Scientific Journal of Impact Factor (SJIF): 4.14

e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 3, Issue 8, August -2016

Backtracking for Timetable Planning

Mohammed Motasim[1], Mohammed Siddiq[2]

[1]Student, Computer Science Engineering, RNSIT, Bangalore,
[2]Student, Computer Science Engineering, RNSIT, Bangalore,

Abstract: *Timetable for any institution is as important as the institution itself. Since, it's the timetable that defines and controls the day to day working of the institution. For years now, Timetable has been planned manually and considered to be a daunting task. This paper attempts to digitize the whole process of planning the timetable. Timetable planning falls under the category of combinatorial problems. We try solving this problem using a very popular technique used for combinatorial problems-Backtracking. If implemented efficiently this technique can provide effective results. We have implemented our approach and at last we present our results and analysis.*

***Keywords*:** *Combinatorial problem, Timetable planning, Digitization, Backtracking*

I. INTRODUCTION

Many universities plan the timetable manually, despite the widespread availability of computing techniques. Here we present an approach to digitize the process of planning the timetable. Backtracking is considered to solve most of the combinatorial problems. Since timetable planning is also one of the combinatorial problems, backtracking can be the go to technique. Timetable planning involves scheduling a teacher (or teachers) to different time slots. This problem, by definition, involves numerous primary constraints. Constraints like: a particular teacher cannot be scheduled to two different time slots, at a given time, no two teachers can be scheduled to the same time slot, and finite number of time slots per week. Additional constraints (University specific) are applicable, such as, workload of each lecturer. Consideration of such constraints is pivotal in planning a timetable. These constraints assist the backtracking algorithm to avoid taking unyielding paths, thereby increasing the efficiency of the algorithm. These constraints must be formulated as a backtracking problem. Recursion is key to backtracking and plays a very vital role in its implantation. The basic working of recursion favours backtracking.

In our implementation, we have considered all the primary constraints and few set of additional constraints, to demonstrate the solution for planning.
In this paper we emphasize on solving the problem in hand, and providing our implemented results and inference.

II. BACKGROUND THEORY

Before introducing the implementation of the solution, we would like to introduce backtracking, the algorithm.
Backtracking: It is a problem solving technique, which is commonly used to solve combinatorial problems with constraints. It approaches the solution by picking a candidate (say 'C') and building up a solution over it. If the algorithm successfully satisfies all the constraints and builds the solution, it is said to have successfully found a solution. If it reaches a dead end, where it no longer can build the solution, it eliminates the candidate 'C' (Backtracks), and picks the next candidate. It repeats the same, until a solution is found or all the candidates are evaluated to be dead ends. In the former case, the algorithm has successfully found the solution to the problem, in the latter there is no feasible solution to the problem. Backtracking is used essentially to solve the constraints specific problems. The following algorithm is used as the basis to plan the Timetable. Timetable planning has numerous constraints and it is a combinatorial problem, which makes backtracking a favourable choice.

Backtracking Algorithm:

```
Pick an initial candidate.
while(Problem is not solved)
    For each path from the initial candidate.
            check if selected candidate satisfies all the constraints, if yes select it
        and make recursive call to rest of the problem
            If recursive calls returns true, then return true.
            else
                Undo the current move and return false.
    End For
    If none of the move works out, return false, NO SOLUTON.
```

Backtracking is a smarter brute force algorithm. Brute force lists all possible solutions without considering the constraints and then later checks each possible solution for its compliance with the constraints. In comparison, a backtracking algorithm checks constraints for a candidate and if the candidate does not lead to a solution, it eliminates possible paths having that candidate.

