khazanah informatika

Jurnal Ilmu Komputer dan Informatika

Complex University Timetabling Using Iterative Forward Search Algorithm and Great Deluge Algorithm

I Gusti Agung Premananda*, Ahmad Muklason
Information Systems Department
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember Surabaya
Surabaya, Indonesia
*igustiagungpremananda@gmail.com

Abstract-University timetabling is an issue that has received more attention in the field of operations research. Course scheduling is the process of arranging time slots and room for a class by paying attention to existing limitations. This problem is an NP-Hard problem, which means the computation time to find a solution increases exponentially with the size of the problem. Solutions to problems of this kind generally use a heuristic approach, which tries to find a sufficiently good (not necessarily optimal) solution in a reasonable time. We go through two stages in solving the timetabling problem. The first stage is to schedule all classes without breaking any predefined rules. The second stage optimizes the timetable generated in the first stage. This study attempts to solve the class timetabling problem issued in a competition called the 2019 International Timetabling Competition (ITC 2019). In the first stage, we use the Iterative Forward Search (IFS) algorithm to eliminate timetable candidates and to generate a schedule. In the second stage, we employ the Great Deluge algorithm with a hyper-heuristic approach to optimize the solution produced in the first stage. We have tested the method using 30 datasets by taking 1,000,000 iterations on each dataset. The result is an application that does schedule elimination and uses the IFS algorithm to produce a schedule that does not violate any of the hard constraints on 30 ITC 2019 datasets. The implementation of the Great Deluge algorithm optimizes existing schedules with an average penalty reduction of 42%.

Keywords: timetabling, class scheduling, iterated forward search, international timetabling competition

Article info: *submitted December 10, 2020, revised March 4, 2021, accepted March 24, 2021*

1. Introduction

Timetabling problems in education have received much attention and have long been studied in the field of operations research [1]. This problem contains how to schedule courses against the available schedule and room. There are two types of constraints pertaining to the challenge, namely hard constraints and soft constraints [2]. Hard constraint is a limit that must be met in scheduling a class [3]. Examples of this limitation include the maximum capacity of a classroom, the number of schedule slots available in a class and two classes that cannot be scheduled simultaneously. Soft constraint is a limit that can be violated, but if it is violated, it will cause the quality of scheduling to decrease [3]. Generally, the penalty value is used on the soft constraints that are violated to measure the quality of the resulting scheduling. Examples of soft constraints such as preferably class a and class b should be scheduled simultaneously, class a should not be scheduled in room x, and class a should be scheduled in a different week from class c.

Scheduling problems involve many rooms and many possible time slots. Hence the number of possible combinations will be very high. This problem has been categorized as an NP-Hard problem [4] which means the computation time required to find a solution increases exponentially with the size of the problem [5]. If there are 3 classes to be scheduled, the number of possible schedule combinations is 6. However, if there are 10 classes to be scheduled, the number of possible schedule combinations increases exponentially to 3,628,800 possibilities. Facing this fact, latest researches focus more on developing algorithms with a heuristic approach, namely an approach to produce a fairly good solution (not necessarily the optimal one) and within a reasonable time (time that allows it to be applied to real problems) [6].

Generally solving scheduling problems consists of two phases, namely the first phase to build an initial solution without breaking existing hard constraints and the next phase is to optimize the solution by reducing the number of penalties for violating soft constraints [7]. In the first phase, Graph Coloring algorithm is generally applied, which is a simple algorithm to convert classes into graphical form and use coloring to ensure that there are no violations of hard constraints [8]. However, in several case studies, sometimes there are too many hard constraints that the Graph Coloring algorithm could not produce an initial solution. This calls for another algorithm to tackle the challenge [7].

Several studies have been conducted to solve timetabling problems. Research conducted by Muller et al in 2004 [9] created a new algorithm that can be used in university timetabling problems in which there are many constraints and hierarchies of courses so that this problem is known as *complex university timetabling* [10]. This algorithm is called Iterative Forward Search (IFS). The trial was carried out on scheduling problems at Purdue University and was able to meet up to 98% of student requests where there were no conflicting schedules between the courses selected by the student. Subsequent research was carried out by Rudová in 2010 [10] where they successfully applied the IFS algorithm to solve scheduling problems in the 2007 ITC dataset. Meanwhile, on the same dataset, Muklason conducted research in 2019 [11] to perform optimization using the Great Deluge algorithm based on hyper heuristics scheme. The result concluded that this algorithm is superior to other algorithms such as simulated annealing and hill climbing algorithms [11].

