CHALMERS

K-Priority Scheduling of Hard Real-Time Implicit-Deadline Periodic Task Systems on Uniprocessor

Master of Science Thesis in Intelligent Systems Design

GILBERT MENSAH

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, August 2011

K-Priority Scheduling of Hard Real-Time Implicit-Deadline Periodic Task Systems on Uniprocessor

Gilbert Mensah

Examiner: Jan Jonsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Abstract:

The Rate Monotonic (RM) scheduling algorithm (static priority scheme) has an advantage in that it is simple to implement and incurs less run time overhead. It however has a disadvantage that it cannot guarantee 100% utilization of the processor for many tasks set having 100% tasks utilization. The earliest deadline first (EDF) scheduling algorithm (dynamic priority scheme) has a significant advantage in terms of processor utilization but increase in runtime overhead undermines this advantage.

In this thesis, a dual priority algorithm is proposed. Each task has two static priorities – lower and upper band priorities with each priority band following the rate monotonic priority ordering. Initially each task executes with the lower band priority and promoted to the upper band priority after a particular (promotion point) time offset from the release of the task. We show by using simulation that 100% processor utilization is possible and our conjecture is that 100% processor utilization is possible.

Key words: Rate monotonic scheduling algorithm, Earliest Deadline first Scheduling algorithm, static priority, dynamic priority, utilization, scheduling, heuristic, dual priority.

Acknowledgement:

I would like to express my deepest appreciation to my supervisor Prof Jan Jonsson for his time and dedication throughout my thesis and for giving me the opportunity to be part of the dependable real-time systems research group.

I would like to give a special thanks to Risat Mahmud Pathan. You were there to help no matter the time or day of the week, and your assistance to me has been priceless.

I am also very grateful to my family and friends who have given me unconditional support and love through this long process.

Contents

Introduction:

Our daily interaction with computers has increased rapidly in the last couple of years. We interact with them when washing, watching TV, playing games, driving etc. One group of computers systems we often interact with is the '*real-time computer systems'* which have timing constraints. The most important timing constraint of real-time systems is about meeting the *deadlines* of tasks. This implies that for a real-time system to have a correct behavior, the logical or functional output should be delivered at the right time. A class of real-time systems where a deviation from the timing constraints can cause catastrophic consequences like loss to human life or a significant economic loss is the *hard real-time system*. As such for hard real-time systems, the utmost concern of designers is about meeting all the deadlines of the application once the system is in mission. This does not mean that the functional correctness is not important, since there is no use for a wrong result produced at the right time anyway. To ensure that the timing constraint of a real-time system is satisfied, the appropriate *task scheduling* algorithm must be used.

Task scheduling is a means of determining the order in which the various tasks in the application are to be taken up by the processor for execution. One way to achieve this is by assigning to each task a priority value which is used by the scheduler to determine the appropriate task to execute on the processor. The higher the priority value, the higher the urgency that the task needs to be executed on the processor. Based on how the priority values are assigned, scheduling algorithms can be divided into static priority and dynamic priority scheduling algorithms. If the priorities of the tasks are assigned before run time and do not change, the priority assignment is static, but if the priority values can change during the execution of the tasks, then the priority assignment is dynamic. An example of the static priority assignment is the Rate Monotonic scheduling algorithm (RM) and the earliest deadline first (EDF) is an example of the dynamic priority assignment policy.

The run time system for RM is very simple and incurs less overhead because the priority values are assigned before run time. These advantages make this scheduling algorithm an ideal candidate for many commercial and safety critical applications. Due to the fact that this algorithm can only guarantee about 70% usage of the processor's capacity for large number of tasks, these advantages are undermined. On the other hand because the priority values of EDF are assigned during run time and can change, the run time system of the EDF are relatively complex and incurs more overheads than the static counterpart. This dynamic priority behavior makes the EDF to use all (100%) of the processors capacity, making it preferred in cases where higher processor usage is needed for large number of tasks. There is therefore the need to develop an algorithm that combines the advantages of both RM and EDF that is, an algorithm that can introduce dynamic behavior into RM to achieve higher processor usage.

This thesis seeks to develop an algorithm that aims to increase the processor utilization of static priority (RM) scheduling algorithm from 70% to 100% (while maintaining its advantages for independent periodic task systems on uni-processors) by introducing minimum dynamic priority behavior.

The first contribution of this thesis deals with the development of a heuristic algorithm called the *dual priority heuristic algorithm.* This algorithm, unlike RM, assigns two static priority values to each task called lower and upper band priorities with all upper band priorities higher than all lower band priorities, and the priority ordering in both bands following the RM priority ordering. The algorithm starts by executing tasks with their priority set to the lower band values, but to achieve a higher utilization (100%) than normally achievable with RM, the algorithm determines the exact time instant from the release of each task (known as the promotion point) at which the lower band priority of the task needs to be changed or promoted to the upper band priority values to avoid a task from missing its deadline. The main challenge in developing this algorithm is finding the promotion points. Assuming a task set with periods T_1 ,T_2 , T_3 , ..., T_n , then an exhaustive search approach would require $T_1 * T_2 * T_3 *\ldots\ldots * T_n$ number of iterations in the worse case to determine the promotion points. The developed algorithm reduces this huge number of iterations to just $T_1 + T_2 + T_3 + \ldots\ldots + T_n$ in the worse case. This represents a huge reduction making this algorithm much faster than existing algorithms that employ the exhaustive search approach.

The second contribution of this thesis is carrying out an extensive simulation using the heuristic algorithm developed to determine if a task set with 100% processor utilization could be found that is not schedulable by the algorithm (in other words no promotion point can be found). This was done by randomly generating task sets which are not schedulable under RM and using the heuristic to determine the promotion points. With the assumption that promotion points could always be found, another contribution of this thesis is to show that the algorithm can achieve the 100% processor utilization bound.

Finally, the algorithm developed has been packaged as a simulation tool which can easily be used by real-time systems engineers to find promotion points. The tool determines the promotion point for a set of n periodic tasks.

The rest of the thesis is organized as follows: Chapter 2 describes the related background of real-time systems and scheduling algorithms. Related research on dual priority algorithm is discussed in Chapter 3 and detail discussion of the dual priority heuristic algorithm developed in this thesis is presented in Chapter 4. The simulation results as well as detailed description of task generation techniques are presented in Chapter 5. Chapter 6 concludes the thesis with future work and conclusion.

Chapter 2

In this chapter, the related background of real-time systems and scheduling is presented.

2.1 Real-time Systems:

A real-time computer system is one in which the result expected is not the only important thing, but also the time as which the result is obtained. Computers that fall under this group can normally be found in safety critical control systems as well as transaction systems. For example, in a brake-by-wire system, the computer that controls the braking systems of the car is a real-time system because when the brakes are applied, it is expected that the car stops within a specified amount of time [14]. There are consequences when the results are not obtained at before a pre-specified deadline. The severity of the consequence of not meeting the deadlines further classifies real-time systems into hard and soft real-time systems. A soft real-time system is one in which the consequence of a result after its deadline is minimal, and does not pose any threat to human life. An example is a live video streaming, if the arrival of the frames are delayed, it will result in a degraded video but not pose severe threat like death or injury. Hard real-time systems on the other hand posses threat to human life or economic loss when the results are produced after the deadline. An air bag system which delays when a car crushes has a severe effect of causing death or injury to the passenger.

This thesis deals with scheduling of periodic tasks in hard real-time systems. This is due to the fact that many monitoring and safety critical systems are modeled as recurrent tasks because they are predictable and well understood [1, 2]. The most relevant real-time scheduling concepts in this thesis are: periodic task systems, implicit deadline task systems, ready task, task priority, preemptive scheduling algorithm, and feasibility conditions of scheduling algorithm.

2.1.1 Periodic task systems:

A task is what actually gets scheduled on a processor; it is therefore the basic component of scheduling. Formally, it can be defined as a unit of work that when executed generates some output or service from an application. This unit of work can be whole of a program of just a code fragment. Examples of tasks include reading from a sensor, sending commands to actuator, doing some computations etc. An instance of a task is known as a *job* of the task. *When a set of tasks are characterized by a period , deadline and worse case execution time (WCET) , then the task set is said to be a periodic task system.*

Period:

This is the time separation between the release times of successive jobs of a task. At each period, a job of the task is released and ready for execution.

