DOI: https://doi.org/10.33103/uot.ijccce.21.4.10

Timetabling Problem Solving Based on Improved Meerkat Clan Algorithm (IMCA)

Mohammed A. Jebur[1], Hasanen S. Abdullah[2]
[1,2]Department of Computer Science, University of Technology, Baghdad, Iraq
[1]cs.19.10@grad.uotechnology.edu.iq, [2]hasanen.s.abdullah@uotechnology.edu.iq

Abstract— *The university courses timetabling problem (UCTP) is a big topic among academics and institutions since it occurs every academic year. In general, UCTP is the distribution of events across slots time for each room based on a list of restrictions provided in one semester, such as (hard constraint and soft constraint), with the objective of avoiding conflicts in such assignments. Hard constraints should never be breached when striving to satisfy as many soft constraints as possible. There are many different methods used in automating the problems of the university timetabling course in higher education institutions. This paper presents an improved algorithm for the Meerkat Clan to solve the UCTP. This is done by studying the behavior of the Meerkat Clan Algorithm and Specifying the points that are able to improve without changing the main behavior of the Meerkat Clan Algorithm. And by testing with four datasets of different sizes, good results were obtained by optimizing this algorithm.*

Index Terms— *University Courses Timetabling Problem UCTP, Meerkat Clan Algorithm MCA, Meta-Heuristics Search.*

I. INTRODUCTION

It requires proper coordination and communication between several groups to prepare the schedule and timetable in universities as it is a multiphase process. Mostly, planning groups often consist of education administrators, department heads, teachers, and students [1]. What is challenging for educational institutions is including a set of events e.g., training courses that have to be allocated to specific rooms and time periods while adhering to a set of constraints [2]. Constraints vary in the university course timetabling problem (UCTP) that determent by the educational institution [3]. Some of these constraints, which are called hard constraints, must necessarily be met. Another class of constraints is soft constraints to assist improve schedule quality [4]. As a result, the best way to find the best solution is to follow the soft constraints. There are unique hard and soft constraints that most institutions define based on their facilities and resources. Previously, university schedules were manually formed. Manually setting up a course timetable is a complex and time immoderate task. Often it is not able to establish a timetable without conflict even after several iterations of repair. UCTP has been proved as an NP-hard [5]. The variety of requirements of different institutions increases the complexity of UCTP. That, suggesting different new methods to process the problem for its importance [6,7]. A well-known problem in the educational institution is the UCTP and is considered a classic in the area of optimization problems. The objective of the problem is to match the number of activities, such as courses, with the number of timeslots and rooms available. This UCTP is summarized by creating a number of preset time slots that are scheduled for a group of

Received 11/8/2021; Accepted 27l8l2021

DOI: https://doi.org/10.33103/uot.ijccce.21.4.10

events, a group-rooms that events can happen, a group-students present at each event, and a group-constraints that must meet [5]. This means a group of Ne events E {e1, e2, …, eNe} arranged in a group of 20 timeslots T = {t1, t2, …, t20} (There are five days in a week and four timeslots per day.), and a group of Nr rooms R {r1, r2, …, rNr}, in which events can happen.

This study only used one type of constraint: the Hard-Constraint, according to previous research [8]. A hard constraint is a requirement that must be met. Soft-Constraints may be considered, but they are not taken into account due to the scope of this study. The problem is considered solved when all events are given a time slot while not violating any of the hard constraints. When all of the following hard constraints are met, a solution (timetable) is considered acceptable:

• Each event in each course is assigned a time slot.
• No student group can hold two events at once.
• No lecturer should be conducting two events at the same time.
• The proper room is used for all activities.
• No two events in the same room happen at the same time.
• No event is held in a room that is smaller than the expected number of students.

In this paper, the improved MCA is used to solve UCTP. The rest of the paper is divided into six sections, as follows: The related work is shown in Section 2. The MCA is shown in Section 3. The IMCA is shown in Section 4. Section 5 shows the results of the testing as well as how it was carried out. Finally, but certainly not least. Section 6 discusses the conclusion and offers suggestions for future research.

