67	66.67	91.67	**100**	**100**
e_{21}	3	**83.33**	66.67	50.00	66.67	**83.33**	50.00
e_{22}	3	41.67	**66.67**	33.33	50.00	50	50.00
e_{23}	3	**83.33**	58.33	**83.33**	25.00	58.33	**83.33**
e_{24}	3	83.33	**100**	83.33	83.33	**100**	**100**
e_{25}	3	**91.67**	50.00	83.33	66.67	66.67	83.33
e_{26}	3	66.67	**83.33**	50.00	50.00	66.67	**83.33**
e_{27}	3	58.33	25.00	58.33	83.33	75	**91.67**
Avg. Satisfaction		**75.62**	**65.19**	**69.61**	**73.14**	**80.92**	**83.92**

than the rest of the methods. The best solutions produced by individual methods are analyzed below.

A new measure of satisfaction of individual instructors is considered in this study for better realization of the quality of the solutions (i.e. the produced course schedules) by different methods. One of the objectives of optimizing UCSP is to satisfy the instructors' preferences as much as possible. Satisfaction of an instructor can be expressed in percentage and 100% satisfaction means courses of the instructor are assigned to timeslots heaving the highest preference values (here 5). Satisfaction of ith particle's/member's jth instructor in a Solution $S_{i,j}$, $Satisfaction(S_{i,j})$ is computed according to the formula:

$$Satisfaction(S_{i,j}) = \frac{F(S_{i,j})}{M(e_j)} * 100 \tag{4.1}$$

where, $M\left(e_j\right)$ is the maximum possible fitness and $F(S_{i,j})$ is the achieved fitness (calculated using Eq. (3.2) of jth instructor's solution. Table 4.5 shows the achieved satisfaction value (in %) for all the instructors in the best solutions produced by the methods whose fitness values are marked in Table 4.4 as the best fitness. The table also includes weekly course load for each instructor and instructor wise best achieved satisfaction values for individual methods are marked in bold face type for a better understanding. Among 27 instructors, PSOSS achieved the best satisfaction values for fifteen instructors. On the other hand, GA, PSO, HS, GSO, and GSOSS achieved the best satisfaction values for seven, six, three, four and eight cases, respectively. Therefore, the average satisfaction value achieved by PSOSS is much higher than the rest of the methods. The average satisfaction of the best solution produced by PSOSS is 83.22. On the other hand, the achieved satisfactions for GA, PSO, HS, GSO, and GSOSS are 75.62, 65.19, 69.61, 73.14, and 80.92, respectively. This is a significant performance indicator of the proposed PSOSS method. Result obtained from GSOSS is also satisfactory compared to other implemented methods.

Table 4.6 (a)-(f) shows schedules for instructor e_1 from the best solutions (marked in Table 4.5) generated by GA, PSO, HS, GSO, GSOSS, and PSOSS. From the schedule generated by PSOSS shown in Table 4.6 (e), it can be seen that e_1 has a Laboratory class CSE 1204 in timeslots 7, 8 and 9 on Sunday which is desirable because e_1 has maximum preference of 4 in these periods for Sunday as stated in Fig 4.1. Similarly, timetable for other days also adheres to instructor e_1's preference in most of the cases. GSOSS also produces good schedule considering instructor e_1's preference. In comparison with schedules produced by other methods (i.e., GA, PSO, HS and GSO), the schedules generated by PSOSS and GSOSS for the instructor e_1 are more satisfactory in terms of e_1's preference values. This observation is similar for most of the instructors in case of PSOSS and thus PSOSS is found to be an effective method for solving USCP.