Table 1. Description of the ITC 2019 Dataset

Dataset	Number of Classes	Number of Rooms	Number of students	Number of Limits
agh-fis-spr17	1239	80	1641	1220
agh-ggis-spr17	1859	44	2116	2690
bet-fal17	983	62	3018	1251
iku-fal17	2641	214	0	2902
mary-spr17	882	90	3666	3947
muni-fi-spr16	575	35	1543	740
muni-fsps-spr17	561	44	865	400
muni-pdf-spr16c	2526	70	2938	2026
pu-llr-spr17	1001	75	27018	634
tg-fal17	711	15	0	501
agh-ggos-spr17	1144	84	2254	1688
agh-h-spr17	460	39	1988	399
lums-spr18	487	73	0	518
muni-fi-spr17	516	35	1469	699
muni-fsps-spr17c	650	29	395	709
muni-pdf-spr16	1515	83	3443	1012
nbi-spr18	782	67	2293	596
pu-d5-spr17	1061	84	13497	1535
pu-proj-fal19	8813	768	38437	7797
yach-fal17	417	28	821	645
agh-fal17	5081	327	6925	7154
bet-spr18	1083	63	2921	1418
iku-spr18	2782	208	0	3488
lums-fal17	502	73	0	597
mary-fal18	951	93	5051	513
muni-fi-fal17	535	36	1685	787
muni-fspsx-fal17	1623	33	1152	1359
muni-pdfx-fal17	3717	86	5651	3501
pu-d9-fal19	2798	224	35213	2746
tg-spr18	676	18	0	426

The 2019 ITC [12] issued 30 new datasets (Table 1) containing timetabling problems based on original data from several universities in the world. This problem contains how to schedule classes without violating existing restrictions and also produce good quality timetable by obtaining the smallest possible penalty. The penalty value at ITC 2019 comes from four types of soft constraints. The first is that each schedule candidate has a different penalty value. So that as much as possible in scheduling a class, it is done on the schedule candidate having the lowest penalty. Furthermore, each room also has a different penalty value so that as much as possible we choose the room with the lowest penalty when scheduling a class. Furthermore, a penalty is applied if a student gets two or more overlapped classes – hence impossible to attend them all. Penalties will be given according to the number of overlapping classes for each student. Finally, penalties are awarded based on violations of 19 types of distribution (Table 2) which are soft constraints.

Table 2. Description of Distribution Boundaries

Distribution Limits	Description
SameStart	The classes covered by this limit must start in the same time slot
SameTime	Classes that fall under this limit must have the same start and end times if the class duration is the same. If the class duration is different, the shorter class must start at the same time or after the class whose duration is longer and must end before or together with the longer duration class.

Distribution Limits	Description
DifferentTime	The opposite of the sameTime limitation.
SameDays	Classes that are subject to this limit must be scheduled on the same day. If a class has fewer days in the week, that class should be scheduled as a subset of the class with the larger number of days.
DifferentDays	The opposite of the SameDays limitation
SameWeeks	These limits are the same as the SameDays limits, but apply on a week basis
DifferentWeeks	The opposite of the SameWeeks constraint
Overlap	Classes that fall under this limitation must be scheduled overlapping in terms of time slots, days and weeks.
NotOverlap	The opposite of the Overlap limit
SameRoom	Classes that fall under this limit must be scheduled in the same room
DifferentRoom	The opposite of SameRoom limitation
SameAttendees	The class that is in this limit must allow one student to take it at the same time. Classes must not be scheduled overlap and must pay attention to the distance to go from one class to another.
Precedence	Classes contained in this constraint must be scheduled according to the order in this constraint. The ordering is only based on the first time meeting in each class.
WorkDay	Classes that are subject to this limit may not be scheduled for the end of the final class plus the initial class start over the S time slot if it is scheduled on the same day and week.
MinGap	Classes that fall under this limit must have a certain distance (x) of time if scheduled on the same day.
MaxDays	Classes that are subject to this limit cannot be scheduled more than (x) days apart.
MaxDayLoad	Classes subject to this limit must be scheduled for no more than (x) time slots on each day.
MaxBreaks	Classes that fall under this limit must be scheduled with no break between classes more than (x) time slots on each day.
MaxBlock	This limitation imposes a limit on the block length (some classes are scheduled with the rest interval between classes is less than (y) time slots) so that there are no more than (x) time slots on each day.