II. RELATED WORKS

Studies have included a lot of evolution-based optimization methods, the Genetic Algorithm (GA) is highly common and many strategies based on GA have been tested for University Courses Timetabling Problem (UCTP). To solve the UCTP, A hybrid local search algorithm was used by Goh et al. [9] that combined Tabu Search (TS) and Simulated Annealing Reheating (SAR). For solving the UCTP, Chen et al. [10] devised a controlled randomization TS algorithm. To accommodate later algorithm creation, a random acceptance technique with a threshold mechanism is proposed. To address the disadvantages of metaheuristic optimization approaches, one of these problems is non-deterministic polynomials. Muklason et al. [11] presented Tabu-Variable Neighborhood Search. Instead of solving the problem directly, their solution used heuristics at each decision point. Goh et al. [12] used a highly tuned Simulated Annealing (SA) and reported superior results. Mazlan et al. [13] have used Ant Colony Optimization (ACO) lately. Their approach has produced more reliable findings. Sarin and Wang [14] presented an integer model for a UCTP with the aim of minimizing the traveled distance between classes by the lecturers. Then, they employed the problem for a university in Virginia Tech and solved it by applying Bender’s partitioning. In this article, we try to improve the Meerkat Clan Algorithm (MCA) for solving the UCTP.

III. MEERKAT CLAN ALGORITHM (MCA)

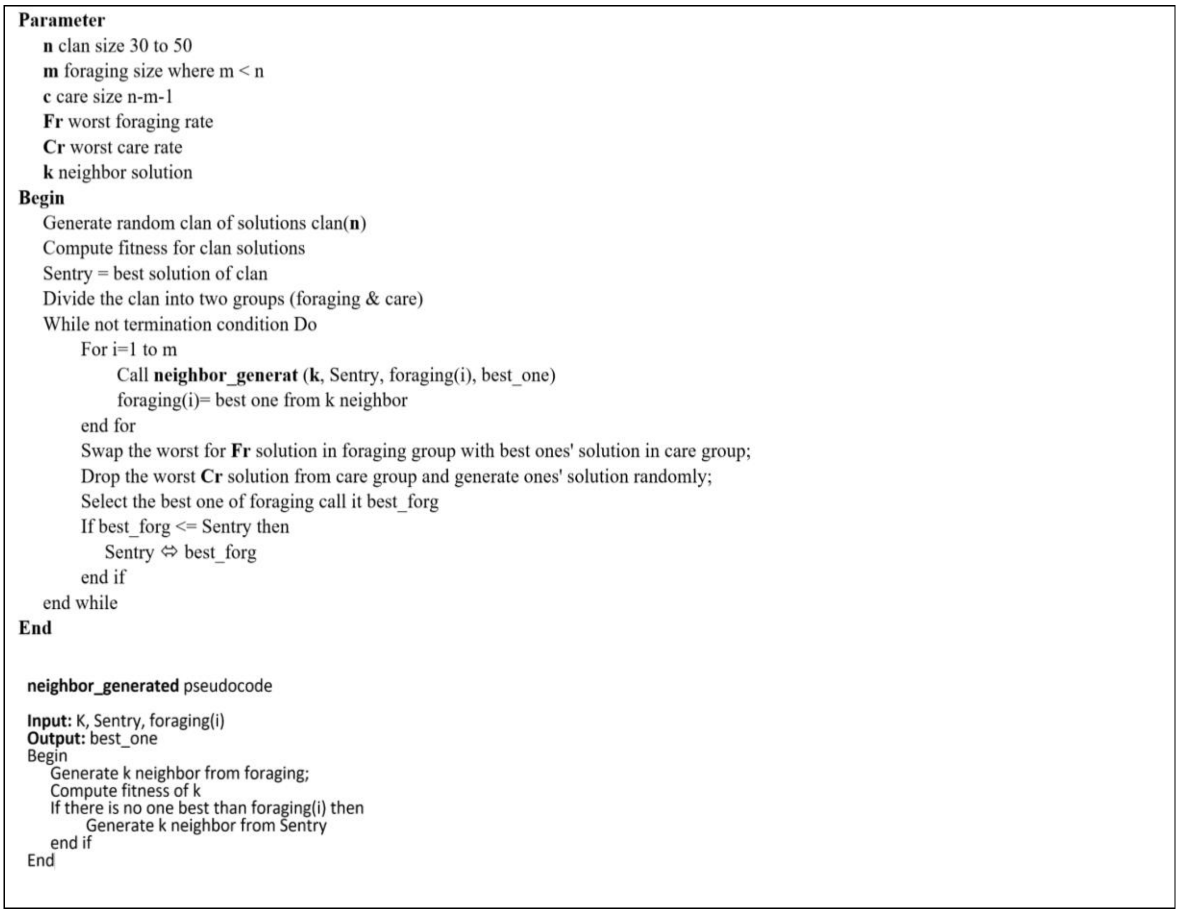
This algorithm is inspired by the behavior of meerkats. Meerkats live socially together in colonies of 5 to 30 individuals. They share parental and caring duties as social beings. Each clan has a leader who can be male or female. Where a clan has its own territory, they move through sometimes if no food is found or when a more ruthless clan forces them. In

