Limited Searching by Problem Category in State Space Tree to Solve Timetable Scheduling Problems

Jamaludin Hakim[1], Retantyo Wardoyo[2*], Sri Hartati[3]

[1,2,3] Department of Computer Science and Electronics
Universitas Gadjah Mada
D.I. Yogyakarta, Indonesia
jamaludin.hakim[at]mail.ugm.ac.id
rw[at]ugm.ac.id* (corresponding author)
shartati[at]ugm.ac.id

Abstract: ***The main problem in making timetable schedules is finding a solution slot that can meet the minimum tolerable constraints. The search for solution slots aims to place all timetable components in timetable slots that fit the constraints. The searching for solution slots does not have to be carried out on all slots in the timetable but is carried out only on slots that have the potential to provide solutions based on problems that arise. The limited search to solution slots in the state space tree allows for finding the best solution slot based on the problems that arise. Limited searching allows the resulting timetable scheduling to meet tolerable constraints and simplify the searching process.***

Keywords: Solution slot, State space tree, Limited searching, Timetable component (TC), Timetable media (TM)

1. Introduction

Timetable scheduling at universities are formed based on mandatory provisions (hard constraints) and based on tolerable conditions (soft constraints). The placement of timetable components (TC) in accordance with the provisions stipulated in the timetable media (TM) slots is a major concern in the process of forming a timetable scheduling. Until now, researchers are still looking for the most appropriate method in placing timetable components in timetable media.

One of the methods in establishing the timetable is to swapping slots for each timetable component that does not meet the constraints. The slot swapping method is carried out with the expectation that the timetable component will find slots that meet the constraints. The swapping of slots can be done by various methods. Either done randomly or follow a certain pattern or certain calculations.

Swapping can be made if the Timetable Component (TM), which consists of lecturers, Department, courses, semesters, classes and the number of participants, can meet the expected Constraints. In the swapping of TM, what need to be tested whether it meets the Constraints is TC for which a solution will be found (TCi) and TC which occupies the solution slot (TCj).

Solution slots are randomly distributed on the Timetable media (SM). The easiest step is to experiment with swapping all existing slots until the desired solution is found. However, this step requires a lot of effort. Another step is to mapping the slots that are candidate solutions. Determination of candidate solutions will map solution slots. Determination of solution slots is to take all slots in all time periods and all days, in rooms that meet capacity.

The searching process is a factor that is one of the key determinants in finding a mapped solution slot. Not all of the mapped solution slots must be checked whether they match or not. Checking the solution slots will be carried out based on the problem categories that have been defined, so the solution slots must be formed in a state space tree that describes the categories of problems that arise.

2. Previous work

Swapping the position of the timetable components in the solution slots is one of the effective ways to find solutions in Timetable scheduling. Genetic Algorithm (GA), is one method that swap slots to find solutions in scheduling [5]. The swapping of positions in the slots that are considered to be a solution is also carried out when using the Ant Colony Optimization method (ACO). Swapping of solution slots is carried out if it is considered that the swap will result in a better timetable [13].

The combination of GA and Simulated Annealing (SA) creates a random step of mutation which is carried out by greedy stochaltic local search and then will be selected randomly through the swap of chromosomes which are considered to represent the appropriate solution slots. The criteria for stopping iterations in the swap of chromosomes/solution slots are determined by SA [12].

The swapping of solution slots which are considered potential for solving problems is still used in research for timetable scheduling problems. The main objective of finding solutions to timetable scheduling problems is the placement of Timetable Components in appropriate slots where the potential for violations of the constraints can be minimized.

3. Timetable scheduling problem

The formulation that describes the problems that arise in the timetable is needed so that the desired solution can be found. The formulation of the problem will be described in a mathematical model so that it is easy to know the provisions that are applied, but before that, the following is a description of the problem which is divided into two categories, namely Hard Constraints (HC) and Soft Constraints (SC).

Hard Constraints (HC)

- Lecturers are only scheduled once at a certain time (HC1)
- Room is only scheduled once at a certain time (HC2)
- Courses at the same semester level are only scheduled once at a certain time (HC3).

Soft Constraints (SC)

- All courses chosen by students at different semester levels should be scheduled once at a certain time (SC1).
- Lecturer teaching time should refer to the lecturer's teaching time preference (SC2).
- Rooms are scheduled based on room capacity (SC3).

