DE GRUYTER OPEN

JAISCR, 2013, Vol. 3, No. 3, pp. 251 – 263
DOI 10.2478/jaiscr-2014-0018

B-TREE ALGORITHM COMPLEXITY ANALYSIS TO EVALUATE THE FEASIBILITY OF ITS APPLICATION IN THE UNIVERSITY COURSE TIMETABLING PROBLEM

Marco Antonio Cruz Chvez, Alina Martnez Oropeza
CIICAp, Universidad Autnoma del Estado de Morelos
Av. Universidad 1001. Col. Chamilpa, C.P. 62209.
Cuernavaca, Morelos, Mxico
mcruz@uaem.mx

Abstract

This paper presents a comparative analysis of complexity between the B-TREE and the Binary Search Algorithms, both theoretically and experimentally, to evaluate their efficiency in finding overlap of classes for students and teachers in the University Course Timetabling Problem (UCTP). According to the theory, B-TREE Search complexity is lower than Binary Search. The performed experimental tests showed the B-TREE Search Algorithm is more efficient than Binary Search, but only using a dataset larger than 75 students per classroom.

1 Introduction

Resource Scheduling is a very important area within computer sciences due to its wide application in different areas, such as manufacturing, the academic environment, and vehicle routing, among others. One of the typical problems in this area is the scheduling in an academic environment [6], where school schedules must be established. Their development requires long periods of time due to the number of variables to assess. Also on occasion, unfeasible solutions are obtained because overlaps could occur. An overlap is the timeslot in which the same event or activity happens simultaneously; in this context, when scheduling classes, teachers and students cannot give or take different classes during the same timeslot.

Designing school schedules is not an easy task, but its complexity depends on the level. For example in an elementary school the complexity is reduced due to the smaller quantity of teachers, classrooms, and the fixed student groups, so they do not have to take into account so many variables. The complexity increases for a high school or a university where there are many constraints to be considered in order to obtain a feasible solution.

To treat the problem of scheduling at a university level, there is a proposed mathematical model and some benchmarks which are used as a paradigm to try to obtain feasible and better solutions. The name of the problem in combinatorial optimization is the University Course Timetabling Problem (UCTP), which has been classified within the set of NP-Complete problems due to its complexity [1, 2]. This means that there is no known deterministic algorithm bound by a polynomial temporal function to solve this kind of problem [3]. Therefore, the scientific community has focused its research on the treatment of NP-Complete problems through non-deterministic algorithms, which are bound by polynomial time. These algorithms (better known as Heuristics) need to be constantly improved in attempt to obtain better solutions or reduce the execution time [4]. There are several techniques that have been used to find feasible solutions to this problem, but the most common are metaheuristics.

To find a good solution to UCTP using any metaheuristic, local search application is necessary. The heart of a local search is the application of a neighborhood structure, which generates exchanges of elements. Iterative local search, guided by an objective function, can improve the solution of a problem as UCTP, but every swap generated by the neighborhood structure through local search could generate an unfeasible solution. To detect a movement of the neighborhood structure that can generate an unfeasible solution in UCTP, it is necessary to use a search algorithm. For example, in a UCTP, moving a student e to take a class on day i, at a specific hour j, in classroom k, (*i, j, k*), may create schedule conflicts (overlap) for the student and other classes that he takes. To determine if there is overlap, it is necessary to perform a search in all classrooms ($k = 1,..., n$) to verify that the student e is not taking another class in the same timeslot (*i, j*).

According to theory, the search algorithms with the least theoretical complexity are B-TREE and Binary search. The B-TREE complexity, in the worst case, is lower than the Binary search complexity. For this reason, in this work, these two search algorithms were tested for UCTP. A set of hard constraints were accounted for, which ensured non-overlapping classes for both teachers and students [5]. The fulfillment of these constraints must be verified; it is necessary to check that students do not take more than one subject per timeslot (*i, j*), and in one classroom k. If a student e takes more than one subject in the same timeslot (*i, j*), the student is overlapping; therefore, the obtained solution is unfeasible.

In this research, the two most efficient search algorithms, according to their theoretical complexity, were used to conduct an experimental comparison.

Application of a search method allows for revision of each student e assigned to an event in a timeslot in a CR classroom. It also allows for revision of possible overlaps with other events the student takes, where an event is a class taken by the student e. According to this reasoning, the methods applied to UCTP require a very efficient search algorithm within their structure due to the iterative nature which is required for this problem. According to theoretical and experimental results, a conclusion was reached as to the best option, based on the efficiency of the search algorithms applied to this specific case.

The contribution of this work is to find the most efficient search algorithm to detect overlap of events for students and teachers in the UCTP.

The present research is divided as follows. Section one provides a general introduction to the problem, and emphasizes the importance of having an efficient method which can obtain good solutions in a reasonable computational time. Section two presents the formal definition of the University Course Timetabling Problem, focusing on the explanation of the set of constraints, which define the specific case. Section three discusses the theoretical complexity managed in the literature for the principal search methods. An explanation is given for the methods of both Binary Search and B-TREE Search. The fourth section presents the temporal functions specified to each algorithm. Section five shows the experimental results, and the comparative analysis of both methods, taking into account their theoretical and experimental complexity. Finally, the conclusions of this research are presented in section six.