Received 11/8/2021; Accepted 27l8l2021

DOI: https://doi.org/10.33103/uot.ijccce.21.4.10

the event that a tougher clan forces them, weaker clans will try to grow in some other manner or stay until they get stronger and regain their lost vulnerabilities. There is a so-called 'sentry' for every clan which means who is guarding a clan and when to detect danger and inform other members if there is danger. The task of the sentry is watching from (the ground, climbing a tree, the bush). The sentry makes a loud beep when it notices danger, then the clan will quickly move into their hiding holes. These being the general stages for MCA; they can be modified based on the type of problem [15].

1- Initial: Set the parameters for size of clan, size of foraging, size of care, and rate of worst foraging and rate of care by generating a clan of individuals at random.
2- Calculate the clan's fitness.
3- Like a 'sentry' I chose the best one.
4- Split up into two parts (foraging & care) from the clan.
5- For a foraging group, create neighbors.
6- Select the worst members of the foraging group and replace them with the best members of the care group.
7- Remove the worst members of the care group and generate a new one at random.
8- Supplant the best forager with a sentry if the best option.
Fig. 1 shows the pseudocode MCA.



```
Parameter
    n clan size 30 to 50
    m foraging size where m < n
    c care size n-m-1
    Fr worst foraging rate
    Cr worst care rate
    k neighbor solution
Begin
    Generate random clan of solutions clan(n)
    Compute fitness for clan solutions
    Sentry = best solution of clan
    Divide the clan into two groups (foraging & care)
    While not termination condition Do
        For i=1 to m
            Call neighbor_generat (k, Sentry, foraging(i), best_one)
            foraging(i)= best one from k neighbor
        end for
        Swap the worst for Fr solution in foraging group with best ones' solution in care group;
        Drop the worst Cr solution from care group and generate ones' solution randomly;
        Select the best one of foraging call it best_forg
        If best_forg <= Sentry then
            Sentry ⇔ best_forg
        end if
    end while
End

neighbor_generated pseudocode

Input: K, Sentry, foraging(i)
Output: best_one
Begin
    Generate k neighbor from foraging;
    Compute fitness of k
    If there is no one best than foraging(i) then
        Generate k neighbor from Sentry
    end if
End
```

FIG. 1. PSEUDOCODE MCA [15].

Received 11/8/2021; Accepted 27l8l2021

DOI: https://doi.org/10.33103/uot.ijccce.21.4.10

This MCA contains many parameters that affect the performance of the algorithm. The first parameter is the size of the clan (n), it represents the number of solutions in the clan or population size of the problem. The performance of the algorithm is influenced by the parameter (n), especially when it is set to a large value, which causes the algorithm to run slowly. It is preferable to choose (n) between 30 and 50 solutions. It does not affair if you choose (n) more than 50 in special cases depending on the needs of the problem. As for what is a parameter for the size of the foraging group (m). The number of solutions from the clan is the parameter (m). It is necessary to make an informed decision because it represents the clan's best options. A large value causes some bad solutions to be added to the foraging group, potentially causing the algorithm to reach a delayed best solution. As a result, (m) must be carefully chosen. The size of the care group is a parameter (c). After division, this is the number of rest solutions in the clan. In the care group, the best options are switched for the bad ones in the foraging group, also the worst options are discarded. The care group is created and random options are added. The worst foraging rate (Fr) is a parameter that represents the percentage of foraging solutions that were swapped with the best solutions in the care group. This rate has an impact in terms of performance, particularly when the foraging group has a lot of good options. The worst care rate (Cr) is a parameter. In the care group, the rate of worst solutions decreased. The crucial parameter is (K), which indicates the total number of generated neighbors. This parameter allows the algorithm to choose the next step with more variety. Height diversity is not always appropriate for some problems, so it must be carefully chosen.

