A Novel and Effective University Course Scheduler Using Adaptive Parallel Tabu Search and Simulated Annealing

Xiaorui Shao[1], Su Yeon Lee[2], and Chang Soo Kim[3*]
[1] Industrial Science Technology Research Center, Pukyong National University, Busan 608737, Korea
[e-mail: shaoxiaorui@pukyong.ac.kr]
[2] CEO, JRTech Co., Ltd, Ulsan, 44781 Republic of Korea
[e-mail : sylee@jungrok.co.kr]
[3] Information Systems, Pukyong National University, Busan, 608737 South Korea
[email :cskim@pknu.ac.kr]
[*]Corresponding author: Chang Soo Kim

Received October 23, 2023; revised January 8, 2024; accepted March 11, 2024; published April 30, 2024

Abstract

The university course scheduling problem (UCSP) aims at optimally arranging courses to corresponding rooms, faculties, students, and timeslots with constraints. Previously, the university staff solved this thorny problem by hand, which is very time-consuming and makes it easy to fall into chaos. Even some meta-heuristic algorithms are proposed to solve UCSP automatically, while most only utilize one single algorithm, so the scheduling results still need improvement. Besides, they lack an in-depth analysis of the inner algorithms. Therefore, this paper presents a novel and practical approach based on Tabu search and simulated annealing algorithms for solving USCP. Firstly, the initial solution of the UCSP instance is generated by one construction heuristic algorithm, the first fit algorithm. Secondly, we defined one union move selector to control the moves and provide diverse solutions from initial solutions, consisting of two changing move selectors. Thirdly, Tabu search and simulated annealing (SA) are combined to filter out unacceptable moves in a parallel mode. Then, the acceptable moves are selected by one adaptive decision algorithm, which is used as the next step to construct the final solving path. Benefits from the excellent design of the union move selector, parallel tabu search and SA, and adaptive decision algorithm, the proposed method could effectively solve UCSP since it fully uses Tabu and SA. We designed and tested the proposed algorithm in one real-world (PKNU-UCSP) and ten random UCSP instances. The experimental results confirmed its effectiveness. Besides, the in-depth analysis confirmed each component's effectiveness for solving UCSP.

Keywords: Timetabling, scheduling, metaheuristic algorithm, university course scheduling problem

---

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Education(2021R1I1A3052605).

http://doi.org/10.3837/tiis.2024.04.002 **ISSN : 1976-7277**

1. Introduction

University course scheduling problem (UCSP), also called timetabling, is vital to implementing intelligent teaching systems in the university to improve efficiency and reduce labor costs [1], [2]. Its purpose is to arrange teaching resources such as professors, students, and rooms within a fixed timeslot to satisfy several constraints [3], [4], and it has been proved to be an NP-complete problem [5]. In previous years, the university staff solve UCSP by hand. They first arrange the simple course and then the completed one, which takes much time and is easily trapped and difficult to extricate oneself since UCSP's continuance. If one fails, the staff needs to adjust all of the others. In addition, with the increasing number of faculties, students, and courses, the real UCSP is increasingly complex. Solving complex UCSP instances by hand is difficult and even not possible. Therefore, one automatic and accurate UCSP method is necessary.

Similar to other optimization problems, such as job shop scheduling problem (JSSP) [6]–[8], route planning (RP) [9], and traveling salesman problems (TSP) [10], [11], UCSP is one optimization problem [12]. They aim to develop one algorithm that can attach the optimal goals. Especially, JSSP is to arrange all jobs at corresponding machines with a minimum make-span; DP breaks one huge-mode problem into several small groups to find the shortest path; and UCSP arranges all teaching resources, including courses, rooms, faculties, students, and timeslots by minimizing one defined fitness score. The only difference between UCSP and JSP is that UCSP has some hard constraints to be satisfied and cannot be broken out. Meanwhile, some soft constraints should be satisfied most during the solving. Therefore, the methods used for solving JSSP and DP also could be applied to solving UCSP. This manuscript mainly gives the related references for JSSP and UCSP since they are highly related.

The current methods for solving the optimization mentioned above problems mainly contain two branches. One is the exact approach [13], [14], which aims at finding the best solution through modeling one complex mathematical function. Another is the approximation-based approach, which aims at finding one near-optimal solution. The exact methods contain mathematical programming (MP) methods. E.g., Gomes et al. [15] used integer linear programming (ILP) to solve one of JSSPs, the flexible job shop scheduling problem (FJSSP). Boland et al. proposed a new ILP to solve UCSP [2]. Even though the exact approach could give optimal solutions for the above optimization problems, it cannot solve large instances to satisfy modern universities' needs since it requires numerous computation and time resources [16]. Therefore, most current research for solving the above optimization problems focuses on approximation-based methods.

The approximation-based methods for solving the above optimization methods could be divided into meta-heuristic and learning-based methods. The recent popular learning-based method could solve UCSP and JSP faster but relies on meta-heuristic approaches to generate labels. Hence, the performance is still far from meta-heuristic methods [7], [8], [17]. As a result, this manuscript mainly investigated the meta-heuristic methods for solving JSSP and UCSP. The metaheuristic methods contain genetic algorithm (GA) [18], Tabu search [19], simulated annealing (SA) [20], particle swarm optimization (PSO) [21], etc. E.g., Kuri [22] proposed a new island GA to solve JSSP, and the experiments on 52 JSSP data sets confirmed its effectiveness; Dehghan-Sanej et al. [14] utilized SA for medium- and large-sized JSSP instances; Xu et al. [23] tested the effectiveness of GA for hydro unit economic load dispatch; A hybrid method based on GA and Tabu are proposed to solve flexible JSSP problems [16], in which GA is used to find the global solution path, the found global path is fed into Tabu search to find the best local path. Besides, several pieces of research related to PSO for JSSP could be found from [24]–[26].

Compared to the application of meta-heuristic methods in the domain of JSSP, the references related to UCSP are relatively few. Mainly, Abdullah et al. [27] proposed a GA-based algorithm by minimizing the number of timetable violations of soft constraints. Awad et al. [28] proposed an adaptive Tabu search algorithm to solve large-scale UCSP instance. Leite et al. [29] proposed a fast SA (FastSA) to solve an examination timetable problem, the experimental results proved that it uses 17% less evaluations, on average, and a maximum of 41% less evaluations on one instance compared to standard SA. Chen et al. [30] utilized PSO with local search to solve timetable problem. Besides, a recent study [31] combined GA and machine learning methods to solve UCSP, in which GA is used to generate the intimal solution, and a machine learning algorithm is used to detect if these models can accurately approximate the evaluation function for UCSP. Turabieh et al. [32] applied fish swarm algorithm to solve this problem on Socha data set. One Tabu-based algorithm with random partial neighborhood search is developed in [33] to solve UCSP. More details about the UCSP's definition, trends, and perspectives can be found in [34], [35]

The above-mentioned metaheuristic approaches already obtained acceptable solutions. However, they only use one algorithm to select the next step in the solution, limiting its performance. Besides, they lacked more comprehensive and in-depth analysis to demonstrate their effectiveness. Therefore, this paper proposes a new approach that combines Tabu search and SA in a parallel mode to solve UCSP effectively and quickly. The proposed method first generates the initial solution by the first fit algorithm, which saves much time compared to the random initialization. Then, we designed one union move selector that consists of two changing move selectors to provide diverse solutions to improve performance. The diverse solutions from two changing move selectors are fed into Tabu search and SA algorithms in parallel to filter out unacceptable moves that break out some soft constraints. At last, we designed an adaptive decision algorithm to choose the best next-step move from Tabu and SA solutions to construct the final solution path. Benefiting from the excellent design of the proposed method structure, it generates a robust initial solution and diverse moves. It fully uses Tabu search and SA algorithms to solve UCSP effectively.