So when formulated for each Constraint is as follows:

$$\text{HC1} \quad \sum_{d=1}^{n}\sum_{p=1}^{b}\sum_{r=1}^{m} D_{d,p,r} \leq 1 \tag{1}$$

$$\text{HC2} \quad R_{d,p,r} \leq 1 \tag{2}$$

$$\text{HC3} \quad \sum_{d=1}^{n}\sum_{p=1}^{b}\sum_{r=1}^{m} M_{d,p,r} \leq 1 \tag{3}$$

$$\text{SC1} \quad \sum_{d=1}^{n}\sum_{p=1}^{b}\sum_{r=1}^{m} MP_{d,p,r} \leq 1 \tag{4}$$

$$\text{SC2} \quad D_{d,p,r} \in P \tag{5}$$

$$\text{SC3} \quad KR_{d,p,r} \geq Pst \tag{6}$$

Where,

$$D_{d,p,r} = \begin{cases} 1, & D \text{ appears in the slot } d,p,r \\ 0, & D \text{ does not appear in the slot } d,p,r \end{cases} \tag{7}$$

$$R_{d,p,r} = \begin{cases} 1, & R \text{ Scheduled 1 course} \\ 2, & R \text{ Scheduled more than 1 course} \\ 0, & R \text{ if the room is not scheduled} \end{cases} \tag{8}$$

$$M_{d,p,r} = \begin{cases} 1, & M \text{ appear in the slot } d,p,r \\ 0, & M \text{ does not appear in the slot } d,p,r \end{cases} \tag{9}$$

$$MP_{d,p,r} = \begin{cases} 1, & MP \text{ appear in the slot } d,p,r \\ 0, & MP \text{ does not appear in the slot } d,p,r \end{cases} \tag{10}$$

For r=room, p=period, d=day, b= weight, n= number of days, m=number of rooms, D=lecture, R= used classroom, M= subject (the same semester, class, and Department), MP= the subjects chosen by students at different semester levels, P= lecturer's teaching time preference, Pst= lecture participants, KR= room capacity.

4. Timetable scheduling construction

The formation of a timetable scheduling with a certain method requires media and compositions that must be formulated. The timetable will be formed in a Timetable Media (TM) which consists of day, period and room components. The combination of the three components in the timetable media will provide information on the position of a slot. Determining and searching for slots on the TM will be the main factor in the formation of the timetable. While the courses, lecturers, number of participants and classes are components of the timetable (TC) which will be placed on the TM.

The process of finding solution slots will begin with the placement of TC on TM based on constraints. TCs that do not find slots that meet the constraints will be collected for the process of searching for solution slots. Next, Any KJ that does not meet the constraints will be placed in empty slots without checking constraints. Furthermore, TC that has been placed without checking for constraints is checked for violations that arise against constraints in the slot. The process is shown in the model in Figure 1.

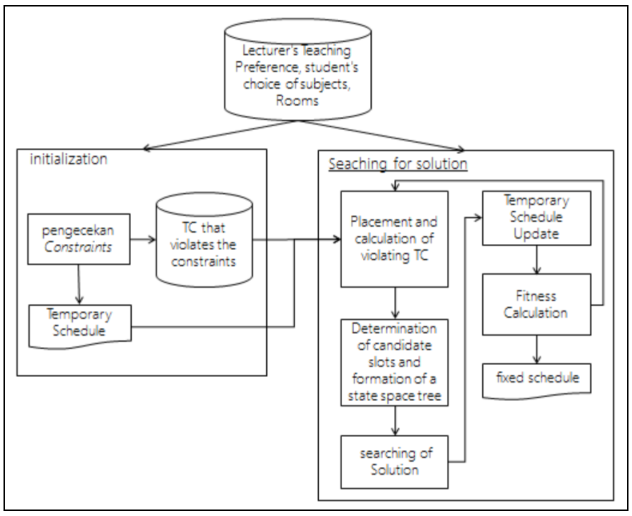

Lecturer's Teaching Preference, student's choice of subjects, Rooms
initialization
pengecekan Constraints
TC that violates the constraints
Temporary Schedule
Seaching for solution
Placement and calculation of violating TC
Determination of candidate slots and formation of a state space tree
searching of Solution
Temporary Schedule Update
Fitness Calculation
fixed schedule

Figure 1: The Model of Timetable scheduling with limited searching

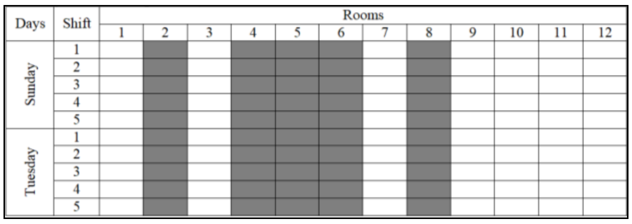
Based on the results of the examination of the constraints, the number of solution slots can be determined and a state space tree can be formed. Limited searching aims to find slots that if tested with constraints will meet fitness. Constraint testing is not only carried out on TC for which a solution is sought (TCi), but also on TC that occupies the solution slot (TCj). The slot swapping will be carried out if the TC test has met the fitness. The fitness calculation was performed on TCi and TCj. The search for a solution slot will be carried out in the next iteration/Level if it does not meet the fitness, and the timetable construction process will stop if the fitness has been met.