This dataset has a higher level of complexity compared to the dataset issued by the same competition in 2011, 2007 and 2002. The increased complexity lies in the limited list of schedules in each class, increased types of hard constraints and soft constraints to 19 types, and the presence of hierarchies in each subject. The ITC 2019 dataset is divided into 3 types, namely 10 early instance datasets, 10 middle instance datasets and 10 late instance datasets. The three groups of the dataset have different levels of difficulty based on the number of classes, the number of rooms, the number of students and the number of distributions. The *early* dataset is the easiest dataset with the smallest average number of classes, number of rooms, number of students, and distribution. Furthermore, the *middle* dataset has a moderate level of difficulty with the average number of classes, the number of rooms, the number of students and distribution more than the *early* group dataset but less than the *late* group dataset. The *late* dataset group is the most difficult dataset with an average number of classes, the number of rooms, the number of students and the most distribution compared to the *early* and *middle* group dataset.

Based on the above background, this study aims to solve the latest complex scheduling problems using the latest dataset from ITC 2019. From this research it is hoped that it can be used by various universities that have similar problems in producing course schedules in their respective departments or universities.

2. Methods

This section explains the stages carried out in this study, starting from data preprocessing to validating the final solution.

a. Data Preprocessing

At this stage there will be elimination of several schedule candidates in each class that are impossible to use. This happens because the candidate's schedule and room have clearly violated the hard constraints. Otherwise, if those candidates were retained, it will interfere with the initial solution search process and the solution optimization.

The first elimination is carried out based on room usage restrictions, where there are rooms that cannot be used at a certain time. Each candidate schedule will be checked as shown in Figure 1. If there is a conflict, the candidate schedule will be deleted.

The next elimination is carried out on schedule candidates who violate the distribution constraints (Table 2) in the form of hard constraints that present in each class. For example if there is a distribution limitation where class (x) must be scheduled on the same day (SameDay) as class (y), then the candidate schedule in class (x) which has the same day as the class schedule candidate (y) will be retained and the rest going to elimination. All schedule candidates in each class will be checked against the distribution boundaries in the form of hard constraints as shown in Figure 2. If it is found that there is a violation of any hard constraint distribution, the candidate schedule will be discarded.

