Optimization of University Course Scheduling Problem using Particle Swarm Optimization with Selective Search


Link to publication record in Ulster University Research Portal

Published in:
Expert Systems with Applications

Publication Status:
Published (in print/issue): 01/08/2019

DOI:
10.1016/j.eswa.2019.02.026

Document Version
Author Accepted version

General rights
Copyright for the publications made accessible via Ulster University's Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Ulster University's institutional repository that provides access to Ulster's research outputs. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact pure-support@ulster.ac.uk.
Optimization of University Course Scheduling Problem using Particle Swarm Optimization with Selective Search

Sk. Imran Hossain¹, M. A. H. Akhand¹, M. I. R. Shuvo¹, Nazmul Siddique² and Hojjat Adeli³

¹Department of Computer Science and Engineering, Khulna University of Engineering & Technology, Bangladesh, e-mail: imran@cse.kuet.ac.bd; akhand@cse.kuet.ac.bd; insan_shuvo@cse.kuet.ac.bd;

² School of Computing, Engineering and Intelligent Systems, Ulster University, United Kingdom, e-mail: nh.siddique@ulster.ac.uk;

³Department of Civil, Environmental and Geodetic Engineering, The Ohio State University, USA, e-mail: adeli.1@osu.edu;

Abstract:
University Course Scheduling Problem (UCSP) is a highly constrained real-world combinatorial optimization problem. 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 researches have reported solution for UCSP using Genetic Algorithm (GA), Particle Swarm Optimization (PSO) and Harmony Search (HS). Among them a few PSO based methods with different adaptations are proved to be effective solving UCSP. In general, the existing methods consider relatively simple UCSPs where the UCSP is first transformed into numeric domain then apply PSO to solve it. In this study, Particle Swarm Optimization with Selective Search (PSOSS), a novel PSO based method has been proposed for solving highly constrained UCSP by introducing swap sequence-based velocity, selective search and forceful swap application with repair mechanism. The proposed method has been tested for optimising course schedule of Computer Science and Engineering Department of Khulna University of Engineering & Technology which is relatively complex with many hard and soft constraints. Experimental results show the superiority of the proposed method compared to other prominent methods (e.g., GA, HS) for tackling the UCSP.

Keywords: University Course Scheduling, Particle Swarm Optimisation, Selective Search, Forceful Swap Application, Swap operator, Repair Mechanism.

1. Introduction

The timetabling is a real-life optimization problem that deals with scheduling of events of fixed number of timeslots and resources satisfying the soft and hard constraints and the necessary objectives as close as possible (Chiarandini et al., 2006; Mencia et al., 2016; Tang et al., 2018). The timetabling problem is NP-hard requiring a huge volume of computation for finding solutions, which grows exponentially with increasing problem size (Yue et al., 2017). Timetabling problem has to satisfy two kinds of constraints namely hard and soft constraints where hard constraints are the conditions that must be satisfied for a working timetable whereas the soft constraints are conditions that may be violated but they affect the solution quality (Pongcharoen et al., 2008).

Timetabling problem has found many applications in different domains such as employee allotment, transport systems, educational organizations, sports activities and industrial applications. An organization may come up with different timetabling problems for different applications. In higher educational institutions, examination and course scheduling are two important common and challenging tasks for optimizing physical and human resources
Among the various timetabling problems, university course scheduling is the most complex task requiring a set of soft and hard constraints to be satisfied. This task is well-known as University Course Scheduling Problem (UCSP) in the literature.

The goal of UCSP is to assign all theoretical classes and laboratory sessions to instructors, rooms and time slots considering the hard and soft constraints in a way such that there is no dispute in these assignments (Feizy-Derakhshi et al., 2012). The challenges of the UCSP are the constraints, for example, instructors’ dispositions, educational policies of the school, students’ cohort, availability of teaching staffs and other physical resources. In UCSP, each instructor can teach one class at a time slot and students can just go to one class at any given time. Other similar kind of constraints are treated as hard constraints that must be satisfied. The common soft constraints of the UCSP are instructors’ preferences for favoured days and timeslots, and preferences for the maximum length of breaks between teachings which must be satisfied to the extent possible. In the UCSP the main issue is to handle room allocation for lectures considering maximum capacity of room, number of enrolled students in a course or class and other related facilities (Shiau, 2011; Azimi, 2005). Both the hard and soft constraints may vary from institution to intuition 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. UCSP is a more complex combinatorial optimization problem than other combinatorial problems such as Travelling Salesman Problem (TSP). Although almost all metaheuristic methods have been applied to TSP, the number of studies on UCSP is much fewer than that of TSP because too many constraints must be satisfied in UCSP.

A number of meta-heuristic approaches (Yang et al., 2016) have been applied to the UCSP in the last few years. Among them are genetic algorithm (GA) (Wang, 2003; Pongcharoen, 2008; Martinez-alvarez et al., 2016; Li et al., 2017; Zhao et al., 2018), integer linear programming (Boland et al., 2008), tabu heuristic Search (Mushi, 2006), simulated annealing (Abramson et al, 1991), hybrid evolutionary algorithm (Prieto et al., 2016) with variable neighborhood search (Abdullah et al., 2007), hybrid GA with local search (Yang and Jat, 2011), hybrid evolutionary approach with nonlinear great deluge (Obit et al., 2012) and hybrid electromagnetism-like mechanism with great deluge (Turabieh et al., 2009) and Harmony Search (HS) algorithm (Al-Betar et al., 2012; Al-Betar et al., 2009). Wang et al. (2003) investigated GA for the UCSP with multiple constraints, which resulted in a timetable more acceptable to instructors. Mushi (2006) proposed a Tabu Search algorithm for UCSP with emphasis on the University of Dar-assalaam that generates schedule by heuristically minimizing penalties over infeasible solutions. Chiarandin et al. (2006) describe a metaheuristic algorithm based on the set of benchmark instances of ‘International Timetabling Competition’. The method combines heuristics, simulated annealing, variable neighborhood descent and tabu search.