Moreover, we illustrate the proposed method’s advantages compared to the current methods, as listed in **Table 1**. The table showed that the exact approach is the most straightforward and optimal but is time- and computation-consumption. Besides, it cannot deal with large UCSP instances. The existing meta-heuristic methods are near-optimal, but they consider one single algorithm, except [16] utilized one cascade GA+Tabu, so the performance can still be improved. The learning-based method is fast but needs to be more accurate since it relies on other metaheuristic algorithms. In contrast, the proposed method is one near-optimal method, whose speed is faster, the performance is higher than the existing metaheuristic algorithms due to the excellent design of the first fit and parallel mode of Tabu and SA algorithms, which has been proved in Section 3.

The main contributions of this work are summarized as follows:

- We proposed a novel and practical framework to solve UCSP problems effectively and fastly. Its effectiveness has been confirmed on one real-world UCSP data set and ten random data sets.
- A parallel structure of Tabu and SA is designed to help the proposed method generate diverse moves to improve performance. Besides, it is easy to apply in other optimization tasks such as JSSP, RP, and TSP.
- We proposed a new evaluation metric for validating the UCSP algorithm, called normalized score in Eq. (3), which normalizes all soft constraints into one same level to compare them fairly.

- We did an in-depth analysis of the proposed method to confirm each component's effectiveness.

Table 1. The comparison between the proposed method for UCSP and others

Methods		References	Advantages	Disadvantages
Exact		[2], [15]	Simple and Optimal	Time- and computation-consumption cannot deal with large UCSP instances
Approximation	Metaheuristic	[14], [16], [22], [23], [24]–[26], [27], [28], [29], [30], [33]	Near-optimal	Single algorithm and performance is still can be improved
	Learning	[7], [8], [17], [31]	Much Faster	Not accurate
Proposed		-	Fast, combined method, near-optimal	-

The remainder of this manuscript is arranged as follows. Section 2 introduces the proposed method in detail. In Section 3, we verified the proposed method on one real-world UCSP instance and did an in-depth analysis to discuss the proposed method. Section 4 concludes this manuscript.

2. The Proposed Methods

2.1 Problem Definition

UCSP is to arrange teaching materials including courses $C = \{C_1, C_2 ..., C_n, ..., C_N\}$, faculties $Pr = \{Pr_1, Pr_2, ..., Pr_p ..., Pr_P\}$, students $S = \{S_1, S_2, ..., S_q ..., ..., S_Q\}$ to a set of rooms $R = \{R_1, R_2, ..., R_o, ..., R_O\}$, timeslots $TS = \{TS_1, TS_2, ..., TS_m, ..., TS_M\}$ accurately. Where N, P, Q, O, and M are corresponding courses, faculty, students, rooms, and timeslot numbers. It requires meeting all hard and most soft constraints during the solving process. If the solution breaks out of any hard constraints, this solution will be treated as an infeasible solution. One good solution satisfies all hard constraints and has the highest scores for all soft constraints.

We utilized a set of problem cases to describe UCSP in this manuscript. Firstly, five hard constraints are given:

H1: ***RoomTimeHardConflict.*** One room R_o can accommodate at most one course at the same timeslot TS_m.

H2 CourseTypeHardConflict: The room R_o should meet the course's functional requirements $room_{type}$. For example, some courses require devices such as computers and functional machines, while others do not.

H3 RoomSizeHardConflict: The room size $room_{size}$ must be equal to or larger than the applied student's numbers $student_{nubmer}$.

H4 TeachingHardConflict: One faculty Pr_p can teach at most one course C_n at the same timeslot TS_m.

H5 StudentHardConflict: One student S_q can only attend one course C_n at most at the same time TS_m.

Secondly, the five soft constraints are listed as follows:

S1 FacultyRoomStability: The faculty Pr_p prefers to teach in one fixed room R_o.

S2 FacultyTimePreference: The faculty Pr_p has his preferred time duration TS for each course R_o.

S3 FacultyTeachingMode: The faculty Pr_p prefers to teach in a sequential mode.

S4 StudentStudyingMode: The student S_q dislikes studying in a sequence mode.

S5 CourseDepartmentConsistency: The course C_n and room R_o should be consistent with its department.

According to the definition of UCSP, we can write the optimization function during solving, as shown in Eq. (1). Where S_{hard} is the score of hard constraints, which is equal to zero by default. If the solution breaks out any hard constraints (occurring hard constraints conflicts), we will give one negative score, indicating that this solution is infeasible. It ensures that all given solutions are feasible. The S_{soft} is the score for soft constraints, the sum of five soft constraints: $S_{S1}, S_{S2}, S_{S3}, S_{S4}$, and S_{S5}. The coefficients a_1, a_2, a_3, a_4, and a_5 illustrate each soft constraint's reward/penalize level. Especially if the solution satisfies one soft constraint, the S_{soft} will be given one reward (positive value) with different coefficients. In contrast, if the solution breaks out one soft constraint, the S_{soft} will be penalized one negative value with corresponding weights. Generally, $a_1 = a_2 = a_3 = a_4 = a_5 = 1(reward)/-1(penalize)$, and its influence will be discussed later in Section 3.5.

$$
\begin{aligned}
Score &= S_{hard} + S_{soft} \\
&= 0 + S_{soft} \\
&= a_1 S_{S1} + a_2 S_{S2} + a_3 S_{S3} + a_4 S_{S4} + a_5 S_{S5}
\end{aligned} \tag{1}
$$

2.2 UCSP Objects

The UCSP solution refers to three objects: Room R, Course C, and Timeslot TS, and they are defined in **Table 2**. Significantly, the UCSP factor (Room R and Timesolt TS) cannot change during solving while Entity (Course C) changes, and Decision object (Timetable $Ttable$) gives the final solution and the corresponding score $\{room_{list}, course_{list}, score\}$. The term 'id' is used to identify each sample of each object, the Fact object Room $R = \{id: int, name: str, room_{type}: str, size: str, department: str\}$, where $name$ is the course name, $room_{type}$ records some the functional type for each room, size is the room capacity, and department records its owner. Fact Timeslot TS records the day of the week dayofweek, course start $start_{time}$, and end time end_{time}. Course C consists of its name $subject$, teaching faculty $faculty$, department department, student group $student_{group}$, and applied number $student_{number}$, course type $course_{type}$, faculty preferable time $refer_{time}$, corresponding timeslot $timesolt$, and room $room$.

Table 2. Defined objects.

Object	Type	Variables
Room R	Fact	$\{id: int, name: str, room_{type}: str, size: str, department: str\}$
Timeslot TS	Fact	$\{id: int, dayofweek: str, start_{time}: datetime, end_{time}: datetime\}$
Course C	Entity	$\{id: int, subject: str, faculty: str, department: str, student_{group}: str,$ $course_{type}: str, prefer_{time}: Timesolt, timeslot: Timesolt, room: Roo$
Timetable $Ttable$	Decision	$\{room_{list}, course_{list}, score\}$

The pseudo-code utilized the defined objects to control hard and soft constraints, as in algorithms 1 and 2. Breaking out the constraint will be penalized by adding one negative value, and satisfying the constraint will be rewarded with one positive value, as described in Eq. (1). Breaking out any hard constraint means the solution is infeasible. In the algorithms, the UCSP solution course list that recorded some course objects $\{C_1, C_2, \ldots, C_i, \ldots, C_N\}$ controls corresponding constraints. The operation “. ” means object’s attributes. Moreover, we defined the faculty-liked and student-disliked time duration as 12 hours, which is flexible and can be changed into any proper time duration.