IV. IMPROVED MEERKAT CLAN ALGORITHM (IMCA)

The new solutions generative resulting from the current solution are divided into two classes of solution. The first class has desired values that are replaced by the current solution. The second class has undesirable values that are compared to previous solutions and replaced. IMCA relies on exploiting the second class of new solutions, which can be replaced by some of the current worst solutions in the foraging group by random selection. On the other hand, undesirable solutions that give fitness to the best value from the current solutions. In detail, IMCA improved the basic MCA by the solution (x) generated from the neighbor_generat function. In addition, choose one solution (y) from the foraging group at random. If the solution (x) is better than solutions (y) then replace it. The pseudocode of the IMCA is shown in *Fig. 2*. The basic MCA ignores the undesirable solutions, while the IMCA algorithm exploited the undesirable solutions to be a substitute for weak solutions in the foraging group. The test results indicated that the IMCA advantage in performance compared to the basic MCA.

V. EXPERIMENTAL RESULTS

The datasets, as well as their most notable features, are described in this section of the paper. It has been shown how to set up parameters for IMCA. After that, the results will be discussed and analyzed.

A. Dataset Description

The Royal Institute of Technology KTH [8] provided the dataset's input data. The number of student groups, courses, and classes in each of the four input data files varies. Each student group has two courses to complete, each with one to three events to plan. Divide the number of events by the number of timeslots to get the event density. The

Received 11/8/2021; Accepted 27l8l2021

DOI: https://doi.org/10.33103/uot.ijccce.21.4.10

datasets are divided into four categories based on event density (S, M, L, and XL). The classified datasets are summarized in Table 1.

```
Parameter
    n = size of clan
    m = size of foraging where m < n
    c = size of care n-m-1
    Fr = rate of worst foraging
    Cr = rate of worst care
    k = neighbor option
Start
    Create a clan(n) of options at random
    Calculate clan fitness options
    Sentry = clan's best option
    Split up into two parts (foraging & care) from the clan.
    While not termination condition Do
          For i=1 to m
                Call neighbor_generat (k, Sentry, foraging(i), best_one)
                If best_one is better than foraging(i)
                    foraging(i) = best_one;
                Else
                    Let solution(x) = best_one;
                    Select randomly solution(y) from foraging group;
                    If solution(x) is better than solution(y);
                              solution(y) ⇔ solution(x);
          end for
          In the foraging group, exchange the worst option for Fr with the best option in the care
          group;
          Remove the worst Cr option from the care group and create an option at random;
          Choose the best foraging option and name it best_forg;
          If best_forg <= Sentry then
              Sentry ⇔ best_forg
          end if
    end while
End

Function of neighbor_generated
In : K, Sentry, foraging(i)
Out : best_one
Start
    Obtain k neighbors by foraging;
    Calculate k's fitness.
    If there is no one best than foraging(i) then
          Obtain k neighbors by Sentry;
    Else
          best one is the better one of neighbor for both foraging and Sentry;
    end if
End
```

FIG. 2. IMPROVED MEERKAT CLAN ALGORITHM (IMCA) ALGORITHM PSEUDOCODE.

Lecture, lesson, and lab are the three types of rooms. A week is divided into five weekdays, from Sunday to Thursday, with each weekday divided into four two-hour timeslots. This equates to a total of 20 timeslots in each room. For example, if there are eight rooms on the schedule, the total time slots are 8x20=160.

Received 11/8/2021; Accepted 27l8l2021

DOI: https://doi.org/10.33103/uot.ijccce.21.4.10

TABLE 1. SUMMARY OF THE DIFFERENT TEST DATA SETS [8]

Input Data File	kth_S	kth_M	kth_L	kth_XL
Lecture Rooms	2	2	3	6
Lesson Rooms	3	5	6	10
Lab Rooms	3	5	7	11
Courses	12	15	21	29
Lecturers	9	12	15	21
Student Groups	6	8	12	21
Total Events	70	115	159	293
Total Time Slots	160	240	320	540
Event Density	0.44	0.48	0.50	0.54

B. Parameter setting

Several tests were carried out to determine the proper parameter values for the examination and to compare the IMC and MC algorithms. The values of the parameters considered are listed in Table 2.

TABLE 2. USED PARAMETER FOR TESTING

Parameter No.	Parameter Name	kth_S	kth_M	kth_L	kth_XL
1	Clan Size	30	40	50	60
2	Foraging Size	18	24	30	36
3	Care Size	11	15	19	23
4	Iteration	60	80	100	120
5	Worst foraging rate	0.25	0.25	0.25	0.25
6	Worst care rate	0.25	0.25	0.25	0.25
7	Neighbor solution	6	8	10	12

1-Clan Size: The size of the population or the number of meerkats, which ranges from 30 to 60 with an increment of 10.

2-Foraging Size: Equals (clan size multiplied by 0.6), which ranges from 10 to 36 depending on the clan size.