2 University Course Timetabling Problem

The University Course Timetabling Problem (UCTP) has been classified within the Complexity Theory as NP-Complete [1]. This problem could be solved by deterministic or non-deterministic methods. Using a deterministic method, it is only feasible to obtain the optimal solution for a small instance, because if the instance size grows, the computation time of a deterministic algorithm increases exponentially. As a result, most researchers have focused on heuristics, which get approximated solutions in a reasonable time; although, there is no guarantee of their optimality.

The features of the components that are considered in this problem, such as students, classrooms, facilities, teachers, and classes, when considered in combination with the features of limited resources, time considerations and the available spaces for assigning the events, make it difficult to get feasible solutions in a reasonable computing time.

The goal of finding solutions to the UCTP is to obtain a scheduling such that both students and teachers can attend all their programmed classes without time difficulties. The UCTP has some specific features that must be respected to obtaining a feasible solution. These features are known as constraints, and are classified into two kinds of constraints.

Hard Constraints: They must be fulfilled completely, and must not be violated. For example a teacher cannot give different classes in the same timeslot.

Soft Constraints: They should, preferably, be satisfied, although some violations could be accepted adding a penalty per each violation. A clear example may be that the students should not have four or more continuous classes. The main reason is that it is not pedagogically recommended, because the students need a rest after taking continuous classes. However, some violation could be accepted due to the time circumstances, and the scheduling.

The specific problem tackled in this research consists of a set of events that have to be programmed in a set of 45 timeslots, which are distributed in five days with nine 50-minute periods per day. There is also a set of classrooms where the events are held, a set of students who attend the events, and a set of facilities in each classroom, which are necessary to perform the events. Each student has to attend a certain number of events, which are scheduled in certain classrooms with specific features. Taking this into account, a feasible solution has to have all the events scheduled in a classroom during a timeslot, respecting the following constraints and trying to improve the value of the objective function (1).

$$Min\ F.O = \sum s1 + \sum s2 + \sum s3 \qquad (1)$$

The objective function minimizes the quantity of soft constraints violated, where S1, S2, and S3 refer to the soft constraints defined to this specific problem [6]. The constraints, both hard and soft, are explained as follows:

Set of Hard Constraints

H1. No student attends more than one event in the same timeslot.

H2. The classroom capacity must be enough to hold all the students assigned to attend the event.

H3. The assigned classroom must have the required facilities to carry out the assigned event.

H4. No classroom can have more than one event in the same timeslot.

H5. No teacher attends more than one event in the same timeslot.

Set of Soft Constraints

S1. A student should not take classes in the last timeslot of the day.

S2. A student should not take two or more uninterrupted classes.

S3. A student should not take only one class during a day.

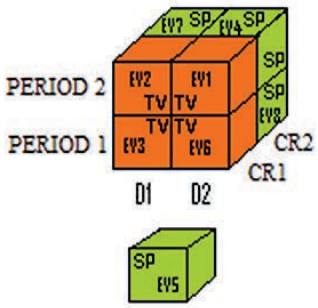
The UCTP could be represented in a schematic form (Figure 1) to improve its understanding.

The representation shown in Figure 1 is made using a tridimensional form, better known as a cube, where each layer, represented by different colors, symbolizes a classroom. A classroom is formed by several 3D boxes (Period, Day, and Classroom) per day, where each one defines a timeslot where an event can be assigned. An event is the intersection between a Period, and a Day in certain Classroom, where an event could be assigned. Each event will be taken by a group of students, who will attend to several events $\{EV1, EV2, \ldots, EVn\}$ per week. For example, the EV1 is taken by students E1, E2, and E3 in the classroom CR2. A set of events form a class.

Each classroom has certain features, which are required by the different events. Those features could be TVs, DVDs, computers, projectors, or whiteboards, among others.

Some of the characteristics that must be taken into account to assign an event to a classroom are its features and size. For example, a mathematics teacher may need a whiteboard in order to give a class. Size of the classroom must also be considered. It is necessary to check the classroom capacity because a common problem is that big groups of students are assigned to small classrooms. In Figure 1 we can observe a possible solution, where all the hard and soft constraints are satisfied in order to get a feasible scheduling to a small instance. For this representation, the quality of the obtained solution,

according to the objective function is not evaluated, because it is only an example of a feasible solution where all the constraints are satisfied. It is very difficult to use a bigger instance because of its time limitations and the nature of the problem.