Algorithm 1: Pseudo-code for controlling hard constraints

1. Given the UCSP instance solution course list $\text{course}_{\text{list=}}\{C_1, C_2, \ldots, C_i, \ldots, C_N\}$
2. If $C_i.course_{type}! = C_i.room.room_{type}$:
3.   Break ***H2 CourseTypeHardConflict.***
4. If $C_i.student_{number} > C_i.room.size$:
5.   Break ***H3 RoomSizeHardConflict.***
6. Doing join operation $[\text{course}_{\text{list}}, \text{course}_{\text{list}}] = \{C_1C_2, C_1C_3, \ldots, C_1C_N, \ldots, C_iC_j, \ldots, C_{N-1}C_N\}$
7. Do C_iC_j in $[\text{course}_{\text{list}}, \text{course}_{\text{list}}]$:
8.   If $C_i.timesolt = C_j.timesolt$ and $C_i.room = C_j.room$ and $C_i.id < C_j.id$:
9.   Break ***H1 RoomTimeHardConflict***.
10.   If $C_i.timesolt = C_j.timesolt$ and $C_i.faculty = C_j.faculty$ and $C_i.id < C_j.id$:
11.   Break ***H4 TeachingHardConflict.***
12.   If $C_i.timesolt = C_j.timesolt$ and $C_i.student_{group} = C_j.student_{group}$ and $C_i.id < C_j.id$:
13.   Break ***H5 StudentHardConflict.***

Algorithm 2: Pseudo-code for controlling soft constraints

1. Given the UCSP instance solution content course list $\text{course}_{\text{list=}}\{C_1, C_2, \ldots, C_i, \ldots, C_N\}$
2. If $C_i.prefer_{time}.dayofweek = C_i.timesolt.dayofweek$ or $C_i.prefer_{time}.start_{time}! = C_i.timesolt.start_{time}$:
3.   Break ***S2 FacultyTimePreference.***
4. If $C_i.department == C_i.room.department$:
5.   Satisfy ***S5 CourseDepartmentConsistency.***
6. Doing join operation $[\text{course}_{\text{list}}, \text{course}_{\text{list}}] = \{C_1C_2, C_1C_3, \ldots, C_1C_N, \ldots, C_iC_j, \ldots, C_{N-1}C_N\}$
7. Do C_iC_j in $[\text{course}_{\text{list}}, \text{course}_{\text{list}}]$:
8.   If $C_i.faculty = C_j.faculty$ and $C_i.id < C_j.id$:
9.   If $C_i.room! = C_j.room$:
10.   Break ***S1 FacultyRoomStability.***
11.   If $C_i.faculty = C_j.faculty$ and $C_i.timesolt.dayofweek = C_j.timesolt.dayofweek$:
12.   If $C_i.timesolt.end_{time} - C_j.timesolt.start_{time} < 12$ hours:
13.   Satisfy ***S3 FacultyTeachingMode.***
14.   If $C_i.subject = C_j.subject$ and $C_i.student_{group} = C_j.student_{group}$ and $C_i.course_{type} = C_j.course_{type}$ and $C_i.timesolt.dayofweek == C_j.timesolt.dayofweek$:
15.   If $C_i.timesolt.end_{time} - C_j.timesolt.start_{time} < 12$ hours:
16.   Break ***S4 StudentStudyingMode***.

2.3 The Proposed Method

This manuscript proposed a novel and effective parallel Tabu and SA structure to solve UCSP, as shown in **Fig. 1**. t consists of five steps: UCSP instance construction, giving initial solution, generating possible moves, Tabu+SA filters out unacceptable moves, and selecting the next move using an adaptive decision algorithm, respectively. Each step is explained in the subsequent subsections.

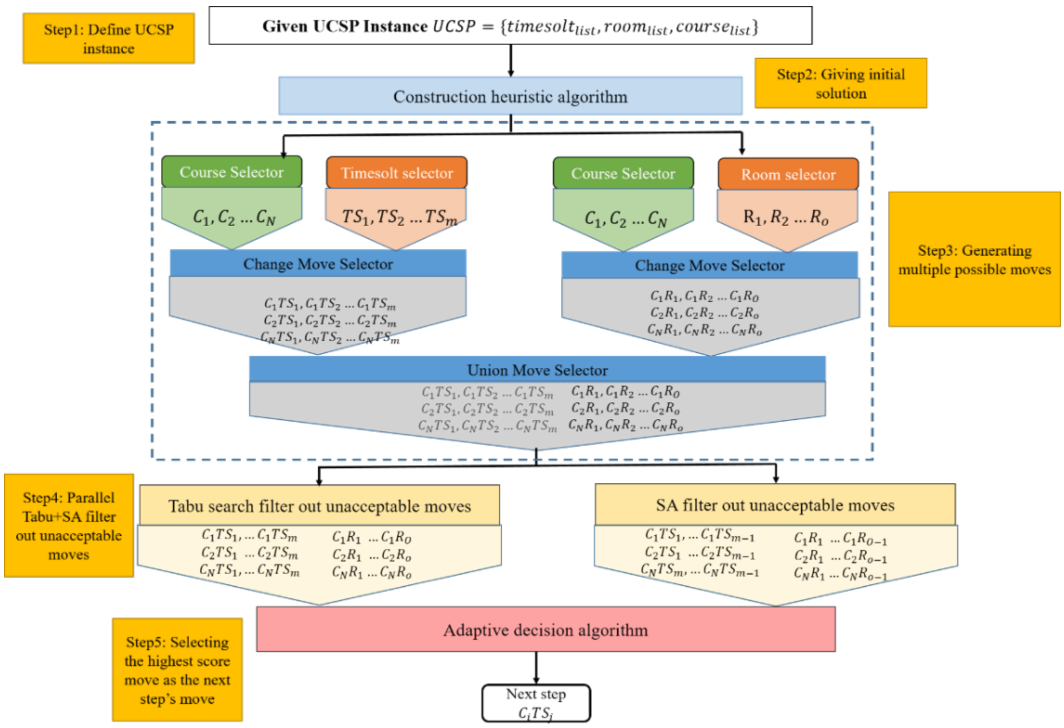


Fig. 1. The proposed parallel Tabu+SA for solving UCSP

2.2.1 UCSP Instance Construction

According to the definition of UCSP in Section 2, it generally consists of three parts: timeslot, room, and course that describe all related teaching. A detailed description of them is shown in Table 2. Depending on the university, each part contains unique samples to be stored in one list object ($Timesolt_{list}$, $Room_{list}$, $and\ Course_{list}$). We define one UCSP instance with M timeslots, O rooms, and N courses, as shown in Eq. (2).

$$UCSP_{instance} = \begin{cases} TS = \{TS_1, TS_2, \dots, TS_m, \dots, TS_M\} \\ R = \{R_1, R_2, \dots, R_o, \dots, R_O\} \\ \mathrm{C} = \{\mathrm{C}_1,\ \mathrm{C}_2 \dots, \mathrm{C}_\mathrm{n}, \dots, \mathrm{C}_\mathrm{N}\ \} \end{cases} \tag{2}$$

2.2.2 Generating Initial Solution

The constructed USCP instance is fed into one construction heuristic algorithm, the first fit algorithm [36], to generate the initial solution $solution_{in}$. The first fit arranges only one entity course C_n at once by its default order and casecondsnnot be assigned in the next round. The algorithm is terminated after all courses are assigned, which can save much time finding a better solution, even if it cannot ensure the optimal solution. It gives one fast available solution to guide other algorithms to find the better solution.