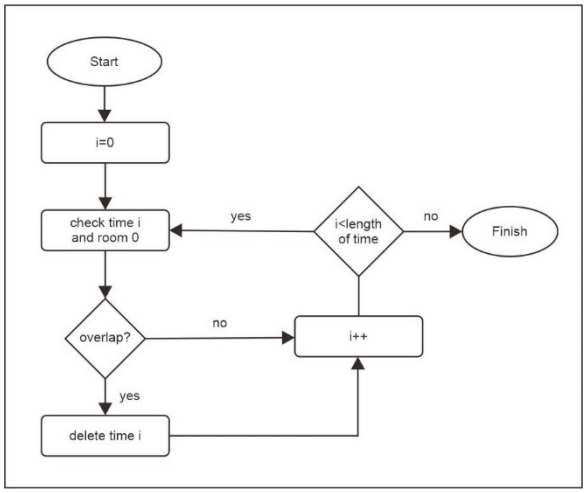

Figure 1. First Elimination Flow

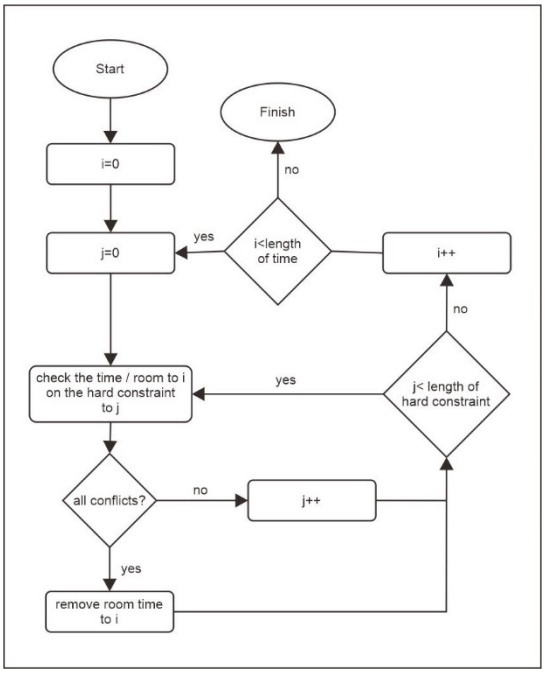

Figure 2. Second Elimination Flow

The next elimination is carried out based on the class which has one schedule candidate. For example, suppose class (x) has only one schedule candidate. Then class (x) will be scheduled on that one candidate schedule because there is no other choice. Apart from class (x), if there is a class that has an overlap with the candidate schedule in class (x), it will be eliminated because it is not possible to use it. The flow of this elimination can be seen in Figure 3. To see how effective this elimination is, the results will be compared with elimination and without elimination.

b. Algorithm design and implementation

At this stage, a design will be carried out to make several changes to the IFS algorithm so that it is able to produce a viable solution. The IFS algorithm initially takes a variable randomly and tries to enter a randomly drawn value from that variable. If there is a conflict, the value in the conflict variable will be temporarily deleted. The pseudo code of the IFS algorithm can be seen in Figure 4.

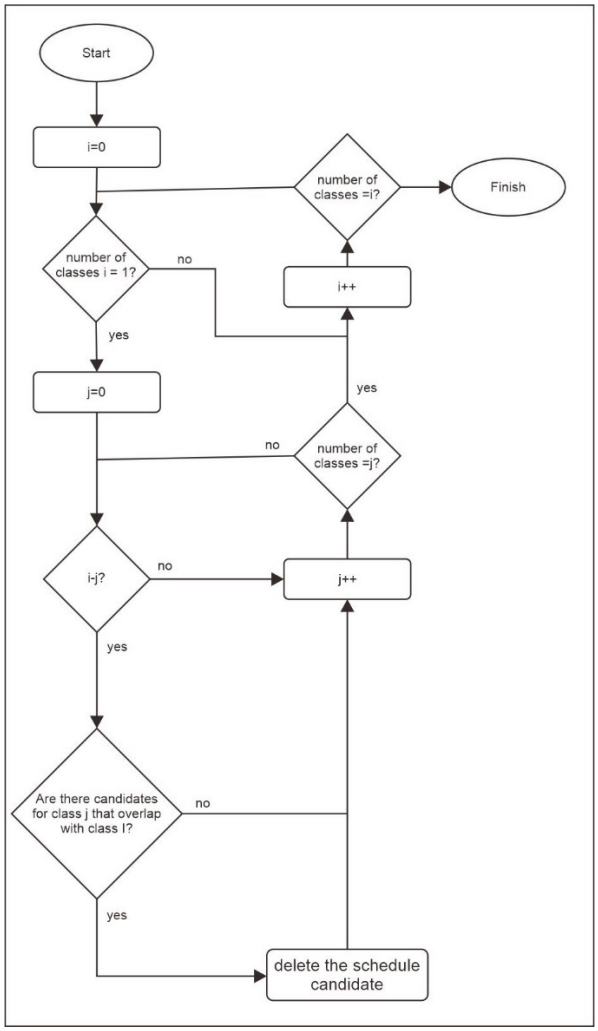

Figure 3. Third Elimination Flow

```
procedure SOLVE(initial)            // initial solution is the parameter
    iteration = 0;                  // iteration counter
    current = initial;              // current solution
    best = initial;                 // best solution
    while canContinue(current, iteration) do
        iteration = iteration + 1;
        variable = selectVariable(current);
        value = selectValue(current, variable);
        UNASSIGN(current, CONFLICTING_VARIABLES(current, variable, value));
        ASSIGN(current, variable, value);
        if better(current, best) then best = current
    end while
    return best
end procedure
```