Recently, various swarm intelligence (SI) based optimization methods have been investigated for UCSP such as Ant Colony Optimization (ACO) (Ayob and Jaradat, 2009; Li and Zhang, 2013), honey-bee mating optimization algorithm (Sabar et al., 2012). Among different SI methods, Particle Swarm Optimization (PSO) is the most popular due to its simplicity and adaptation ability (Alexandridis et al., 2017). Various PSO based strategies have been examined for UCSP. Shiau (2011) proposed an algorithm considering a bunch of constraints and a repair mechanism for all infeasible solutions. Tassopoulos and Beligiannis (2012) proposed a hybrid PSO algorithm for generating timetable. Chen and Shih (2013) investigated two different versions of PSO, the inertia weight version and the constriction version. Osman (2015) proposed a PSO approach for UCSP of Najran
University. The algorithm consists of two steps: first, the representation of the solutions as particle (i.e. particle encoding) and second the adjustment of the fitness function. The author used the basic PSO equations to adjust each particle’s position and velocity.

The main objective of this paper is to solve the highly constrained UCSP problem using a modified PSO based technique. Existing methods transform UCSP to numeric domain and then apply PSO for obtaining a viable solution (Tassopoulos and Beligiannis, 2012; Chen and Shih, 2013). In these cases, the conventional PSO method, used for function optimization, is chosen for the given UCSP. Instead of transforming the UCSP to numeric domain, a better approach would be the modification of the algorithm. Moreover, current methods considered simple instances of UCSP, which fail to provide quality solution for highly constrained scenario. In this paper, swap sequence has been adapted for velocity calculation in UCSP and selective search as well as forceful swap application with repair mechanism has been introduced for handling highly constrained nature of UCSP.

The proposed algorithm for the solution of the UCSP has been applied to the scheduling of department of Computer Science and Engineering of Khulna University of Engineering & Technology (KUET). The UCSP-KUET is considered a highly constrained realistic environment, where course scheduling is a difficult job. The rationale for working with UCSP-KUET is that: resource is much scarcer in KUET compared to western universities which makes it a more difficult task to create an optimal schedule. Experimental study shows that our proposed technique performs better compared to other traditional methods such as GA, HS, Producer-Scrounger Method (PSM) and PSO.

The rest of the paper is structured as follows: Section 2 describes the proposed method for UCSP. Section 3 presents the experimental studies with comparative analysis among algorithms. Finally, section 4 provides some concluding remarks.

2. Optimising UCSP using PSOSS

The following sections contain a brief overview of PSO and detailed description of the proposed PSOSS method for solving UCSP.

2.1 Overview of PSO

PSO developed by Kennedy and Eberhart (1995; 2001) is an optimization algorithm based on the social behaviour of swarms. In PSO, a bird of a flock or fish of a school is represented by a particle, and the swarm is a collection of particles (Chen and Shih, 2013). Every particle in the swarm indicates a candidate solution to the optimization problem. Every particle adjusts its position in the multidimensional search space based on the experience of its adjacent particles and personal experience. Particle uses its personal best position and the best position among its neighbours to move towards an optimal solution. The fitness of each particle is calculated using a fitness function which is associated with the problem at hand. PSO has been a popular technique for solving different constrained optimization problems.

Initially, PSO creates a population of particles randomly. The number of particles to be used in a population is problem dependent. Every particle representing a solution to the problem has three parameters namely velocity, position and fitness. At every iteration, a particle uses its personal best position and also the best position among its neighbours to update its position. This process continues until it reaches a stopping criterion.
Consider a search space of \( D \) dimensions consisting of \( M \) 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, velocity of a particle \( V_p \) is calculated using

\[
V_p^{(t)} = i V_p^{(t-1)} + l_1 r_1 (B_p - X_p^{(t-1)}) + l_2 r_2 (G - X_p^{(t-1)})
\]

(1)

where, \( i \) is the inertia factors, \( \{l_1, l_2\} \) are learning factors, and \( \{r_1, r_2\} \) are random values ranging from 0 to 1.

The position of the particle is updated using

\[
X_p^{(t)} = X_p^{(t-1)} + V_p^{(t)} \times T
\]

(2)

where, \( T \) represents time and is assumed to be unity. Position of a particle represents a solution and initially each particle is given a random position and a random velocity. In every iteration, updated velocity of each particle is determined using Eq. (1) and particle’s position is updated according to Eq. (2). The fitness value of each particle is updated for the new position and the personal best position \( B_p \) gets updated if a better fitness value is found compared to the previous one. The global best position \( G \) is also updated in the same manner. \( G \) is considered as the final result after termination of the operation. The algorithm ends when the stopping criterion is satisfied (Montero et al., 2011).

2.2 PSOSS for solving UCSP

Proposed PSOSS method works with a population of particles in which individual particle represents 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 application. The particle encoding, swap operator and swap sequence, velocity computation, forceful swap application with repair mechanism, selective search, fitness calculation and other operations are described in the following sections.

A. Particle Encoding

PSOSS works with a population of particles and each particle represents the complete schedule for instructors, students, classrooms and laboratories. A particle’s solution (\( S_p \)) is represented by instructor-wise solutions in a

\[
S_p = \begin{bmatrix}
   I1 & I2 & I3 & \ldots & Im-1 & Im
\end{bmatrix}
\]

(a) Instructor-wise summary view of a particle.

45 timeslots X \( m \) instructors

\[
\begin{array}{cccccccc}
\text{45 timeslots} & \text{45 timeslots} & \text{45 timeslots} \\
1 & 2 & 3 & \ldots & 44 & 45 & 41 & 42 & 43 & \ldots & 89 & 90 & \ldots & 45(m-1) +1 & 45(m-1) +2 & \ldots & 45m & 45m \\
\end{array}
\]

Instructor 1

Instructor 2

Instructor \( m \)

(b) Detailed view of particle

Figure 1: Particle representation of UCSP for KUET instance.
B. Swap Operator and Swap Sequence

A Swap Operator (SO) denotes the index of items to be swapped in a list. Consider the following list L,

\[
L = \begin{bmatrix}
    a & b & c & d \\
    0 & 1 & 2 & 3
\end{bmatrix}
\]

A SO(1,3) produces a new list \( L' \) as follows:

\[
L' = L + SO(1,3)
\]

\[
L' = \begin{bmatrix}
    a & d & c & b \\
    0 & 1 & 2 & 3
\end{bmatrix}
\]

here, ‘+’ does not mean any arithmetic operation rather it means the swap operation SO(a,b) on \( L \).