Deadline:

When a job of a task is released at each period, it has a maximum time form this release time by which it must complete its execution. This time is known as the *relative deadline* of the job and is the same for all jobs of a particular task. The time instant represented by the addition of the release time of a job and the relative deadline is known as the *absolute deadline*.

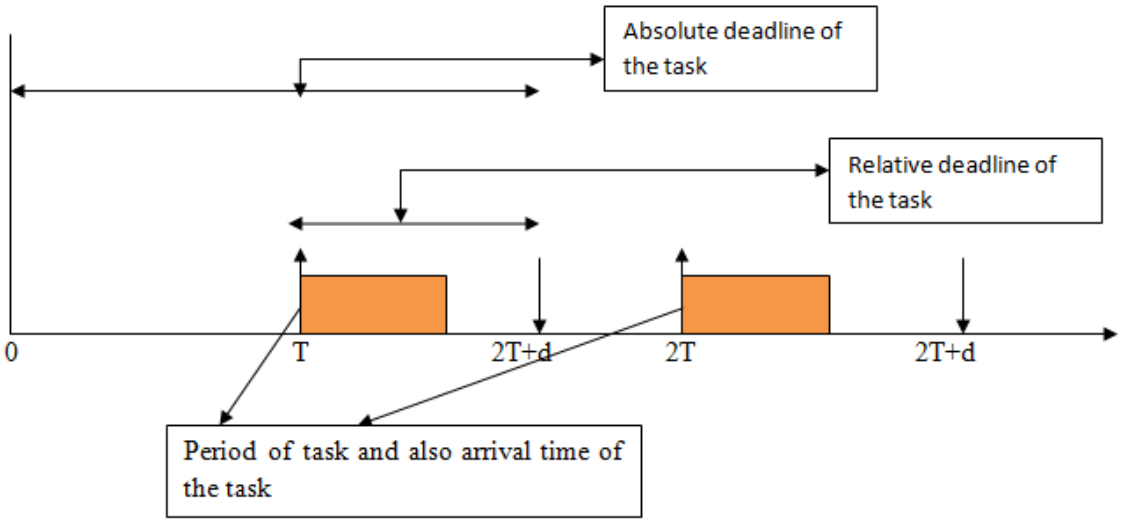

Fig 1, period, arrival time, relative and absolute deadline of a task

Worse Case Execution Time (WCET):

This is the maximum amount of time a job may take to execute after it becomes eligible for execution. This time must be less than the relative deadline of the task or in the worse case be equal to it.

2.1.2 Implicit deadline task systems:

A task is said to be constrained if the relative deadline is less or equal to its period. *The task becomes an implicit deadline task if the relative deadline is exactly equal to the period*. If the task is neither constrained nor implicit, then it is arbitrary. This thesis considers the scheduling of implicit deadline periodic task systems.

2.1.3 Ready tasks:

At each period, a job of a task is released for periodic task systems. This released job is eligible for execution and thus can be executed on the processor when dispatched. The set of all such jobs is kept in a ready (active) queue from where they are selected by the run time dispatcher to be executed on the processor based on priority.

2.1.4 Task priorities:

Scheduling basically involves assigning tasks to processors when they are ready. But when two or more tasks are ready, then certain rules must be applied in order to select one task for execution. The rule applied is based on the task with highest *priority*. The priority of a task is a value assigned to the task that suggests the relative *urgency* of that task for execution. The priority can be fixed (static) or dynamic.

2.1.4.1 Fixed Priority (FPS):

This is also known as static priority scheduling in that the priorities of the tasks are assigned before run time and jobs of a task has the same priority relative to other jobs. The deadline monotonic (DM) and the rate monotonic scheduling (RM) algorithms are examples of fixed priority algorithm [2, 15]. DM assigns to each task a priority value based on the relative deadline to the task, the task with the smallest deadline has the highest priority and the one with the largest deadline has the lowest priority. RM assigns each task a unique priority based on its period; the shorter the period, the higher the priority, thus the priority can be determined as the inverse of the task periods. The focus of this thesis is on RM for static priority algorithms. It has been found by Liu and Layland in [2] that the rate monotonic algorithm is optimal for scheduling task set under fixed priority on uni-processors for implicit deadline tasks systems. [2, 3, 4, 5].

2.1.4.2 Dynamic Priority:

Unlike the fixed priority, there is no restriction to the manner in which priorities are assigned to individual jobs of a task in the dynamic priority scheme. This implies that different jobs of a task may have different priorities relative to other jobs. The earliest deadline first algorithm (EDF) is an example of a dynamic priority algorithm. It assigns priority to tasks based on the deadlines of their current job, thus the highest priority goes to the task whose current job has its deadline nearest while the lowest priority goes to the task whose current job has its deadline furthest. The EDF algorithm has been found also by Liu and Layland in [2] to be optimal for scheduling tasks under dynamic priority for uni-processors. [3, 4, 5].

2.1.5 Preemptive Scheduling:

A scheduling algorithm is preemptive if the release of a new job of task with higher priority can preempt the job of a currently running task with lower priority.If the scheduling algorithm is non-preemptive, then the lower priority task will be allowed to finish its execution first RM and EDF are examples of preemptive scheduling algorithms. [2, 3, 4, 5]. The scheduling algorithm developed in this thesis is a preemptive scheduling algorithm on a uni-processor platform.

2.1.6 Feasibility Analysis:

To determine whether a given scheduling algorithm ensures that all the timing constraints of the set of tasks are met, a feasibility analysis has to be conducted. Baker and Shaw in [16] proposed a simple way to determine feasibility for periodic task systems known as the cyclic schedule (time line schedule). They observed that after a length of time which is equal to the least common multiple (LCM) of the periods of the task set, the schedule is just a repeat of the initial schedule prior to the LCM time value. This suggest that if the task set can be scheduled up to the LCM of the periods of the task set, then it will be schedulable for all time instants. Although simple, it becomes computationally impractical for large task sets. Feasibility tests (which are mathematical expressions) can be used to overcome this problem. Given a schedule, a feasibility test can be used to determine whether the task set is schedulable or not under the given run-time system. Two main feasibility tests are of concern to this thesis, the sufficient test and the exact test. Before discussing the two feasibility test, a brief task model is presented [3, 4].

2.1.6.1 Task Model:

Each task has a period T_i, a computation time also known as the WCET C_i, a deadline D_i which is the same as the period. With these values the utilization of each task which is the ratio between the task WCET and its period can be calculated. The total utilization of the task set is the sum of all the task utilizations.

2.1.6.2 Sufficient feasibility test:

Given a scheduling algorithm, the sufficient feasibility test guarantees that all deadlines of the tasks will be met if it produces a positive answer. No decision can be made when the answer is negative. Liu and Layland provided a sufficient test for RM scheduling algorithm in [2] based on the total utilization of the task set, given as,

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \qquad \text{(Eq 1)}$$

Where n is the number of tasks. The above test shows that as the number of tasks gets very large, the utilization value reduces to about 69.3%. [2, 3, 4].

2.1.6.3 Exact feasibility test:

Given a schedule, the exact test produces a positive answer if the task set is schedulable and a negative answer if it is not. Liu and Layland provided in [2] an exact test for EDF algorithm based on the task utilization of the processor,

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1, \qquad \text{(Eq 2)}$$

The above test shows that, as long as the task utilization is less than or equal to 100%, EDF can schedule the task set. [2, 3, 4].