3-Care Size: Equal (clan size minus foraging size minus 1), which ranges from 11 to 23 depending on the clan size and the foraging size.

4-Iteration: The number of iterations (60, 80, 100, and 120) is determined by the size of the dataset.

5-Worst foraging rate: probability with a value of 0.25.

6-Worst care rate: probability with a value of 0.25.

7-Neighbor solution: Equals (clan size multiplied by 0.2), which ranges from 6 to 12 depending on the clan size.

Received 11/8/2021; Accepted 27l8l2021

DOI: https://doi.org/10.33103/uot.ijccce.21.4.10

C. Discussion and analysis of the results

Several experiments were performed according to the parameters listed in Table 2. It uses C# programming language and computer specification (CPU Core i5, RAM 8MB, OS Windows 10 64-bit, HDD 512GB).

A fitness level function is used to grade the solution by calculating the number of hard constraints that were violated as shown In the formula:

fitness(timetable) = (HC1 + HC2 + HC3 + HC4)

where HC1, HC2, HC3, and HC4 are as follows:

CH1 is the number of double-booked student groups.

CH2 is the number of double-booked lecturers.

CH3 is the number of room capacity breaches.

CH4 is the number of room-type breaches.

This is achieved by taking ten readings of the average fitness values on each iteration. Also calculating the ratio of change for the average fitness values for each iteration by formula:

$$rate = \frac{f(B) - f(A)}{f(A)}$$

Where f(B) is the average fitness value of ten readings on each iteration for IMCA, also f(A) is the average value of fitness for MCA for the same number of readings. In other words, the f(A) and f(B) values are average fitness values in the same iterations for calculating the difference of increase or decrease to average fitness. The rate value may be positive or negative depending on the difference in fitness value. The negative value refers to improvement in performance. And vice versa. The obtained test results are shown in Table 3, Table 4, Table 5, and Table 6 by datasets.

TABLE 3: RESULTS OF TESTING FOR KTH_S DATASET.

Iterations	$f(A)$	$f(B)$	$f(B)-f(A)$	$\frac{f(B)-f(A)}{f(A)}$
1	55.5	55.5	0	0.00%
4	49.2	47.5	-1.7	-3.46%
7	42.7	42.9	0.2	0.47%
10	37.7	38.9	1.2	3.18%
13	32.7	32.9	0.2	0.61%
16	29.5	26.4	-3.1	-10.51%
19	25.2	24	-1.2	-4.76%
22	21.4	20.7	-0.7	-3.27%
25	17.5	16.3	-1.2	-6.86%
28	14.6	15.1	0.5	3.42%
31	10.5	10.8	0.3	2.86%
34	9.6	7.9	-1.7	-17.71%
37	7.8	6	-1.8	-23.08%
40	5.5	4.2	-1.3	-23.64%
43	5.1	3.8	-1.3	-25.49%
46	3.3	3	-0.3	-9.09%
49	2.6	1.4	-1.2	-46.15%
52	2.5	0.6	-1.9	-76.00%
51	1.4	0.5	-0.9	-64.29%
58	1.2	0.6	-0.6	-50.00%

TABLE 4: RESULTS OF TESTING FOR KTH_M DATASET.

Iterations	$f(A)$	$f(B)$	$f(B)-f(A)$	$\frac{f(B)-f(A)}{f(A)}$
1	100.3	102.1	1.8	1.79%
5	92.8	90.6	-2.2	-2.37%
9	82.3	80.8	-1.5	-1.82%
13	73.7	73.2	-0.5	-0.68%
17	66.1	67.2	1.1	1.66%
21	59	57.5	-1.5	-2.54%
25	53.5	52.4	-1.1	-2.06%
29	46.4	43.3	-3.1	-6.68%
33	40.7	39.6	-1.1	-2.70%
37	35.6	34.1	-1.5	-4.21%
41	30.3	28.3	-2	-6.60%
45	26.9	23.3	-3.6	-13.38%
49	20.7	19.1	-1.6	-7.73%
53	18.3	14.6	-3.7	-20.22%
57	16.2	12.5	-3.7	-22.84%
61	10.5	9.4	-1.1	-10.48%
65	8.7	6.4	-2.3	-26.44%
69	8.7	5.1	-3.6	-41.38%
73	5.1	2.9	-2.2	-43.14%
77	3.6	2.1	-1.5	-41.67%

Received 11/8/2021; Accepted 27l8l2021

DOI: https://doi.org/10.33103/uot.ijccce.21.4.10

TABLE 5: RESULTS OF TESTING FOR KTH_L DATASET.