A Swap Sequence (SS) is a group of SOs defined as follows:

\[
SS = \{ SO_1, SO_2, SO_3, \ldots SO_n \}
\] (3)

Employment of a SS means application of all the SOs in a 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

\[
SS = B - A
\] (4)

For example, if \( SS = \{(1,3), (2,0)\} \) then,
\[ L' = L + SS \]

The figure shows instructor-wise swap sequences (SSs) of complete SS.

**C. Velocity Computation using Swap Operator and Swap Sequence**

SO and SS has been used in the proposed PSOSS method for velocity calculation. The SS to convert one particle’s solution to another particle’s solution is a collection of swap sequences which is measured in an instructor-to-instructor basis. Consider a UCSP consisting of two instructors I1 and I2 each having two courses C1, C2 and C3, C4 respectively. Figure 3 shows two different solutions A and B for the UCSP in consideration. In solution A, instructor I1 has a course C1 in slot no 1 and another one C2 in slot no 3 whereas in solution B, course C1 is at slot no 0 and C2 is at slot no 2. So, the required SS for converting the schedule of I1 in solution A to schedule of I1 in solution B is \( SS_{I1} = \{(1,0), (3,2)\} \). Similarly, \( SS_{I2} = \{(0,3), (2,1)\} \). So the complete SS for converting solution A to solution B is \( SS = \{SS_{I1}, SS_{I2}\} \).

In the proposed method, swap sequence SS is treated as velocity to update a particle’s position at each iteration which is calculated using Eq. (5)

\[
SS = \gamma SS_{PA} \otimes \alpha (S_{GB} - S_{P}) \otimes \beta (S_{PB} - S_{P}) \quad \alpha, \beta, \gamma \epsilon [0,1]
\]

where \( SS_{PA} \) is the previously applied velocity, \( S_{PB} \) is the previous best solution of the particle, \( S_{GB} \) is the global best solution of the swarm and \( \alpha, \beta, \gamma \) are selection probabilities for selecting a bunch of SOs from the corresponding SS. The equation \( SS_{GB} = S_{GB} - S_{P} \) represents instructor-wise SSs to reach \( S_{GB} \) from \( S_{P} \) and \( SS_{PB} = S_{PB} - S_{P} \) is the instructor-wise SSs to reach \( S_{PB} \) from \( S_{P} \). However, such simple SS calculation and operation is not suitable for solving UCSP. In UCSP, the solution for one instructor depends on others and intended swapping in an instructor’s solution may not be feasible due to unavailability of slots and resources which is held by others.

To make the operation useful, Eq. (5) is represented in a different form as follows:

\[
SS = \alpha (S_{GB} - S_{P}) + \beta (S_{PB} - S_{P}) \otimes \gamma SS_{PA}
\]

(6)

As \( SS_{GB} = S_{GB} - S_{P} \) and \( SS_{PB} = S_{PB} - S_{P} \), Eq. (6) can be written as:

\[
SS = \alpha \ast SS_{GB} + \beta \ast SS_{PB} \otimes \gamma SS_{PA}
\]

(7)

After selection of SOs with \( \alpha, \beta, \gamma \), SS becomes
\[ SS = SS_{GB} + SS_{PB} \otimes SS_{PA} = SS_{GB} + SS_{M} \]  

where, \( SS_{GB}, SS_{PB}, SS_{PA} \) are the selected SS from \( SS_{GB}, SS_{PB} \) and \( SS_{PA} \) respectively and \( SS_{M} \) is the swap sequence resulting from merger of \( SS_{PB} \) with \( SS_{PA} \).

As \( SS_{GB} \) and \( SS_{M} \) can contain redundant SOs, redundant swaps are removed from them and \( SS_{GB} \) and \( SS_{M} \) become \( SS_{MSGB} \) and \( SS_{MM} \), respectively, after removing redundant SOs. Finally, \( SS \) becomes:

\[ SS = SS_{MSGB} + SS_{MM} \]  

This final velocity \( SS \) is then applied using forceful swap application with repair mechanism (described in section 2.2C) and selective search (described in section 2.2D).

C. Forceful Swap Application with Repair Mechanism

In the proposed method, \( SS \) for solving UCSP consists of swaps to global best (\( SS_{GB} \)), swaps to personal Best (\( SS_{PB} \)) and Previously Applied Swaps (\( SS_{PA} \)). UCSP is highly constrained in nature and most of the constraints are interrelated. Consequently, if a class needs to be shifted to a new time slot then all the involved members such as instructor, students and room need to be free in that time slot. As a result, most of the selected swaps can’t be applied because of violation of constraints. Therefore, a portion of \( SS_{GB} \) is forcefully applied to present solution \( S_P \) to ensure that \( S_P \) moves a little towards \( SS_{GB} \). Forcefully applying a SO can result in conflicts so, a repair mechanism is involved in forceful SO application to make sure that no invalid solution results in that process. The repair mechanism works by randomly moving conflicting courses to non-conflicting positions.

D. Selective Search

In proposed method, velocity \( SS \) is applied using selective search mechanism. In selective search, each solution generated by applying a SO of \( SS \) is considered as an intermediate solution and 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.

Suppose, \( SS = \{SO_1, SO_2, SO_3, \ldots SO_n\} \) then the selective search can be written as

\[
S_p^1 = S_p + SO_1 \\
S_p^2 = S_p^1 + SO_2 \\
\vdots \\
S_p^n = S_p^{n-1} + SO_n
\]

In the above cases, \( S_p^1, S_p^2, \ldots, S_p^n \) are the intermediate solutions and the intermediate solution having the highest fitness becomes the final solution \( S_P \) in selective search as defined by the following equation:

\[ S_p = \max\{S_p^j\}, \ j = 1, 2, \ldots n \]  

Finally, the velocity is \( SS = \{SO_1, SO_2, SO_3, \ldots SO_j\}, \ 1 < j \leq n \).