Table 4.6: Sample Timetable for Instructor e_1 generated by GA, PSO, HS, GSO, GSOSS and PSOSS methods

(a) GA

Day	Timeslot									
	1	2	3	4	5	6		7	8	9
Sun					CSE2201 $(r_5\|b_2\|e_1)$		LUNCH BREAK		CSE2201 $(r_3\|b_2\|e_1)$	
Mon			CSE1203 $(r_2\|b_1\|e_1)$			CSE1203 $(r_5\|b_1\|e_1)$		CSE1204 $(r_9\|b_1\|v_1\|e_1)$		
Tue	CSE2202 $(r_8\|b_2\|v_2\|e_1)$					CSE2201 $(r_1\|b_2\|e_1)$				
Wed	CSE1203 $(r_5\|b_1\|e_1)$			CSE2202 $(r_8\|b_2\|v_1\|e_1)$				CSE1204 $(r_9\|b_1\|v_2\|e_1)$		
Thu										

(b) PSO

Day	Timeslot									
	1	2	3	4	5	6		7	8	9
Sun							LUNCH BREAK	CSE2202 $(r_8 \mid b_2 \mid v_2 \mid e_1)$		
Mon				CSE1204 $(r_9 \mid b_1 \mid v_1 \mid e_1)$						
Tue				CSE1203 $(r_3 \mid b_1 \mid e_1)$		CSE1203 $(r_1 \mid b_1 \mid e_1)$		CSE2201 $(r_3 \mid b_2 \mid e_1)$		
Wed				CSE2202 $(r_8 \mid b_2 \mid v_1 \mid e_1)$					CSE2201 $(r_2 \mid b_2 \mid e_1)$	
Thu	CSE2201 $(r_3 \mid b_2 \mid e_1)$					CSE1203 $(r_5 \mid b_1 \mid e_1)$		CSE1204 $(r_9 \mid b_1 \mid v_2 \mid e_1)$		

(c) HS

Day	Timeslot									
	1	2	3	4	5	6		7	8	9
Sun						CSE1203 $(r_4 \mid b_1 \mid e_1)$	LUNCH BREAK	CSE2201 $(r_5 \mid b_2 \mid e_1)$		CSE1203 $(r_5 \mid b_1 \mid e_1)$
Mon				CSE2202 $(r_8 \mid b_2 \mid v_1 \mid e_1)$				CSE1204 $(r_9 \mid b_1 \mid v_1 \mid e_1)$		
Tue								CSE1203 $(r_4 \mid b_1 \mid e_1)$		
Wed		CSE1204 $(r_9 \mid b_1 \mid v_2 \mid e_1)$				CSE2201 $(r_3 \mid b_2 \mid e_1)$		CSE2201 $(r_3 \mid b_2 \mid e_1)$		
Thu	CSE2202 $(r_8 \mid b_2 \mid v_2 \mid e_1)$									

(d) GSO

Day	Timeslot									
	1	2	3	4	5	6		7	8	9
Sun							LUNCH BREAK	CSE2202 $(r_8 \mid b_2 \mid v_2 \mid e_1)$		
Mon			CSE1203 $(r_4 \mid b_1 \mid e_1)$		CSE2201 $(r_4 \mid b_2 \mid e_1)$			CSE2202 $(r_8 \mid b_2 \mid v_1 \mid e_1)$		
Tue				CSE1204 $(r_9 \mid b_1 \mid v_2 \mid e_1)$				CSE1203 $(r_5 \mid b_1 \mid e_1)$		
Wed			CSE1204 $(r_9 \mid b_1 \mid v_1 \mid e_1)$					CSE2201 $(r_5 \mid b_2 \mid e_1)$		CSE2201 $(r_4 \mid b_2 \mid e_1)$
Thu	CSE1203 $(r_1 \mid b_1 \mid e_1)$									

(e) GSOSS

Day	Timeslot									
	1	2	3	4	5	6		7	8	9
Sun							LUNCH BREAK	CSE2201 $(r_1 \mid b_2 \mid e_1)$		CSE1203 $(r_3 \mid b_1 \mid e_1)$
Mon				CSE1203 $(r_1 \mid b_1 \mid e_1)$		CSE2201 $(r_5 \mid b_2 \mid e_1)$		CSE1204 $(r_9 \mid b_1 \mid v_2 \mid e_1)$		
Tue				CSE2202 $(r_8 \mid b_2 \mid v_1 \mid e_1)$				CSE2201 $(r_3 \mid b_2 \mid e_1)$		
Wed				CSE1204 $(r_9 \mid b_1 \mid v_1 \mid e_1)$				CSE1203 $(r_2 \mid b_1 \mid e_1)$		
Thu	CSE2202 $(r_8 \mid b_2 \mid v_2 \mid e_1)$									