In [13], Audsley *et-al* proposed an exact test for RM known as the response time analysis (RTA). In RTA, the maximum time interval between the release of a job of a task and its completion time called the worse – case response time (WCRT) of the task is computed. If this WCRP is less or equal to the deadline of the respective tasks, then the task set is feasible or schedulable and vice versa. The WCRT equation proposed in [13] is given as

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil C_j \qquad \text{(Eq 3)}$$

Where R_i and C_i represents the response time and WCET of task T_i . The set $hp(i)$ represents the set of higher priority tasks with respect to task T_i and P_j and C_j represent the period and wcet of such high priority tasks. The term $\sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{P_j} \right\rceil C_j$ represents the maximum interference from all high priority tasks than T_i . This term combined with WCET of T_i gives the WCRT. Since there is R_i on both sides of Eq 3, the equation can only be solved recursively and is transformed into Eq 4

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{P_j} \right\rceil C_j \qquad \text{(Eq 4)}$$

Where R_i^n represent the response time in the nth iteration

2.1.7 Difference between EDF and RM:

RM is supported by many programming languages and operating systems because it is easier to implement as the priority is static. Since EDF is dynamic, it requires a complex run-time system and hence has more overhead. [3]

Also RM is more predictable under overload conditions as compared to EDF which becomes unpredictable and can experience domino effect where large number of tasks misses their deadlines. [3]

In terms of utilization and feasibility, EDF is superior to RM since it can schedule all tasks RM can schedule but not all task scheduled by EDF can be schedule by RM. When the utilization is 69.3% (69.3% in the worse case if n is infinity; 82% to 69% for n = 2 to n = infinity) $< U \leq 100\%$, then RM cannot guarantee that all deadlines will be met. [3].

Chapter 3

In this chapter, the related research on dual priority algorithm is developed to form the basis for the work done in chapter 4.

3.0Why dual priority?

Due to the disjoint advantages of RM and EDF, it becomes desirable to have a single algorithm that combines their advantages. That is, an algorithm which has static properties like RM with less overhead and at the same time can achieve 100% utilization. That algorithm is the dual priority algorithm. The dual priority algorithm is a minimally dynamic priority algorithm in the sense that it assigns to each task two static priorities, hence the name '*dual priority'*. Initially each task runs with the first priority and after some elapsed time known as the promotion time, the task is promoted to the second priority to enable it meet its deadline. The algorithm has dynamic behavior since it is allowed to change the task priority during run- time and it also has static behavior since these priorities are already assigned to the task before run – time. This is basically an attempt to increase the processor utilization of fixed priority algorithm such as RM.

The problem of increasing the utilization of fixed priority scheduling based on dual priority concept is not a new one. In 1991, Harbour et al [10] observed that by breaking a task into precedent constrained subtasks, and allowing the subtask to increase its priority after a period of computation time, it was possible to schedule task set that were otherwise not possible with fixed priority. The subtasks in their scheme performed the same computation on each release.

However, in 1993, A. Burns and A. Wellings [6] found a counter example of a task set that could not be schedulable under the scheme proposed by Harbour in [10] and introduced the concept of dual priority scheduling as the alternative means of increasing the utilization of fixed priority scheduling. Tasks were split into subtask just as in [10] but the subtasks were allowed to increase their priority only after a period of elapsed time (promotion points) from the release time of the task, making the subtask not perform the same computation on each release. A look at their algorithm is presented below.

3.1 Dual priority algorithm: Burns and Wellings:

The concept of dual priority introduced by Burns and Wellings in [6] allows a low priority task that would otherwise miss its deadline to steal execution time from a higher priority task by increasing its priority. To achieve this, each task has two priority levels, called lower band and upper band. The concept of promotion point (S_i) was then introduced as the time instant after the release of the task where the priority of the task is increased to a value in the upper band. Each task has a period T_i, a computation time C_i, lower band priorityP_i^1, upper band priority P_i^2 and a promotion point S_i. As such during the time interval [0, S_i) the task execute with a priority in the

lower band and if the task still has some computation to do for the time period $[S_i, D_i]$ then it execute with a priority value in the upper band. Just for clarity, an example of the dual priority algorithm for a task set that will fail if scheduled with RM is given below.

Task set

T_i	3	4	6
C_i	1	2	1
U_i	0.3333333	0.5	0.1666667
U	1		
LCM	12		

Table 1, task information for three task with 100% U

Based on the periods of the above task set, the LCM value is 12 and as such the schedule will repeat itself after 12. This means for a cyclic (timeline) schedule, it is enough to simulate up to 11 that is the interval [0, 11) A timeline schedule using RM for the task set in table 1 is shown below

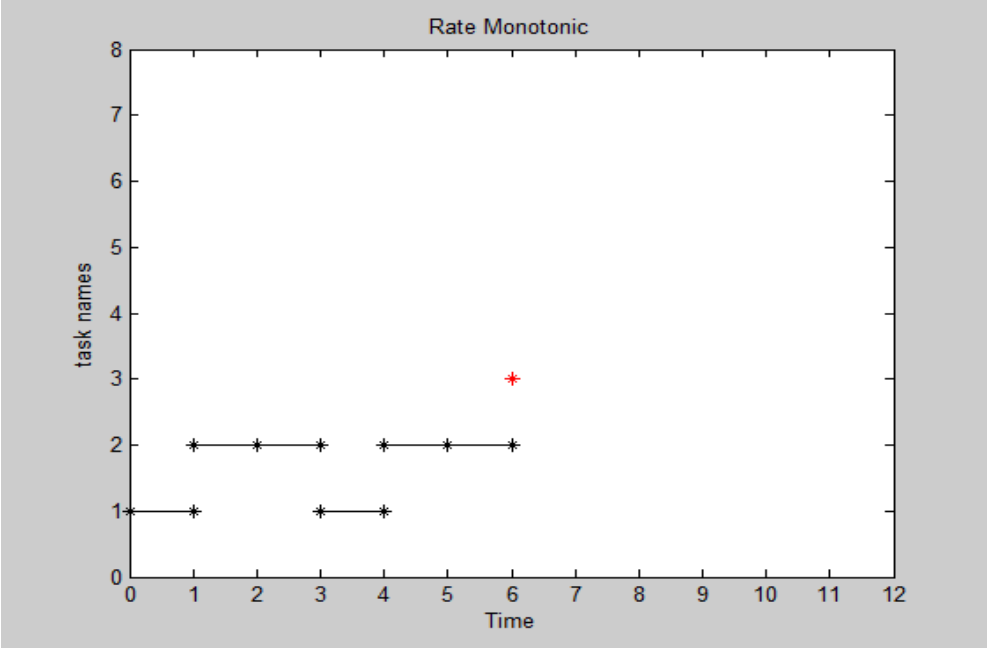

Fig 2. Time line schedule produced when task set in Table 1 is scheduled using RM. Initially all 3 tasks are released and put into the ready queue at time 0. Task 1 which has the highest priority starts its execution and completes at 1. It is immediately followed by task 2 which finishes at 3. Task 1 again is released at 3 and immediately starts executing because it has higher priority than task 3 which is still in the queue, finishing at 4. Task 2 again gets released at 4 and starts immediately and finishes at 6 forcing task 3 to miss its deadline because it did not have the opportunity to execute due to the presence of high priority tasks.

It can be seen that task 3 misses its deadline by 1 time unit at 6 because it could not find any free space between [0, 6] to execute at its initial rate monotonic priority. However, if we promote (give it a higher priority than the task that is causing it to miss its deadline, in this case task 2) task 3 at 5, then it will have a higher priority at 5 and can then execute to avoid missing it deadline. This is shown in the figure below.

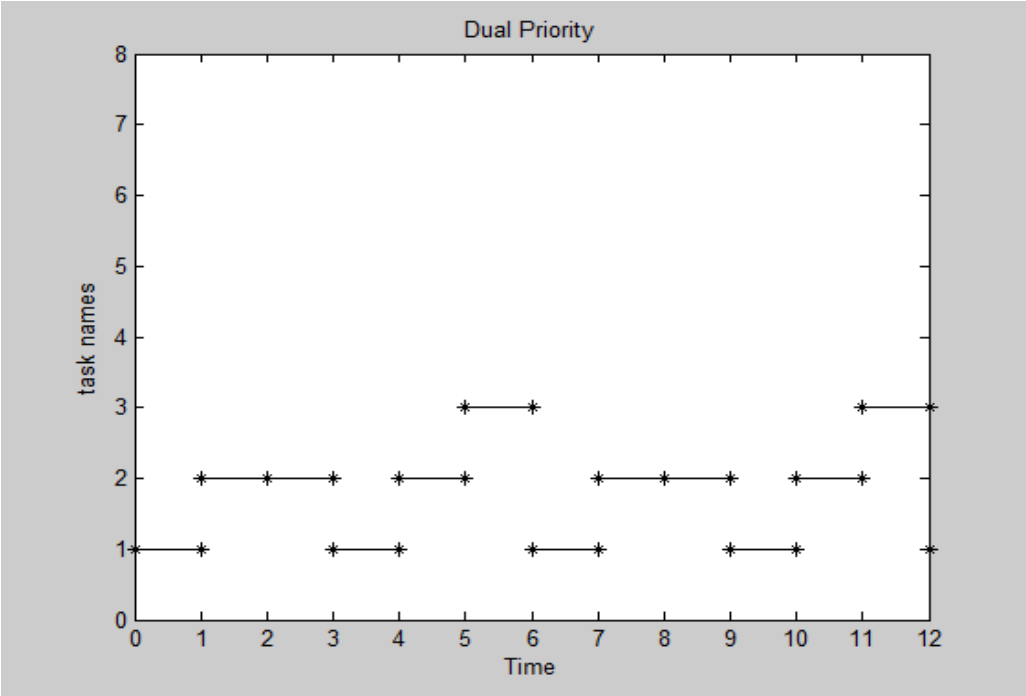

Fig 3. Result of the dual priority algorithm after promoting task 3 at 5.All task meets their deadlines as shown in the graph above.