Iterations	$f(A)$	$f(B)$	$f(B)-f(A)$	$\frac{f(B)-f(A)}{f(A)}$
1	147	148	1	0.68%
6	131.2	132.7	1.5	1.14%
11	122.3	120.2	-2.1	-1.72%
16	108.8	108.5	-0.3	-0.28%
21	98.9	100.9	2	2.02%
26	88.6	86.8	-1.8	-2.03%
31	76	76.4	0.4	0.53%
36	69.4	67.5	-1.9	-2.74%
41	59.7	56.9	-2.8	-4.69%
46	49.7	50.5	0.8	1.61%
51	47	42.8	-4.2	-8.94%
56	39.1	37.5	-1.6	-4.09%
61	34.3	31.3	-3	-8.75%
66	26	25.2	-0.8	-3.08%
71	22.7	19.7	-3	-13.22%
76	17.8	14.1	-3.7	-20.79%
81	12.5	9.3	-3.2	-25.60%
86	8.7	6.6	-2.1	-24.14%
91	6.7	5	-1.7	-25.37%
96	5.2	2.7	-2.5	-48.08%

TABLE 6: RESULTS OF TESTING FOR KTH_XL DATASET.

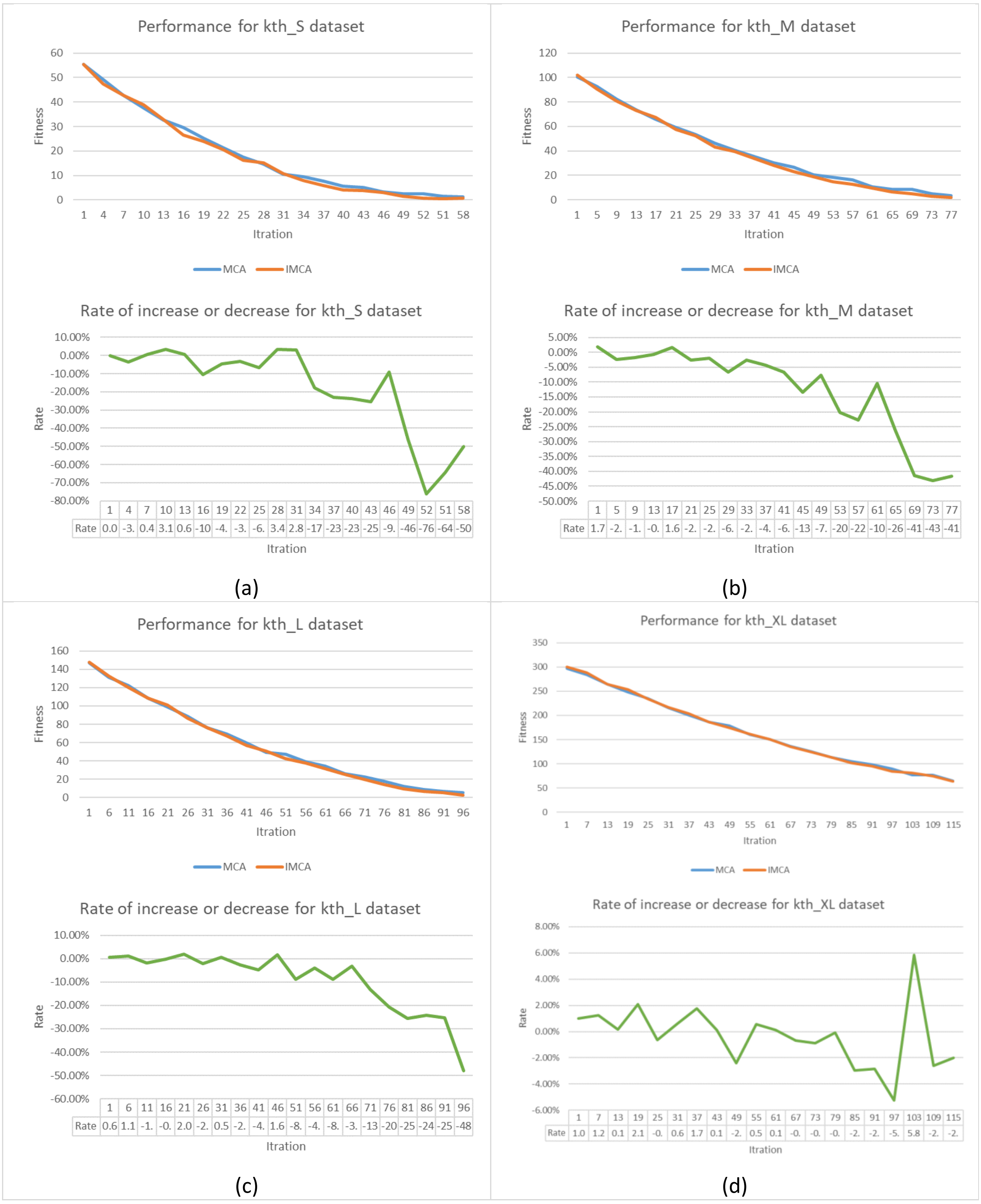
Iterations	$f(A)$	$f(B)$	$f(B)-f(A)$	$\frac{f(B)-f(A)}{f(A)}$
1	296.9	299.9	3	1.01%
7	284.3	287.8	3.5	1.23%
13	264.2	264.6	0.4	0.15%
19	247.9	253.1	5.2	2.10%
25	235.6	234.1	-1.5	-0.64%
31	215.8	217.1	1.3	0.60%
37	200.6	204.1	3.5	1.74%
43	186.6	186.8	0.2	0.11%
49	178.3	174	-4.3	-2.41%
55	160.5	161.4	0.9	0.56%
61	150	150.2	0.2	0.13%
67	136.1	135.2	-0.9	-0.66%
73	125.3	124.2	-1.1	-0.88%
79	113.5	113.4	-0.1	-0.09%
85	104.9	101.8	-3.1	-2.96%
91	98.5	95.7	-2.8	-2.84%
97	89.3	84.6	-4.7	-5.26%
103	77.1	81.6	4.5	5.84%
109	76.3	74.3	-2	-2.62%
115	65.1	63.8	-1.3	-2.00%