The ultimate solution \( S_P \) in selective search is the intermediate solution possessing the highest fitness value. Thus, the selective search technique explores the opportunity of getting better solution from the intermediate solutions.

E. Fitness Calculation
Each instructor’s preference for conducting a class in a particular time slot is represented by an integer value as shown in Fig. 4. A higher value corresponds to a higher preference of an instructor to conduct the class in that particular time slot. Whereas, a negative value shows the instructor’s non-preference. The fitness of a particle’s solution is calculated by considering fitness of each of the instructors’ solution which belongs to that particle’s solution using following equation:

\[
F_{GS} = \sum_{i=0}^{m} F_{IS_i}
\]  

where, \( F_{GS} \) is the fitness of the particle’s solution, and \( F_{IS} \) is the fitness of the instructors’ solution.

Now, fitness of each instructor’s solution is calculated by considering quality and violation of the instructor’s solution using the following equation:

\[
F_{IS} = Q_{IS} - V_{IS}
\]

where, \( Q_{IS} \) is the quality of the instructors’ solution, and \( V_{IS} \) is the violation of the instructor’s solution.

Preference values of corresponding positions where courses were assigned to an instructor are summed up to calculate the satisfaction of each instructor’s solution. Violation of the instructors’ solution is calculated using the following equation:

\[
V_{IS} = \sum_{i=1}^{n} 2^{zi}
\]

where, \( n \) is the total number of blocks of consecutive classes in an instructor’s solution and \( z \) is the number of classes in a block.

2.3 PSOSS Algorithm for UCSP

The proposed PSOSS algorithm for solving UCSP is shown in Algorithm 1. The notations and inputs of the proposed algorithm are listed at the beginning of the Algorithm 1.

**Algorithm 1: UCSP-PSOSS**

**Input:**
Instructors’ Information, Batches’ Information,
Courses’ Information, Classrooms’ Information, Break Times’ information,
\( N \) - total number of iteration
\( N_P \) - total number of particles
\( \alpha \) - selection probability of a swap from swap sequence to global best solution
\( \beta \) - selection probability of a swap from swap sequence to personal best solution
\( \gamma \) - selection probability of a swap from previously applied swap sequence

\( F_p \) - percentage of swaps to be forced towards global best solution

**Output:** An optimal solution of UCSP

**Variables:**

- \( S_{PB} \) - personal best solution
- \( S_{GB} \) - global best solution
- \( S_P \) - current solution
- \( S_R \) - A random solution
- \( SS_{CA} \) - currently applied swap sequence
- \( SS_{PA} \) - previously applied swap sequence
- \( SS_R \) - random swap sequence for all Instructors
- \( SH \) - swap sequence holder for selective search
- \( S \) - set of particles
- \( T \) - set of Instructors
- \( CC \) - set of conflicting classes

1. create \( N_P \) particles and append them to \( P \)
2. **for all** \( p \in P \) **do**
   3. \( S_P \leftarrow \) a random solution
   4. calculate fitness of \( S_P \) as described in section 2.2E
   5. \( S_{PB} \leftarrow S_P \) //initially current solution is assigned as personal best solution
   6. \( SS_{PA} \leftarrow \emptyset \)
   7. **end for**
8. \( S_{GB} \leftarrow \text{solution}(\max P) \) //select solution having highest fitness among all the particles
9. **for** \( i \leftarrow 1 \) **to** \( N \) **do**
   10. **for all** \( p \in P \) **do**
       11. \( SSH \leftarrow \emptyset \)
       12. \( SH \leftarrow \emptyset \)
       13. **if** \( SS_{PA} = \emptyset \) **then**
           14. \( SS_{PA} \leftarrow SS_R \)
       **end if**
       15. \( SS_{GB} \leftarrow S_{GB} - S_P \)
       16. \( SS_{PB} \leftarrow S_{PB} - S_P \)
       17. \( SS_{SG} \leftarrow \alpha \times SS_{GB} \)
       18. \( SS_{SP} \leftarrow \beta \times SS_{PB} \)
       19. \( SS_{SPA} \leftarrow \gamma \times SS_{PA} \)
       20. \( SS_{SM} \leftarrow SS_{PB} \oplus SS_{PA} \) //merge \( SS_{PB} \) with \( SS_{PA} \)
       21. \( SS_{SMGB} \leftarrow \text{swapMinimizer}(SS_{SM}) \) //remove redundant swaps
       22. \( SS_{SM} \leftarrow \text{swapMinimizer}(SS_{SM}) \)
   23. **for all** \( t \in T \) **do**
       24. \( SS_{GB[t]} \leftarrow SS_{SMGB[t]} \) //select swap sequence for Instructor \( t \)
       25. \( NS_{F} \leftarrow F_p \times \vert SS_{GB[t]} \vert \)
       26. **for** \( a \leftarrow 1 \) **to** \( NS_F \) **do**
           27. \( S_P \leftarrow S_P + SS_{GB[t][a]} \) forcefully //apply \( SS_{GB[t][a]} \) forcefully to \( S_P \)
           28. \( CC \leftarrow \text{list of conflicting classes in } S_P \text{ resulting from } SS_{GB[t][a]} \text{ application} \)
           **if** \( CC \neq \emptyset \) **then**
               29. **for all** \( cc \in CC \)
                   30. move \( cc \) to a randomly selected non-conflicting position
               **end for**
           **end if**
           31. \( SS_{CA} \leftarrow SS_{CA} \cup \{ SS_{GB[t][a]} \} \)
           32. \( \text{selectiveSearch}(S_P, SS_{CA}, SH, SSH) \)
       **end for**
       **for** \( a \leftarrow NS_F + 1 \) **to** \( \vert SS_{GB[t]} \vert \) **do**
       **if** \( SS_{GB[t][a]} \) applicable **then**
           33. \( S_P \leftarrow S_P + SS_{GB[t][a]} \)
           34. \( SS_{CA} \leftarrow SS_{CA} \cup \{ SS_{GB[t][a]} \} \)
           35. \( \text{selectiveSearch}(S_P, SS_{CA}, SH, SSH) \)
       **end if
In the proposed algorithm, initial population of particles is generated by creating specified number of particles. Each particle's current solution $S_P$ gets initialized by a random solution. The fitness of each particle’s current solution is calculated as described in section 2.2E. $S_P$ also becomes the personal best solution $S_{PB}$ initially. Also, the previously applied swap sequence $SS_{PA}$ is initially empty. Then, the solution having the highest fitness among all the particles is selected as the global best solution $S_{GB}$. In each iteration, for each particle $SH$ and $SSH$ are emptied to be used for selective search. $SH$ is used to hold best intermediate solution and $SSH$ holds the swap sequence that produces $SSH$. A random swap sequence is assigned to $SS_{PA}$ if it is empty. Then instructor-wise swap sequences to reach $S_{GB}$ and $S_{PB}$ from $S_P$ are calculated which are represented by $SS_{GB}(=S_{GB} - S_P)$ and $SS_{PB}(=S_{PB} - S_P)$ respectively. Some swaps are selected for each instructor from $SS_{GB}$ based on the selection probability $\alpha$ denoted by $SS_{GB}(=\alpha \cdot SS_{GB})$. Similarly, $SS_{PB}(=\beta \cdot SS_{PB})$ and $SS_{SPA}(=\gamma \cdot SS_{SPA})$ are the selected swaps from $SS_{PB}$ and $SS_{SPA}$ respectively. $SS_{PB}$ and $SS_{SPA}$ are merged together to $SS_{M}=SS_{PB} \cup SS_{SPA}$. Redundant swaps are removed from $SS_{GB}$ and $SS_{M}$ using $swapMinimizer()$ function, results of which are denoted by $SS_{SMGB}$ and $SS_{MM}$ respectively. After that, for each instructor a portion of $SS_{SMGB}$ is selected using the equation $F_P \cdot |SS_{GB}|$, where $F_P$ is the force percentage and $SS_{GB}$ is the swap sequence corresponding to an instructor $t$. Swaps of this selected portion are forcefully applied to $S_P$ and resulting conflicts are resolved by randomly moving the conflicting classes to non-conflicting positions. Rest of the swaps are applied to $S_P$ if they do not create any conflicts. Similarly, the swaps from $SS_{MM}$ are applied only if they are applicable. Each applied swap gets added
to $SS_{CA}$ which holds currently applied swap sequence and. The selective search technique is used after applying each swap to ensure that best intermediate solution is retained. Algorithm 1.1 shows the required steps of selective search. It simply updates $SH$ and $SSH$ with $SP$ and $SS_{CA}$ respectively only if $SP$ is found better than $SH$. Finally, after the application of all the swaps the best intermediate solution $SH$ becomes particle’s solution $SP$ and the swap sequence $SSH$ that produces $SH$ becomes $SS_{PA}$ for next iteration. Then $SP_B$ is updated if $SP$ is found better than $SP_B$. Finally, $S_{GB}$ is recalculated and algorithm goes to next iteration. The algorithm uses a predefined number of iterations $N$ as the termination criteria. After termination $S_{GB}$ is considered as the final solution.