EVENT	STUDENTS	CHARACTERISTIC
EV1:	E1, E2, E3.	TV
EV2	E2, E4, E5.	N/A
EV3:	E3. E4. E5.	N/A
EV4:	E4, E6, E6.	SP
EV5:	E8, E2, E7.	N/A
EV6:	E7. E8, E3.	TV
EV7:	E6, E7, E1.	N/A
EV8:	E5, E1, E6.	SP

CLASSROOM	CHARACTERISTIC
CR1	SLIDE PROJECTOR (SP)
CR2	TELEVISION (TV)

Figure 1. Schematic representation of a possible solution applied to a small UCTP instance

Finding a feasible solution is not easy; it requires an efficient process dedicated to exploring the solution space and verifying that all the constraints were satisfied. The process of ensuring that a solution is feasible requires the use of a search algorithm, which has to examine each set of students and teachers in each classroom to make certain that no subject and no students were overlapped. Even if a feasible solution is found, it is necessary to continue exploring for new feasible solutions in order to obtain the best possible solution. The aim of the present research is to find an efficient search algorithm to apply to the specific problem of UCTP explained in this section.

The next section provides a general explanation of some of the most efficient search algorithms. Examination of their theoretical complexity allowed for the selection of the two best algorithms. An experimental comparison was performed, using a computer program to select the best algorithm for this specific problem.

3 Search Algorithms

Search algorithms have been widely used to perform searches in data structures, such as trees, where they have been applied successfully. Many types of search algorithms exist, such as linear search, Binary search, B-TREE search, and some hybrid searches, which have been applied to different optimization problems.

The main difference among these search algorithms is the computational effort required to carry out a certain search, because many of them need the keys (elements of the total set) in a specific order, which increases the complexity of the algorithm. For this research, three of the most common search algorithms are used, in addition to one hybrid (Quicksort + Binary search). According to their complexity, theoretically, B-TREE search is more efficient that the others.

The complexity of an algorithm (Table 1) depends on the number of questions that it must ask in the worst case in order to find a specific key within a set of n keys.

Using the information from Table 1, the graph below (Figure 2) was made, which clearly shows the theoretical behavior of each search method. In the case of B-TREE search, a tree to the order of 5 was used.

According to the comparison presented in Table 1 and Figure 2, it can be observed that the most efficient algorithm is B-TREE search, and the second most efficient is Binary search. It is important to note that these two algorithms work with ordered keys. They have the previously mentioned complexity only if the set of keys used are already ordered; otherwise, the complexity increases depending on the sorting algorithm applied.

A comparison of the requirements of the search algorithm (Table 1) with the features of the specific problem tackled in this paper, show that it is possible to apply them to the UCTP. The set of keys used corresponds to the students assigned to differ-

Table 1. Theoretical complexity in the worst case for four commonly used search algorithms

Search Algorithm	Theoretical Complexity
B-TREE ***m > 2***	$d_{\min} = \log_m (n+1)\ d_{\max} = 1 + \log_{\left(\frac{m+1}{2}\right)} \left(\frac{n+1}{2}\right)$
Binary Search	$iterations = \log_2 (n)$
Linear Search	$iterations = n$
QuickSort + Binary Searchc = 3.321928095	$iterations = \log_{10} (n^{n+c})$

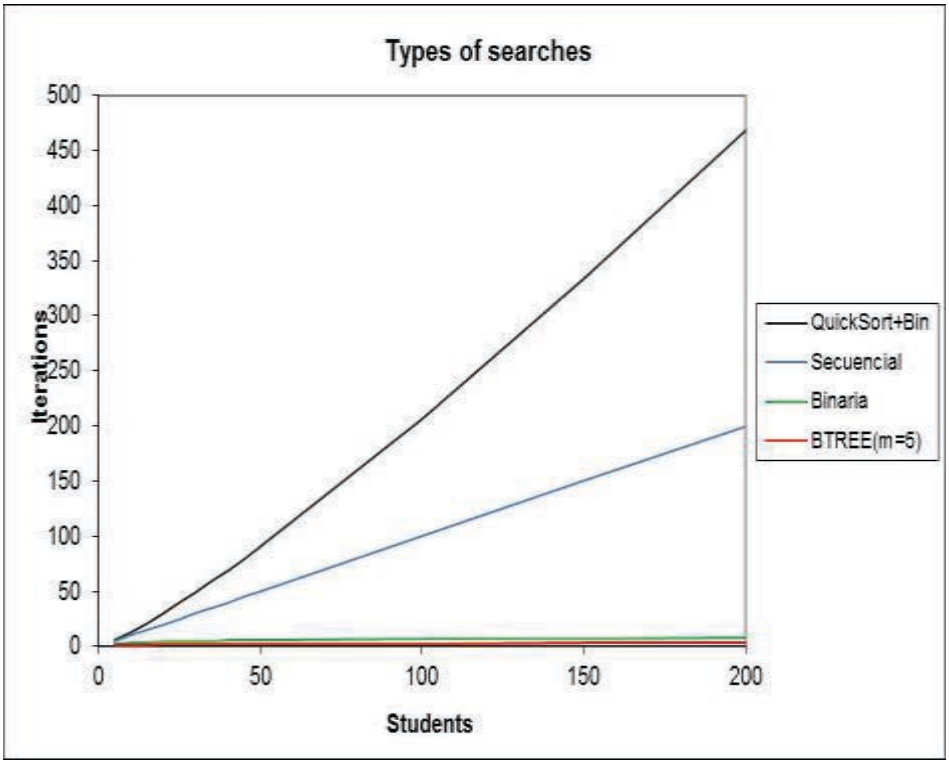


Figure 2. Theoretical behavior of each search method, as shown in Table 1.

ent events. Those data are always ordered, thus a sorting algorithm is not needed and the algorithms maintain their theoretical complexity. For this reason, the two most efficient algorithms were chosen which are, according to Table 1, the Binary search and B-TREE algorithms are.