4.1 Determination of solution slots

Solution slots need to be determined which ones have potential. The number and position of the solution slots to be used depends on the number of participants taking a course at TC. The number of participants will determine the classrooms that can be used. The more participants TC has, the fewer the number of solution slots. This is related to the smaller number of large capacity rooms and later the rooms will be categorized based on room capacity (RC). The number of solution slots (SoS) formed is obtained by using equation (11).

SoS=#day*#period*#RC (11)

Visually, the solution slots will spread throughout the day and period, and in the rooms that fall into categories according to the capacity of the participants. Figure 2, for example describing the position of the solution slot in rooms with a large capacity, so that only a few rooms are included in the solution slot.



Days	Shift	1	2	3	4	5	6	7	8	9	10	11	12
		Rooms											
Sunday	1		▨		▨	▨	▨		▨				
	2		▨		▨	▨	▨		▨				
	3		▨		▨	▨	▨		▨				
	4		▨		▨	▨	▨		▨				
	5		▨		▨	▨	▨		▨				
Tuesday	1		▨		▨	▨	▨		▨				
	2		▨		▨	▨	▨		▨				
	3		▨		▨	▨	▨		▨				
	4		▨		▨	▨	▨		▨				
	5		▨		▨	▨	▨		▨				

Figure 2: The Model of Timetable scheduling with limited searching

The process of forming solution slots is a step related to the construction of a state space tree. Each solution slot formed will be specific to each TC for which a solution will be sought. Prior to the construction of the solution slot, violations that occurred for each TC that had not occupied a slot in the initialization process would be calculated. The calculation of violations will determine the category of violations owned by TC. The problem category will be used for the creation of the state space tree and the search process. The determination of the violation is determined based on what constraints are violated (12).

There are three main problems that arise in the violation checking process. First, the schedule component does not meet all the constraints. Second, schedule components meet the constraints except SC3. Third, the schedule component only meets SC3. Table 1, is an explanation of the categories of violations and the actions that will be taken when the construction of the state space tree and searching of solution slots. The problem category will be a guide in the process of construction the state space tree. The category of violation determines the movement searching of solution slot (Table 2). The problem category is the key in the limited searching process.

$$CoV = \begin{cases} 1, & \text{Violation Only on } SC3 \\ 2, & \text{All Constraints violated Except } SC3 \\ 3, & \text{All Constraints violated} \end{cases} \qquad (12)$$

Table 1: Types of violations, problem categories and actions

Violation	Problem Categories	Actions
SC3	1	Moving room without moving period
HC1-HC3, SC1, SC2	2	Moving period without moving room
HC1-HC3, SC1-SC3	3	Moving period and Moving room

The category of violation (CoV) will consist of three parts. If the violation only occurs in the 3rd Soft Constraint (SC3), then it becomes a first category (1st category). For violations that occur on all constraints except SC3, it will meet the second problem category (2nd category). If all constraints are violated, it will meet the third problem category (3th category).

Table 2: Movement for next solution slot

Problem category	Solution movement
1	r → next RC
2	p → p+1
3	p → p+1 and r → next RC

4.2 Construction of the State Space Tree

The formation of a state space tree refers to the category of problems. The problem category is divided into three parts causing the state space tree to have tree nodes. The first node will contain the solution slots for the first category, the second node will contain the second category solution slots and the third node will contain the solution slots for the third category. The state space tree will represent the position of the solution slots based on the problem category.

Figure 3, is an illustration of a state space tree based on problem categories. The tree will be built on three nodes representing the problem categories. In Figure 4, the tree is defined as consisting of three nodes and three pieces of data containing information on day, period and space. The state space tree will be construction according to the problem category. Each node will contain nodes of all categories of problems, which can be construction nodes at the next level.