2.4 Illustration of the Mechanism of PSOSS with a Sample Problem

Figure 5: Illustration of the mechanism of PSOSS with a sample problem.
A schematic representation of the proposed method in shown in Fig. 5. Suppose, a system consisting of three instructors 11, 12 and 13, each having five weekly slots that need to be scheduled. In Fig. 5, $S_p$ is a particle’s present solution which consists of individual solution of all three instructors, $S_{GB}$ is the global best solution and $S_{PB}$ is the particle’s personal best solution. $S_{GB}(=S_{GB} - S_p)$ represents instructor-wise swap sequences to reach $S_{GB}$ from $S_p$, and $S_{PB}(=S_{PB} - S_p)$ is the instructor-wise swap sequences to reach $S_{PB}$ from $S_p$. $S_{PA}$ is the previously applied instructor-wise swap sequences. The circle (○) symbol inside the swap sequences represents a swap operator.

In the first step, some swaps are selected for each instructor from $SS_{GB}$ based on the selection probability $\alpha$ denoted by $SS_{SMGB}(=\alpha * SS_{GB})$. Similarly, $SS_{SPA}(=\beta * SS_{PB})$ and $SS_{PSA}(=\gamma * SS_{PA})$ are the selected swaps from $SS_{PB}$ and $SS_{PA}$ respectively. The redundant swaps are removed from $SS_{SMGB}$ using $swaMinimizerr()$ function, result of which is denoted by $SS_{SMGBA}(=$swaps numbered 1, 2, 3, 4, 5 and 6 in Fig. 5). Then, $SS_{SPB}$ and $SS_{SPA}$ are merged together to $SS_{SPA}(=SS_{SPB} \oplus SS_{SPA})$ before removing the redundant swaps from them. The redundant swaps are removed from $SS_{SPA}$ using $swaMinimizerr()$ function, result of which is denoted by $SS_{SMMA}(=$swaps numbered 7, 8, 9, 10, 11 and 12 in Figure 5). After that, a portion of $SS_{SMGB}$ is selected using the equation $F_p \cdot SS_{SMGB}$, where $F_p$ is the force percentage. This selected portion of $SS_{SMGB}$ is denoted by $SS_{SMGBA}(=$swaps numbered 1, 3 and 5 in Fig. 5) and the rest of the swaps are denoted by $SS_{SMGBA}(=$swaps numbered 2, 4 and 6 in Fig. 5). Swaps of $SS_{SMGBA}$ are forcefully applied to $S_p$ and then the swaps of $SS_{SMGBA}$ are applied to $S_p$ if they do not create any conflicts.

Any conflict resulting from forceful swap application is handled by repair mechanism as described in section 2.2C. Similarly, the swaps from $SS_{SMMA}$ are applied only if they are applicable. In Fig. 5, 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 $S_p$ then it becomes $S_p^1$ and similarly applying swap 3 on $S_p^3$ makes it $S_p^5$. In the example shown in Fig. 5, swaps numbered 1, 3 and 5 are forcefully applied on initial solution $S_p$ making it $S_p^5$. Then swap numbered 2 gets applied on $S_p^5$ resulting in $S_p^7$ as there is no conflicts. Solution stays at $S_p^7$ 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 then do not give any conflicts. The best one among these solutions is then picked as particle’s solution. Accordingly, $S_{PB}$ and $S_{GB}$ are updated for the next iteration.

