IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 27, Issue 1, Ser. 4 (Jan. – Feb. 2025), PP 24-32
www.iosrjournals.org

Developing Timetable Scheduling Software Based On The Test And Set Algorithm

Vuong Quoc Dung
Faculty Of Information Technology, Hanoi University Of Industry

Abstract
The problem of creating a timetable is a very difficult problem to solve, this is a problem belonging to the class of NP-complete. Nowadays, there are many algorithms to solve this problem, most of which are in the fields of artificial intelligence. The software products based on these algorithms, which are still at the level of supporting timetable scheduling tasks, still violate hard constraints such as one teacher teaching two classes at the same time, one classroom being assigned to two classes at the same time, or one class being scheduled to study two subjects at the same time Therefore, the executor of scheduling job still has to check, adjust manually, which is very complicated and tiring. Test and Set algorithm is used for allocating shared resources among multiple processes in computer operating systems and has proven to satisfy the requirements of mutual exclusion and bounded waiting. In this paper, I investigate and extend the Test and Set algorithm to apply it in solving the credit-based learning timetable scheduling problem at universities without violating the basic hard constraints as noted above. The achieved result is a software program which allocates and organizes classroom and teacher schedules for independent classes rationally without resource conflicts, thus saving time and effort for timetable schedulers (not having to check, review, and readjust the arrangements).
Keywords: *Test and Set; resources; process; mutual exclusion; bounded waiting; timetable schedule.*

Date of Submission: 12-02-2025 Date of Acceptance: 22-02-2025

I. Introduction

The problem of creating a timetable is a very difficult problem to solve, this is a problem belonging to the class of NP-complete. Currently, there are many algorithms to solve this problem including genetic algorithm [1], [3], Tabu Search algorithm, Simulated Annealing algorithm (SA) [4], [5], Variable Neighborhood Search algorithm (VNS), Memetic algorithm, Particle Swarm Optimization algorithm (PSO) [6] and Bees Algorithm, Branch and Bound algorithm, Hill Climbing algorithm, Graph Coloring algorithm, Approximation algorithm slag,... These are difficult algorithms, still violating the hard constraints of the scheduling problems. Figure 1 [1] below illustrates the experimental results of solving the problem using Genetic algorithms. The scheduling results still violate the hard constraints.

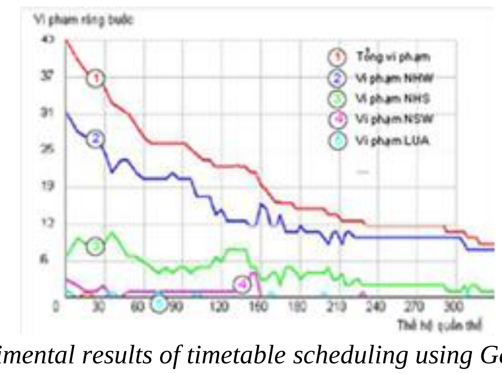


Figure 1. Experimental results of timetable scheduling using Genetic algorithms.

I realize that the timetable scheduling problem is a problem of scheduling, managing, distributing and exploiting the use of resources for classes in the school to ensure the smooth operation of classes, satisfying certain specified constraints

One of the basic functions of the operating system is to manage, distribute, exploit and effectively use the resources of the computing system. Currently there are many algorithms to solve this problem. Among them, the Test and Set[2] algorithm used to distribute a shared resource to n processes satisfies the requirements of mutual exclusion and bounded waiting.

The Test and Set algorithm can be extended to distribute multiple shared resources to multiple processes. This expanded algorithm can be applied in building a timetable program for schools. The idea of expanding the Test and Set algorithm is as follows: Each class corresponds to a process, and the simultaneous resources required for each class are classrooms, subjects, and teachers. Among these, classroom resources and teacher resources are two types that cannot be shared at the same time, these are time-shared resources, which are refered as classroom time resources and teacher time resources. And the algorithm's task is to reasonably allocate classroom time resources, subjects and teacher time resources for classes in the school.

II. Methodology

Some basic concepts

Critical resource: A resource that is simultaneously required by multiple concurrent processes, yet restricted by their shared usability.