Figure 4. IFS algorithm pseudocode

Changes made in this study involve changing the selection of values, or in this case in the form of a schedule candidate from being randomly selected to trying each candidate schedule until it is found that there is no conflict with other variables or in this case a class. If it is found that there are no schedule candidates that can be used in a scheduled class, one of the schedule candidates will be selected randomly and the other classes that conflict with the schedule candidate will be returned to the unscheduled

condition. The iteration will be repeated until it is found that all the class conditions have been scheduled and no one has violated the constraints. Figure 5 shows the changes made to the IFS algorithm.

To see if there is an impact of changes made to the IFS algorithm, this study runs and compares two algorithms, namely by using the IFS algorithm as shown in Figure 4 and the IFS algorithm which has been changed as in Figure 5.

```
Algorithm 1: Modified Iterated Forward Search
  i=0;
  solution=getAllClass();
  while !cekFeasible(solution) do
      result=cekAllSchedule(solution.Class(i));
      if !result then
          r=random(solution.Class(i).CombinationTimeRoom.size());
          unassign(ConflictingClass(solution.Class(i),r),solution);
      else
          i++;
      end
      if i >sizeClass() then
          i=0;
      end
  end
```

Figure 5. Pseudocode of IFS Algorithm Change Results

After obtaining the initial solution, optimization will be carried out using the Great Deluge algorithm with a hyper-heuristic approach. In the hyper-heuristic approach, there are two parts, namely move acceptance to choose whether or not a solution is accepted and low level heuristic (LLH) to change existing solutions.

In the move acceptance section, the Great Deluge algorithm is used. A new solution will be accepted if it produces a better result than the previous solution or if the solution is better than the level parameter value. The level parameter value is obtained from the initial solution value and will decrease continuously during the iteration. Figure 4 illustrates how this algorithm works.

In the LLH section, mutations are used to make changes to the solution. Mutations work by changing the schedule of one of the classes. Figure 7 illustrates the changes made by mutations. Initially there was a class 6 class that had been scheduled. Class 1 is scheduled for time slot 10, class 2 is scheduled for time slot 76, class 3 is scheduled for time slot 23 and so on. Mutations are applied to class 3 by scheduling at a different time slot, namely 27 time slots. The mutation only affects one randomly selected class. Meanwhile, other classes will not change.

Furthermore, the algorithm design will be implemented through the Java programming language with a trial environment as shown in Table 3.

Table 3. Algorithm Implementation Environment

Device	Specification
Processor	AMD Ryzen 7 3700U (8 CPUs)
Ram	16GB
OS	Windows
Intellij	IDE 8.0.2

c. Final Solution Validation

To ensure that the final result is a valid solution, the final result will be uploaded to https://www.itc2019.org/validator to check the solution. The validator will check whether any hard constraints are violated or not. In addition, the validator will calculate the penalty value so that the penalty value on the validator web can be compared with the penalty value that is owned in this test to ensure that the program of implementing the algorithm that has been designed produces the correct results. Solutions that have been deemed valid by the web validator will be stored on the website.

```
Algorithm 1: Great Deluge algorithm
  solution=getIntitialSolution();
  bestSolution=solution;
  estimatedQuality=calculatePenalty(solution)/10;
  NumOfIte=1000000;
  level=calculatePenalty(solution);
  notImprovingCounter=0;
  decreasingRate=(calculatePenalty(solution)-estimatedQuality)/NumOfIte;
  iteration=0;
  while iteration < NumOfIte do
      newSolution=solution r=random(getSizeClass());
      mutation(newSolution.Class(r));
      if calculatePenalty(newSolution)<calculatePenalty(bestSolution) then
          solution=newSolution;
          bestSolution=newSolution;
          notImprovingCounter=0;
      else
          if calculatePenalty(newSolution)<=level then
              solution=newSolution;
              notImprovingCounter=0;
          else
              notImprovingCounter++;
              if notImprovingCounter==100 then
                  solution=bestSolution;
                  decreasingRate=(calculatePenalty(solution)-estimatedQuality)/(NumOfIte-
                    iteration);

                  notImprovingCounter=0;
              end
          end
      end
      level=level-decreasingRate;
      iteration++;
  end
```