The complete task information from table 1 is given in table 2. When the task information in table 2 is given to a dual priority scheduler, the above schedule in Fig 3 is produced where task 3 gets promoted at 5 and all the other tasks do not get promoted due to the fact that their promotion point is the same as their period.

Task set

T_i,	3	4	6
C_i	1	2	1
P_i^1	4	5	6
P_i^2	1	2	3
S_i	-	-	5
U_i	0.3333333	0.5	0.1666667
U	1		

Table 2, complete task information after the dual priority algorithm of the task set in table 1

3.2 Advantage of dual priority algorithm:

The main advantage of the dual priority algorithm lies in the fact that it combines both RM and EDF into one; as such it has all the good properties of RM and also possibly has 100% utilization.

Davis and Wellings showed in [7] that the dual priority algorithm provides an efficient and effective means of scheduling tasks with soft deadlines (including periodic, sporadic and adaptive tasks) with tasks with hard deadlines. They compared the performance of the dual priority scheme to the well known bandwidth preserving schemes such as Background, Extended Priority Exchange and dynamic Slack Stealing methods and found it to be superior.

In [8] Burns hinted that the dual priority algorithm can be used with priority-based non-preemptive communication protocols such as CAN. In CAN, the priorities (the 11bit identifiers) of the nodes (devices) are statically assigned, and at any point the node with the highest priority identifier is allowed to transmit messages. With the static priority assignment, dynamic priority algorithms such as EDF become difficult to implement on the CAN. With a minor change to the protocol, that is by equipping each node with two 11bit identifiers (two priorities), a message which has been in a node's output buffer for a defined amount of time could have its priority promoted (first 11bit is changed to the second 11 bit identifier) so it could be transmitted, making the protocol more robust

Davis also showed in [9] that the dual priority scheme can be used to construct an efficient *O(n)* acceptance test for providing online guarantees to hard real-time tasks, which he admitted was not optimal, but was sufficient and effective .

3.3 Challenges with Dual priority algorithm proposed by Burns and Wellings:

The algorithm developed by Burns and Wellings has some shortcomings and a discussion of these shortcomings as well as how they are can be resolved is given below.

3.3.1 Determining priority assignments and promotion points in Burns and Wellings:

Each task has two priority values, the lower band priority and the upper band priority. The lower band priority is the initial priority of the task, and the upper band priority is the priority value to which the task is promoted to at the time instant known as the promotion point of the task.

First the lower band priorities are assigned using rate monotonic priority assignment ordering and all promotion points S_i of the tasks are set to the value of their periods T_i. When a task misses its deadline, its promotion point is progressively moved backwards from T_i until the task

meets its deadline and the upper band priority (the upper band priority ordering is not necessarily RM) is set high enough so that the task can steal execution from the "*appropriate task or tasks*".

An appropriate task for t_i refers to the task that has a higher priority than t_i and due to that causes t_i to miss its deadline, but has the property that if t_i can be made to execute before it (by increasing t_i's priority), it will still not miss its deadline. For example if t_3 is allowed to execute at time instant k (because it has the highest priority among all ready task) and that causes t_4 to miss its deadline, but when t_4 is made to executes at k instead(due to priority promotion at k), t_3 does not miss its deadline, then t_3 is the appropriate task with respect to t_4. At such t_4 must steal execution from t_3 in other to meets its deadline. This is done by setting the upper band priority of t_4 between the lower band priorities of t_3 and t_2 so that t_4 can preempt t_3 to finish its execution. This makes this scheme complex and computationally intensive since the appropriate task must be found and the upper band priority is based on the lower band priority of this appropriate task. An example of a complete task set information scheduled with this scheme is shown below in table 3

	T	C	S	P^1	P^2	U
τ_1	4	1	-	4	-	25%
τ_2	6	1	-	5	-	16.67%
τ_3	12	3	-	7	-	25%
τ_4	30	5	29	9	6	16.67%
τ_5	36	6	23	10	8	16.67%
Total Utilisation	100%					
LCM	180					

Table 3 taken from [6] .Complete task set information with the promotion points. From the table, T, C, , S, P^1, P^2 and U represents the period ,WCET, the promotion points, the lower band priorities ,the upper band priorities and the task utilization for the tasks. Tasks with smaller priority values have higher priority than higher priority valued tasks. Tasks t_1, t_2 and t_3 need not steal from any other task and as such do not have any promotion point. This is shown by the dash (-) at their respective promotion points(S) and upper priority values. However, task t_4 has its upper band priority set to a value between the lower band priority of t_3 and t_2 and as such must steal execution from t_3 to meet its deadline, making t_3 the appropriate task for t_4. Also t_5 has its upper band priority set to a value between t_4 and t_3 and as such must steal execution from t_4 to meet its deadline. This makes t_4 the appropriate task for t_5.

Table 4 is another example which shows how complex the priority assignment in the upper band can become.

	T	C	S	P^1	P^2	U
τ_1	16	4	-	4	-	25%
τ_2	20	5	19	5	3	25%
τ_3	28	11	19	6	2	39.29%
τ_4	56	6	50	7	1	10.71%
Total Utilisation	100%					
LCM	560					

Table 4 taken from [6]. Complete task set information with the promotion points. Task t_2 needs to steal from task t_1 at time 20 and as such has its upper band priority set to a lower than the lower band priority of t_1. Task t_3 also needs to steal from t_2 and thus has its upper band priority set between the lower band priority of t_2 and t_1. Task t_4 needs to steal from t_3 at 50 and has its upper band priority set between the lower band priority of t_3 and t_2 .

From table 4, it can be seen that the priority assignment is not trivial.

It can be observed that while the assignment of the priorities in the lower band follows the rate monotonic priority assignment, that of the upper band is completely arbitrary.

The major challenges of the existing dual priority algorithm are bulleted below:

- There is difficulty in determining the priority values in both lower and upper band

- The algorithm is computationally intensive because it involves "determining the appropriate tasks"

In 2010, Burns in [8] hinted that it was possible for both the lower band and upper band priority levels to both follow the rate monotonic priority ordering with all the upper band priorities higher than all the lower band priorities. If that were the case, then it eliminates the need to determine the "appropriate tasks" in the initial algorithm.

3.4 Goals of this thesis:

- The first goal of this thesis is to eliminate the difficulty in determining the priority values in both lower and upper bands. This is achieved by allowing both bands to follow the RM priority ordering.

- The second goal deals with reducing the computational intensive nature of old algorithms by eliminating the need to determine the "appropriate tasks".

Chapter 4

In this chapter, the dual priority heuristic algorithm developed in this thesis is presented.

4.0 Assumptions:

The algorithm developed in this thesis is a preemptive algorithm; as such any task with a higher priority can preempt any other task with a lower priority.
Also because independent periodic task systems are considered, there are no task dependencies and as such tasks do not suspend themselves other than at the end of their computation.
To make the analysis simpler, the cost associated with context switches and priority changes etc are summed up in the task's WCET.