Critical section: The critical section is a program segment that uses critical resources.

Critical section problem

Suppose there are n processes denoted as P0,P1,.. ,Pn-1 cooperating, each process has a critical segment. Find a method whereby the processes traverse their critical sections without affecting the operation of the entire system.

In operating system development, there is a software-based solution to the critical section problem. However, software-based solutions are not guaranteed to enable the operating system to function on modern computer architectures.

In the next part of the report, I present some solutions to tackle the critical section problem by leveraging advancements in hardware technology combined with available software API functions for kernel developers and application programmers. All of these solutions are based on the premise of using locks - that is, protecting critical sections through the use of locks. As we will see, designing locks as mentioned will be absolutely perfect.

We begin by examining a few simple hardware instructions that are available on many systems and find that they can be used effectively in solving the critical section problem. Hardware features make programming tasks easier and improve system efficiency.

The critical section problem can be solved simply in a single-processor environment, if we can prevent interrupts from occurring when a shared variable A is being modified. In this way, we can be sure that the current sequence of commands will be allowed to execute in an order without precedence. No other command can execute barge in, so there is no unexpected changes affecting the shared variable A.

Many modern computer systems provide special hardware instructions that allow us to examine and change the contents of a variable or swap the contents of two variables without interruption. We can use special instructions to solve problems in a relatively simple way.

The Test and Set() function can be defined as shown in figure 2 [2].

```
boolean Test_and_Set(boolean*target)
{
      boolean rv = *target;
      *target  = true;
      return rv;
}
```

Figure 2. The definition of Test and Set() function

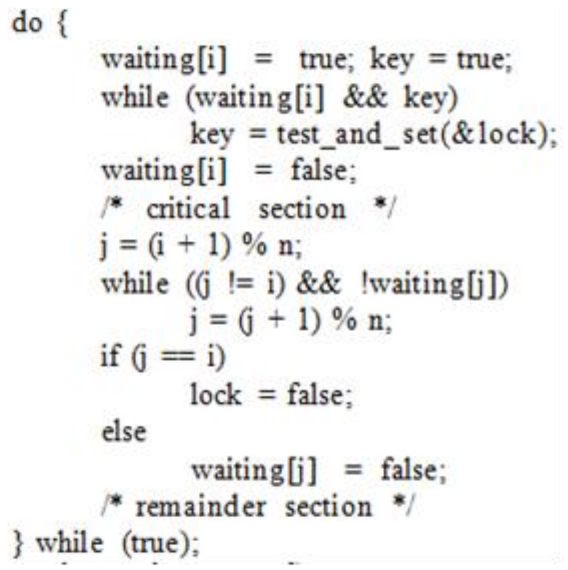
The important feature of this function is that it is executed without interruption. If the machine supports the Test and Set() function, we can implement mutual exclusion between processes using shared resources by declaring a boolean lock variable reflecting the status of resource R. If it is still free, the lock is set to False. Otherwise, if R is being used by a process, the lock is set to True. In the beginning, the lock is initialized to FALSE (the R resource has not been used by any process). The program structure for process Pi is shown in Figure 3.

```
do {
      while (Test_and_Set(&lock))
      ; /* do nothing */
      /* critical section */
      lock = false;
      /* remainder section */
} while (true);
```

Figure 3. Mutual exclusion implementation with the Test and Set() function

In Figure 4 below, another algorithm using the function Test and Set()[2] is introduced, which meets all the requirements for solving the critical segment problem. Common data structures are boolean waiting[n]; boolean lock;

The waiting[n] and lock data structures are initialized to false. The array waiting[i]= false for all i (i = 1 ÷ n) means there is no process waiting to use resource R (no need to use R), and as mentioned, lock = false implies that R is currently free.



```
do {
        waiting[i]  =  true; key = true;
        while (waiting[i]  &&  key)
                key = test_and_set(&lock);
        waiting[i]  =  false;
        /*  critical   section  */
        j = (i + 1) % n;
        while ((j  != i) &&  !waiting[j])
                j = (j + 1) % n;
        if (j == i)
                lock  = false;
        else
                waiting[j]  =  false;
        /* remainder  section  */
} while  (true);
```

Figure 4. Resolving mutual exclusion and bounded waiting with the Test and Set() function

Specifically, prove that the mutual exclusivity requirement is met as follows: process Pi can enter its critical section only if either waiting[i] == false or key == false. The key's value can become false only when the Test and Set() function is executed. The first process executes the Test and Set() function and finds that key == false, it uses the resource; all other processes must wait. The value waiting[i] becomes false only when another process i exits its critical section; only one variable waiting[i] is set to false, and the mutual exclusion requirement is maintained.