2.2.3 Generating Multiple Possible Moves

The moves defined in this manuscript are to construct a new UCSP neighbor solution of the original solution to keep the diversity of the solution to increase the performance. We especially defined a union of change move selectors to generate the possible moves for the next step. One is for course object $C = \{C_1, C_2 \ldots C_N\}$ and timeslot value $TS = \{TS_1, TS_2 \ldots TS_M\}$, another one is for object course and room value $R = \{R_1, R_2 \ldots R_o\}$. The principle of change move is automatically changing the value into another one. For example, from $C_1 TS_1$ to $C_1 TS_2$, the original solution assigns C_1 at TS_1 while the neighbor solution assigns it at TS_2. Each move selector will iterate all possible moves to the construction solution during the solving. After two move selections, one union move selector combines $C_n TS_m$ and $C_n R_o$ for the next step, increasing the solution's diversity in both timeslots and rooms to improve the performance.

Algorithm 3: Tabu search to filter out unacceptable moves

Input: Possible next steps solution $PS = \{C_1TS_1, C_1TS_2 \ldots C_1TS_N, \ldots, C_1R_1 \ldots C_1R_o\}$
Output: Acceptable next step solution $AP \in PS$.
Defined current solution i,

1. Randomly generate one solution $i \in PS$, set best solution $s = i$, Tabu list $H = \{\}$ with a 2% Entity Object length, move step $ms = 7$, and set iteration $k = 0$. Calculate the fitness score f(s) using Eq. (1).
2. While not stop:
3. $A = N(i, H)$ #generate the neighbors of the solution set A from H.
4. $i = \text{SelectionBestSolution}(A)$ #set the current solution i
5. Calculate the fitness score f(i) using Eq. (1).
6. Update the H considering Tabu list length and move step ms. # The Tabu list will be full as a length of 2% of Entity Object, and one Tabu object cannot be used before seven steps.
7. if $f(i) < f(s)$:
8. $f(s) = f(i)$
9. End if
10. $k = k + 1$
11. End While
12. $AP = s$

Algorithm 4: SA algorithm to filter out unacceptable moves

Input: Possible next steps solution $PS = \{C_1TS_1, C_1TS_2 \ldots C_1TS_N, \ldots, C_1R_1 \ldots C_1R_o\}$
Output: Acceptable next step solution $AP \in PS$.
Define starting temperature $T_s = 1000$, and the cooling ratio is 0.99.

1. Randomly generate one solution $i \in PS$, setting best solution $s = i$, and calculate its fitness score f(s) by Eq. (1).
2. While not stop
3. Generate a random neighbor s', and calculate $f(s')$ by Eq. (1)
4. $\Delta E = f(s) - f(s')$
5. If $\Delta E < 0$ then $s = s'$
6. Else accept s' with probability $e^{\frac{-\Delta E}{T_s}}$
7. Update $T_s = G(T_s)$
8. Until T_s<1
9. $AP = s$

2.2.4 Filter Out Unacceptable Moves

To select efficient moves, we designed a Parallel Tabu+SA structure to filter out unacceptable moves generated by step 3. Mainly, Tabu search is used to find USCP's global solution that can avoid falling into local optimal, and SA is used to find its robust solution by several steps. This good design makes the solution take into account both the advantages of Tabu and SA and keeps the diversity of choice to select the potential next step's move. A detailed explanation of Tabu and SA is shown in algorithms 3 and 4.

2.2.5 Selecting Next Move

Select the next move using one adaptive decision algorithm. The algorithm selects the move with the highest score. If Tabu and SA have similar score, they will randomly select one to execute. To complete one USCP instance, the proposed method will run iteratively until all courses satisfy all hard constraints and most soft constraints.

3. Experimental Verification

To verify the effectiveness of the proposed method for solving UCSP, we implement the proposed method based on the operating system of Windows 10 with intel(R) Core(TM) i7-7700 CPU @3.60GHz, and RAM 32.0 GB, on one real-world data set, the department of information systems at Pukyong National University (PKNU) and ten random UCSP instances. The programming language is Python 3.6, and the basic library is Optapy [37].

3.1 PKNU-UCSP Instance

The UCSP instance for the department of information systems at PKNU, for graduate students only, consists of eight rooms $R = \{R_1, R_2, .., R_8\}$, and two functional room types are considered: teaching room and practice room that consists of some computers. Where R_1 and R_4 are for practice rooms, and others are teaching rooms. The eight rooms' capacities are {40,40,20,25,30,20,10,40}; and the owners of these rooms are departments $\{D_1, D_4, D_3, D_2, D_2, D_2, D_2, D_4\}$. The courses only can be arranged from Monday to Friday. Each day has three courses starting from 9:00 am and ending at 21:30. That is, fifteen timeslots $TS = \{TS_1, TS_2, \dots, TS_{15}\}$ corresponding to 9:00-12:30, 13:00-16:30, and 18:30-21:30 for five days are taken into account, respectively. Besides, one week has 17 courses $C = \{C_1, C_2, C_3, \dots, C_{17}\}$ touched by six faculties $Pr = \{Pr_1, Pr_2, \dots, Pr_6\}$, where $C_7, C_8, C_9, C_{10}, C_{14}$ and C_{15} are practice courses and others are not; and the corresponding student groups for 17 courses are {17th, 18th, 17th, 19th, 20th, 17th, 19th, 21st, 20th, 18th, 19th, 20th, 22nd, 17th, 18th, 18th, 19th}, which indicated the students entrance year. More details of the $Course_{list}$ object are given in **Table 3**. Notice that the faculties' preferred time is a subset of their available time using a questionnaire.

Table 3. PKNU UCSP instance $\boldsymbol{Course_{list}}$ description.

Course	Faculty	Department	$Student_{group}$	$Student_{number}$	$Course_{type}$	$Prefer_{time}$
C_1	Pr_1	D_1	17th	20	Teaching	TS_1
C_2	Pr_2	D_2	18th	21	Teaching	TS_3
C_3	Pr_3	D_3	17th	34	Teaching	TS_2
C_4	Pr_4	D_1	19th	25	Teaching	TS_2
C_5	Pr_5	D_2	20th	25	Teaching	TS_{13}
C_6	Pr_1	D_1	17th	40	Teaching	TS_{14}
C_7	Pr_2	D_2	19th	20	Practice	TS_5
C_8	Pr_4	D_1	21st	40	Practice	TS_5

C_9	Pr_2	D_2	20^{th}	20	Practice	TS_9
C_{10}	Pr_6	D_4	18^{th}	20	Practice	TS_9
C_{11}	Pr_2	D_2	19^{th}	10	Practice	TS_5
C_{12}	Pr_2	D_2	20^{th}	25	Teaching	TS_5
C_{13}	Pr_6	D_4	22^{nd}	5	Teaching	TS_5
C_{14}	Pr_6	D_2	17^{th}	25	Practice	TS_3
C_{15}	Pr_5	D_2	18^{th}	20	Practice	TS_5
C_{16}	Pr_2	D_2	18^{th}	25	Teaching	TS_2
C_{17}	Pr_1	D_1	19^{th}	35	Teaching	TS_2

3.2 Solving PKNU UCSP Instance using The Proposed Method

The results use the proposed method for solving the PKNU-UCSP instance, as shown in **Table 4**. The results are shown with the format of the $Course(Faculty|Student_{group}|Room)$. The results showed that the proposed method satisfied all hard and soft constraints. It arranges all practice courses correctly and only uses three rooms, which reduces the management cost for cleaning the room, delivering keys, etc. For instance, one room was arranged once at one timeslot (H1). Assigning practice courses $C_7, C_8, C_9, C_{10}, C_{14}$ at rooms R_4, R_1, R_4, R_4, R_4 to satisfy (H2), accordingly; Assigning C_8 at R_1, not R_4, since applied C_8 students are 40 and the size of R_4 is 25, which satisfies (H3). Besides, all professors and students only take one course simultaneously (H4 and H5).