Binary Search

This type of search works with a set of ordered keys, which divides in two parts and compares a sought key with the central one. If it is not equal, the limits of the rank are defined, depending on whether the central key is greater or less than the sought one, which considerably reduces the search set. This process is performed iteratively until the sought key is found. Binary search always finds the sought key if it exists. The worst case would be that the desired key would be the last one compared. Consequently, the number of iterations would be defined by (2), where the number of iterations is directly affected by the number of keys in the sample.

$$iterations = \log_2(n) \tag{2}$$

B-TREE Search

B-TREE search performs a specific search for a key in a balanced tree. This type of search makes a multi-way branching decision according to the number of children per node (order). As input, B-TREE search takes a pointer to the root node and searches for a key in a sub tree [10].

According to this, a B-TREE of order D can store up to 2D keys in each of its tree nodes, with a maximum of 2D + 1, which corresponds to the maximum number of children per node. This is true for all nodes, except the root node, which could have at least one key, and as a result two pointers [8]. Therefore, the depth d is between a lower (LB) and an upper bound (UB), which are defined by equations 3 and 4 [9].

$$LB = \log_m(n+1) \tag{3}$$

$$UB = 1 + \log_{\left(\frac{m+1}{2}\right)}\left(\frac{n+1}{2}\right) \tag{4}$$

In equations 3 and 4, m = 2D + 1 indicates the total number of pointers per node in the tree, and n is the number of keys in the tree. A graphical representation is shown in Figure 3, and is based on the lower bound (Equation 3) of a B-TREE with D = 1. It has three pointers (m = 3) and two keys per node (2D = 2), giving a depth of three (LB = 3) and n = 26 keys. The depth is very important because is the number of iterations required to find the sought key.

Applying Equation 4, the upper bound of the same B-TREE can be obtained. The result is a B-TREE with four levels. In the same way, a balanced tree is obtained, which is shown in Figure 4. It is not complete because it does not use the maximum number of pointers per node.

The levels vary, due to the handling of balancing, and how the keys are accommodated into the nodes. For example, in Figure 2, the maximum number of pointers per node is used. This is unlike Figure 4, where the nodes do not use all their keys, and most of them have only two pointers, which increases the levels of the tree (UB = 4.75).

4 Binary and B-TREE Search Algorithms

The two algorithms explained in the last section were programmed using C standard language. Before the experimental testing, a complexity analysis of both algorithms was performed, obtaining their temporal function in the worst case. The algorithms' programs are shown below (Figure 5, and 6).

```
int binary_search(int *array, int n, int num)
{
    int upper, lower, test;

    upper = n;
    lower = -1;
    while(upper - lower > 1){
        test = (upper + lower) / 2;
        if(array[test] == num) return test;
        if(array[test] < num) lower = test;
        else upper = test;
    }
    return -1;
}
```

Figure 5. Binary search algorithm

The Binary search algorithm presented in Figure 5 shows the process of checking the feasibility of the solutions in a general way. First of all, the function receives some parameters; the first one contains the array of students or teachers (keys) assigned to different events in a timeslot (*i, j*), which