This algorithm has proved that the resource requirements of the processes are met; the arguments presented for mutual exclusion also apply here, since a process i exits the critical section, set lock to false or waiting[j] to false. Both allow a waiting process to enter its critical section to continue.

And the proof that the bounded waiting requirement is met is as follows: when a process exits its critical section, it traverses the waiting array in circular order (i + 1, i + 2, .. ., n − 1, 0, ..., i − 1). It assigns the first process in this order to enter the critical section (waiting[j] == true becomes waiting[j] = false), thereby allowing the next process (process j) to enter the critical section. Any process waiting to enter its critical section will do so within n - 1 turns.

III. Principles Of My Proposal

Extend the Test and Set algorithm

In practice, I observe many parallel processes that need to share various different types of resources. I propose an extended Test and Set algorithm for n processes, sharing m resources that satisfies the requirements of mutual exclusion and bounded waiting. Specifically, the extended algorithm is as follows: There are n parallel processes P1, P2,..., Pn , with many different resources, the service capacity of each resource is 1. When processes request resources, they will be allocated available resources sequentially. If a resource has been used and there is still a process requesting it, that resource becomes a critical resource.

- The algorithm performs scheduling through the critical section is as follows:

Use a target array, m Boolean elements, each element reflects the state of a resource Rj (j = 1 ÷ m). When R[j] = true is resource Rj in use, and when R[j] = false, resource Rj is currently free.

- Use a waiting array consisting of (n x m) Boolean elements, each element reflects the state of a process that needs to use a certain resource. When process Pi (i = 1 ÷ n) needs to use Rj, waiting[i,j] = true, and when Pi is using Rj or does not need to use Rj (Pi does not have to wait to use Rj) then waiting[i,j] = false.
- Variable n indicates the number of processes, i is the process index.
- Variable m indicates the number of resource types, j is the resource index.

boolean Waiting[255][255];

```
boolean R[255];
unsigned int i, j, k, n, m;
boolean TS(boolean *target); /*Pi calls TS(&R[j]) when Pi wants to use resource Rj */
{
CPU interrupts are prohibited;
Boolean rv = *target;
*target = true;
Return rv;
CPU interrupts are accepted;
}
void main()
{           cout <<  "Enter the process number n =  "; cin >> n ;
cout << "Enter the resource number m = "; cin >> m;
for (j = 1; j <= m; j++)
R[j] = false;         //All resources are assigned a free state
for (i = 1; i <= n; i++)
for (j = 1; j <= m; j++)
Waiting[i,j] = false;           /* All processes Pi are not waiting for resources of
type Ri*/
for (i = 1; i <= n; i++)
{     j = 1;
do
{   do
{
Waiting [i,j] = true; // Pi is in need of using Rj
While (waiting[i,j] && TS(&R[j])) ;             /* Pi continues to check Rj
when Pi is waiting for Rj and Rj is still in use by another process*/
Waiting[i,j]=false;  // Pi is used Rj – Pi in the critical section
k    :=    (i    mod    n)    +1;             /*    Consider    putting    process    Pk    adjacent    to    Pi    into
                                                  use Rj*/
While ((k <> i) && (!waiting[k,j]))
k = (k mod n) +1; /* Consider the next process if Pk does not need to
               use resource Rj */
if  (i == k)
R[j] = false; /* There are no more processes that need to use resource
Rj*/
else
waiting[k,j] = false;           /* Process Pk enters the critical section, Pg with
g ≠ k continue to do other work*/
while     (R[j] == false);  // The loop stops when resource Rj is in a free state
j = (j mod m) +1;
while (j > m);
}
}
```

The primitive Test and Set algorithm presented in section 2.3 applies to n processes, sharing a common resource that satisfies the requirements of mutual exclusion and bounded waiting.