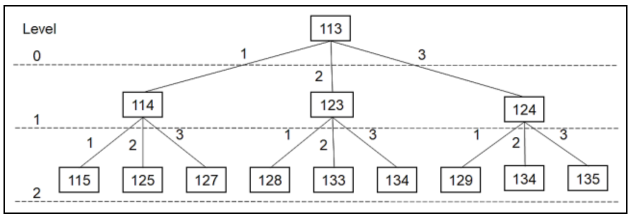
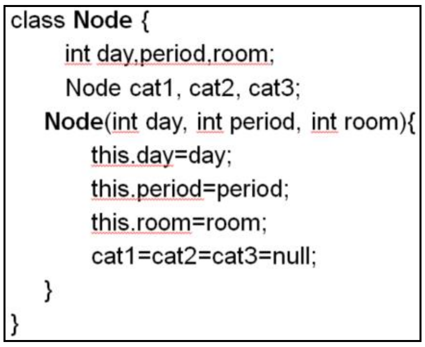


Figure 3: State space tree visualization

Visually in Figure 3, Day, period and room information will be placed on nodes based on the problem category. Taking the next solution slots that will be placed on the tree, will refer to the last position of the slot. While the movement of taking slots can be seen in Table 2.



```
class Node {
    int day,period,room;
    Node cat1, cat2, cat3;
  Node(int day, int period, int room){
      this.day=day;
      this.period=period;
      this.room=room;
      cat1=cat2=cat3=null;
  }
}
```

Figure 4: State space tree class node

If the problem category is one, then the next solution slot is to take a slot in the next room category (RC) without moving the period and day. If the problem category is two, from the last slot position, take the slot in the next period without moving room and day. For violation with category three, take a slot after moving room and Period.

The solution slots that have been determined based on the problem category are placed on the nodes according to the problem category. The root node will contain the solution slot with the smallest day, period and room number from the group of solution slots that appear. For example in Figure 2, the room category where the solution slots at room 2,3,4,5,7,8, then the sequence of the rooms will refer to that sequence. So if the slot position on the node above it is in room 2, for nodes in category 1, the solution slot room is in room 3 (next room category).

The 2nd node will give a slot with a category 2 problem. The solution to the category 2 problem is a moving of period. The solution slot that will occupy the 2nd node is the slot in the next period from the slot in the node above it (period ←period+1). The 3rd node of the root will contain a solution slot for category three. The solution to the problem in category three is the moving of the room category and the moving of the period from the slot on the parent node.

4.3 Fitness Calculation

The process of searching solution slots in the state space tree can occur in several iterations. The iteration will continue until the desired conditions are met. There are several conditions that must be met in order for the iteration to be stopped. There are minimum conditions that must be met so that the process of searching the solution slot is considered complete. The minimum conditions are constructed in certain formulas which are used as fitness in the searching. Each of iteration means that it is conducted at a certain level in the state space tree.

There are two types fitness that must be met to complete the searching process. The first is the fitness for each constraint and the second is the fitness for the entire constraints. The fitness of each constraint will limit the conditions that must be met. Constraints HC1, HC2, HC3 are constraints that must be met, while SC1, SC2, SC3 are constraints that can be tolerated if they are not met.

$$f(HC1) = \sum_{d=1,p=1,r=1}^{n,n,o} HC1_{d,p,r} = 0 \tag{13}$$

$$f(HC2) = \sum_{d=1,p=1,r=1}^{n,m,o} HC2_{d,p,r} = 0 \tag{14}$$

$$f(HC3) = \sum_{d=1,p=1,r=1}^{n,m,o} HC3_{d,p,r} = 0 \tag{15}$$

$$f(SC1) = \sum_{d=1,p=2,r=1}^{n,m,o} SC1_{d,p,r} \leq 1 \tag{16}$$

$$f(SC2) = \sum_{d=1,p=1,r=1}^{n,m,o} SC2_{d,p,r} \leq 1 \tag{17}$$

$$f(SC3) = \sum_{d=1,p=1,r=1}^{n,m,o} SC3_{d,p,r} = 0 \tag{18}$$

Each TC will be checked for constraints for all slots in that period (d,p,r=1…o). d is a position or describes day, p is a position that describes the period and r is a position that describes the room, and o describes the amount of room in TM.

Hard Constraints (HC1, HC2, HC3) must be met so that it must be scored 0 (13)(14)(15). Soft Constraints (SC1, SC2) can be not met or can be scored 1 (16)(17), meanwhile SC3 must be met or must be scored 0 in the test (18). Testing the fitness value for each constraint on the position of the solution slot and replacement slot, can be called a local fitness test (19).

$$f(TC) = \frac{f(HC1) + f(HC2) + f(HC3) + f(SC1) + f(SC2) + f(SC3)}{m} \leq 0{,}33 \tag{19}$$

The local fitness f(TC) test is done by summing the fitness of each constraint and dividing by the number of slots in one period (m). If the result is less than or equal to 0.33 (there are a maximum of 2 violations) then TC can be placed in that slot.