IMCA accelerates to the optimal solution from MCA because do does not ignore undesirable solutions. The result has shown the performance of IMCA over MCA by a small difference because MCA has a high-performance exploitative behavior. The performance difference between MCA and IMCA is clearly illustrated in *Fig.* 3 (a, b, c, and d).

Where (a) included testing with the kth-S dataset and the rate of average fitness value refers to up to (-76%) with an average (-17.69%). And (b) included testing with the kth-M dataset and the rate of average fitness value is refers to up to (-43%) with an average (-12.67%). Also, (c) and (d) was the test with the kth-L and kth-XL datasets were rate of average fitness value for both refer to up to (-48% and -5%) with an average (-9.38% and -0.34%).

Received 11/8/2021; Accepted 27l8l2021

DOI: https://doi.org/10.33103/uot.ijccce.21.4.10



	1	4	7	10	13	16	19	22	25	28	31	34	37	40	43	46	49	52	51	58
Rate	0.0	-3.	0.4	3.1	0.6	-10	-4.	-3.	-6.	3.4	2.8	-17	-23	-23	-25	-9.	-46	-76	-64	-50

	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61	65	69	73	77
Rate	1.7	-2.	-1.	-0.	1.6	-2.	-2.	-6.	-2.	-4.	-6.	-13	-7.	-20	-22	-10	-26	-41	-43	-41

	1	6	11	16	21	26	31	36	41	46	51	56	61	66	71	76	81	86	91	96
Rate	0.6	1.1	-1.	-0.	2.0	-2.	0.5	-2.	-4.	1.6	-8.	-4.	-8.	-3.	-13	-20	-25	-24	-25	-48

	1	7	13	19	25	31	37	43	49	55	61	67	73	79	85	91	97	103	109	115
Rate	1.0	1.2	0.1	2.1	-0.	0.6	1.7	0.1	-2.	0.5	0.1	-0.	-0.	-0.	-2.	-2.	-5.	5.8	-2.	-2.

FIG. 3. AVERAGE FITNESS VALUE VERSUS ITERATIONS AND THE RATE OF INCREASE OR DECREASE.