Table 4. Solving PKNU UCSP instance using the proposed method.

Day	Timeslot		
	9:00-12:30	13:00-16:30	18:30-21:30
Monday	$C_4(Pr_4\|19\text{th}\|\text{R}_1)$ $C_{14}(Pr_6\|17\text{th}\|\text{R}_4)$	$C_8(Pr_4\|21st\|\text{R}_1)$ $C_2(Pr_2\|18th\|\text{R}_4)$	$C_{11}(Pr_2\|19\text{th}\|\text{R}_4)$
Tuesday	$C_{13}(Pr_6\|22\text{nd}\|\text{R}_4)$	$C_{10}(Pr_6\|18\text{th}\|\text{R}_4)$	
Wednesday			
Thursday	$C_{16}(Pr_2\|18\text{th}\|\text{R}_4)$	$C_3(Pr_3\|17\text{th}\|\text{R}_1)$ $C_7(Pr_2\|19\text{th}\|\text{R}_4)$ $C_5(Pr_5\|20\text{th}\|\text{R}_5)$	$C_{15}(Pr_5\|18\text{th}\|\text{R}_4)$
Friday	$C_{17}(Pr_1\|19\text{th}\|\text{R}_1)$	$C_1(Pr_1\|17\text{th}\|\text{R}_1)$ $C_{12}(Pr_2\|20\text{th}\|\text{R}_4)$	$C_6(Pr_1\|17\text{th}\|\text{R}_1)$ $C_9(Pr_2\|20\text{th}\|\text{R}_4)$

Moreover, the solution satisfies the most soft constraints. For instance, faculties Pr_1, Pr_2, $and\ Pr_4$ like teaching in only one fixed room, R_1, which satisfies S1. Most courses are arranged according to the faculty's preferred time (S2). Each faculty took courses in a sequence mode (S3), while students who took courses are not (S4). Professor Pr_1 likes teaching on Friday in the same room, R_1, in a sequence mode, but the students are not. Besides, the course and room are arranged with the consistency of department, like C_4 and R_1 belonging to department D_1. However, some soft constraints have been broken out, such as Pr_5 teaches C_{15} at R_4 and C_5 at R_5 (S1).

3.3 Comparative Analysis

To validate the effectiveness of the proposed method and explore each component's effectiveness for UCSP, we designed and compared the proposed method with other optimization algorithms, including Tabu search algorithm [19], SA [20], and Tabu+SA without first fit construction heuristic algorithm (Proposed$_{wo}$).

The scheduling scores, defined in Eq. (1), showed that all methods could solve the PKNU-UCSP instance since they do not break out any hard constraints. However, the proposed parallel Tabu+SA performs the best, obtaining the highest soft constraints score (4). All methods broke once for S1 and received the same score of 25 for S2, except SA is 24. For S3 and S4, all methods broke out 17 or 18 times, and the proposed method outperforms the others with a total score of 4. The performance of each comparative method is ranked as The proposed>Tabu>Proposedwo>SA.

From the findings, we can conclude that the usage of Tabu has improved by three scores, and SA has improved by one score, which is conducted by comparing the proposed method with SA and Tabu alone accordingly. Besides, the first fit could give the initial solution to help the proposed method meet S5.

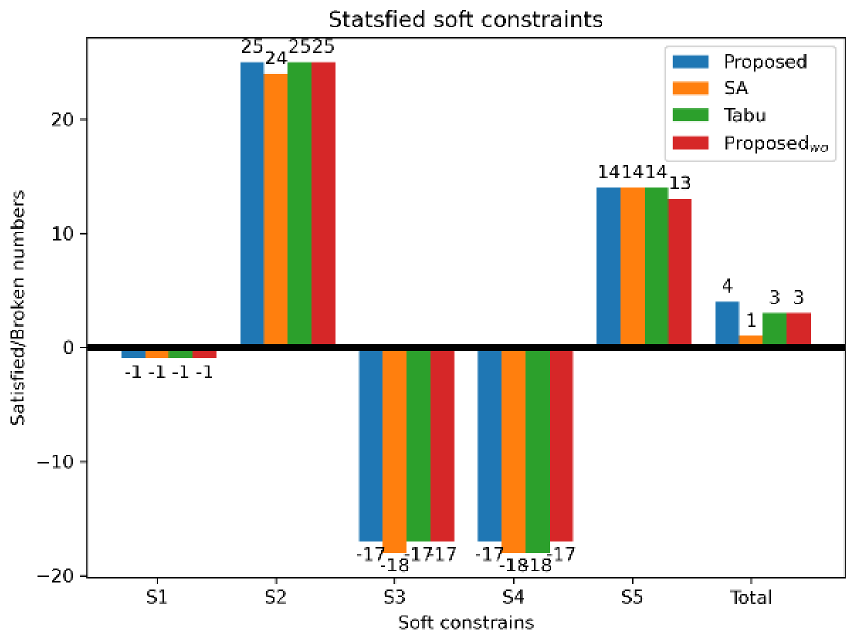


Fig. 2. The comparative results in terms of satisfied soft constraints

3.4 The Effectiveness of Parallel Structure Design

The proposed method adopts one parallel Tabu+SA structure to keep the diversity of the moves. To validate its effectiveness, we compare the proposed method with the cascade Tabu and SA, which is motivated by [16]. The results use scheduling scores, as shown in **Fig. 3**. The proposed method has a parallel structure, while Proposed$_{cascade}$ has a cascade structure. The settings of the Proposed$_{cascade}$ are identical to the proposed method except for the structure. The results showed that the proposed method outperforms the cascade structure, which wins one score in S1 soft constraint, and others are identical. Significantly, the proposed method only breaks out once for S1 while the Proposed$_{cascade}$ breaks out twice, which causes more inconvenience for the faculties.