4.4 Limited searching

A limited search to find a solution slot will be performed on the state space tree. After the tree is constructed, the searching will start from the root node of the tree. The solution slot data on the root node (day, period, room) will be the initial solution slot for TCi. The calculation of the constraints will be carried out on TCi if it is in the solution slot whose data is taken from the node using equation (20). The position of the slot where the TC is located will be labeled with the letter "i", and the position of the slot in the node that is a candidate for the solution slot will be labeled with the letter "j". If the solution slot has been occupied by TCj, a violation calculation will be carried out on TCj with the slot occupied by TCi using equation (21).

$$f(TC)_i = \frac{f(HC1)_j + f(HC2)_j + f(HC3)_j + f(SC1)_j + f(SC2)_j + f(SC3)_j}{m} \leq 0{,}33 \tag{20}$$

$$f(TC)_j = \frac{f(HC1)_i + f(HC2)_i + f(HC3)_i + f(SC1)_i + f(SC2)_i + f(SC3)_i}{m} \leq 0{,}33 \tag{21}$$

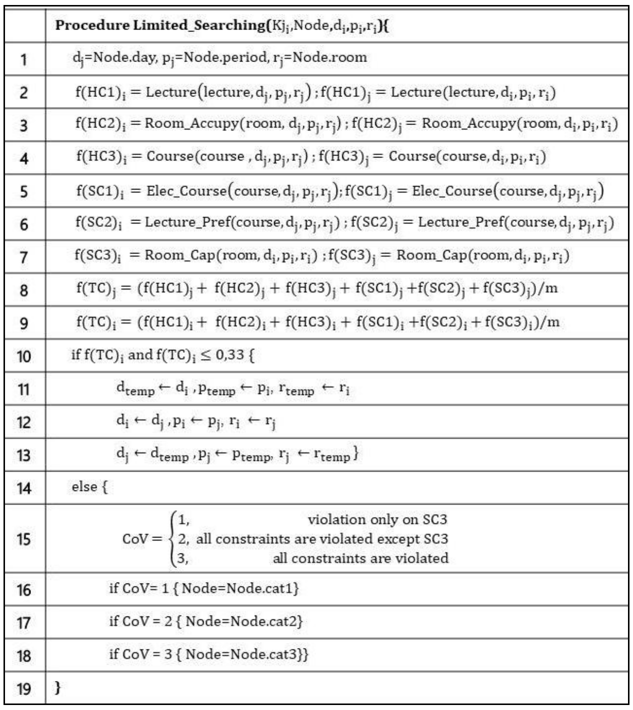
If one of the calculations in equation (20) or (21) does not meet the fitness, then the searching will continue on the node at the next level (child node). The searching for nodes in the next level will depend on the problem category. The category of violation of equation (12) depends on the results of the fitness calculation TCi (20). Furthermore, the slot information (d,p,r) on the node will be used for calculating the fitness f(TC)i (20). After the calculations in equations (20) and (21), the next step to take if f(TC)i and f(TC)j have been met fitness is to swapping slot positions between KJi and KJj. The limited searching algorithm directs the searching to the slots that have the potential to reduce the violation of the constraints (Figure 5). The Searching that refer to the problem category, direct the searching so that the slot visited is a slot that is still close to the slot that was first offered in the tree. So that the searching movement on the node will reduce the violations encountered from the slots on the parent node.

In Figure 5, line 10 checks whether the fitness TCi and TCj are less than or equal to 0.33. If it meets then swapping the slot position (line 11 to 13). If the fitness value is not met, the slot searching will be carried out at the next level node (child) based on the problem category. On line 15 of the limited search algorithm, testing the category of violations encountered (CoV) was carried out. Based on the results of

CoV, the searching for slots at the child node depends on the category of violation (line 16 to 18).