VI. CONCLUSION

An academic, fascinating, and challenging subject is the University Course timetabling problem (UCTP). The researcher in this field should focus on improving techniques for solving UCTP. The Improved Meerkat Clan Algorithm (IMCA) is presented in this research

Received 11/8/2021; Accepted 27l8l2021

DOI: https://doi.org/10.33103/uot.ijccce.21.4.10

to solve the UCTP. The improvement idea is that finding a new subroutine for Meerkat Clan Algorithm MCA to get the best results. And that by studying the behavior of MCA and Specifying the points that are able to improve without changing the main behavior of MCA. Therefore, the focus was on exploiting solutions that were ignored. The weak solutions in the foraging group are randomly replaced with it. The IMCA's experimental findings are encouraging. Despite the problem's great complexity and NP-hard, the performance of the IMCA is good. Within a processing period, good results were in the range of -17.69% to -0.34% of the average rate depending on the scale of the dataset. The percentage decreases as the dataset grows. Therefore, it is recommended to use it with low-complexity datasets to get a good result. In this sense, other ways should be studied as well, despite the IMCA's strong performance. As a result, in the second supplement of this study, we would want to develop and experiment with different heuristic strategies in order to compare their performance.

REFERENCES

[1] A. M. Hambali, Y. A. Olasupo, and M. Dalhatu, “Automated university lecture timetable using heuristic approach,” Nigerian Journal of Technology, vol. 39, no. 1, pp. 1–14, 2020.

[2] E. K. Burke, B. Mccollum, A. Meisels, S. Petrovic, and R. Qu, “A graph-based hyper-heuristic for educational timetabling problems,” European Journal of Operational Research, vol. 176, no. 1, pp. 177–192, 2007.

[3] A. Abuhamdah, M. Ayob, G. Kendall, and N. R. Sabar, “Population based Local Search for university course timetabling problems,” Applied Intelligence, vol. 40, no. 1, pp. 44–53, 2013.

[4] C. Akkan and A. Gülcü, “A bi-criteria hybrid Genetic Algorithm with robustness objective for the course timetabling problem,” Computers & Operations Research, vol. 90, pp. 22–32, 2018.

[5] A. A. Gozali, S. Fujimura, “Solving University Course Timetabling Problem Using Multi-Depth Genetic Algorithm”, SHS Web of Conferences, vol. 77, pp. 01001, 2020.

[6] H. Babaei, J. Karimpour, and A. Hadidi, “Applying Hybrid Fuzzy Multi-Criteria Decision-Making Approach to Find the Best Ranking for the Soft Constraint Weights of Lecturers in UCTP,” International Journal of Fuzzy Systems, vol. 20, no. 1, pp. 62–77, 2017.

[7] T. L. June, J. H. Obit, Y.-B. Leau, J. Bolongkikit, and R. Alfred, “Sequential Constructive Algorithm incorporate with Fuzzy Logic for Solving Real World Course Timetabling Problem,” Lecture Notes in Electrical Engineering Computational Science and Technology, pp. 257–267, 2020.

[8] A. Salman, R. Hanna, “A Comparative Study between Genetic Algorithm, Simulated Annealing and a Hybrid Algorithm for solving a University Course Timetabling Problem,” Degree Project in Computer Science, KTH, Stockholm, Sweden (2018).

[9] S. L. Goh, G. Kendall, N. R. Sabar, and S. Abdullah, “An effective hybrid local search approach for the post enrolment course timetabling problem,” Opsearch, vol. 57, no. 4, pp. 1131–1163, 2020.

[10] M. Chen, X. Tang, T. Song, C. Wu, S. Liu, and X. Peng, “A Tabu search algorithm with controlled randomization for constructing feasible university course timetables,” Computers & Operations Research, vol. 123, p. 105007, 2020.

[11] A. Muklason, R. G. Irianti, and A. Marom, “Automated Course Timetabling Optimization Using Tabu-Variable Neighborhood Search Based Hyper-Heuristic Algorithm,” Procedia Computer Science, vol. 161, pp. 656–664, 2019.

[12] S. L. Goh, G. Kendall, and N. R. Sabar, “Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem,” Journal of the Operational Research Society, vol. 70, no. 6, pp. 873–888, 2018.

[13] M. Mazlan, M. Makhtar, A. F. K. A. Khairi, and M. A. Mohamed, “University course timetabling model using ant colony optimization algorithm approach,” Indonesian Journal of Electrical Engineering and Computer Science, vol. 13, no. 1, p. 72, 2019.

[14] S. C. Sarin, Y. Wang, and A. Varadarajan, “A university-timetabling problem and its solution using Benders’ partitioning—a case study,” Journal of Scheduling, vol. 13, no. 2, pp. 131–141, 2009.

[15] A. T. S. Al-Obaidi, H. S. Abdullah, and Z. O. Ahmed, “Meerkat Clan Algorithm: A New Swarm Intelligence Algorithm,” Indonesian Journal of Electrical Engineering and Computer Science, vol. 10, no. 1, p. 354, 2018.

Received 11/8/2021; Accepted 27l8l2021