4.1 Task model:

Each task has a period T_i, a computation time also known as the worse case execution time C_i, a deadline D_i which is the same as the period, two priority levels lower band(P_i^1) and upper band (P_i^2), a task name t_i , and a promotion point S_i which is the time instant after the task is released at which its initial priority P_i^1 is promoted to its P_i^2 priority . For all task, $0 < S_i \leq D_i$. During the time interval $[0, S_i)$ the task execute with its P_i^1 priority and after time between $[S_i, D_i]$ if there is still some computation left for the task in the ready queue, that remaining computation will be executed with its P_i^2 priority.

4.2 The dual priority (DP) algorithm:

The dual priority algorithm basically looks like the well known rate monotonic scheduling algorithm with the simple modification that at the promotion point of a task, if there is some computation left, the priority of the task in the ready queue will be changed to its upper band priority.

4.2.1 Data structures used in DP algorithm:

The algorithm maintains a data structure called *releaseTimes* which stores the next release time of each task. Additionally, a data structure called *arrivedTasks* which is the same as the ready queue in most algorithms is used store all tasks that have been released. This data structure contains other structures such as *arrivedTaskName*, *arrivedTaskPriority*, *arrivedTaskWCET*, *arrivedTaskPromotion* and *arrivedTaskDeadline*.

When a task is released, its next release time is calculated and the value in its *releaseTimes* (initially set to 0) data structure is updated as the sum of the current time and its period. The task is then added to the *arrivedTasks* (initially empty) *as* follows: - the *arrivedTaskName* field is updated to the task name t_i , the *arrivedTaskPriority* field is updated to the lower band priority of the task, the *arrivedTaskWCET* is updated to the WCET value C_i of the task,

arrivedTaskPromotion field is updated to the sum of the release time of the task and its promotion point value S_i and the *arrivedTaskDeadline* is updated to the relative deadline of the task.

The *arrivedTasks* structure is then sorted based on the *arrivedTaskPriority* field values so that the task with the highest priority comes first. The algorithm ensures that the *arrivedTasks* data structure is always sorted. This is done by sorting it at times known as event times. An event time (*eventTime*) represents times which can cause a change in the *arrivedTasks* data structure. These times include the release time of a task (a task is added to the *arrivedTasks*), the finishing time of a task (a task is deleted from the *arrivedTasks*) and the promotion time of a task (the *arrivedTaskPriority* value is changed to the upper band priority value).

4.2.2 Details of the DP algorithm:

If the *arrivedTasks* data structure is not empty, then the first task (the highest priority task) is selected. The next occurrence of an event (event time) is determined as the minimum of the release times of the tasks (obtained from the *releaseTimes* data structure), the finishing time of the selected task (obtained as the sum of the current time value and the *arrivedTaskWCET* value of the selected task) and the promotion times of the arrived task (obtained from the *arrivedTaskPromotion* field of the *arrivedTasks* structure). The difference between the current time and the next occurrence of an event (*eventTime*) is calculated as eventTimeDiff (event time difference) and the selected task is executed for eventTimeDIff time units. The execution is done by subtracting the eventTimeDIff value from the task's *arrivedTaskWCET* value and if a value of zero results (task has finished executing), the task is deleted from the *arrivedTasks* data structure. The process is repeated till all tasks are scheduled.

As an example, let t_1, t_2, t_3 be three tasks with periods 3, 4, 6, WCET 1, 2, 1 and promotion points values 3, 4, 5 respectively. The lower band priorities of the task are 4, 5, 6 while the upper band priorities values are 1, 2, 3.Let the *releaseTimes* data structure contain 6, 8, 6 for the next release times of the three tasks respectively. Then at time t=4, only t_2 and t_3 will be in the *arrivedTasks* data structure with *arrivedTaskWCET* equal to 2, 1, *arrivedTaskPriority* being 5, 6, *arrivedTaskPromotion* equal to 8, 5 and *arrivedTaskDeadline* equal to 8, 6. t_2 will then be selected for execution and it's finishing time is calculated as 4+2=6.The next occurrence of an event will be calculated as eventTime= min(6,8,6,6,8,5) which will result in a value of 5. The eventTimeDIff is then calculated to be (5-4)=1, as such t_2 will be executed for only 1 time unit after which the *arrivedTasks* will change (t_3 gets promoted).

If the eventTime is equal to a promotion point of a task (*arrivedTaskPromotion* value), then the *arrivedTaskPriority* value of the task is changed to its upper band priority value at a time equal to the eventTime.

The algorithm can be summarized as follows:

1. *Select the task with highest priority for execution*
2. *Determine the next occurrence of an event (eventTime)*
3. *Determine the difference between the current time and eventTime known as eventTimeDiff.*
4. *Execute the selected task for eventTimeDiff time units by subtracting it from its arrivedTaskWCET value. If a value of zero results, delete the task from the arrivedTask data structure.*
5. *If the eventTime is equal to a promotion point of a task (arrivedTaskPromotion value), then the arrivedTaskPriority value of the task is changed to its upper band priority value at a time equal to the eventTime.*
6. *Repeat till LCM value*

Fig 4. The dual priority (DP) algorithm

4.3 Heuristic algorithm to generate the promotion points (Dual priority heuristic algorithm):

The above algorithm assumes that the promotion points have already been found. Once the promotion points can be found successfully for a task set, then the above algorithm can produce a schedule for that task set. Thus, determining the promotion point is the most important issue to the dual priority algorithm. A heuristic algorithm called the dual priority heuristic algorithm has been developed in this thesis which can be used to determine the promotion points. The heuristic algorithm makes use of the DP algorithm above. Details of the heuristic algorithm are discussed below.

Since the *arrivedTasks* data structure is always sorted according to decreasing order of priority, it is very easy to determine the highest priority task that misses its deadline by comparing the current time to the values of the task's *arrivedTaskDeadline* . Once the current time is equal or greater than a task's *arrivedTaskDeadline* value, then the task has missed its deadline. The amount of computation left for this missed-deadline task is the value of its *arrivedTaskWCET*.

Initially the promotion point S_i of each task is set to the value corresponding to the task's period T_i . The tasks are then assigned their lower and upper band priorities in accordance to the rate monotonic priority assignment with the lower band priority values ranging from $n + 1 \leq P_i^1 \leq 2n$, and the upper band priority levels ranging from $1 \leq P_i^2 \leq n$ for task $t_1, t_2 \ldots \ldots t_n$, which have been sorted according to their priorities with t_1 having the highest priority and t_n having the lowest priority. Table 5 below show the initial task set information for three tasks.

Example: (initial setting)

	WCET	T_i	P^1	P^2	S_i
Task t_1	C_1	T_1	4	1	T_1
Task t_2	C_2	T_2	5	2	T_2
Task t_3	C_3	T_3	6	3	T_3

Table 5, initial task set information to determine the promotion points of three task. The initial value of the promotion points of each task is set to the value of its period.

Given the initial task information as shown in table 5, the DP algorithm is used to schedule the task set(no task gets promoted since the promotion points are the same as their period which is also the same as the deadlines of the tasks). If a successful schedule is produced, then the no task set needs to be promoted and the task set can be scheduled by the rate monotonic scheduling algorithm. If the DP algorithm fails, then the first high priority task that misses its deadline is determined together with how much computation is left for this missed-deadline task (let that value be equal to C_x . In the worse case this will be equal to C_i).

The new promotion point for task t_i is then calculated as $S_i = S_i - C_x$, which is the previous promotion point minus C_x.This means that at time S_i from each release of task t_i, the *arrivedTaskPriority* value of the task is set to its upper band priority value in the *arrivedTasks* data structure if it still has some computation left (this remaining computation will be executed at the upper band priority). The task set is rescheduled using the DP algorithm, which repeats the above process for all tasks that misses their deadline till a termination criterion is reached.

There exist two termination criteria. The first is when the algorithm finds promotion points for all tasks that miss their deadline so that the DP algorithm produces a feasible schedule. In this case the heuristic algorithm terminates with a success signal. The second has to do with the possible range of values of the promotion points. The promotion point S_i of a task t_i lies between $0 \leq S_i \leq T_i$. If the promotion point is 0, then the task gets promoted as soon as it is released, as such the value of the task's *arrivedTaskPriority* is always set to the upper band priority value of the task. If the promotion point value is T_i , then the task never gets promoted since the *arrivedTaskPromotion* value of the task will be the same as the *arrivedTaskDeadline* value of the task. Since the promotion point value of a task is initially set to T_i and is reduced every time it misses its deadline, then the promotion point of a task that always misses its deadline can be a negative value. A negative value means the task needs to be promoted even before it is released, which is not possible as such the algorithm terminates with a failure signal.