5. Result and discussion

Initialization process is important in solving timetable scheduling problems using limited searching. Initiation process is to place TC on TM. The TC sorting method in placing it on TM (Greedy) affects the number of TC results that do not get a slot. There are 566 TCs that need to be plotted on the TM which has 609 occupied slots.



```
    Procedure Limited_Searching($Kj_i$,Node,$d_i$,$p_i$,$r_i$){
1     $d_j$=Node.day, $p_j$=Node.period, $r_j$=Node.room
2     $f(HC1)_i = Lecture(lecture, d_j, p_j, r_j)$ ; $f(HC1)_j = Lecture(lecture, d_i, p_i, r_i)$
3     $f(HC2)_i = Room\_Accupy(room, d_j, p_j, r_j)$ ; $f(HC2)_j = Room\_Accupy(room, d_i, p_i, r_i)$
4     $f(HC3)_i = Course(course, d_j, p_j, r_j)$ ; $f(HC3)_j = Course(course, d_i, p_i, r_i)$
5     $f(SC1)_i = Elec\_Course(course, d_j, p_j, r_j)$; $f(SC1)_j = Elec\_Course(course, d_j, p_j, r_j)$
6     $f(SC2)_i = Lecture\_Pref(course, d_j, p_j, r_j)$ ; $f(SC2)_j = Lecture\_Pref(course, d_j, p_j, r_j)$
7     $f(SC3)_i = Room\_Cap(room, d_i, p_i, r_i)$ ; $f(SC3)_j = Room\_Cap(room, d_i, p_i, r_i)$
8     $f(TC)_j = (f(HC1)_j + f(HC2)_j + f(HC3)_j + f(SC1)_j + f(SC2)_j + f(SC3)_j)/m$
9     $f(TC)_i = (f(HC1)_i + f(HC2)_i + f(HC3)_i + f(SC1)_i + f(SC2)_i + f(SC3)_i)/m$
10    if $f(TC)_i$ and $f(TC)_j \le 0{,}33$ {
11        $d_{temp} \leftarrow d_i$ , $p_{temp} \leftarrow p_i$, $r_{temp} \leftarrow r_i$
12        $d_i \leftarrow d_j$ , $p_i \leftarrow p_j$, $r_i \leftarrow r_j$
13        $d_j \leftarrow d_{temp}$ , $p_j \leftarrow p_{temp}$, $r_j \leftarrow r_{temp}$}
14    else {
15        $CoV = \begin{cases} 1, & \text{violation only on SC3} \\ 2, & \text{all constraints are violated except SC3} \\ 3, & \text{all constraints are violated} \end{cases}$
16        if CoV= 1 { Node=Node.cat1}
17        if CoV = 2 { Node=Node.cat2}
18        if CoV = 3 { Node=Node.cat3}}
19  }
```

Figure 5: Limited searching algorithm

Placement of TC on initialization if it is done by sorting using the largest weight first, the TC that does not get a slot is 55. Placement based on the smallest weight first, the TC that does not get a slot is 54. Placement based on the largest TC participant first, the TC that does not get a slot is 14. There is a significant difference when sorting using the number of TC participants, which means that the placement of TCs with a large number of participants causes more TCs to get slots.

There are 14 TCs that haven't got a slot, which needs to find the slot that meets the constraints. After the initialization process, it was identified that the percentage of constraints violated for the 14 TCs in Figure 6.

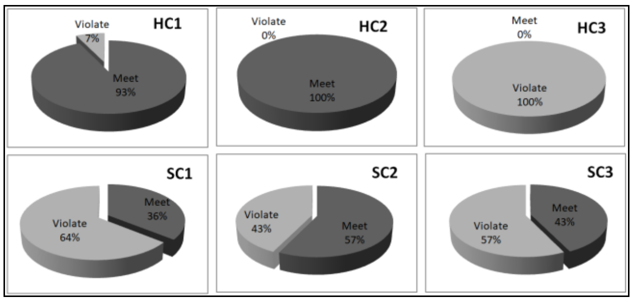


Figure 6: Percentage of TCi on constraints after initialization

In Figure 6, From the constraints that must be met (not to be violated) (HC1, HC2, HC3), there are HC1 which are still 7% violated and HC3 are 100% violated. While the constraints that should be met (SC1, SC2, SC3), all these constraints are violated by varying numbers.

After searching the state space tree to find solution slots and successfully swapping TCi, the results obtained are shown in Figure 7.

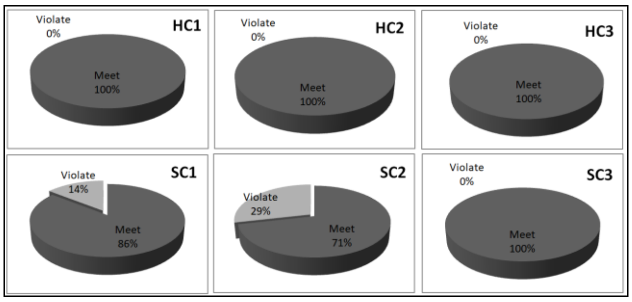


Figure 7: Percentage of TCi on constraints after limited searching

Result after limited searching, all Hard Constraints have been met. Meanwhile, there are Soft Constraints that cannot be met, namely SC1 violates 14% and SC2 violates 29%. That means there are about 14% and 29% (SC1 and SC2) of the 14 TCi that have not been met. SC1 is a student elective course in a different semester, where 14% of the time placement is still the same as other elective courses. SC2 is a constraint that lecturers are occupied with the desired teaching time preferences, so there are 29% of lecturers' teaching time preferences that cannot be met.

It should also be noted that the TC slot is occupied a solution slot (TCj), in swapping for the TCj slot; it must also get a slot that meets the constraints. Figure 8 shows the percentage of TCj after the process of searching and swapping slots.