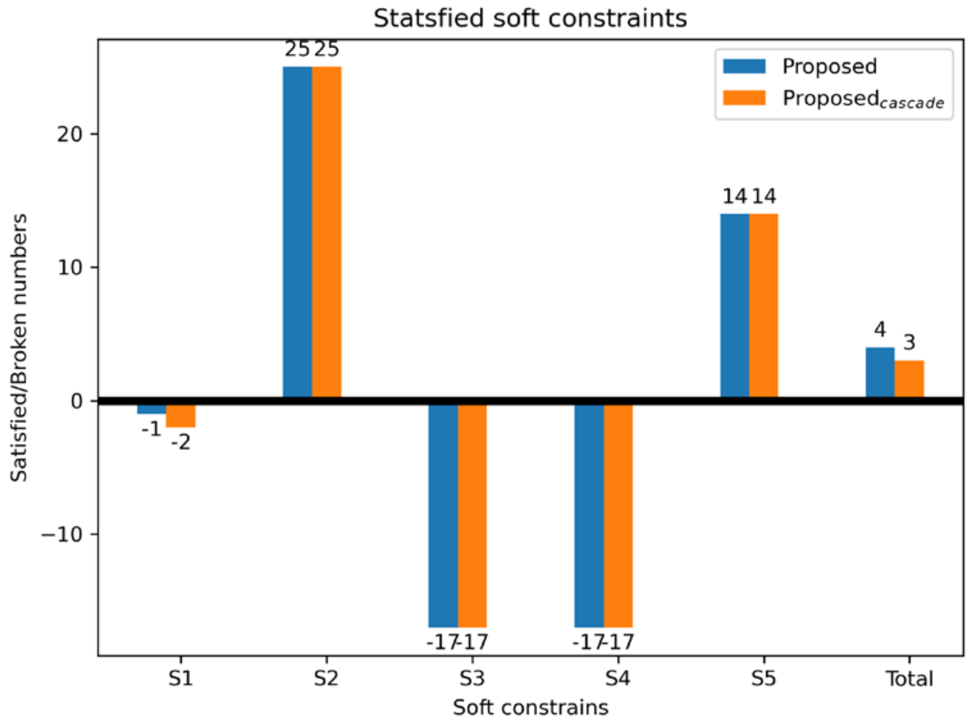


Fig. 3. The effectiveness of parallel Tabu+SA structure.

Moreover, we give the scheduling results of the cascade Tabu+SA structure to show the difference, as shown in **Table 5**. Compared to the proposed method using parallel structure, the cascade structure requires four rooms $\{R_1, R_2, R4, R_8\}$ while the parallel structure only requires three rooms $\{R_1, R_4, R_5\}$, which reduces maintenance cost. Besides, the parallel structure arranges all courses for faculty Pr_1 on Friday, which satisfies the S1. However, the cascade structure arranges them on Monday and Friday. These findings confirmed the effectiveness of the proposed parallel structure. In addition, the results showed that the parallel structure solves the PKNU-UCSP instance using 100.68 seconds while the cascade structure spent 106.73 seconds, which proved that the proposed parallel structure is faster.

Table 5. Solving PKNU-UCSP instance using the Propsoed$_{\text{cascade}}$.

Day	Timeslot		
	9:00-12:30	13:00-16:30	18:00-21:30
Monday	$C_{17}(\text{Pr}_1\|19th\|R_1)$ $C_2(\text{Pr}_2\|18th\|R_4)$	$C_6(\text{Pr}_1\|17th\|R_1)$ $C_7(\text{Pr}_2\|19th\|R_4)$	
Tuesday			
Wednesday		$C_8(\text{Pr}_4\|21st\|R_1)$ $C_{16}(\text{Pr}_2\|18th\|R_4)$	$C_4(\text{Pr}_4\|19th\|R_1)$ $C_3(\text{Pr}_3\|17th\|R_2)$
Thursday	$C_{14}(\text{Pr}_6\|17th\|R_1)$ $C_{15}(\text{Pr}_5\|18th\|R_4)$	$C_{10}(\text{Pr}_6\|18th\|R_1)$ $C_5(\text{Pr}_5\|20th\|R_4)$	$C_{13}(\text{Pr}_6\|22nd\|R_8)$
Friday	$C_{12}(\text{Pr}_2\|20th\|R_4)$	$C_1(\text{Pr}_1\|17th\|R_1)$ $C_{11}(\text{Pr}_2\|19th\|R_4)$	$C_{16}(\text{Pr}_2\|18th\|R_4)$ $C_9(\text{Pr}_2\|20th\|R_4)$

3.5 The Effectiveness of Each Soft Constrain Weight

The fitness score given in Eq. (1) for the proposed method sets all soft constraints as the same impact by default, that is, $a_1 = a_2 = a_3 = a_4 = a_5 = 1/-1$. We designed some subexperiments to explore its effectiveness. Significantly, the weights for each soft constraint are set from 1 to 5, ignoring the positive or negative, causing the case of satisfying or breaking decide it. The total score of each setting is given in **Table 6**. The x-axis sets all weights the same from 1 to 5, while the y-axis sets different weights for each soft constraint. The results with the format of $Nscore(S_{S1}, S_{S2}, S_{S3}, S_{S4}, S_{S5})$, where Nscore is a total normalized score calculated using Eq. (3), one related score element-wise calculation. The $Weights_{vector}$ is one five-tuple vector that records the corresponding weights. For instance, the values of $Weights_{vector}$ at a_3=2 and $a_1 = a_2 = a_4 = a_5 = 2$ equals (2,2,2,2,2), and its Nscore = $\frac{-2}{2} + \frac{50}{2} + \frac{-34}{2} + \frac{-34}{2} + \frac{28}{2}$=4.

Table 6. The effectiveness of each soft constrain weight ('-' indicated did not break out the constrain)

Weight	1	2	3	4	5
a_1=1	4(-1,25,-17,-17,14)	4(-3,52,-34,-34,30)	5(-2,78,-51,-51,45)	6(-2,108,-68,-68,60)	6(-2,135,-85,-85,75)
a_2=2	-25.5(-1,25,-34,-17,14)	4(-2,50,-34,-34,28)	5(-3,75,-34,-51,42)	38(-4,100,-34,-68,56)	51.7(-5,125,-34,-85,70)
a_3=3	**66.33(-3,78,-17,-17,14)**	24.17(-2,81,-34,-34,26)	4(-3,75,-51,-51,42)	-6.17(-,78,-68,-68,56)	-15.73(-,78,-85,-85,70)
a_4=4	4(-1,25,-17,-68,14)	4(-2,50,-34,-68,28)	4(-3,75,-51,-68,42)	4(-4,100,-68,-68,56)	4(-5,125,-85,-68,70)
a_5=5	4(-2,25,-17,-17,75)	-13(-6,52,-34,-34,75)	6(-,78,-51,-51,70)	9.5(-,104,-68,-68,70)	4(-5,125,-85,-85,70)

$$Nscore = sum(\frac{(S_{S1}, S_{S2}, S_{S3}, S_{S4}, S_{S5})}{Weights_{vector}}) \quad (3)$$

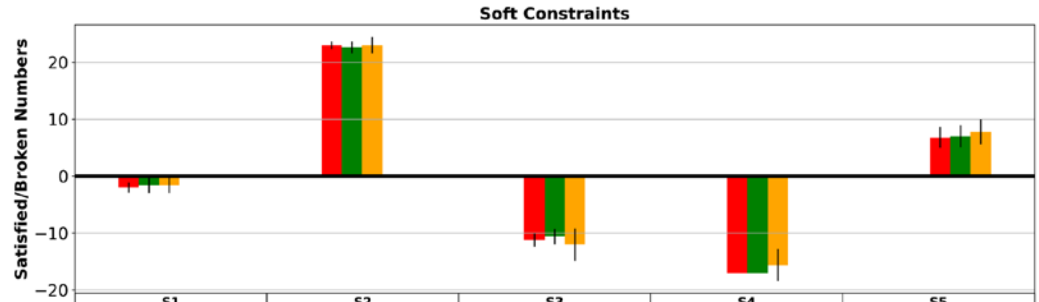
The findings indicated that the proposed method is sensitive to soft constrain weights. Setting the rate a_1: a_2: a_3: a_4: a_5 with 3:1:1:1:1 receives the highest score of 66.33. Besides, setting them in one same ratio performs the same, which receives four scores. Some cases, such as a_2=2 and others are 1, a_3=3, and others are 4 or 5, a_5=5, and others are 2, receive negative scores that indicate they break out most soft constraints. However, it just takes a small ratio, four out of 25 cases in total, which shows that the proposed method is easy to adjust. The results suggested that we set a relatively big weight (3) for S3 and tiny weights (1) for others that could receive the best results. The best timetabling results for the PKNU-UCSP instance are given in **Table 7**. The most significant difference between **Table 7** and **Table 4** is that the faculties in the former teach the course in a remarkably continuous mode.