A couple of examples will help better understand how the above algorithm works.

Example 1

T_i,	6	8	8
C_i	3	2	2
P_i^1	4	5	6
P_i^2	1	2	3
S_i	6	8	8

Table 6. Initial task information for three tasks before dual priority algorithm. The promotion point of all the tasks is set to the values of their respective periods.

When the above task set is scheduled with the dual priority heuristic algorithm, the schedule obtained is shown below,

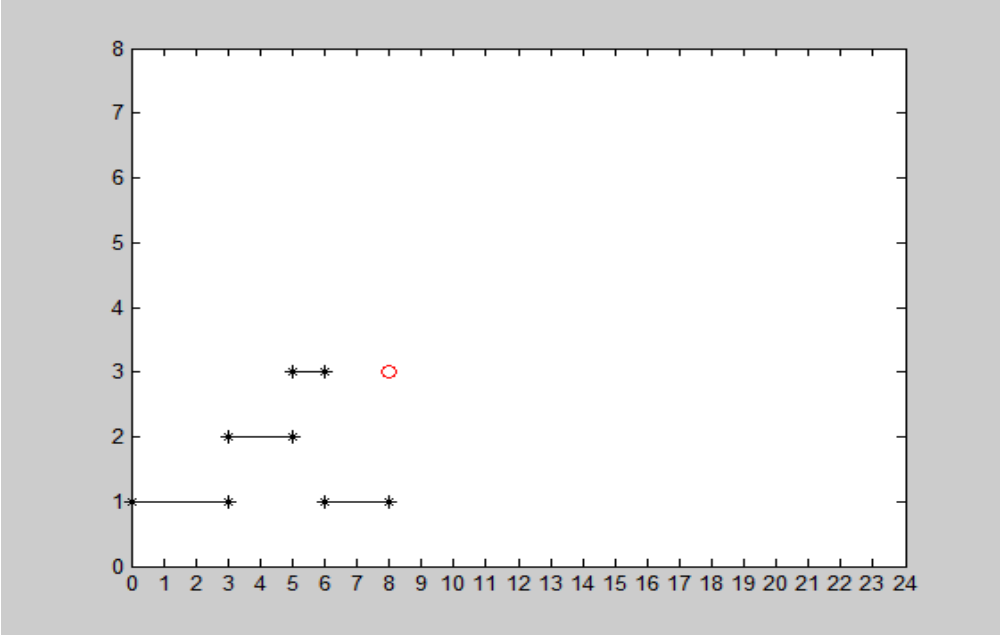

Fig 5. Schedule produced when task information in table 6 is scheduled by dual priority heuristic algorithm. Task t_3 missed its deadline and needs to update its promotion point value.

It can be seen from Fig 4 that task t_3 misses its deadline at time 8 indicated by the red circle in Fig 4. Since it is the only task that missed its deadline, it becomes the first high priority task to miss its deadline. Also since it has already executed for 1 time unit, then the value in its *arrivedTaskWCET* will be 1 which represents the amount of computation left for this task. Thus its promotion point should be set to a value of S_3=S_3 -1 = 8-1=7. The task set is then scheduled again with this new promotion point for task t_3. When this is done, the schedule obtained is shown below.

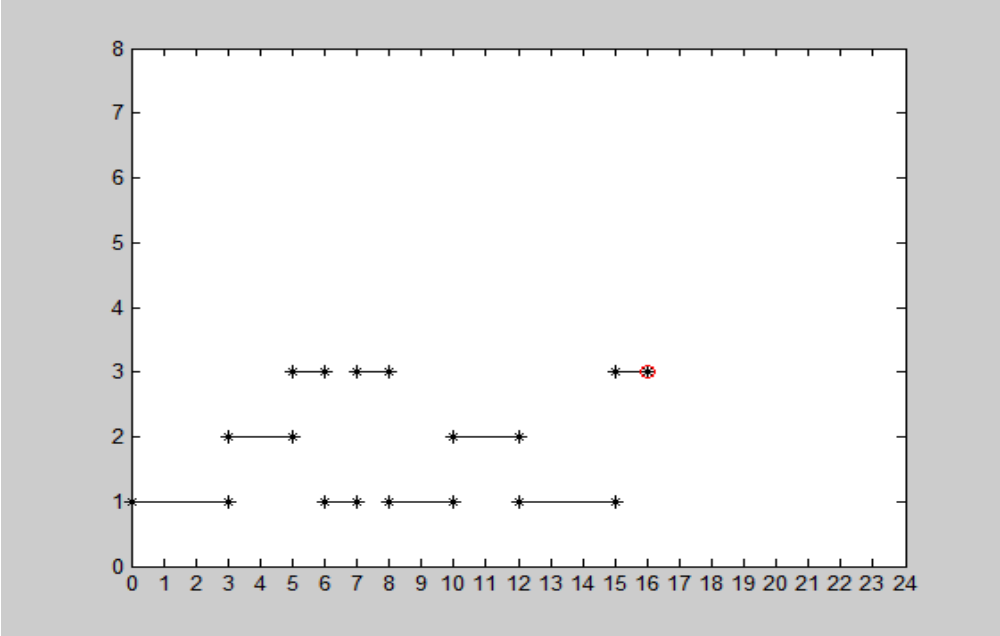

Fig 6. Schedule after task t_3 is promoted at 7. Task t_3 again misses its deadline and its promotion point value needs to be updated again.

It can be seen that, task t_3 again misses its deadline at time 16. Having been released at time 8 and executing only for 1 time unit, then the amount of computation left is 1 as such its promotion point is recalculated as of S_3=S_3 -1 = 7-1=6. This new promotion point for task t_3 is used to reschedule the task set. The new schedule is given below.

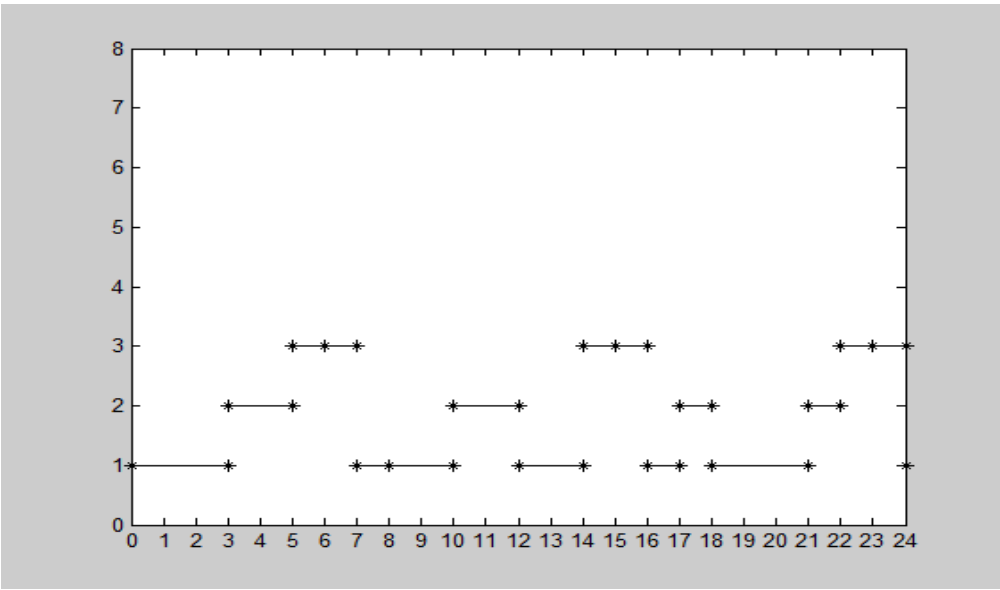

Fig 7. Schedule after task t_3 is promoted at 6. With this new promotion point, a feasible schedule is found by the dual priority heuristic algorithm.