(f) PSOSS

Day	Timeslot									
	1	2	3	4	5	6		7	8	9
Sun							LUNCH BREAK	CSE1204 $(r_9\|b_1\|v_1\|e_1)$		
Mon				CSE2202 $(r_8\|b_2\|v_2\|e_1)$					CSE1203 $(r_3\|b_1\|e_1)$	
Tue				CSE2201 $(r_1\|b_2\|e_1)$		CSE2201 $(r_1\|b_2\|e_1)$		CSE2201 $(r_2\|b_2\|e_1)$		CSE1203 $(r_3\|b_1\|e_1)$
Wed				CSE2202 $(r_8\|b_2\|v_1\|e_1)$				CSE1204 $(r_9\|b_1\|v_2\|e_1)$		
Thu	CSE1203 $(r_5\|b_1\|e_1)$									

CHAPTER V

Conclusions

University Course Scheduling Problem (UCSP) is one of the toughest timetabling problems and solving UCSP has been an active research area for several decades. In this thesis two novel Particle Swarm Optimization (PSO) and Group Search Optimizer (GSO) based methods namely PSO with Selective Search (PSOSS) and GSO with Selective Search (GSOSS) have been proposed for solving UCSP. This chapter will draw a short summary of the key points of this thesis and possible future research directions based on the outcome of the present work.

5.1 Findings

This work proposes two innovative methods called PSOSS and GSOSS incorporating swap sequence based velocity/movement calculation, selective search and forceful swap application with repair mechanism to solve UCSP.

The proposed methods differ from existing methods, including many variants of PSO-based approaches, where UCSP is transformed into an equivalent floating point numeric domain. The proposed PSOSS approach uses a swap sequence based discrete PSO with a number of modifications. The velocity swap sequence is managed in two different parts: sequence for global best; and sequence combining personal best and previous velocity. A portion of swap sequence to global best is considered to be applied forcefully with repair mechanism to change other dependent schedules. After applying SOs one by one, the best intermediate solution is considered as the final solution based on selective search.

Proposed GSOSS method utilizes swap sequence for movement of scroungers and dispersed members and also uses forceful swap application with repair mechanism for updating both the scroungers and dispersed members.

The results obtained by our proposed PSOSS and GSOSS methods show significant improvement in solving UCSP compared to other traditional methods. PSOSS outperformed other implemented traditional methods in terms of quality of solutions.

5.2 Future Research Directions

- Proposed PSOSS and GSOSS methods can be utilized to generate schedule for whole university instead of a single department.
- Selective search and forceful swap application with repair mechanism techniques introduced in this thesis can be incorporated with other algorithms for solving different optimization problems.
- It would be an interest research topic to see how PSOSS and GSOSS perform considering co-teaching constraint.
- More sophisticated method for improving producer's solution in case of GSOSS and creating an initial random solution considering instructors' preference may produce better results.

PUBLICATION FROM THE THESIS

Sk. Imran Hossain, M. A. H. Akhand, M. I. R. Shuvo, N. Siddique, and H. Adeli, "Optimization of University Course Scheduling Problem using Particle Swarm Optimization with Selective Search," *Expert Systems with Applications*, vol. 127, pp. 9–24, Aug. 2019. https://doi.org/10.1016/j.eswa.2019.02.026 (IF:3.768).

REFERENCES

[1] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria, “An effective hybrid algorithm for university course timetabling,” *J. Sched.*, vol. 9, no. 5, pp. 403–432, Oct. 2006.