3. Experimental Studies

This section investigates the effectiveness and performance of the PSOSS algorithm on UCSP-KUET instance for obtaining a viable timetable. The performance of the proposed algorithm has been compared with the performances of GA, HS, PSO and PSM for the same UCSP-KUET instance with the same experimental and parameter settings. This section also contains an experimental analysis for better understanding of the performance of the proposed method.

GA is a search and optimization algorithm mimicking natural selection and genetic mechanisms which includes crossover and mutation (Pillon et al., 2016; Kyriklidis and Douinas 2016; Padillo et al., 2018). The main notion of GA is the survival of the fittest (Rostami Neri, 2016). GA obtains a solution with the highest fitness after several iterations, which is considered the optimal solution (Adeli and Hung, 1995; Siddique and Adeli, 2013; Siddique, 2014). For ease of implementation, single point crossover is used in this paper with a crossover probability of 0.70. Mutation is performed by randomly changing the time slot of a course for a randomly selected instructor with mutation rate of 0.20. Elitism is also considered for implementation with an elite list of size 2.

HS algorithm is based on the notion of harmonic phenomena in musical performance. It is a population based algorithm inspired by improvisation process of musicians [Siddique and Adeli, 2017; 2015a; 2015b; 2015c]. There
are two unique operators in HS: Harmony Memory Consideration Rate (HMCR) and Pitch Adjustment Rate (PAR) which are used to produce and modify a solution, respectively [Wang et al., 2015]. HS is implemented with a HMCR of 0.95 and a PAR of 0.1 respectively.

In PSM, solution having the best fitness becomes the producer, some solutions having the worst fitness become dispersed members and the rest of the solutions are considered as scroungers. In each iteration, producer tries to find a better solution, scroungers move toward the producer with the hope of finding better solution and dispersed members move randomly for finding new solutions [Akhand et al., 2015]. PSM is implemented with a swap selection probability of 0.3 in this paper.

Standard PSO is also investigated for comparison in this paper. Picked values of the tuning parameters alpha, beta and gamma for implementation of PSO and PSOSS are 0.3, 0.5 and 0.2 respectively. Also a force rate of 100% has been used for PSOSS.

The hard constraints of UCSP-KUET instance are:
- 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.

The soft constraints of UCSP-KUET instance are:
- Maintain preference of instructor as much as possible.
- Keep the amount of consecutive classes as few as possible for instructors.

The algorithm has been implemented in Visual C++ of Microsoft’s Visual Studio 2013 on Windows 10 platform on Intel® Core™ i7-7700 CPU @ 3.60 GHz processor, and 8 GB RAM.

3.1 Experimental Environment

In the experimental environment, both instructors’ flexibility, and students’ flexibility are considered. The weekly time slots for instructors and their preferences are given in Fig. 2 and Fig. 4 respectively. The preferences varied from -1 to 5, where -1 means the lowest preference and 5 means the highest preference. Experiments with real-world input data taken from 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 time slots of 45 minutes duration. The theory and laboratory classes are conducted by a single instructor and the duration of each laboratory session as well as an M.Sc. class is three consecutive time slots.

3.2 Input Data Preparation

Table 1 lists the used preference values of all the instructors. There are five batches of students in the CSE Department at KUET: four batches in the undergraduate level and one batch at the postgraduate (MSc) level. In total 38 courses are taught by 27 instructors. Odd-numbered courses represent theory courses and even-numbered courses represent laboratory courses. Table 2 shows which courses belong to which batch, the required credit hours for each course, 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 3 shows the number of courses assigned to an instructor, the courses allocated to each instructor and the weekly workload of each instructor. Table 4 shows the class room id, room type, maximum seating capacity and allowable courses that can be taught in the class room. There are two types of class 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.
|   | 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 |
|   | 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 |

Table 1: Input preference values for instructors.
3.3 Experimental Results and Analysis

The size of the initial population of all the algorithms investigated in this study (i.e. GA, PSO, PSM, HS and PSOSS) is kept equal. The population size is one of the important parameters. Its impact is evaluated as the computation cost increases with growing population size. A second parameter is the maximum number of iterations for convergence to an optimal solution. On the other hand, individual algorithms have their own parameters. In this section, first the effect of the number of iterations and population size on the algorithms is investigated. Next, algorithms are compared based on instructors’ satisfaction followed by the sample timetables generated for an instructor by the algorithms.
For better understanding of the effect of varying population size on the algorithms, fitness values are calculated by varying the population size from 5 to 300 while keeping the iteration number fixed at 100. The results of varying population sizes are shown in Fig. 6. It is clear from the figure that PSOSS outperforms all the other algorithms for varying population sizes because of the use of the force in PSOSS that assures all the particles