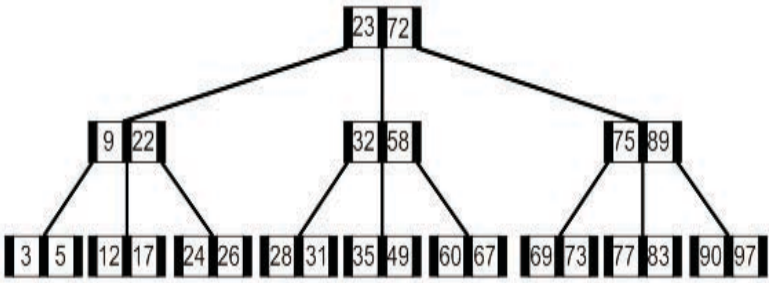


Figure 3. Lower Bound of a B-TREE order three, complete and balanced

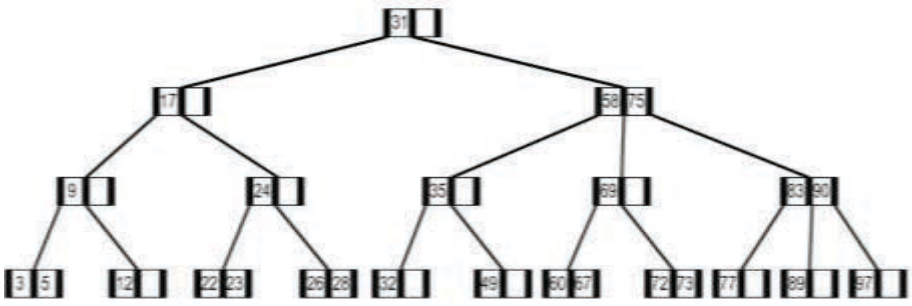


Figure 4. Upper Bound of a B-TREE order three, incomplete and balanced

have to be carefully checked in each classroom of the same timeslot (*i, j*) to avoid overlaps. The second one contains the number of keys (students or teachers) in the variable array, and the last one takes the value of the sought key (desired student or teacher), which is to be searched into the array.

The body of the function performs an iterative process, which compares the sought key, continuously halves the search space until the desired key has been found, and returns the position of the specific key found. If the key has not been found, the algorithm returns -1, indicating that the student does not exist in that list. The following algorithm is the B-TREE search, which is shown in Figure 6.

```
int b-tree_search(int key)
{
    pbnode node = entry;
    int i;

    while(node)
        i = 0;
        while((i < node -> Nodekeys )&&
        (node -> key[i].key)) i++;
    if(node -> key[i].key == key)
        return node -> key[i].key;
    else node = node -> pointer[i];
}
```

Figure 6. B-TREE search algorithm

The function responsible for the search receives a parameter, which contains the sought key. This value enters into an iterative process, which performs a search descending through the branches of the tree, depending whether the desired key is higher or lower than the root node. If the sought key is found, the algorithm returns the value of the key. It returns -1 if the student or teacher has not been found.

5 Computational Results

Two types of testing were performed to prove the efficiency and efficacy of both algorithms. First of all, the range of iterations required to find a specific student or teacher (sought key) was calculated theoretically. For B-TREE, equations 3 and 4 were used to obtain the lower and upper bound corresponding to a search with certain characteristics, such as number of students or teachers to be searched. For Binary search, the same conditions were used to calculate the maximum number of iterations. Secondly, experimental tests were performed, which were executed on the same computer and used the same instances to be able to directly compare the obtained results.

5.1 Evaluation of Complexity Theory

The theoretical tests were performed to evaluate the efficiency of both the Binary and B-TREE search algorithms. In order to be able to make a comparison between the algorithms, they were executed using the same values and features.

The theoretical evaluation of the complexity of the Binary search algorithm was performed using Equation 2, which calculates the number of iterations needed to find a sought student in the worst case. In case of B-TREE search, Equations 3 and 4 were used to calculate the lower bound and the upper bound, respectively.

In the experimental tests, a set of n students was used, which corresponds to the number of students assigned to a set of events in all classrooms in a timeslot (i, j) at a time. The values of n were 50, 1000, and 1, 000, 000 students. In the case of the B-TREE algorithm, the number of iterations to find a specific student depends on the depth and the amount of pointers on the tree.

The theoretical tests in the B-TREE algorithm were performed using three different keys (20, 1000, and 1000000). Based on the iterations, the lower and upper bounds were calculated for each number of keys. There is a range of iterations required to find a specific student when using the B-TREE algorithm. The obtained results are shown in Figure 7.

The range shown in Figure 7 for each quantity of students indicates the possible number of iterations required to find a specific student, which is a function of the input size. The range could be defined as R = UB – LB, where UB is the maximum number of iterations, and LB refers to the minimum number of iteration that can be generated by the B-TREE search according to Equations 3 and 4. It is worth mentioning that for any value of n, the value of R will be large if the number of pointers is small. Therefore, the value of R decreases and remains almost constant as the number of pointers increases.

As was done with the B-TREE search, theoretical calculations were made with the Binary search.

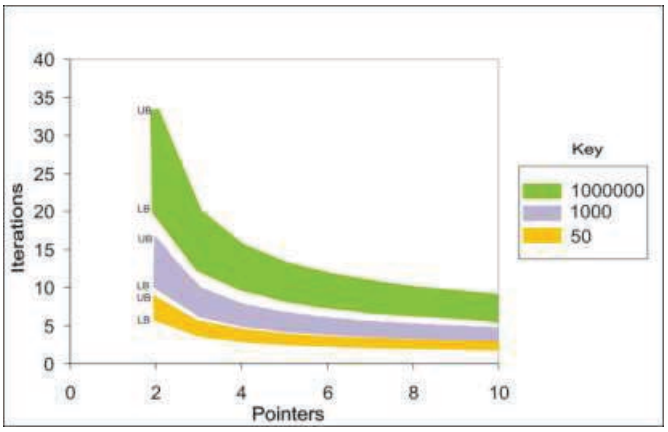


Figure 7. Theoretical results of B-TREE search using different values of keys

The number of iterations required to find a sought student was calculated, using the same number of students as in B-TREE. The calculations were performed using Equation 2. These results were compared with results obtained from Equations 3 and 4 for a B-TREE with m = 5. The comparison is shown in Figure 8.