I find that this algorithm can be extended to n processes, sharing m resources, also satisfying the requirements of mutual exclusion and bounded waiting. This expansion can be applied to the development of a credit timetabling program for schools.

The idea of expanding the Test and Set algorithm to apply to build a timetable program is as follows: Each class corresponds to a process, the resources needed for each class are classroom time and time of teacher (each time slice is a class period). And the algorithm's task is to reasonably distribute classroom time resources, and teacher time resources corresponding to the number of lessons of the subject for classes in the school.

The problem of timetable scheduling for the credit-based education system

**** Characteristics of the scheduling problem***

- Resources: these are the input data sources of the problem. These resources may or may not be recoverable.
- Task: evaluated through performance standards such as execution time, cost, resource consumption.
- Constraints: these are conditions that need to be satisfied for the problem to have the best solution

- Objective: evaluate the optimality of the solution schedule of the problem. When the objectives are satisfied, the constraints must also be satisfied

**** Introduce the problem***

The problem of arranging school schedules in general and arranging school schedules in schools according to the credit-based education system is a difficult problem. The complexity of the problem lies not only in finding a timetable for classes that meets time constraints, professional constraints, and regulations of the Ministry of Education and the school, but also in a more difficult problem of finding a good timetable suitable for all teachers. It must satisfy conditions, time requirements, limit the number of empty periods in a day, and the number of teaching days for teachers per week on the timetable.

So the problem of arranging a timetable for a school following the credit system is to arrange the class schedule in a way that is both appropriate and most convenient.

**** Data of the problem***

- List of courses
- List of classes, subjects and number of periods included in the semester
- List of lecturers/teachers
- Roster of lecturers/teachers teaching in classes (roster of expertise of faculties)
- Table of teachers' constraint requirements to the timetable.
- The requirements table constraint the subjects to the timetable.

The problem of building a timetable needs to satisfy the following hard and soft constraints:

**** Hard constraints:***

H1. It is not allowed to have 2 or more classes sharing the same classroom during the same class time (period) or session.
H2. In one hour (period) of a class, only one subject is scheduled, no more than 2 subjects are allowed.
H3. At the same class hour (period), a lecturer/teacher cannot teach simultaneously in 2 or more rooms.
H4. A lecturer/teacher cannot teach at a predetermined period of unavailability (long sick leave/maternity leave, the break time between 2 teaching shifts does not accommodate shifting shifts due to 2 lecture halls being too far apart…).
H5. A lecturer/teacher does not teach more than 30 periods per week.
H6. A class can only last consecutively for a maximum of the number of periods in a class session, that is in the morning (from period 1 to period 6) or in the afternoon (from period 7 to period 12) or in the evening (from period 1 to period 6). Lesson 13 to Lesson 16).
H7. A class must not be placed in a certain time slot (time-slot) where students cannot study.
H8. A class can only be placed in a classroom when the room's capacity is greater than or equal to the class size.

**** Soft constraints:***

S1. A class's schedule should minimize the number of days that include both morning and afternoon classes.
S2. Class schedules should not have empty hours (periods) between two subjects
S3. The study hours (periods) of a subject for a class in a session should be arranged consecutively
S4. The sessions, hours (periods) of a subject for a class should be evenly distributed throughout the week (The number of consecutive periods of a subject in a class session should not exceed the prescribed number of classes in a cluster of lesson periods).
S5. All periods of a particular subject for a class must be taught by the same lecturer/teacher.
S6. Do not schedule a teacher to teach the last period in the morning and immediately follow it with a period at the beginning of the afternoon, nor schedule a teacher to teach the first period class in the morning and immediately follow it with a class at the end of the morning
S7. All lecturers/teachers should have the same number of teaching hours.
S8. The timetable should be arranged so that the lecturer/teacher's teaching schedule minimizes the number of sessions per week and per semester.
S9. Study time for some subjects should be placed in reasonable study hours (periods) (difficult subjects should not be placed in the last period, physical education subjects should be placed in the first period of the morning or the last period of the afternoon and It's best to schedule a separate session).
S10. Subjects in the same session of a class should be taught in the same room, and a class should be taught in a fixed classroom.
S11. Classroom utilization efficiency should be the highest (avoid leaving empty classrooms causing waste).