Table 7. Solving PKNU UCSP instance using the proposed method with the best parameters

Day	Timeslot		
	9:00-12:30	13:00-16:30	18:00-21:30
Monday		$C_4(\mathrm{Pr}_4\|19th\|R_1)$ $C_5(\mathrm{Pr}_5\|20th\|R_4)$	$C_8(\mathrm{Pr}_4\|21st\|R_1)$ $C_{15}(\mathrm{Pr}_5\|18th\|R_4)$
Tuesday	$C_{17}(\mathrm{Pr}_1\|19th\|R_1)$ $C_3(\mathrm{Pr}_3\|17th\|R_2)$	$C_6(\mathrm{Pr}_1\|17th\|R_1)$	$C_1(\mathrm{Pr}_1\|17th\|R_1)$
Wednesday	$C_{16}(\mathrm{Pr}_2\|18th\|R_4)$		
Thursday	$C_7(\mathrm{Pr}_2\|19th\|R_4)$	$C_9(\mathrm{Pr}_2\|20th\|R_4)$	$C_2(\mathrm{Pr}_2\|18th\|R_4)$
Friday	$C_{14}(\mathrm{Pr}_6\|17th\|R_4)$	$C_{12}(\mathrm{Pr}_2\|20th\|R_4)$ $C_{13}(\mathrm{Pr}_6\|22nd\|R_5)$	$C_{10}(\mathrm{Pr}_6\|18th\|R_1)$ $C_{11}(\mathrm{Pr}_2\|19th\|R_4)$

3.6 The Generality of the Proposed Method

To validate the proposed method's generality for solving UCSP, we randomly generate 10 UCSP instances based on the PKNU-UCSP instance, described in Table 3 but the element (course, timeslot, and room) will be randomly matched, the other configurations are same to the original PKNU data set. The reason we did not compare the proposed method on other public data sets is they have different constraints. It is hard to compare fairly without their source codes. We generated 10 UCSP instances randomly but five times broke the hard constraints for the room size caused by random matching (the room limitations). The successful five-time average results that broke or satisfied soft constraints are shown in **Fig. 4**. The findings indicated that the proposed method has a significant priority for solving UCSP. Especially, it won four cases out of five constraints: S1, S2, S4, and S5 with average satisfied/broken numbers of {-1.6,23.0,-15.60,7.80}, respectively. For the S3, the Tabu search algorithm performs slightly better than the proposed method. Besides, we calculated the normalized score for each UCSP instance to see their differences, as shown in **Fig. 5**. The results indicated that the proposed method performs the best for all five instances. Especially for the fourth instance, the proposed method receives a score of 6 while the other two are 2. The average normalized score for five UCSP instances showed that the proposed method received the highest score of 1.6 while SA and Tabu received scores of -0.4 and 0.4, respectively. These findings confirmed the generality of the proposed method for solving UCSP.



	S1	S2	S3	S4	S5
SA	-2.00±0.89	23.00±0.63	-11.20±1.17	-17.00±0.00	6.80±1.83
Tabu	-1.60±1.36	22.60±1.02	-10.60±1.36	-17.00±0.00	7.00±1.90
Proposed	-1.60±1.36	23.00±1.41	-12.00±2.83	-15.60±2.80	7.80±2.23

Fig. 4. The generality test based on ten random UCSP instances.

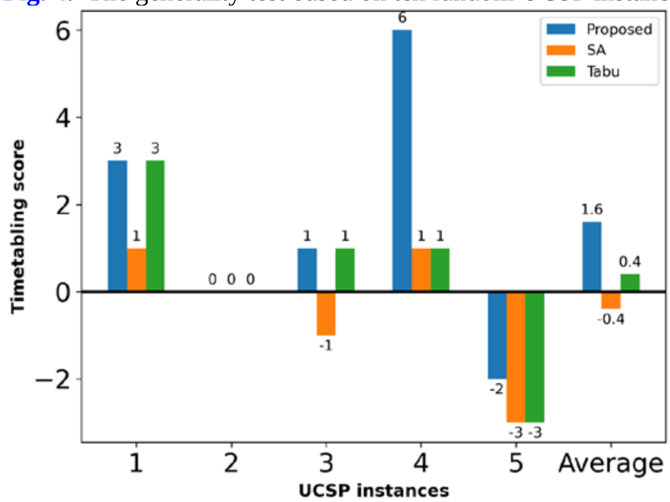


Fig. 5. The average timetabling score for each UCSP instance.

4. Conclusions

This paper presents one novel and practical approach, parallel Tabu+SA, to solve UCSP. Firstly, it utilizes the first fit algorithm to give the initial solutions, and then one union change move selector is designed to increase the diversity of potential moves. The Tabu search and SA algorithms are designed in parallel to select the next move with the highest score. This sound design fully uses Tabu search and SA to help the proposed method increase accuracy. Besides, an adaptive decision selection algorithm is designed to select the next move from Tabu and SA solutions automatically. The experimental verification based on the PKNU real case and ten random UCSP instances verified the proposed method's effectiveness. Moreover, one ablation study confirmed each component's effectiveness in the proposed method.

In summary, the proposed method could solve USCP and easily transfer to other scheduling problems. Besides, the proposed normalized score calculation could be utilized to compare other optimization problems more fairly. In the future, we will test the proposed method's effectiveness on other optimization problems and develop one editable web application for easy use. Also, we will compare the proposed method with more leading algorithms to validate its effectiveness.

Acknowledgement

Thanks to the officer at the department of the information system of PKNU, who assisted us in collecting the data PKNU-UCSP to test the proposed method's generality. The data can be requested from the corresponding author on a reasonable basis.

References

[1] R. Tian, H. Si, Z. Guo, X. Zhao, and Y. Feng, "Realization of course scheduling system based on improved genetic algorithm," in *Proc. of ICCSE 2021 - IEEE 16th Int. Conf. Comput. Sci. Educ.*, pp. 1072–1076, 2021. Article (CrossRef Link)

[2] 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, 2008. Article (CrossRef Link)

[3] E. K. Burke and S. Petrovic, "Recent research directions in automated timetabling," *Eur. J. Oper. Res.*, vol. 140, no. 2, pp. 266–280, 2002. Article (CrossRef Link)

[4] S. 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 Syst. Appl.*, vol. 127, pp. 9–24, 2019. Article (CrossRef Link)

[5] M. Azmi Al-Betar and A. Tajudin Khader, "A hybrid harmony search for university course timetabling," in *Proc. of Multidiscip. y Int. Conf. Sched. Theory Appl.*, no. August, pp. 10–12, 2009. Article (CrossRef Link)

[6] T. Gabel and M. Riedmiller, "Adaptive Reactive Job-Shop Scheduling With Reinforcement Learning Agents," *Int. J. Inf. Technol. Intell. Comput.*, vol. 24, no. 4, 2008. Article (CrossRef Link)

[7] X. Shao and C. S. Kim, "Self-Supervised Long-Short Term Memory Network for Solving Complex Job Shop Scheduling Problem," *KSII Trans. Internet Inf. Syst.*, vol. 15, no. 8, pp. 2993–3010, 2021. Article (CrossRef Link)

[8] X. Shao and C. S. Kim, "An Adaptive Job Shop Scheduler Using Multilevel Convolutional Neural Network and Iterative Local Search," *IEEE Access*, vol. 10, pp. 88079–88092, 2022. Article (CrossRef Link)