The graph shown in Figure 8 represents a comparison between the analyzed algorithms, Binary search and B-TREE search, with respect to the number of iterations required to find a desired student if the quantity of students varies (Keys). The results showed that the B-TREE search needs less iterations than Binary search in all the cases. In the case of B-TREE (Figure 8), while the number of students (keys) increases, the number of iterations becomes constant. Binary search, on the other hand, increases its number of iterations.

According to the results shown in Figure 8, it can be concluded the B-TREE search is more efficient than Binary search for $5 \leq n \leq 200$, due to the theoretical analysis which demonstrated that B-TREE needs fewer iterations to find a specific student than Binary search, working under the same conditions (number of students n, and value of m).

Nevertheless, the exact number of iterations required for the B-TREE search will depend on the implementation of the algorithm.

5.2 Experimental Results

The experimental tests of both algorithms B-TREE search (Figure 6), and Binary search (Figure 5) were performed in a personal computer with an Intel Xeon processor 3.0 GHz, 2 GB RAM, and windows XP professional as the operating system. The compiler used to program both algorithms was Visual C++ 2008.

The testing problems used during the experimental tests were randomly generated. The sets of n students evaluated by both algorithms to prove their efficiency were generated using pseudo-random numbers ranked in ascending order. Each testing problem was evaluated using both algorithms (Fig. 5 and Fig. 6). In the case of B-TREE search, the experimental tests were performed using different m ($m = 3$, $m = 4$, $m = 5$ and $m = 30$).

The experimental tests were computed using several testing problems varying the value of n (students), where $5 \leq n \leq 1000$. Each algorithm was executed until a sought student was found 10,000,000 times, each testing problem was executed 30 times by each algorithm, and the average was calculated, which is shown in Figure 9.

The comparison presented in Figure 9 shows the relationship between the number of students used and the time required to find a specific student for both algorithms. Some changes were applied to the B-TREE algorithm; the variations were the value of m used in the tree.

According to the experimental results shown in Figure 9, it can be seen that the efficiency of both algorithms, taking into account the three versions of B-TREE search, is similar when n is between 5 and 10 students. After 15 students, the behavior of B-TREE search (according to the number of iterations), using trees with different m, is not uniform; it varies depending on the number of students. This is unlike the behavior shown by the Binary search, which is a function of the number of students considered.

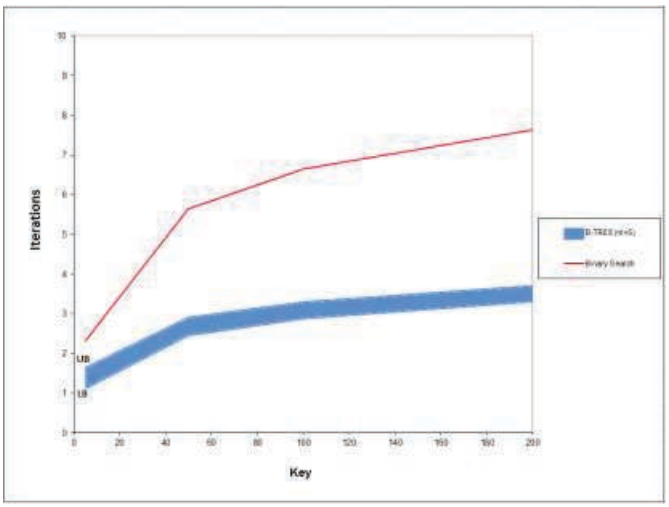


Figure 8. Comparative graph of theoretical results of B-TREE search vs. Binary search

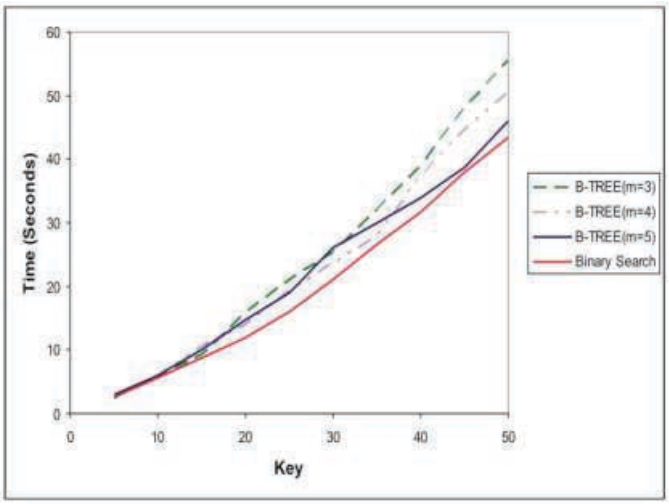


Figure 9. Experimental results: B-TREE search (applying different orders) vs. Binary search, $5 \leq key \leq 50$.