Figure 6. Great Deluge Algorithm pseudocode

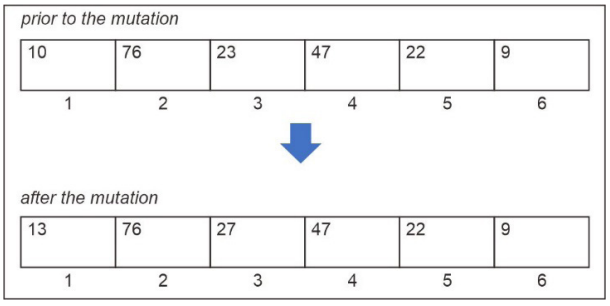

Figure 7. Examples of Mutation Operations in Scheduling

3. Results and Discussion

This section will explain the results obtained in this study.

a. Data Preprocessing

After the data preprocessing process was carried out, a reduction in schedule candidates was obtained in each dataset. Tables 3, 4 and 5 display the number of schedule candidates, the number of eliminated schedule candidates

and the percentage of eliminations performed. Overall there is a reduction in schedule candidates by an average of 19.4% of the 30 datasets tested.

To see the impact of the data preprocessing process, a comparison was made between using the data preprocessing process and those that did not use the data preprocessing process. The result is that without the data preprocessing process, there are 8 datasets for which a solution cannot be found without violating existing hard constraints. The 8 datasets are Agh-fis-spr17, Nbi-spr18, Pu-d5-spr17, Pu-proj-fal19, Agh-fal17, Muni-fspsx-fal17, Muni-pdf-fal17 and Pu-d9-fal19.

The data preprocessing process has proven to be able to help find initial solutions without breaking existing hard constraints. This happens because if the schedule candidate is not eliminated in the data preprocessing process, the schedule candidate will interfere with the process of producing the initial solution. Schedule candidates who are unlikely to be used, can be selected in the process of generating the initial solution. This will have an impact on other classes and cause conflicts between classes.

b. Initial Solution and Final Solution

The initial solution was carried out after the data preprocessing stage by running the IFS algorithm which was adjusted to the 2019 ITC problem. The result was that this algorithm was able to produce an initial solution without breaking the hard constraints that existed in the entire dataset (30 datasets).

To see the impact of changes made to the IFS algorithm, two experiments were carried out using the original IFS algorithm and using the IFS algorithm that had been changed. The result is that using the original IFS algorithm without making any changes is only able to produce an initial solution on one dataset, namely the Mary-spr17 dataset. This happens because the original IFS algorithm selects schedule candidates randomly, so that the search for solutions is not well focused on finding solutions without breaking the existing hard constraints.

The next step is to optimize each dataset by running 1000,000 iterations. The result was that the largest penalty reduction was in the Lums-spr18 dataset, which was 78% and the smallest penalty reduction was in the bet-fal17 dataset, which was 7%. Overall, the average penalty reduction was 42%. Tables 6, 7 and 8 show the results of the comparison between the initial solution and the final solution after the penalty is applied.

In the Pu-proj-fal19, Bet-fal17, Agh-fal17, Bet-spr18, Muni-fspsx-fal17 and Muni-pdfx-fal17 dataset, optimization can only reduce penalties by a percentage of between 7-11%, especially in the late dataset. This happens because the late dataset is the most difficult dataset by having more classes, rooms and boundaries than the middle and early dataset groups. The Great Deluge algorithm cannot explore solutions on a dataset like this because when a solution changes, hard constraint violations often occur so that the new solution cannot be accepted.