Apply the extended Test and Set algorithm to apply to build a timetable program.

The idea of extending the Test and Set algorithm for application in building a timetable program is as follows: Each class corresponds to a process, and the resources required for each class are classroom time (classroom hours) and teacher time (teacher hours) (each time slot represents one class period). The algorithm's task is to distribute the classroom time resources and teacher time resources corresponding to the number of class periods for each class in the school.

The characteristic of classes in the credit-based education system is: Each subject for a major forms an independent class, with a separate class code, called an independent class code

- Each independent class will occupy the classroom for a certain number of class hours/class periods (a certain number of slices of classroom time) during the week, and the number of weeks the independent class occupies the time slots is calculated as follows:

sotuan = sotietLT/number_of_study_hours_in_the_week

or sotuan = sotietTH/number_of_study_hours_in_the_week

- Each independent class is essentially a subject for a prioritized class, assigned to a specific instructor. This means that each independent class will occupy that instructor a number of class hours/class periods (slices of teacher time) per week, and the number of weeks occupying the teacher's hours coincides with the time that the classroom is already occupied.

- When the course ends, the independent class completes its study period (the number of weeks allocated to the class), and the classroom and lecturer occupied by the independent class during that class/period (room hour resources and teacher hour resources) are simultaneously released. And the room hour resources and teacher hour resources that have just been freed up can be allocated to other independent classes waiting for classroom and lecturer availability.

**** Apply the extended Test and Set algorithm as follows:***

The extended Test and Set algorithm applies n processes P1, P2,..., Pn concurrently, with m different resources, each resource's service capacity being 1, wherein:

- n independent classes with parallel learning activities corresponding to n processes, each independent class will be assigned a classroom based on the available classroom corresponding to the required time slice of the independent class.

- Initially, classroom resources are freely available for all classrooms and time slots. Each independent class will sequentially be assigned one classroom corresponding to the class's scheduled time, meaning multiple independent classes will share the same classroom at different time slices (class periods), which have not yet been specifically allocated to any particular independent class. If there are insufficient classrooms, one or more independent classes may temporarily wait to be scheduled in subsequent learning period.

- In the second step, based on the specialization assignment table, allocate teacher hours to independent classes (which also means allocating classroom hours to independent classes taught by the teacher). When independent classes request classroom hour resources, these will be allocated sequentially to the remaining available classroom hours. If all available classroom hours are exhausted and there are still independent classes requesting, classroom time becomes a constrained resource, and independent classes must wait until classroom time resources are freed up.

The Test and Set algorithm is applied to develop software for scheduling independent credit-based classes as follows:

The Test and Set function is defined as follows:

```
bool Test and Set(ref bool test_room, ref bool test_lecture)
{
bool _arranged = true;
if ((test_room == false) &&
(test_lecture == false))
{ điều đỉnh
_arranged = false ;
}
else
{
test_room = true;  test_lecture = true;
}
return _arranged;
}
```

The algorithm for scheduling classes for n independent classes uses a number of data structures as follows:

- independent_class independent class array with n elements, each element includes the following basic information:
+ bool _scheduled: indicates whether the independent class has been scheduled or not.
+ string room_name: indicates the name of the classroom.
+ string teacher_code: indicates teacher code
+ … some information is data to produce a schedule file.
- An array teacher_list, showing information about teachers in the school, an array of k elements with index t_index (t_index = 0 ÷ (k – 1)), including the following basic information:
+ string teacher_code: indicates teacher code
+ bool class_period[]: The class_period array consists of m elements, each element reflects the status of a class hour (class period) that the teacher has been scheduled or not..
+ … some information is data to produce a schedule file.
- A classroom_list array, providing information about classrooms, the array consists of z elements, each element has an index r_index (r_index = 0 ÷ (z – 1)), including the following basic information:
+ string room_name: indicates the name of the classroom (location where the independent class will study).`
+ bool class_period[]: class_period array consists of m elements, each element reflects the status of an class hour (class period) that the classroom has been scheduled or not.
+ … some information is data to produce a schedule file.