[2] R. Mencía, M. R. Sierra, C. Mencía, and R. Varela, “Genetic algorithms for the scheduling problem with arbitrary precedence relations and skilled operators,” *Integr. Comput. Aided. Eng.*, vol. 23, no. 3, pp. 269–285, Jun. 2016.

[3] Y. Tang, R. Liu, F. Wang, Q. Sun, and A. A. Kandil, “Scheduling Optimization of Linear Schedule with Constraint Programming,” *Comput. Civ. Infrastruct. Eng.*, vol. 33, no. 2, pp. 124–151, Feb. 2018.

[4] M. A. Al-Betar and A. T. Khader, “A hybrid harmony search for university course timetabling,” in *Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2009)*, 2009.

[5] M. R. Garey and D. S. Johnson, “A Guide to the Theory of NP-Completeness,” *A Series of Books in the Mathematical Sciences*. 1979.

[6] P. Pongcharoen, W. Promtet, P. Yenradee, and C. Hicks, “Stochastic Optimisation Timetabling Tool for university course scheduling,” *Int. J. Prod. Econ.*, vol. 112, no. 2, pp. 903–918, Apr. 2008.

[7] Y. Yue, J. Han, S. Wang, and X. Liu, “Integrated Train Timetabling and Rolling Stock Scheduling Model Based on Time-Dependent Demand for Urban Rail Transit,” *Comput. Civ. Infrastruct. Eng.*, vol. 32, no. 10, pp. 856–873, Oct. 2017.

[8] A. . Mushi, “Tabu search heuristic for university course timetabling problem,” *African J. Sci. Technol.*, vol. 7, no. 1, pp. 34–40, Jun. 2010.

[9] M.-R. Feizi-Derakhshi, H. Babei, and J. Heidarzadeh, “A survey of approaches for university course timetabling problem,” in *Proceedings of 8th International Symposium on Intelligent and Manufacturing Systems (IMS)*, 2012, pp. 307–321.

[10] Z. Naji Azimi, “Hybrid heuristics for Examination Timetabling problem,” *Appl. Math. Comput.*, vol. 163, no. 2, pp. 705–733, Apr. 2005.

[11] D.-F. Shiau, “A hybrid particle swarm optimization for a university course scheduling problem with flexible preferences,” *Expert Syst. Appl.*, vol. 38, no. 1, pp. 235–248, Jan. 2011.

[12] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan

Kaufmann Publishers Inc., 2001.

[13] J. Kennedy and R. C. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, 1995, vol. 4, pp. 1942–1948.

[14] A. Iglesias, A. Gálvez, and M. Collantes, “Multilayer embedded bat algorithm for B-spline curve reconstruction,” *Integr. Comput. Aided. Eng.*, vol. 24, no. 4, pp. 385–399, Sep. 2017.

[15] F. A. B. S. Ferreira, H. A. S. Leitão, W. T. A. Lopes, and F. Madeiro, “Hybrid firefly-Linde-Buzo-Gray algorithm for Channel-Optimized Vector Quantization codebook design,” *Integr. Comput. Aided. Eng.*, vol. 24, no. 3, pp. 297–314, Jun. 2017.

[16] R.-M. Chen and H.-F. Shih, “Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search,” *Algorithms*, vol. 6, no. 2, pp. 227–244, Apr. 2013.

[17] E. Montero and L. Altamirano, “A PSO algorithm to solve a Real Course+Exam Timetabling Problem,” in *International conference on swarm intelligence*, 2011, pp. 14–15.

[18] R. Tavakkoli-Moghaddam, M. Azarkish, and A. Sadeghnejad-Barkousaraie, “A new hybrid multi-objective Pareto archive PSO algorithm for a bi-objective job shop scheduling problem,” *Expert Syst. Appl.*, vol. 38, no. 9, pp. 10812–10821, Sep. 2011.

[19] T.-H. Wu, J.-Y. Yeh, and Y.-M. Lee, “A particle swarm optimization approach with refinement procedure for nurse rostering problem,” *Comput. Oper. Res.*, vol. 54, pp. 52–63, Feb. 2015.