Table 4. Preprocessing results on 10 datasets of early instances

Dataset	Number of Candidates Schedule	Number of Reduced Schedule Candidates	Percentage Reduction
Agh-fis-spr17	4373766	2715519	62%
Agh-ggis-spr17	353772	90790	26%
Bet-fal17	621508	101715	16%
Iku-fal17	3653791	102996	3%
Mary-spr17	169069	2177	1%
Muni-fi-spr16	48183	7371	15%
Muni-fsps-spr17	33810	2580	8%
Muni-pdf-spr16c	915725	43754	5%
Pu-llr-spr17	178424	47201	26%
Tg-fal17	77498	8116	10%

Table 5. Preprocessing results on 10 middle instance datasets

Dataset	Number of Candidates Schedule	Number of Reduced Schedule Candidates	Percentage Reduction
Agh-ggos-spr17	2331431	1155289	50%
Agh-h-spr17	2693914	1539477	57%
Lums-spr18	591475	199391	34%
Muni-fi-spr17	57843	8508	15%
Muni-fsps-spr17c	439903	84340	19%
Muni-pdf-spr16	915752	43754	5%
Nbi-spr18	144682	3000	2%
Pu-d5-spr17	86802	9991	12%
Pu-proj-fal19	1937292	259255	13%
Yach-fal17	93648	3857	4%

Preprocessing results on 10 late instance datasets

Dataset	Number of Candidates Schedule	Number of Reduced Schedule Candidates	Percentage Reduction
Agh-fal17	6012899	2574797	62%
Bet-spr18	649353	98402	15%
Iku-spr18	3588585	122101	16%
Lums-fal17	606929	220968	36%
Mary-fal18	187962	5295	3%
Muni-fi-fal17	46341	6374	14%
Muni-fspsx-fal17	579535	156832	8%
Muni-pdfx-fal17	5054749	1223449	24%
Pu-d9-fal19	677889	79274	12%
Tg-spr18	85330	11529	10%

c. Validate the final solution results

The final stage to ensure the results of the research are valid, validation of the final solution is carried out on the ITC 2019 website. The results are as in Figure 8, the 30 solutions produced have proven to be solutions that do not violate existing hard constraints by storing these solutions on the validator web. The penalty results for each solution are also stored on the website.

Table 6. Initial Solution and Final Solution on 10 dataset early instances

Dataset	Early Solution Penalty	Final Solution Penalty	Percentage Reduction
Agh-fis-spr17	38233	10858	72%
Agh-ggis-spr17	199432	107558	46%
Bet-fal17	414061	385290	7%
Iku-fal17	162211	112187	31%
Mary-spr17	77804	29161	63%
Muni-fi-spr16	24471	11222	54%
Muni-fsps-spr17	277511	150459	46%
Muni-pdf-spr16c	600126	517913	14%
Pu-llr-spr17	140771	55275	61%
Tg-fal17	26072	14548	44%

Initial Solution and Final Solution on 10 middle instance dataset

Dataset	Early Solution Penalty	Final Solution Penalty	Percentage Reduction
Agh-ggos-spr17	77059	22730	71%
Agh-h-spr17	55312	32717	41%
Lums-spr18	1638	361	78%
Muni-fi-spr17	23116	10158	56%
Muni-fsps-spr17c	657434	488529	26%
Muni-pdf-spr16	337279	212005	37%
Nbi-spr18	128733	58516	55%
Pu-d5-spr17	58746	30762	48%
Pu-proj-fal19	931627	831950	11%
Yach-fal17	29297	8382	71%

Initial Solution and Final Solution on 10 late instance datasets

Dataset	Early Solution Penalty	Final Solution Penalty	Percentage Reduction
Agh-fal17	552095	493834	11%
Bet-spr18	482804	443992	8%
Iku-spr18	198190	149340	25%
Lums-fal17	2695	1467	46%
Mary-fal18	59431	23326	61%
Muni-fi-fal17	30133	12044	60%
Muni-fspsx-fal17	1132578	1002804	11%
Muni-pdfx-fal17	952705	871244	9%
Pu-d9-fal19	627582	480907	23%
Tg-spr18	104276	36242	65%