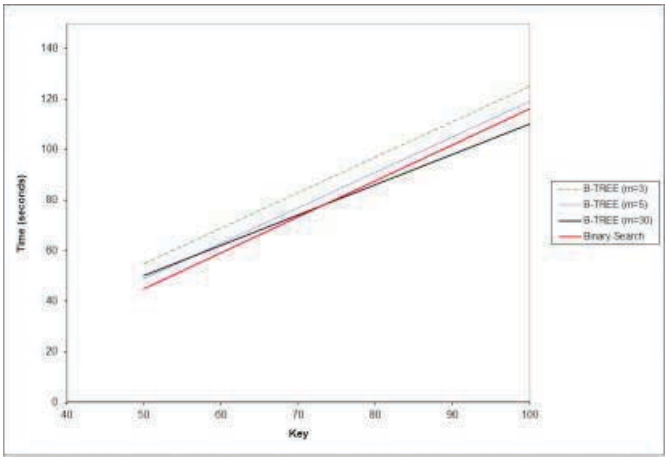


Figure 10. Experimental results: B-TREE search (applying different orders) vs. Binary search, $50 \leq key \leq 100$.

In spite of the theoretical results obtained in section 5.2, where the B-TREE search was the most efficient algorithm, the experimental results presented in Figure 9 show a different situation. Binary search demonstrated convincingly that it is more efficient than any variation of B-TREE search for small ranges; in this case a range of $5 \leq n \leq 50$ was used.

In spite of the theoretical results obtained in section 5.2 that show the B-TREE search as the most efficient algorithm, Figure 10 shows a different situation for up to 75 students. Binary search demonstrated convincingly that it is more efficient than any variation of B-TREE search for small ranges, where $50 \leq n \leq 75$.

In Figure 11, the student ranges increase, where $50 \leq n \leq 1000$. For B-TREE search, trees of the order 3, 5, and 30 were used. The obtained results show a different situation when compared with Figures 9 and 10. In this case, Binary search is not the best algorithm. It can be observed that, unlike for small ranges, for large values of n, the m value of the tree used in B-TREE search is important.

According to Figure 11, B-TREE search is the best option, because it involves less computing time than Binary search. It is interesting because according to the results for $75 < n \leq 1000$, the best option is B-TREE search using a tree order 5. B-TREE with a tree order 30 is a good option too because it is more efficient than Binary search. On the contrary, B-TREE with a tree order 3 presents worse behavior than Binary search.

Using $5 \leq key \leq 50$, some experimental tests were performed with the B-TREE search increasing the order of the tree (m = 8, 9, 10, 18, 19, and 20) to analyze how the order of the tree affects the efficiency of the algorithm. The results are shown in Figure 12.

According to Figure 12, we can conclude that increasing the tree order does not necessarily improve the algorithm efficiency, considering that the behavior of the algorithm is very similar in the six cases shown for a specific range of key.

Contrary to their theoretical complexity reported in literature, Binary search demonstrated itself to be better than B-TREE search in all cases on all the tests performed using a small rage of students $5 \leq key \leq 75$. For bigger values of key > 75, B-TREE search with m = 5, and m = 30, demonstrated itself to be more efficient than Binary search.

This study proves that B-TREE search is a very efficient algorithm when it is necessary to manage a lot of data. Due to this, B-TREE search and its variants are commonly used in Relational Database Management Systems to access data quickly [10, 11].

The theoretical complexity is an asymptotic complexity which is evaluated in the worst case. The results in Section 5.1 are justified for large values of n. This is seen in Figure 11, where B-TREE shows better performance for a value of $m \geq 5$. One of the reasons that the Binary search is more efficient than B-TREE search for small ranges of students is the number of instructions that must be evaluated. B-TREE requires fewer iterations, but it evaluates more instructions in each one, unlike Binary search, which performs a greater number of iterations, but each iteration evaluates fewer instructions.

6 Conclusion

In conclusion, according to theoretical complexity of B-TREE search and Binary search, the number of iterations to be evaluated is smaller in B-TREE search than in Binary search, but the number of instructions evaluated experimentally in Binary search is smaller than B-TREE to n ¡ 200.

Theoretically, there are two cases for B-TREE. The first one generates a complete tree and the second one generates an incomplete one. In the second case, due to its incompleteness, the tree depth tends to be higher than a complete one. As a result, the number of iterations required to find a sought student increases.

Experimental results showed that Binary search is more efficient than any version of B-TREE search evaluated in this research, for less than 200 students. However, for more than 200 students, B-TREE demonstrated itself to be faster. The reason for this behavior is the number of instructions to be evaluated. B-TREE requires fewer iterations, but it evaluates more instructions in each one. This is unlike Binary search, which performs a greater number of iterations, but it evaluates fewer instructions per iteration.