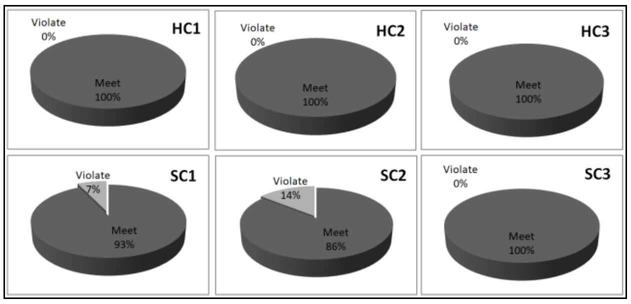


Figure 8: Percentage of TCj on constraints after Limited Searching and swapping slots

HC1, HC2 and HC3 of TCj have met the constraints. However, for SC1 and SC2 there are still 7% and 14% violations. So that there are 7% of different semester elective courses that violate their time placement and 14% of lecturers' teaching time preferences that are violated.

Student elective courses that still do not meet the constraints can be caused by students choosing general courses and or courses with a large number of participants. The Lecturers teaching time preference that cannot be met by TCi and TCj could be due to a less wide preference for teaching time, which is combined with the number of courses taught by a lecturer. The rules regarding teaching time preferences must be analyzed so that teaching preferences can be met.

The number of nodes that must be visited to find a solution slot is not the same for each TC. There is a root node/level 0, a solution slot has been found. But there are also those that up to the 10th level node, a solution slot are found (Table 3). In Table 3, there are several TC that have to go up to level 10 in the state space tree to get a slot that can meet the conditions. Although not all TCs do that, because the search process is limited, so it doesn't continue to searching at the next level if it meets fitness.

Table 3: Movement for next solution slot

Timetable Component	At Level	Number of Violation	
		i	j
1	10	1	1
2	6	0	0
3	6	1	0
4	0	0	1
5	0	1	0
6	5	1	1
7	6	1	1
8	0	0	1
9	10	1	0
10	6	0	1
11	1	1	2
12	4	0	1
13	2	1	0
14	0	0	0

The use of limited searching to state space trees, providing a new perspective in solving timetable scheduling problems. The existence of a problem category has a very good impact on the searching process of the expected solution slots. There is a decrease in the violation of constraints on every visit to the next level node. Not all of the TCs experienced a decrease in the number of violations when moving to the next level, but almost all showed a decrease in the number of violations when moving on to the next level as shown in Figure 9. In Figure 9, the number of violations tends to decrease with each level moving. The decrease in the number of violations each time you enter the next level (child) in the searching, has an indication that the slot found is a suitable solution slot. This shows that if you searching using problem categories on child nodes, you will find nodes that contain slots that are better than the parent node slots. So that a solution slot that meets fitness will be found.

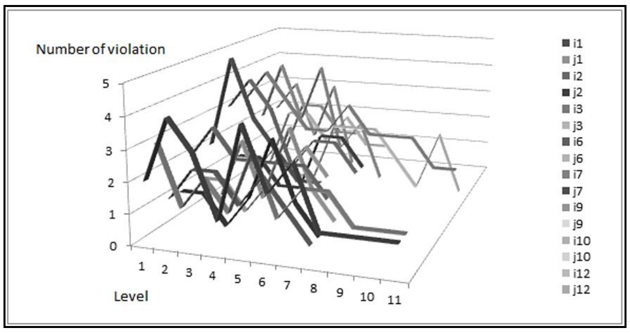


Figure 9: TCi and TCj graphs on decreasing the number of constraints violation at each Level in limited searching

In general, the limited searching process has solved the timetable scheduling problem. Constraints that have not been met have become a condition that needs to be found a way to solve them. However, with limited search, 70% of the constraints have been met (4 of 6 constraints). This method needs improvement, such as adding fitness. Addition of fitness can be done by comparing the difference in violations at each level node visited with the previous level node. If there is a decrease in the number of violations then the searching can continue to the next level. If there are more violations, the slot with the smallest difference in violations is taken.

Searching for solution slots by using the problem category provides a new view that the search will be more efficient in finding solutions. The accuracy in finding the best solution slot cannot be said to be perfectly successful because there are still violations that occur. However, the violations that occur still meet fitness. But there are also TCs that do not have violations with this method (TCs 2 and 14 in Table 3). Limited searching by problem category, will direct the searching to the slot that is expected to provide a solution. So it can be concluded that the limited searching based on the problem category, has found a solution according to fitness.

6. Conclusion

Sorting in initialization is quite influential in SC assignment. Proper sorting will result in SCs that haven't got fewer slots. Fewer SCs need to find a solution slot, reducing the displacement of SCs that meet constraints.