Specifically, the algorithm is as follows:

Step 1. Initialize the state of every classroom and the state of every teacher at every time slice as free, the status of all independent classes is unsorted

Step 2. Schedule classes for independent classes according to the available classroom with the corresponding time slot that the independent class will study:

```
for (int i = 0; i < independent_class.Length; i++)
{
if independent_class[i]._scheduled == true)
continue;            // Consider the next independent class (i = i + 1)
uint r_index = 0, t_index = 0; j = 0;
uint remaining_periods = Week_number_periods;
if (_global._TKB_independent_class[i]._scheduled == false)
{
//Find a satisfying independent class
if  independent_class[i].teacher_code = teacher_list[t_index].teacher_code
{
independent_class[i].room_name = Classroom_list[r_index].room_name
for (j = 0; j < 112; j++)
{
if (Test and Set(ref Classroom_list[r_index].class_period[j], ref teacher_list[t_index].class_period[j]))
continue;  //Continue to consider class_period[j+1]
else
{
independent_class[i] is allocated resources from Classroom_list[r_index].class_period[j] and teacher's hour
from teacher_list[t_index].class_period[j]
//The r_index classroom has been scheduled for class_period[j]
//Teacher has been scheduled to class_period[j]
}
If the independent class has been scheduled for the full number of class periods per week, then
independent_class._scheduled = true and proceed to schedule the next independent class.
}
}
}
}
```

The scheduling algorithm in Step 2 utilizes the Test and Set() function, indicating that independent class i is assigned the lecturer's hour and classroom hour j only if both the classroom hour j of classroom r_index and the lecturer hour j of lecturer t_index are not scheduled yet

Both (Classroom_list[r_index].class_period[j] and teacher_list[t_index].class_period[j]) are false. Therefore, if teacher t_index has already been scheduled to teach class period j, then this period j of the teacher cannot be rescheduled for any other independent class, or if classroom r_index has already been scheduled for class period j, then this period of the classroom cannot be scheduled for any other independent class. This is the crucial point of the algorithm to help avoid violating the fundamental hard constraints H1, H2, H3 in the section of hard constraints in section 3.2. This is the difference in the scheduling problem using a variant of the Test and Set() algorithm compared to other algorithms mentioned above, which, as we know, still violate the hard constraints..

IV. Experimental Results

a. Introducing input data

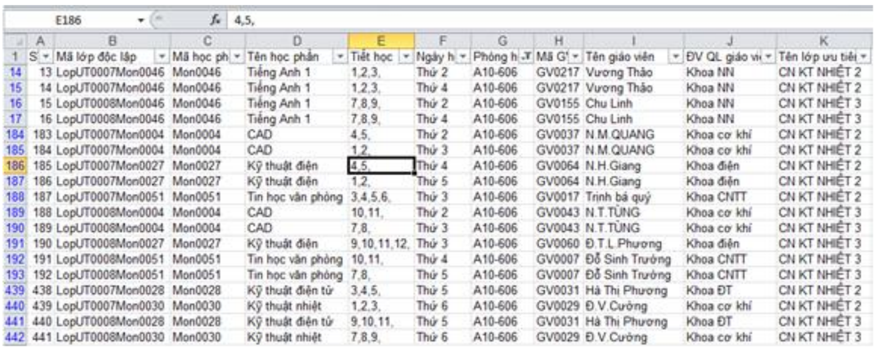
- File DM_Phong.xlsx: provides information about the current classrooms available at the school (manually created). The file contains a list of 56 classrooms.
- File DM_LopUT.xlsx: provides information about the priority classes in the school (manually created). The file contains a list of 99 priority classes.
- File DM_mon.xlsx: provides information about the subjects in the semester (manually created). The file contains a list of 59 subjects for 99 classes in the semester, each subject with a different duration (ranging from 2 to 6 credits depending on the subject).
- File DS_GV.xlsx: provides information about the teachers participating in teaching during the semester (manually created). The file contains a list of 225 teachers involved in teaching during the semester.
- File Phan_cong_CM_SDEC.xlsx: provides information about the results of subject specialization assignments for each teacher per semester (manually created). This file consists of 590 records aggregated from the results of subject specialization assignments by specialized units within the school, corresponding to 590 independent classes. This file shows that each priority class has a different number of teaching periods per week, with a maximum of 30 teaching periods per week.