<table>
<thead>
<tr>
<th>Instructor ID</th>
<th>No. of Courses</th>
<th>Course Code</th>
<th>Weekly Workload (Hrs/Week)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>4</td>
<td>CSE 1203, CSE 1204, CSE 2201, CSE 2202</td>
<td>18</td>
</tr>
<tr>
<td>I2</td>
<td>4</td>
<td>CSE 3211, CSE 3212, CSE 4239, CSE 6225</td>
<td>12</td>
</tr>
<tr>
<td>I3</td>
<td>2</td>
<td>CSE 3201, CSE 3202</td>
<td>9</td>
</tr>
<tr>
<td>I4</td>
<td>1</td>
<td>CSE 3200</td>
<td>6</td>
</tr>
<tr>
<td>I5</td>
<td>3</td>
<td>CSE 4211, CSE 4212, CSE 6471</td>
<td>9</td>
</tr>
<tr>
<td>I6</td>
<td>2</td>
<td>CSE 4207, CSE 6465</td>
<td>6</td>
</tr>
<tr>
<td>I7</td>
<td>1</td>
<td>CSE 2207</td>
<td>3</td>
</tr>
<tr>
<td>I8</td>
<td>1</td>
<td>CSE 1201</td>
<td>3</td>
</tr>
<tr>
<td>I9</td>
<td>1</td>
<td>CSE 2208</td>
<td>3</td>
</tr>
<tr>
<td>I10</td>
<td>2</td>
<td>CSE 2200, CSE 3207</td>
<td>9</td>
</tr>
<tr>
<td>I11</td>
<td>1</td>
<td>CSE 3203</td>
<td>3</td>
</tr>
<tr>
<td>I12</td>
<td>1</td>
<td>CSE 1202</td>
<td>6</td>
</tr>
<tr>
<td>I13</td>
<td>1</td>
<td>CSE 4208</td>
<td>3</td>
</tr>
<tr>
<td>I14</td>
<td>1</td>
<td>CSE 2213</td>
<td>3</td>
</tr>
<tr>
<td>I15</td>
<td>1</td>
<td>CSE 3204</td>
<td>6</td>
</tr>
<tr>
<td>I16</td>
<td>1</td>
<td>EEE 1217</td>
<td>3</td>
</tr>
<tr>
<td>I17</td>
<td>1</td>
<td>EEE 1218</td>
<td>3</td>
</tr>
<tr>
<td>I18</td>
<td>1</td>
<td>EEE 2217</td>
<td>3</td>
</tr>
<tr>
<td>I19</td>
<td>1</td>
<td>EEE 2218</td>
<td>6</td>
</tr>
<tr>
<td>I20</td>
<td>1</td>
<td>MATH 1207</td>
<td>3</td>
</tr>
<tr>
<td>I21</td>
<td>1</td>
<td>MATH 2207</td>
<td>3</td>
</tr>
<tr>
<td>I22</td>
<td>1</td>
<td>ECE 3215</td>
<td>3</td>
</tr>
<tr>
<td>I23</td>
<td>1</td>
<td>ME 1270</td>
<td>3</td>
</tr>
<tr>
<td>I24</td>
<td>1</td>
<td>IEM 4227</td>
<td>3</td>
</tr>
<tr>
<td>I25</td>
<td>1</td>
<td>CHEM 1207</td>
<td>3</td>
</tr>
<tr>
<td>I26</td>
<td>1</td>
<td>CHEM 1208</td>
<td>3</td>
</tr>
<tr>
<td>I27</td>
<td>1</td>
<td>HUM 4207</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 3: Course information for each instructor.
move a little towards the global best and the personal best solutions as described in section 2.2C. Among the other algorithms PSM works well for low population size because the producer improves its fitness in each iteration.

The effect of varying iterations on the algorithms is shown in Fig. 7. The maximum number of iterations is varied from 5 to 300 while keeping the population size fixed at 50. It is clear from the figure that PSOSS performs better than other algorithms for all max number of iterations. Standard PSO is the second best performer.

One of the objectives of optimizing UCSP is to satisfy the demands of instructors’ having high workload as much as possible. Therefore, algorithms are compared based on the percentage of instructors’ satisfaction. The percentage of satisfaction for an instructor is computed using the formula:

\[
Satisfaction_I = \frac{F_{IS}}{MF_{IS}} \times 100
\]  

Table 4: Information for classrooms and laboratories.

<table>
<thead>
<tr>
<th>Room ID</th>
<th>Room Type</th>
<th>Room capacity</th>
<th>Allowable Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>Lecture</td>
<td>60</td>
<td>Any Theoretical Subjects</td>
</tr>
<tr>
<td>CR2</td>
<td>Lecture</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>CR3</td>
<td>Lecture</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>CR4</td>
<td>Lecture</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>CR5</td>
<td>Lecture</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>LB1</td>
<td>Laboratory</td>
<td>30</td>
<td>CSE 2200, CSE 3202, CSE 4208, CSE 4212</td>
</tr>
<tr>
<td>LB2</td>
<td>Laboratory</td>
<td>30</td>
<td>CSE2200, CSE4212, CSE3212, CSE2208, CSE3202</td>
</tr>
<tr>
<td>LB3</td>
<td>Laboratory</td>
<td>30</td>
<td>CSE 1202, CSE 2202</td>
</tr>
<tr>
<td>LB4</td>
<td>Laboratory</td>
<td>30</td>
<td>CSE1204</td>
</tr>
<tr>
<td>LB5</td>
<td>Laboratory</td>
<td>30</td>
<td>CSE3204</td>
</tr>
<tr>
<td>LB6</td>
<td>Laboratory</td>
<td>30</td>
<td>ME1270</td>
</tr>
<tr>
<td>LB7</td>
<td>Laboratory</td>
<td>30</td>
<td>EEE1218, EEE2218</td>
</tr>
<tr>
<td>LB8</td>
<td>Laboratory</td>
<td>30</td>
<td>CHEM1208</td>
</tr>
</tbody>
</table>

Figure 7: Variation effect of Iterations.
where, $MF_{IS}$ is the maximum possible fitness of an instructor’s solution, $FiS$ is the achieved fitness of an instructor’s solution as described by Eq. (12).

Table 5 shows the achieved satisfaction value (in %) for all the instructors for all the algorithms. It also includes weekly work load for each instructor. It is seen from the table that the instructor I1 has the highest workload among all others and PSOSS achieved the highest satisfaction value (in %) for instructor I1 which is also the case for the second highest work load for instructor I2. Though for some instructors with some less work load, the satisfaction value achieved by PSOSS is not higher than other algorithms but the average satisfaction value achieved by PSOSS is much higher than other algorithms. This shows a significant performance indicator.

Table 6 shows the timetable for instructor I1 generated by the algorithms GA, PSO, HS, PSM and PSOSS. In the generated timetable G0 denotes a batch and G1 and G2 represent subgroups. From the generated timetable by PSOSS shown in Table 6 (e), it can be seen that I1 has a Laboratory class CSE 1204 in timeslots 7, 8 and 9 on Sunday which is desirable because I1 has maximum preference of 4 in these periods for Sunday as stated in Table 1. Similarly, timetable for other days also adheres to instructor I1’s preference in most of the cases. Overall, the timetable generated by PSOSS for instructor I1 is satisfactory compared to timetables generated by other algorithms. It is to be noted that KUET has working days from Sunday to Thursday.
Table 6: Sample Timetable for Instructor 1 generated by GA, PSO, HS, PSM and PSOSS.