[20] K.-H. Chao, Y.-S. Lin, and U.-D. Lai, “Improved particle swarm optimization for maximum power point tracking in photovoltaic module arrays,” *Appl. Energy*, vol. 158, pp. 609–618, Nov. 2015.

[21] S. He, Q. H. Wu, and J. R. Saunders, “A Group Search Optimizer for Neural Network Training,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006, pp. 934–943.

[22] S. He, Q. H. Wu, and J. R. Saunders, “Group Search Optimizer: An Optimization Algorithm Inspired by Animal Searching Behavior,” *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 973–990, Oct. 2009.

[23] C. J. Barnard and R. M. Sibly, “Producers and scroungers: A general model and its application to captive flocks of house sparrows,” *Anim. Behav.*, vol. 29, no. 2, pp.

543–550, May 1981.

[24] J. H. Holland, “Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence,” *MIT Press*. 1992.

[25] C. Kyriklidis and G. Dounias, “Evolutionary computation for resource leveling optimization in project management,” *Integr. Comput. Aided. Eng.*, vol. 23, no. 2, pp. 173–184, Mar. 2016.

[26] F. Padillo, J. M. Luna, F. Herrera, and S. Ventura, “Mining association rules on Big Data through MapReduce genetic programming,” *Integr. Comput. Aided. Eng.*, vol. 25, no. 1, pp. 31–48, Dec. 2017.

[27] P. E. Pillon, E. C. Pedrino, V. O. Roda, and M. C. Nicoletti, “A hardware oriented ad-hoc computer-based method for binary structuring element decomposition based on genetic algorithms,” *Integr. Comput. Aided. Eng.*, vol. 23, no. 4, pp. 369–383, Sep. 2016.

[28] D. B. Fogel, “An introduction to simulated evolutionary optimization,” in *Evolutionary Computation: The Fossil Record*, 1998.

[29] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, 1989.

[30] C. K. H. Lee, “A review of applications of genetic algorithms in operations management,” *Eng. Appl. Artif. Intell.*, vol. 76, pp. 1–12, Nov. 2018.

[31] H. Adeli and S.-L. Hung, *Machine Learning - Neural Networks, Genetic Algorithms, and Fuzzy Systems*. John Wiley and Sons, New York, 1994.

[32] N. Siddique, *Intelligent Control*, vol. 517. Cham: Springer International Publishing, 2014.

[33] N. Siddique and H. Adeli, *Computational Intelligence*. Oxford, UK: John Wiley & Sons Ltd, 2013.

[34] R. L. Kadri and F. F. Boctor, “An efficient genetic algorithm to solve the resource-constrained project scheduling problem with transfer times: The single mode case,” *Eur. J. Oper. Res.*, vol. 265, no. 2, pp. 454–462, Mar. 2018.

[35] C. Yu, Q. Semeraro, and A. Matta, “A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility,” *Comput. Oper. Res.*, vol. 100, pp. 211–229, Dec. 2018.

[36] Zong Woo Geem, Joong Hoon Kim, and G. V. Loganathan, “A New Heuristic

Optimization Algorithm: Harmony Search," *Simulation*, vol. 76, no. 2, pp. 60–68, Feb. 2001.

[37] S. Das, A. Mukhopadhyay, A. Roy, A. Abraham, and B. K. Panigrahi, "Exploratory Power of the Harmony Search Algorithm: Analysis and Improvements for Global Numerical Optimization," *IEEE Trans. Syst. Man, Cybern. Part B*, vol. 41, no. 1, pp. 89–106, Feb. 2011.

[38] Z. Yang *et al.*, "Multi-objective inventory routing with uncertain demand using population-based metaheuristics," *Integr. Comput. Aided. Eng.*, vol. 23, no. 3, pp. 205–220, Jun. 2016.