b. Introducing output data

After implementing the algorithm and running the program, we obtain 2 files:

- File DM_LopUT_PhHoc.xlsx: Is the result of the classroom assignment step for each priority class. This is an intermediate result to prepare for the timetable scheduling step. This file is generated by the program from two files: DM_Phong.xlsx and DM_LopUT.xlsx
- File TKB_HK4_2016-11-30_22h30_750rows: Is the result of scheduling classes for each independent class in the semester. The output result is displayed in the file comprising 750 records, corresponding to 750 class sessions for all independent classes in the school.

I have verified the output result by filtering information for each teacher, each classroom on each school day to ensure there are no overlapping conflicts regarding time and location, no empty periods between two subjects for the same priority class. This means that the scheduling result does not violate the fundamental hard constraints as mentioned above and some soft constraints. Figure 5 below is an image reflecting the output result: an image of data filtering from the output file is the timetable scheduling result to check the allocation of classroom resources for independent classes. The result shows no conflicts in classroom resource allocation, ensuring maximum efficiency of classroom resources, no empty periods between scheduled classes, and no conflicts in teacher resource allocation. Therefore, the applied algorithm did not violate the aforementioned hard constraints.



E186 | 4,5,

	A	B	C	D	E	F	G	H	I	J	K
1	S	Mã lớp độc lập	Mã học ph	Tên học phần	Tiết học	Ngày h	Phòng h	Mã G'	Tên giáo viên	ĐV QL giáo vi	Tên lớp ưu tiê
14	13	LopUT0007Mon0046	Mon0046	Tiếng Anh 1	1,2,3,	Thứ 2	A10-606	GV0217	Vương Thảo	Khoa NN	CN KT NHIỆT 2
15	14	LopUT0007Mon0046	Mon0046	Tiếng Anh 1	1,2,3,	Thứ 4	A10-606	GV0217	Vương Thảo	Khoa NN	CN KT NHIỆT 2
16	15	LopUT0008Mon0046	Mon0046	Tiếng Anh 1	7,8,9,	Thứ 2	A10-606	GV0155	Chu Linh	Khoa NN	CN KT NHIỆT 3
17	16	LopUT0008Mon0046	Mon0046	Tiếng Anh 1	7,8,9,	Thứ 4	A10-606	GV0155	Chu Linh	Khoa NN	CN KT NHIỆT 3
184	183	LopUT0007Mon0004	Mon0004	CAD	4,5,	Thứ 2	A10-606	GV0037	N.M.QUANG	Khoa cơ khí	CN KT NHIỆT 2
185	184	LopUT0007Mon0004	Mon0004	CAD	1,2,	Thứ 3	A10-606	GV0037	N.M.QUANG	Khoa cơ khí	CN KT NHIỆT 2
186	185	LopUT0007Mon0027	Mon0027	Kỹ thuật điện	4,5,	Thứ 4	A10-606	GV0064	N.H.Giang	Khoa điện	CN KT NHIỆT 2
187	186	LopUT0007Mon0027	Mon0027	Kỹ thuật điện	1,2,	Thứ 5	A10-606	GV0064	N.H.Giang	Khoa điện	CN KT NHIỆT 2
188	187	LopUT0007Mon0051	Mon0051	Tin học văn phòng	3,4,5,6,	Thứ 3	A10-606	GV0017	Trịnh bá quý	Khoa CNTT	CN KT NHIỆT 2
189	188	LopUT0008Mon0004	Mon0004	CAD	10,11,	Thứ 2	A10-606	GV0043	N.T.TÙNG	Khoa cơ khí	CN KT NHIỆT 3
190	189	LopUT0008Mon0004	Mon0004	CAD	7,8,	Thứ 3	A10-606	GV0043	N.T.TÙNG	Khoa cơ khí	CN KT NHIỆT 3
191	190	LopUT0008Mon0027	Mon0027	Kỹ thuật điện	9,10,11,12,	Thứ 3	A10-606	GV0060	Đ.T.L.Phương	Khoa điện	CN KT NHIỆT 3
192	191	LopUT0008Mon0051	Mon0051	Tin học văn phòng	10,11,	Thứ 4	A10-606	GV0007	Đỗ Sinh Trường	Khoa CNTT	CN KT NHIỆT 3
193	192	LopUT0008Mon0051	Mon0051	Tin học văn phòng	7,8,	Thứ 5	A10-606	GV0007	Đỗ Sinh Trường	Khoa CNTT	CN KT NHIỆT 3
439	438	LopUT0007Mon0028	Mon0028	Kỹ thuật điện tử	3,4,5,	Thứ 5	A10-606	GV0031	Hà Thị Phương	Khoa ĐT	CN KT NHIỆT 2
440	439	LopUT0007Mon0030	Mon0030	Kỹ thuật nhiệt	1,2,3,	Thứ 6	A10-606	GV0029	Đ.V.Cường	Khoa cơ khí	CN KT NHIỆT 2
441	440	LopUT0008Mon0028	Mon0028	Kỹ thuật điện tử	9,10,11,	Thứ 5	A10-606	GV0031	Hà Thị Phương	Khoa ĐT	CN KT NHIỆT 3
442	441	LopUT0008Mon0030	Mon0030	Kỹ thuật nhiệt	7,8,9,	Thứ 6	A10-606	GV0029	Đ.V.Cường	Khoa cơ khí	CN KT NHIỆT 3