[9] Y. Lan, Q. Zhai, X. Liu, and X. Guan, "Fast Stochastic Dual Dynamic Programming for Economic Dispatch in Distribution Systems," *IEEE Trans. Power Syst.*, vol. 38, no. 4, pp. 3828–3840, 2022. Article (CrossRef Link)
[10] G. Laporte, H. Mercure, and Y. Nobert, "Generalized travelling salesman problem through n sets of nodes: the asymmetrical case," *Discret. Appl. Math.*, vol. 18, no. 2, pp. 185–197, 1987. Article (CrossRef Link)
[11] S. Abbas, F. Ashraf, F. Jarad, M. S. Sardar, and I. Siddique, "A Drone-Based Blood Donation Approach Using an Ant Colony Optimization Algorithm," *C. - Comput. Model. Eng. Sci.*, vol. 136, no. 2, pp. 1917–1930, 2023. Article (CrossRef Link)
[12] A. Bashab et al., "Optimization Techniques in University Timetabling Problem: Constraints, Methodologies, Benchmarks, and Open Issues," *Comput. Mater. Contin.*, vol. 74, no. 3, pp. 6461–6484, 2023. Article (CrossRef Link)
[13] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the Flexible Job-shop Scheduling Problem," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3202–3212, 2008. Article (CrossRef Link)
[14] K. Dehghan-Sanej, M. Eghbali-Zarch, R. Tavakkoli-Moghaddam, S. M. Sajadi, and S. J. Sadjadi, "Solving a new robust reverse job shop scheduling problem by meta-heuristic algorithms," *Eng. Appl. Artif. Intell.*, vol. 101, p. 104207, 2021. Article (CrossRef Link)
[15] M. C. Gomes, A. P. Barbosa-Póvoa, and A. Q. Novais, "Optimal scheduling for flexible job shop operation," *Int. J. Prod. Res.*, vol. 43, no. 11, pp. 2323–2353, 2005. Article (CrossRef Link)
[16] X. Li and L. Gao, "An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem," *Int. J. Prod. Econ.*, vol. 174, pp. 93–110, 2016. Article (CrossRef Link)
[17] G. R. Weckman, C. V. Ganduri, and D. A. Koonce, "A neural network job-shop scheduler," *J. Intell. Manuf.*, vol. 19, no. 2, pp. 191–201, 2008. Article (CrossRef Link)
[18] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, pp. 8091-8126, 2021. Article (CrossRef Link)
[19] G. Fred, "Tabu Search: A Tutorial," *Interfaces (Providence)*, vol. 20, no. 4, pp. 74–94, 1990. Article (CrossRef Link)
[20] B. Dimitirs and S. John, "Simulated annealing," *Stat. Sci.*, vol. 8, no. 1, pp. 10–15, 1993. Article (CrossRef Link)
[21] L. Asadzadeh, "A local search genetic algorithm for the job shop scheduling problem with intelligent agents," *Comput. Ind. Eng.*, vol. 85, pp. 376–383, 2015. Article (CrossRef Link)
[22] M. Kurdi, "An effective new island model genetic algorithm for job shop scheduling problem," *Comput. Oper. Res.*, vol. 67, pp. 132–142, 2016. Article (CrossRef Link)
[23] B. Xu, P. A. Zhong, Y. F. Zhao, Y. Z. Zhu, and G. Q. Zhang, "Comparison between dynamic programming and genetic algorithm for hydro unit economic load dispatch," *Water Sci. Eng.*, vol. 7, no. 4, pp. 420–432, 2014. Article (CrossRef Link)
[24] L. Gao, C. Peng, C. Zhou, and P. Li, "Solving flexible job-shop scheduling problem using general particle swarm optimization," in *Proc. of 36th Int. Conf. Comput. Ind. Eng. ICC IE 2006*, pp. 3018–3027, 2006. Article (CrossRef Link)
[25] W. Wisittipanich and V. Kachitvichyanukul, "Differential Evolution Algorithm for Job Shop Scheduling Problem," *Ind. Eng. Manag. Syst.*, vol. 10, no. 3, pp. 203–208, 2011. Article (CrossRef Link)
[26] H. W. Ge, L. Sun, Y. C. Liang, and F. Qian, "An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling," *IEEE Trans. Syst. Man, Cybern. Part ASystems Humans*, vol. 38, no. 2, pp. 358–368, 2008. Article (CrossRef Link)
[27] S. Abdullah and H. Turabieh, "Generating university course timetable using Genetic Algorithms and local search," in *Proc. of 3rd Int. Conf. Converg. Hybrid Inf. Technol. ICCIT 2008*, vol. 1, pp. 254–260, 2008. Article (CrossRef Link)
[28] F. H. Awad, A. Al-Kubaisi, and M. Mahmood, "Large-scale timetabling problems with adaptive tabu search," *J. Intell. Syst.*, vol. 31, no. 1, pp. 168–176, 2022. Article (CrossRef Link)
[29] N. Leite, F. Melício, and A. C. Rosa, "A fast simulated annealing algorithm for the examination timetabling problem," *Expert Syst. Appl.*, vol. 122, pp. 137–151, 2019. Article (CrossRef Link)

[30] 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, 2013. Article (CrossRef Link)
[31] P. Kenekayoro, "Incorporating Machine Learning to Evaluate Solutions to the University Course Timetabling Problem," *arXiv:2010.00826*, 2020. Article (CrossRef Link)
[32] Turabieh, H., Abdullah, S., McCollum, B., McMullan, P., "Fish Swarm Intelligent Algorithm for the Course Timetabling Problem," in *Proc. of Rough Set and Knowledge Technology, RSKT 2010*, pp. 588-595, 2010. Article (CrossRef Link)
[33] Y. Nagata, "Random partial neighborhood search for the post-enrollment course timetabling problem," *Comput. Oper. Res.*, vol. 90, pp. 84–96, 2018. Article (CrossRef Link)
[34] M. C. Chen, S. N. Sze, S. L. Goh, N. R. Sabar, and G. Kendall, "A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities," *IEEE Access*, vol. 9, pp. 106515–106529, 2021. Article (CrossRef Link)
[35] S. Ceschia, L. Di Gaspero, and A. Schaerf, "Educational timetabling: Problems, benchmarks, and state-of-the-art results," *Eur. J. Oper. Res.*, vol. 308, no. 1, pp. 1–18, 2023. Article (CrossRef Link)
[36] A. A. Bertossi, L. V Mancini, and F. Rossini, "Fault-Tolerant Rate-Monotonic First-Fit Scheduling in Hard-Real-Time Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 9, pp. 934–945, 1999. Article (CrossRef Link)
[37] https://github.com/optapy/optapy.

Xiaorui Shao received his Ph. D degree in the department of information system from Pukyong National University in 2022. His research interest includes the fault diagnosis, deep learning, and job shop scheduling.

Su Yeon Lee She received her doctoral degree from the Department of Information Systems Engineering at Pukyong National University in 2024. In 2016, she took office as CEO of Jeongrok Co., Ltd. and is still working hard. Her current research interests include plasma, environment, big data, and AI.

Chang-Soo Kim received Ph. D. degree in the Department of Computer Engineering from Chung Ang University in 1991, Seoul, Korea. He became a professor in the Department of IT Convergence and Application Engineering, Pukyong National University, Pusan, South Korea, in 1992 and has continued until the present. He has been the Vice President of Korea Multimedia Society since 2011. His current research interests include scheduling of smart factories, big data simulation.