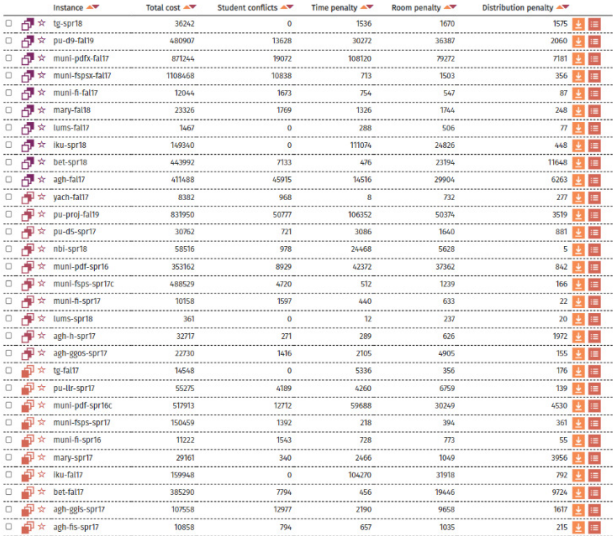

Figure 8. Validation Results

4. Conclusion

This research focuses on solving complex scheduling problems using the latest dataset from ITC 2019. The solution to this problem is carried out in two phases. The first phase is building an initial solution without breaking existing hard constraints. The second phase is optimizing the solution to make it better by reducing the number of penalties for violations of soft constraints.

The first phase is done by implementing the elimination of schedule candidates when preprocessing data and applying the IFS algorithm with some changes. As a result, all ITC 2019 datasets (30 datasets) were found to be viable initial solutions without breaking the hard constraints.

The second phase is optimization by applying the Great Deluge algorithm with a hyper heuristic approach using mutation LLH. The results obtained an average penalty reduction of 42%.

Reference

1. Lindahl, M., Mason, A. J., Stidsen, T. and Sørensen, "A strategic view of University timetabling," *European Journal of Operational Research,* vol. 226, no. 1, pp. 35-45, 2018.
2. V. I. Skoullis, . I. X. Tassopoulos and G. N. Beligiannis, "Solving the high school timetabling problem using a hybrid cat swarm optimization based algorithm," *Applied Soft Computing,* vol. 52, pp. 277-289, 2017.
3. 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 and Operations Research,* pp. 1-20, 2020.
4. J. S. Tan, S. L. Goh, G. Kendall and N. R. Sabar, "A survey of the state-of-the-art of optimisation

methodologies in school timetabling problems," *Expert Systems With Applications,* vol. 165, 2021.

5. T. Thepphakorn and P. Pongcharoen , "Performance improvement strategies on Cuckoo Search algorithms for solving the university course timetabling problem," *Expert Systems with Applications,* no. 161, 2020.
6. L. Saviniec and A. A. Constantino, "Effective local search algorithms for high school timetabling problems," *Applied Soft Computing,* vol. 60, pp. 363-373, 2017.
7. A. Rezaeipanah, S. S. Matoori and G. Ahmadi , "A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search," *Applied Intelligence,* no. 51, p. 467–492, 2021.
8. T. Song, S. Liu, X. Tang, X. Peng and M. Chen, " An iterated local search algorithm for the University Course Timetabling Problem," *Applied Soft Computing,* vol. 68, pp. 597-608, 2018.
9. T. M¨uller, R. Bart´ak and H. Rudov´a, "Iterative Forward Search Algorithm: Combining Local Search with Maintaining Arc Consistency and a Conflict-Based Statistics," *Principles and Practice of Constraint Programming - CP,* vol. 3258, 2004.
10. Rudová, H., Müller, T. and Murray, K., "Complex university course timetabling," *Journal of Scheduling,* vol. 14, no. 2, p. 187–207, 2010.
11. A. Muklason, G. B. Syahrani and A. Marom, "Great Deluge Based Hyper-heuristics for Solving Real-world University Examination Timetabling Problem: New Data set and Approach," *The Fifth Information Systems International Conference 2019,* pp. 647-655, 2019.
12. T. M¨uller , ·. H. Rudov´a and Z. M¨ullerov´a, "University course timetabling and International Timetabling Competition 2019," *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018),* pp. 5-31, 2018.