As shown in Fig 6, the task set is now schedulable, as such the complete task information for the task set is shown in table 7

T_i,	6	8	8
C_i	3	2	2
P_i^1	4	5	6
P_i^2	1	2	3
S_i	6	8	6

Table 7. Complete task information for task set in table 6 after finding promotion points.

Example 2

T_i,	6	6	8	8	8
C_i	2	1	2	1	1
P_i^1	6	7	8	9	10
P_i^2	1	2	3	4	5
S_i	6	6	8	8	8

Table 8. Initial task set information for five tasks before the promotion points are found. The promotion point of all the tasks is set to the values of their respective periods.

When the above task set is scheduled with the dual priority heuristic algorithm, the schedule obtained is shown below,

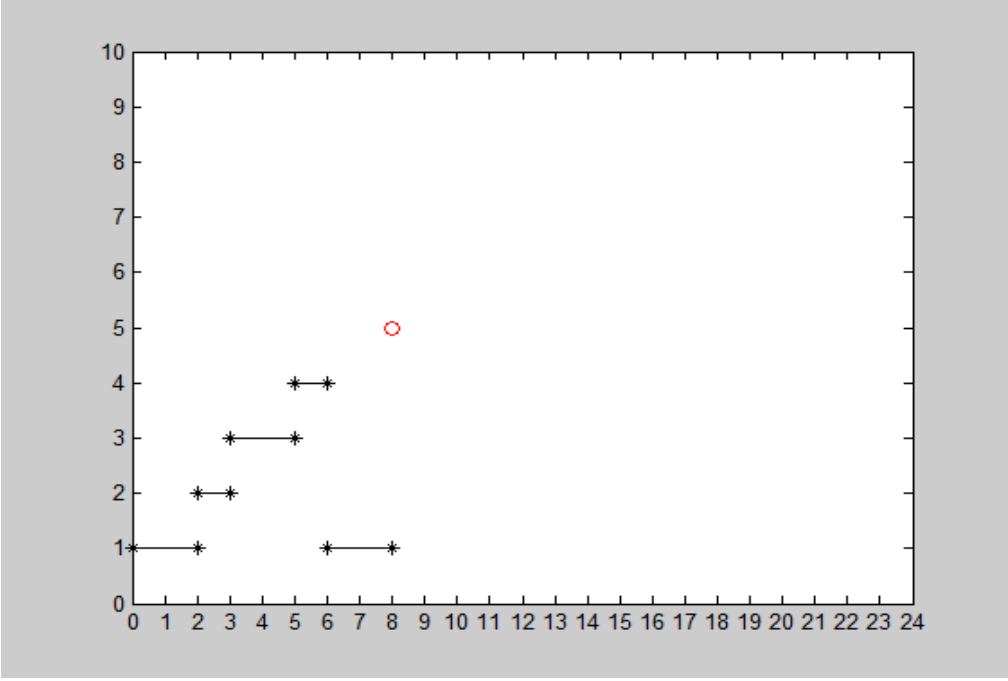

Fig 8. Schedule after task in table 8 is scheduled by dual priority algorithm. Task t_5 misses its deadline indicated by the red circle and its promotion points needs to be updated.

From Fig 7, task t_5 misses its deadline at time 8 and as such becomes the first high priority task to miss its deadline. Since it did not find any space to execute, then the amount of computation left for this task is 1, as such its promotion point is calculated as 8-1= 7. The new task set is then schedule again to produce the schedule in Fig 8.

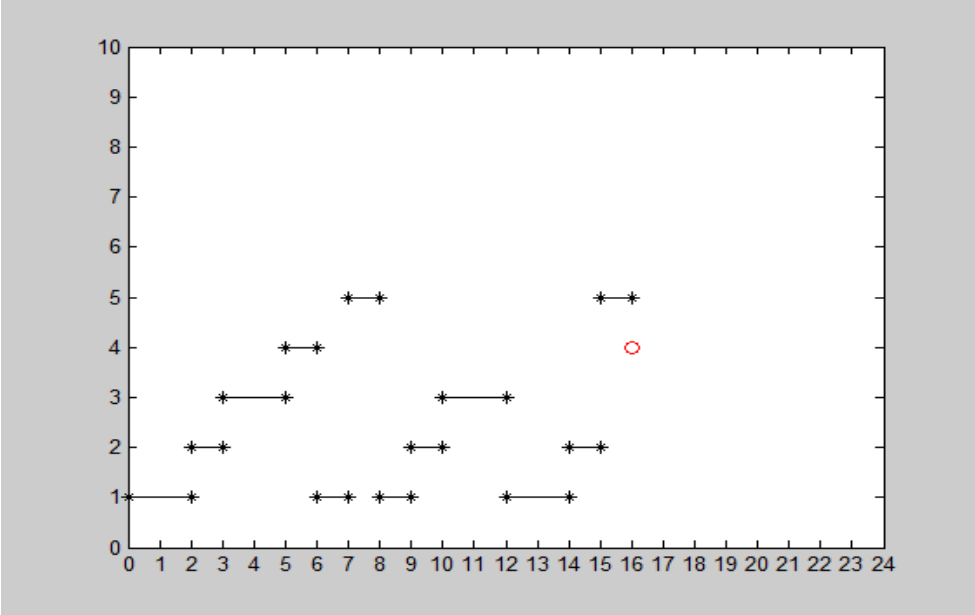

Fig 9. Schedule after task t_5 is promoted at 7. But task t_4 misses its deadline and its promotion point must be updated.

From the above schedule, task t_4 misses its deadline at time 16 and the amount of computation left is 1, as such its promotion point is calculated as 8-1= 7. The new task set is then rescheduled to produce the schedule in Fig 9.

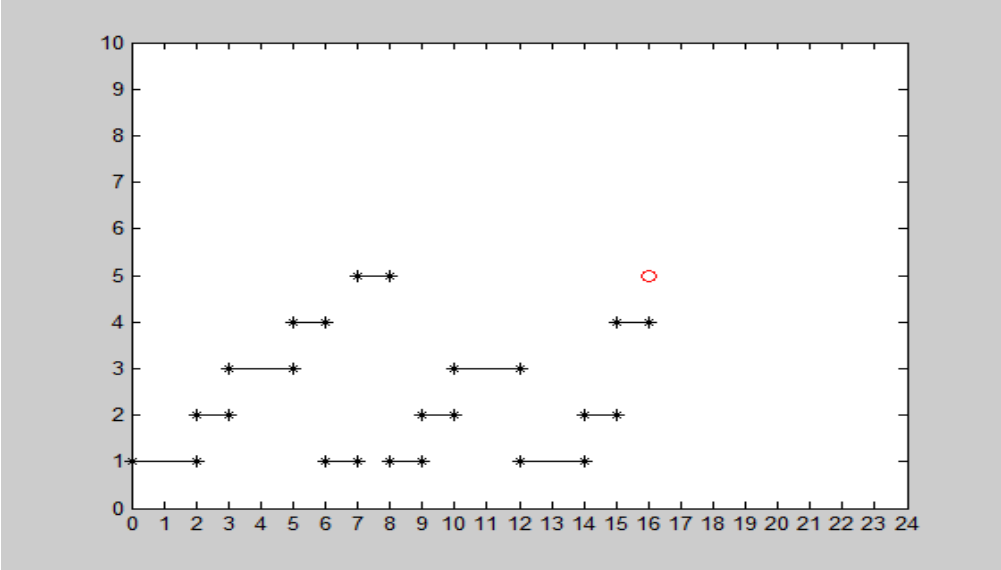

Fig 10. Schedule after both t_5 and t_4 gets promoted at 7. Now task t_5 misses its deadline and its promotion point must be updated again.

From Fig 9, task t_5 again misses its deadline at 16 by 1 and as such its promotion point is further reduced to 6. The new task set is rescheduled again to obtain the schedule in Fig 10.