Limited searching based on the problem category in the state space tree to get a solution slot that satisfies the fitness. Searching by problem category directs searching based on problems that arise when testing constraints. Each level moving in the searching there is a reduction in the number of violations until it meets fitness. The limited searching by problem category leads to a searching that leads to a solution slot that satisfies the fitness effectively.

Further research needs to be done for fitness which is not only on the calculation of violations, but also on changes in violations that occur during constraints testing. The smaller the number of violations that appear, the searching will continue until there are no changes or the number of violations does not decrease. This method can be applied to all timetable schedules that can be presented in the form of rows and columns (grids).

Acknowledgment

This paper using data of Faculty of Mathematics and Natural Sciences, Universitas Gadjah Mada (FMIPA UGM) Yogyakarta Indonesia. The data is closed and not distributed outside UGM. The use of space together by several departments in FMIPA makes scheduling quite complex and deserves to be researched.

References

[1] Al-Mahmud and M.A.H Akhand, ‘ACO with GA operators for solving University Class Scheduling Problem with flexible preferences’, 2014 International Conference on Informatics, Electronics and Vision, ICIEV 2014. doi: 10.1109/ICIEV.2014.6850742

[2] C. K. Teoh, A. Wibowo and M. S. Ngadiman, ‘Review of state of the art for metaheuristic techniques in Academic Scheduling Problems’, Artificial Intelligence Review, 44(1), pp. 1–21., 2015, doi: 10.1007/s10462-013-9399-6

[3] D. Adrianto, ‘Comparison using particle Swarm optimization and genetic algorithm for timetable scheduling’, Journal of Computer Science, 10(2), pp. 341–346, 2014, doi: 10.3844/jcssp.2014.341.346.

[4] D. Suryadi and R. Pilipus, ‘Genetic Algorithm for University Timetable Planning in FTI’, in Proceeding of the 2012 International Conference on Industrial Engineering and Operation Management. Istambul, Turkey, pp. 656–664.,2012

[5] D. T. Long, ‘A Genetic Algorithm Based Method For Timetabling Problems Using Linguistics Of Hedge Algebra In Constraints’, Journal of Computer Science and Cybernetics, 32(4), pp. 285–301., 2017, doi: 10.15625/1813-9663/32/4/7962.

[6] E. A. Abdelhalim and G.A. El Khayat,‘A Utilization-based Genetic Algorithm for Solving the University Timetabling Problem (UGA)’, Alexandria Engineering Journal, 55(2), pp. 1395–1409, 2016, doi: 10.1016/j.aej.2016.02.017.

[7] F. Alaul, I. Zabalawi, and A. Wasfy, ‘A Sat-Based Approach To Solve The Faculty Course Scheduling Problem’ Department of Computer Science & Engineering American University of Sharjah , UAE’,2013

[8] L. Yang and C. Xie, ‘Research on Model of Course Scheduling System for Ideological and Political Teaching in Colleges and Universities Based on Particle Swarm Optimization’, 32, pp. 657–663., 2017

[9] M. Jouya and S. Khayati, ‘Review Local Search Algorithms In Artificial Intelligence’, International Academic Journal of Science and Engineering, 4(1), pp. 190–195.,2017

[10] R. A. Komijan and M. N. Koupaei, ‘a Mathematical Model for University Course Scheduling : a Case Study’, International Journal of Technical Research and Aplications, 19(19), pp. 20–25., 2015

[11] S. I. Hossain, S. et al., ‘Optimization of University Course Scheduling Problem using Particle Swarm Optimization with Selective Search’, Expert Systems with Applications, 127, pp. 9–24. 2019, doi: 10.1016/j.eswa.2019.02.026.

[12] S. Susan and A. Bhutani (2019) ‘A novel memetic algorithm incorporating greedy stochastic local search mutation for course scheduling’, Proceedings - 22nd IEEE International Conference on Computational Science and Engineering and 17th IEEE International Conference on Embedded and Ubiquitous Computing, CSE/EUC 2019, pp. 254–259., 2019, doi: 10.1109/CSE/EUC.2019.00056.

[13] T. Thepphakorn, P. Pongcharoen, and C. Hicks, ‘An ant colony based timetabling tool’, International Journal of Production Economics, 149, pp. 131–144.,2014, doi: 10.1016/j.ijpe.2013.04.026.

[14] W. Wen-Jing, (2018) ‘Improved adaptive genetic algorithm for course scheduling in colleges and universities’, International Journal of Emerging Technologies in Learning, 13(6), pp. 29–42., 2018, doi: 10.3991/ijet.v13i06.8442.