[39] R. Bruni and P. Detti, "A flexible discrete optimization approach to the physician scheduling problem," *Oper. Res. Heal. Care*, vol. 3, no. 4, pp. 191–199, Dec. 2014.

[40] N. Boland, B. D. Hughes, L. T. G. Merlot, and P. J. Stuckey, "New integer linear programming approaches for course timetabling," *Comput. Oper. Res.*, vol. 35, no. 7, pp. 2209–2233, Jul. 2008.

[41] W. Li *et al.*, "Mountain Railway Alignment Optimization with Bidirectional Distance Transform and Genetic Algorithm," *Comput. Civ. Infrastruct. Eng.*, vol. 32, no. 8, pp. 691–709, Aug. 2017.

[42] Y.-Z. Wang, "Using genetic algorithm methods to solve course scheduling problems," *Expert Syst. Appl.*, vol. 25, no. 1, pp. 39–50, Jul. 2003.

[43] W. Zhao, S. Guo, Y. Zhou, and J. Zhang, "A Quantum-Inspired Genetic Algorithm-Based Optimization Method for Mobile Impact Test Data Integration," *Comput. Civ. Infrastruct. Eng.*, vol. 33, no. 5, pp. 411–422, May 2018.

[44] O. Valenzuela, X. Jiang, A. Carrillo, and I. Rojas, "Multi-Objective Genetic Algorithms to Find Most Relevant Volumes of the Brain Related to Alzheimer's Disease and Mild Cognitive Impairment," *Int. J. Neural Syst.*, vol. 28, no. 09, p. 1850022, Nov. 2018.

[45] R. Paz, E. Pei, M. Monzón, F. Ortega, and L. Suárez, "Lightweight parametric design optimization for 4D printed parts," *Integr. Comput. Aided. Eng.*, vol. 24, no. 3, pp. 225–240, Jun. 2017.

[46] A. Martínez-Álvarez, R. Crespo-Cano, A. Díaz-Tahoces, S. Cuenca-Asensi, J. M. Ferrández Vicente, and E. Fernández, "Automatic Tuning of a Retina Model for a Cortical Visual Neuroprosthesis Using a Multi-Objective Optimization Genetic Algorithm," *Int. J. Neural Syst.*, vol. 26, no. 07, p. 1650021, Nov. 2016.

[47] D. Abramson, “Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms,” *Manage. Sci.*, vol. 37, no. 1, pp. 98–113, Jan. 1991.

[48] N. Siddique and H. Adeli, “Simulated Annealing, Its Variants and Engineering Applications,” *Int. J. Artif. Intell. Tools*, vol. 25, no. 06, p. 1630001, Dec. 2016.

[49] A. Prieto, F. Bellas, P. Trueba, and R. J. Duro, “Real-time optimization of dynamic problems through distributed Embodied Evolution,” *Integr. Comput. Aided. Eng.*, vol. 23, no. 3, pp. 237–253, Jun. 2016.

[50] J. Wright and I. Jordanov, “Quantum inspired evolutionary algorithms with improved rotation gates for real-coded synthetic and real world optimization problems,” *Integr. Comput. Aided. Eng.*, vol. 24, no. 3, pp. 203–223, Jun. 2017.

[51] S. Abdullah, E. K. Burke, and B. McCollum, “A hybrid evolutionary approach to the university course timetabling problem,” in *2007 IEEE Congress on Evolutionary Computation*, 2007, pp. 1764–1768.

[52] J. Henry, J. Henry Obit, D. Ouelhadj, D. Landa-Silva, and R. Alfred, “) An evolutionary non- Linear great deluge approach for solving course timetabling problems An Evolutionary Non-Linear Great Deluge Approach for Solving Course Timetabling Problems,” *Int. J. Comput. Sci. Issues*, vol. 9, no. 4, pp. 1–13, 2012.

[53] H. Turabieh, S. Abdullah, and B. McCollum, “Electromagnetism-like Mechanism with Force Decay Rate Great Deluge for the Course Timetabling Problem,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, pp. 497–504.