III. MPLEMENTATION AND RESULT ANALYSIS

Considering the problem to be formulated with all the constraints and requirements, the following algorithm illustrates the planning of timetable.

```
//TIMETABLE_ARRAY [ ] [ ] is a 2d array representing the timetable .Rows represent the days and columns
represent the periods (time slots).

//TIMETABLE () function generates the timetable using backtracking (uses recursion).

//PLACEABLE() function checks whether the subject picked satisfies the constraints &hence can be scheduled

// 'DAY' is an integer representing a particular day, 'PERIOD' represents the time slot

//COMPLETE() function checks if the timetable generation is complete

void TIMETABLE (int TIMETABLE_ARRAY[ ][ ] ,int PERIOD, int DAY)
{
        Initialise the TIMETABLE_ARRAY to -1;//representing all slots are available

        if  (PERIOD > maximum periods in DAY )
        {
                        PERIOD=1;
                        DAY++; // go to the next day and the first slot.

        }




        if  (TIMETABLE_ARRAY[DAY][PERIOD] == -1) // if the slot is empty
```



```
        {
                for (each subject S)
                {
                        if(PLACEABLE(S))// if subject satisfies all the constraints
                        {

                                TIMETABLE_ARRAY[DAY][PERIOD]=S;
                                // allocate the time slot for subject S
                                if(COMPLETE())
                                {
                                        timetable successfully generated;
                                        Exit;
                                }
                                else
                                {
                                        TIMETABLE(TIMETABLE_ARRAY[ ][ ],PERIOD++,DAY);
                                        // recursive call to timetable
                                        TIMETABLE_ARRAY[DAY][PERIOD]=-1;
                                        // Backtrack and Restore previous data
                                }
                        }
                }
        }
        TIMETABLE(TIMETABLE_ARRAY[][],PERIOD++,DAY);
        // recursive call to try for the next unallocated time slot
}
```

The above algorithm plans the timetable for a class. If there exists a solution, then timetable generation is successful and the algorithm stops. If there is no solution, then the algorithm has checked all possibilities and it is impossible to plan a time table with the given constraints. That is, no feasible solution exists for the given constraints. The given constraints have to be relaxed to generate a viable timetable

Here are some of the results produced by the algorithm mentioned above.

DAY/TIME	1	2		3	4		5	6	7
MON	LAB	LAB	B	LAB	SUB_4	L	SUB_6	SUB_3	SUB_5
TUE	SUB_2	SUB_3	R	SUB_1	SUB_5	U	LAB	LAB	LAB
WED	SUB_3	SUB_2	E	SUB_4	SUB_1	N	SUB_5	SUB_6	SUB_2
THR	LAB	LAB	A	LAB	SUB_3	C	SUB_2	SUB_4	SUB_6
FRI	SUB_4	SUB_1	K	SUB_5	SUB_6	H	SUB_3	SUB_5	SUB_1
SAT	SUB_1	SUB_4		SUB_6	SUB_2				

The above timetable is generated for a particular university and it complies with all the constraints of that university. For example, this particular university provides the 'lab' timetable and mandates the algorithm to accord with this constraint. This university also mandates that no same time slot in a week be given to the same subject. As evident from the timetable, this constraint is also satisfied. All these constraints are checked in the PLACEABLE function.

PLACEABLE function:

All the constraints can be checked in the PLACEABLE function. The PLACEABLE function returns true only if all the constraints are satisfied. This function can also be used to modify the efficiency of the planned timetable. For example, a teacher taking for two different classes can be scheduled in such a way that he has at least one hour gap between two successive time slots allotted to him. One more way in which the function can be used is specify prioritized constraints. That is, the constraints can be prioritized in the order of their importance. The highest priority constraints have to be satisfied, the lower priority constraints can be compromised to generate a solution which otherwise, would not have been possible. Ultimately, PLACEABLE function is subjective, one can use it as per the requirements.

The prototype of the PLACEABLE function is shown below:

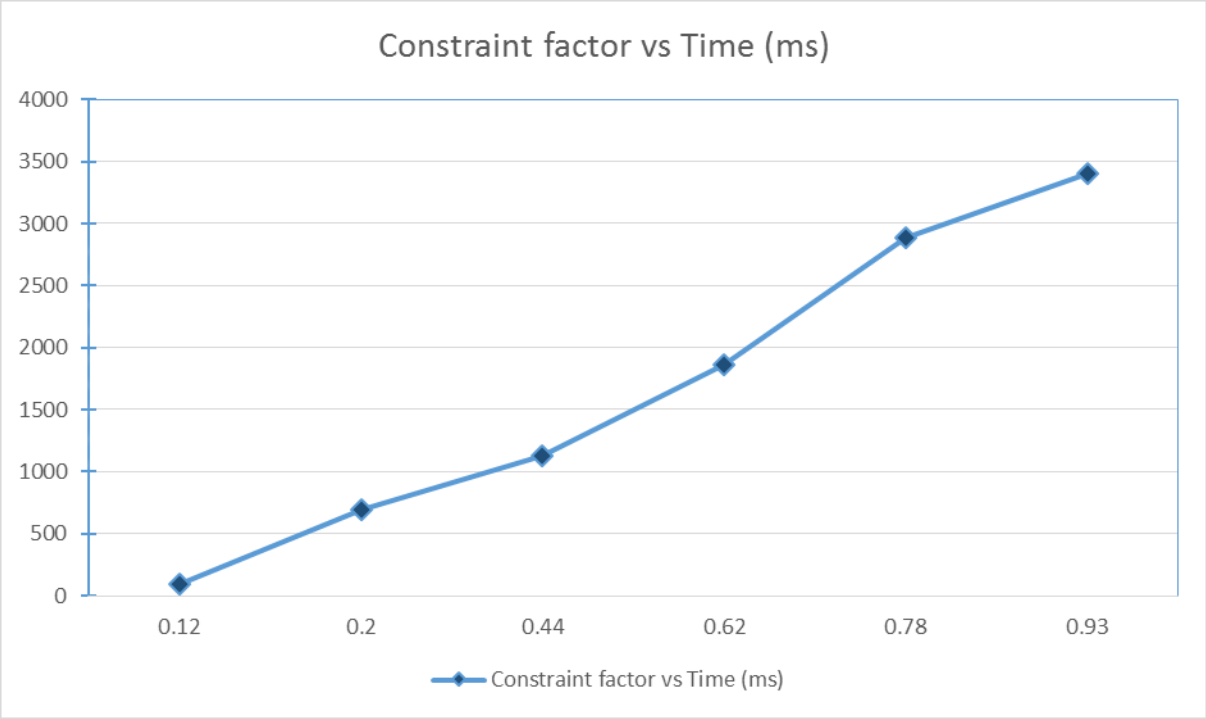
```
Boolean PLACEABLE(int subject)
{
        for every constraint C in constraints [ ]
        {
                if ( C is satisfied )
                        continue;
                else
                        return false;
        }
        return true;  //only if all the constraints are met
}
```

However, the performance of the algorithm dips with increasing number of constraints and severity of the constraints. The algorithm is said to be strained, and it takes more time than normal to produce a solution. If constraints are way too severe and numerous, then a feasible solution may not exist. The table below shows the time taken by our implementation for different values of constraint factor.

	Constraint Factor	Time in milliseconds
Trial 1	0.12	97
Trial 2	0.2	697
Trail 3	0.44	1125
Trial 4	0.62	1863
Trial 5	0.78	2886
Trail 6	0.93	3404
Trial 7	1.2	No feasible solution possible

Constraint factor: Constraint factor is a single value representing the severity and number of constraints. It takes into account the basic (primary) constraints and University specific constraints. Constraint factor signifies how strained the algorithm is, that is how difficult it is for the algorithm to produce a feasible solution. The value of constraint factor lies in between 0 and 1. 0 indicates that only basic constraints are considered, and a value more than 1 indicates that constraints are so severe that a feasible solution is not possible.

Given below is a plot of constraint factor versus the time taken in milliseconds.



As one can infer from the graph, time taken to produce a solution increases with increasing constraint factor value.

V. CONCLUSION

Timetable planning is like any other combinatorial problems, which can be solved using computing techniques, and hence digitized. We have presented one approach to tackle this, using backtracking. It is necessary to consider the primary constraints which essentially defines timetable, and further any specific constraints can be considered. The efficiency varies with constraints and can be improved with smarter implementation.

VI. REFERENCES

[1] Anany Levitin: *Introduction to design and analysis of algorithms.*
[2] Elliz Horowitz, Sartaj sahni, Rajasekaran *: Fundamentals of computer algorithms*