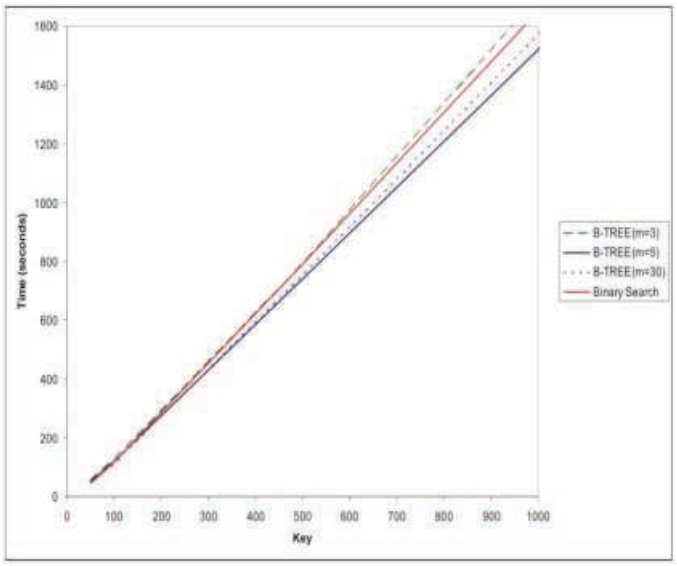


Figure 11. Experimental results: Comparison between B-TREE search, and Binary search using large student ranges, $50 \leq key \leq 1000$.

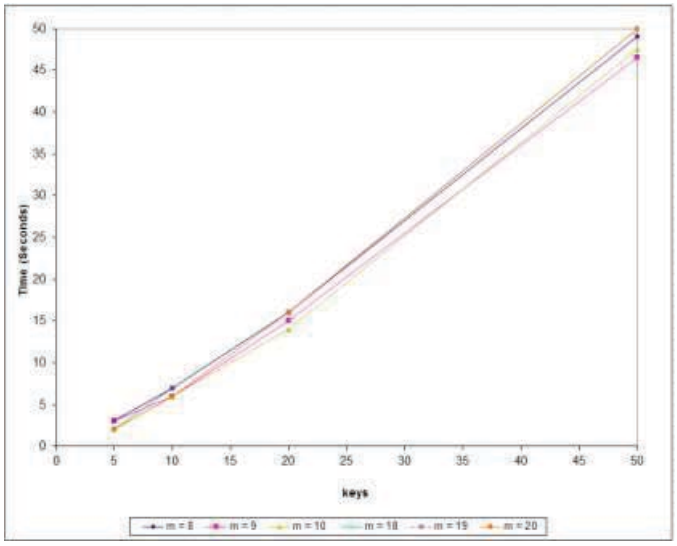


Figure 12. Experimental results: B-TREE search increasing the order of the tree for $5 \leq key \leq 50$

With the results obtained experimentally, it can be concluded that in the case of the University Course Timetabling Problem, the best option is to implement a combination of a Binary search and a B-TREE algorithm, because the number of students is different in each class, ranging from 5 to 300 students in a typical case of UCTP. To find overlaps of students in timeslots when a class has up to 75 students, a Binary search should be used. When a class has over 75 students, a B-TREE search should be used. Following these guidelines, testing to ensure satisfaction of H1 and H2 type constraints in UCTP models can be done more efficiently.

References

[1] M.R. Garey, and D.S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, USA, ISBN 0-7167-1044-7, 1979.

[2] S. Even, A. Itai, and A. Shamir, On the Complexity of Timetable and Multicommodity Flow Problems, SIAM Journal on Computing, 5(4):691-703, ISSN 0097-5397, 1976.

[3] C. H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Dover Publications Inc., U.S.A., ISBN 0-486-40258-4, p. 496, 1998.

[4] R. Johnsonbaugh, Discrete Mathematics, 6th Edition, Prentice Hall, U.S.A., ISBN 0-13-117686-2, ISBN 0-13-117686-2.

[5] B. Paechter, R. C. Ranking, A. Cumming and T. C. Fogarty, Timetabling the Classes of an Entire University with an Evolutionary Algorithm. Parallel Problem Solving from Nature (PPSN) V. Lectures Notes in Computer Science 1498, Springer-Verlag, Berlin, pp. 865-874, 1998.

[6] O. Rossi-Doria, M. Samples, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Sttzle. A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem, Lecture Notes in Computer Science, Vol. 2740 pp. 329-351. Springer-Verlag, Berlin, Germany, 2003.

[7] J. F. Korsh, Data Structures, Algorithms and Program Style, ISBN: 0871509369, PWS Computer Science, USA, p. 499, 1986.

[8] O. Cair, and S. Guardati., Data Structures, ISBN: 9701059085, McGraw-Hill, p. 423, Mxico 2006.

[9] D. F. Stubbs and N. W. Webre, Data Structures with Abstract Data Types and Pascal, ISBN: 0534092640, Brooks/Cole Pub. Co., p. 471, 1994.

[10] T. H. Cormen, C. E. Leiserson and C. D. Stein, Introduction to Algorithms, 2nd Edition. Mit Press, U.S.A., ISBN 0-262-03293-7, 2001.

[11] T. Cunnolly and C. Begg, Database Systems, A Practical Approach to Design, Implementation and Management, Fourth Edition, Addison Wesley, U.S.A., ISBN 0-201-70857-4, 2004.