[54] M. A. Al-Betar and A. T. Khader, “A harmony search algorithm for university course timetabling,” *Ann. Oper. Res.*, vol. 194, no. 1, pp. 3–31, Apr. 2012.

[55] M. A. Al-Betar, A. T. Khader, and M. Zaman, “University Course Timetabling Using a Hybrid Harmony Search Metaheuristic Algorithm,” *IEEE Trans. Syst. Man, Cybern. Part C (Applications Rev.*, vol. 42, no. 5, pp. 664–681, Sep. 2012.

[56] W. Erben and J. Keppler, “A genetic algorithm solving a weekly course-timetabling problem,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1996, pp. 198–211.

[57] S. Yang and S. N. Jat, “Genetic Algorithms With Guided and Local Search Strategies for University Course Timetabling,” *IEEE Trans. Syst. Man, Cybern. Part C (Applications Rev.*, vol. 41, no. 1, pp. 93–106, Jan. 2011.

[58] C. Akkan and A. Gülcü, “A bi-criteria hybrid Genetic Algorithm with robustness objective for the course timetabling problem,” *Comput. Oper. Res.*, vol. 90, pp. 22–32, Feb. 2018.

[59] M. Ayob and G. Jaradat, “Hybrid Ant Colony systems for course timetabling problems,” in *2009 2nd Conference on Data Mining and Optimization*, 2009, pp. 120–126.

[60] H. Li and H. Zhang, “Ant colony optimization-based multi-mode scheduling under renewable and nonrenewable resource constraints,” *Autom. Constr.*, vol. 35, pp. 431–438, Nov. 2013.

[61] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, “A honey-bee mating optimization algorithm for educational timetabling problems,” *Eur. J. Oper. Res.*, vol. 216, no. 3, pp. 533–543, Feb. 2012.

[62] A. Alexandridis, E. Paizis, E. Chondrodima, and M. Stogiannos, “A particle swarm optimization approach in printed circuit board thermal design,” *Integr. Comput. Aided. Eng.*, vol. 24, no. 2, pp. 143–155, Mar. 2017.

[63] I. X. Tassopoulos and G. N. Beligiannis, “Solving effectively the school timetabling problem using particle swarm optimization,” *Expert Syst. Appl.*, vol. 39, no. 5, pp. 6029–6040, Apr. 2012.

[64] I. X. Tassopoulos and G. N. Beligiannis, “A hybrid particle swarm optimization based algorithm for high school timetabling problems,” *Appl. Soft Comput.*, vol. 12, no. 11, pp. 3472–3489, Nov. 2012.

[65] Al-Mahmud and M. A. H. Akhand, “ACO with GA operators for solving University Class Scheduling Problem with flexible preferences,” in *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, 2014, pp. 1–6.

[66] T. Ferdoushi, P. K. Das, and M. A. H. Akhand, “Highly constrained university course scheduling using modified hybrid particle swarm optimization,” in *2013 International Conference on Electrical Information and Communication Technology (EICT)*, 2014, pp. 1–5.

[67] F. Sharifa and R. Afroza, “A Study on Grammatical Evolution,” Khulna University of Engineering & Technology, 2003.

[68] I. Khan and M. K. Maiti, “A swap sequence based Artificial Bee Colony algorithm for Traveling Salesman Problem,” *Swarm Evol. Comput.*, vol. 44, pp. 428–438, Feb. 2019.

[69] Kang-Ping Wang, Lan Huang, Chun-Guang Zhou, and Wei Pang, “Particle swarm optimization for traveling salesman problem,” in *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, 2003, pp. 1583–1585.

[70] M. A. H. Akhand, P. C. Shill, M. F. Hossain, A. B. M. Junaed, and K. Murase, “Producer-Scrounger Method to Solve Traveling Salesman Problem,” *Int. J. Intell. Syst. Appl.*, vol. 7, no. 3, pp. 29–36, Feb. 2015.

[71] “Boost C++ Libraries.” [Online]. Available: https://www.boost.org/.