Figure 5. The output data is the arranged timetable.

V. Conclusions

Expanding and applying the Test and Set() algorithm to timetable scheduling has indeed not violated the crucial hard constraints, namely constraints H1, H2, H3 in the section of hard constraints in section 3.2. above, as the algorithm has been demonstrated to satisfy the requirements of mutual exclusion and bounded waiting. Meanwhile, algorithms in the field of artificial intelligence still violate hard constraints when solving scheduling problems with the timetable scheduling problem solved as above, the other hard constraints have not been considered, fulfilling the remaining hard constraints as well as soft constraints entirely becomes feasible by adding conditions to the timetable scheduling program according to the extended Test and Set algorithm

I perceive that this algorithm can entirely be used to build and develop software applications to solve scheduling problems in general, especially timetable scheduling problems in schools or educational institutions in particular.

Acknowledgment

I would also like to sincerely thank Hanoi University of Industry for supporting me in completing this paper.

I would like to sincerely thank Comrade Vu Ngoc Anh , along with all the staff in the Training Department of Hanoi University of Industry. I also extend my gratitude to Teacher Pham Thi Ha, Vice Principal of Ly Thuong Kiet Secondary School in Nam Dinh, for her dedicated advice and assistance, which helped me grasp the basic process and key points when scheduling. Thanks to this, I was able to expand and apply the Test and Set algorithm effectively to solve the problem.

I would like to sincerely thank my family, friends, and colleagues for their encouragement and valuable input to help me complete this paper.

References

[1] Pham Anh Tuan (2012), "Application Of Genetic Algorithms To Schedule Credit System For Universities", Master's Degree In Engineering, Code 60.48.01, Danang University.

[2] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne (2013), Operating System Consepts, 9th,John Wiley &Sonspublisher, USA.

[3] Dipesh Mittal, Hiral Doshi, Mohammed Sunasra, Renuka Nagpure (2015), Automatic Timetable Generation Using Genetic Algorithm, Nternational Journal Of Advanced Research In Computer And Communication Engineering Vol. 4, Issue 2.

[4] E. Aycan And T. Ayav, Solving The Course Scheduling Problem Using Simulated Annealing.

[5] Olivier Catoni, Solving Scheduling Problems By Simulated Annealing (2015), Siam J. Control Optim Vol. 36, No. 5, Pp. 1539 – 1575.

[6] Shu-Chuan Chu, Yi-Tin Chen, Jiun-Huei Ho, Timetable Scheduling Using Particle Swarm Optimization (2006), Innovative Computing, Information And Control, 2006. ICICIC '06. First International Conference On, Volume: 3.