(a) GA

<table>
<thead>
<tr>
<th>Day</th>
<th>Time Slot</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>CSE1204</td>
<td>LB4</td>
<td>B1</td>
<td>G1</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1203</td>
<td>LB4</td>
<td>B1</td>
</tr>
<tr>
<td>Mon</td>
<td>CSE1203</td>
<td>CR4</td>
<td>B2</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1201</td>
<td>CR4</td>
<td>B2</td>
</tr>
<tr>
<td>Tue</td>
<td>CSE1202</td>
<td>CR5</td>
<td>B1</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1201</td>
<td>CR4</td>
<td>B2</td>
</tr>
<tr>
<td>Wed</td>
<td>CSE1204</td>
<td>LB4</td>
<td>B1</td>
<td>G2</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1201</td>
<td>CR4</td>
<td>B2</td>
</tr>
<tr>
<td>Thu</td>
<td>CSE1202</td>
<td>CR5</td>
<td>B2</td>
<td>G2</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1201</td>
<td>CR4</td>
<td>B2</td>
</tr>
</tbody>
</table>

(b) PSO

<table>
<thead>
<tr>
<th>Day</th>
<th>Time Slot</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>CSE2202</td>
<td>LB3</td>
<td>B2</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1203</td>
<td>LB4</td>
<td>B1</td>
</tr>
<tr>
<td>Mon</td>
<td>CSE2201</td>
<td>CR4</td>
<td>B2</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1201</td>
<td>CR4</td>
<td>B2</td>
</tr>
<tr>
<td>Tue</td>
<td>CSE1202</td>
<td>CR5</td>
<td>B1</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1202</td>
<td>LB3</td>
<td>B2</td>
</tr>
<tr>
<td>Wed</td>
<td>CSE1204</td>
<td>LB4</td>
<td>B2</td>
<td>G1</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1201</td>
<td>CR4</td>
<td>B2</td>
</tr>
<tr>
<td>Thu</td>
<td>CSE2201</td>
<td>CR4</td>
<td>B2</td>
<td>G1</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1201</td>
<td>CR4</td>
<td>B2</td>
</tr>
</tbody>
</table>

(c) HS

<table>
<thead>
<tr>
<th>Day</th>
<th>Time Slot</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>CSE2201</td>
<td>LB3</td>
<td>B2</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1204</td>
<td>LB4</td>
<td>B1</td>
</tr>
<tr>
<td>Mon</td>
<td>CSE2201</td>
<td>CR4</td>
<td>B2</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1201</td>
<td>CR4</td>
<td>B2</td>
</tr>
<tr>
<td>Tue</td>
<td>CSE1203</td>
<td>CR4</td>
<td>B1</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1202</td>
<td>LB3</td>
<td>B2</td>
</tr>
<tr>
<td>Wed</td>
<td>CSE2202</td>
<td>CR3</td>
<td>B1</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1203</td>
<td>LB4</td>
<td>B1</td>
</tr>
<tr>
<td>Thu</td>
<td>CSE2201</td>
<td>CR4</td>
<td>B1</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1204</td>
<td>LB4</td>
<td>B1</td>
</tr>
</tbody>
</table>

(d) PSM

<table>
<thead>
<tr>
<th>Day</th>
<th>Time Slot</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>CSE1204</td>
<td>LB4</td>
<td>B1</td>
<td>G2</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1203</td>
<td>LB4</td>
<td>B1</td>
</tr>
<tr>
<td>Mon</td>
<td>CSE2201</td>
<td>LB4</td>
<td>B1</td>
<td>G2</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE2202</td>
<td>LB3</td>
<td>B2</td>
</tr>
<tr>
<td>Tue</td>
<td>CSE2201</td>
<td>LB4</td>
<td>B1</td>
<td>G2</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE2202</td>
<td>LB3</td>
<td>B2</td>
</tr>
<tr>
<td>Wed</td>
<td>CSE2202</td>
<td>LB3</td>
<td>B2</td>
<td>G1</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE2201</td>
<td>LB4</td>
<td>B1</td>
</tr>
<tr>
<td>Thu</td>
<td>CSE2201</td>
<td>CR4</td>
<td>B2</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1203</td>
<td>CR5</td>
<td>B1</td>
</tr>
</tbody>
</table>

(e) PSOSS

<table>
<thead>
<tr>
<th>Day</th>
<th>Time Slot</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>CSE1204</td>
<td>LB4</td>
<td>B1</td>
<td>G2</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1204</td>
<td>LB4</td>
<td>B1</td>
</tr>
<tr>
<td>Mon</td>
<td>CSE2201</td>
<td>LB4</td>
<td>B1</td>
<td>G2</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE2202</td>
<td>LB3</td>
<td>B2</td>
</tr>
<tr>
<td>Tue</td>
<td>CSE2201</td>
<td>CR3</td>
<td>B1</td>
<td>G0</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE2201</td>
<td>CR3</td>
<td>B1</td>
</tr>
<tr>
<td>Wed</td>
<td>CSE2202</td>
<td>LB3</td>
<td>B2</td>
<td>G1</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE2201</td>
<td>CR3</td>
<td>B1</td>
</tr>
<tr>
<td>Thu</td>
<td>CSE1204</td>
<td>LB4</td>
<td>B2</td>
<td>G2</td>
<td>I1</td>
<td></td>
<td></td>
<td>CSE1204</td>
<td>LB4</td>
<td>B2</td>
</tr>
</tbody>
</table>
4. Conclusions

In this paper, a PSO based innovative technique PSOSS has been proposed to solve UCSP. The proposed method differs from existing methods, including many variants of PSO-based approaches, where UCSP is transformed into an equivalent numerical 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: sequences for global best; and sequences combining personal best and previous velocity. A portion of global sequence portion is considered to be applied forcefully with repair mechanism to change other dependent schedule. After applying SOs one by one, the best intermediate solution is considered as the final solution based on selective search. The results obtained by our proposed method show significant improvement in respect of quality of solutions compared to other traditional methods.

References