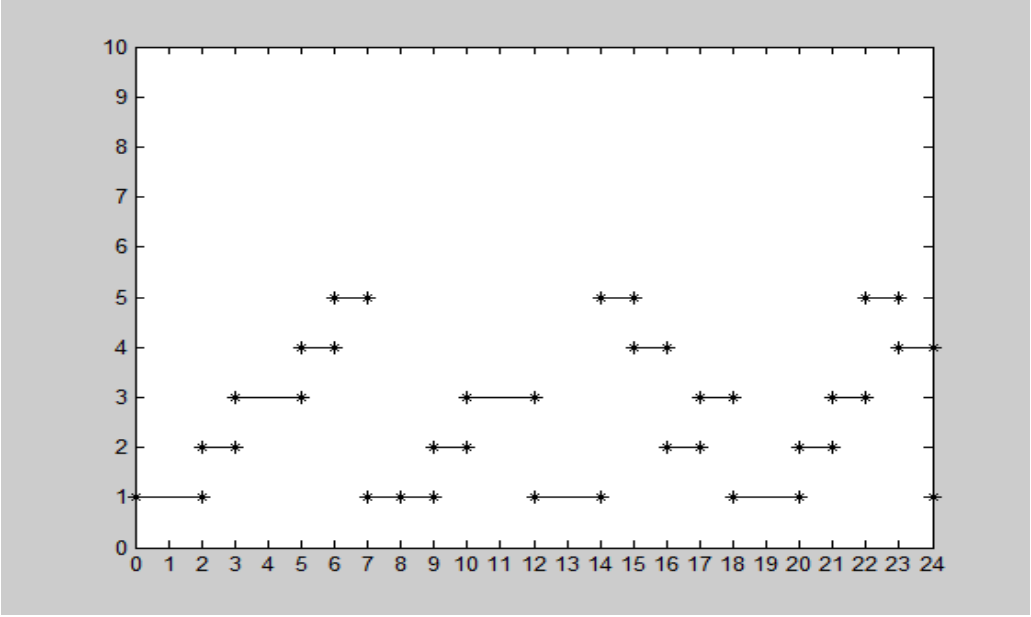

Fig 11. Schedule after t_5 is promoted at 6 and t_4 is promoted at 7. Now all task meets their deadlines and the algorithm terminates.

From fig 10, it can be seen that no task misses its deadline and the task set is schedulable. The complete task information is presented in table 9.

T_i,	6	6	8	8	8
C_i	2	1	2	1	1
P_i^1	6	7	8	9	10
P_i^2	1	2	3	4	5
S_i	6	6	8	7	6

Table 9. Complete task information for task set in table 8 after finding promotion points.

The above algorithm can be summarized into the steps below.

1. *Assign the tasks their lower and upper band priorities in the interval* $n + 1 \leq P_i^1 \leq 2n$ *and* $1 \leq P_i^2 \leq n$ *respectively.*
2. *Set the* S_i *of each task to the value of its period* T_i.
3. *While (task set is not schedulable by DP algorithm or no fail signal is issued by dual priority heuristic algorithm)*
    a. *Determine the highest priority task that misses its deadline and the amount of computation left (value of* C_x *(execution time) by which it misses its deadline)*
    b. *Calculate the new promotion point for this task as* $S_i = S_i - C_x$
    c. *If* $S_i < 0$ *, issue a fail signal.*
4. *Terminate the algorithm.*

Fig 12. The dual priority heuristic algorithm

4.4 Comparing the heuristic developed to algorithm by Burns and Wellings:

In [8] Burns posited that there may be range of promotion points that may all lead to a schedulable system. This has been found to be true as the promotion points obtained by the heuristic algorithm developed in this thesis differs in some situations completely from the ones obtained by the algorithm developed by Burns and Wellings. The principle of reducing the promotion points when a task misses its deadline is the same for both algorithms but the main reason for the difference may be due to the difference in priority assignment in both lower and upper bands. As discussed in *section 3.3.1*, the dual priority heuristic algorithm developed is superior to the algorithm developed by Burns and Welligns in terms of the amount of computation required to determine the promotion points.

Below are some examples provided in [6] and [8] with their promotion points and then the same examples with the promotion points obtained by the heuristic developed.

Example 3 from [6]

T_i,	12	16	20	20
C_i	3	4	4	6
P_i^1	4	5	6	7
P_i^2	-	-	2	1
S_i	-	-	13	14

Table 11. Complete task set information using burns and wellings dual priority algorithm

When the task set in table 11 is schedule with the heuristic algorithm developed in this thesis, the promotion points obtained is summarized in the table below

T_i	12	16	20	20
C_i	3	4	4	6
P_1	5	6	7	8
P_2	1	2	3	4
S_i	12	16	19	13

Table 12. Complete task information using the dual priority heuristic algorithm developed in this thesis.

Example 4 from [8]

T_i	28	100	160
C_i	21	15	16
P_1	4	5	6
P_2	1	2	3
S_i	9	84	130

Table 17. Complete task set information using burns and wellings dual priority algorithm

When the task set in table 17 is schedule with the heuristic algorithm developed in this thesis, the promotion points obtained is summarized in the table below

T_i	28	100	160
C_i	21	15	16
P_1	4	5	6
P_2	1	2	3
S_i	7	82	130

Table 18. Complete task information using the dual priority heuristic algorithm developed in this thesis.

4.5 Proof that heuristic achieves 100% utilization:

A proof that the algorithm can achieve 100% utilization is presented under certain assumptions. The proof follows the proof presented by Burns and Wellings in [6]. The assumptions made are presented followed by the proof.

4.5.1 Assumptions:

To make a proof, an assumption is made that every task set that is not schedulable by the rate monotonic scheduling algorithm can be scheduled by the dual priority scheduling algorithm, implying that there exist promotion points for all such tasks. With this assumption, then the possible proof sketch is shown below.

4.5.2 Proof:

Assume a task t_i with deadline at D_i and a promotion point at S_i

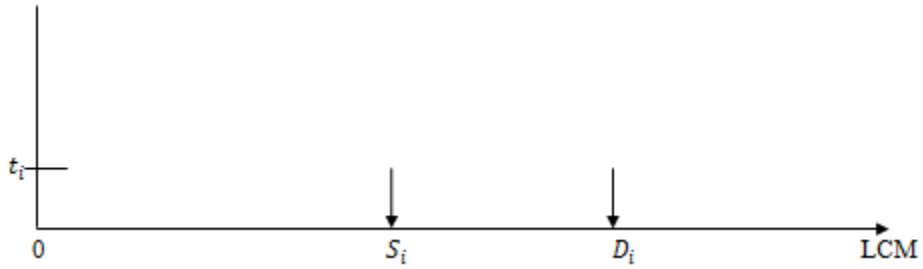

Fig 13. Time line with promotion point and deadline for task t_i

Then all other tasks can be classified as

$\alpha =$ tasks with lower P^1 and P^2 priorities than t_i

β = tasks with higher P^1 and P^2 priorities than t_i

Then the following can be deduced,

1. In the worst case, all α tasks with deadlines at or before D_i should have finished by S_i , meaning they have their promotion points before S_i as shown in Fig 15.

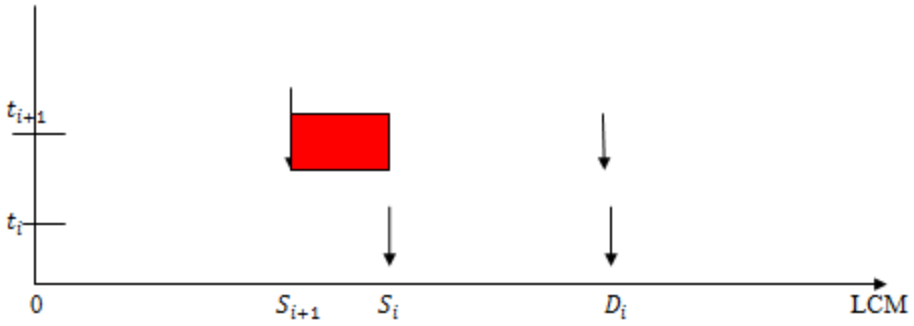

Fig 14. Shows where lower priority tasks with deadline before D_i can execute.

This is due to the fact that between $[S_i, D_i]$,task t_{i+1} has lower priority in both bands and cannot preempt task t_i.

2. Within the interval $[D_i - S_i\ , D_i]$, only tasks which belong to β which have their promotion point within this interval due to the fact that their deadlines is at or beyond D_i and task t_i itself can execute . Let the earliest time of such promotion point of β be R, then it means that within the interval [$R - S_i$, R] only task t_i can execute and must finish its execution by R as shown in Fig 15

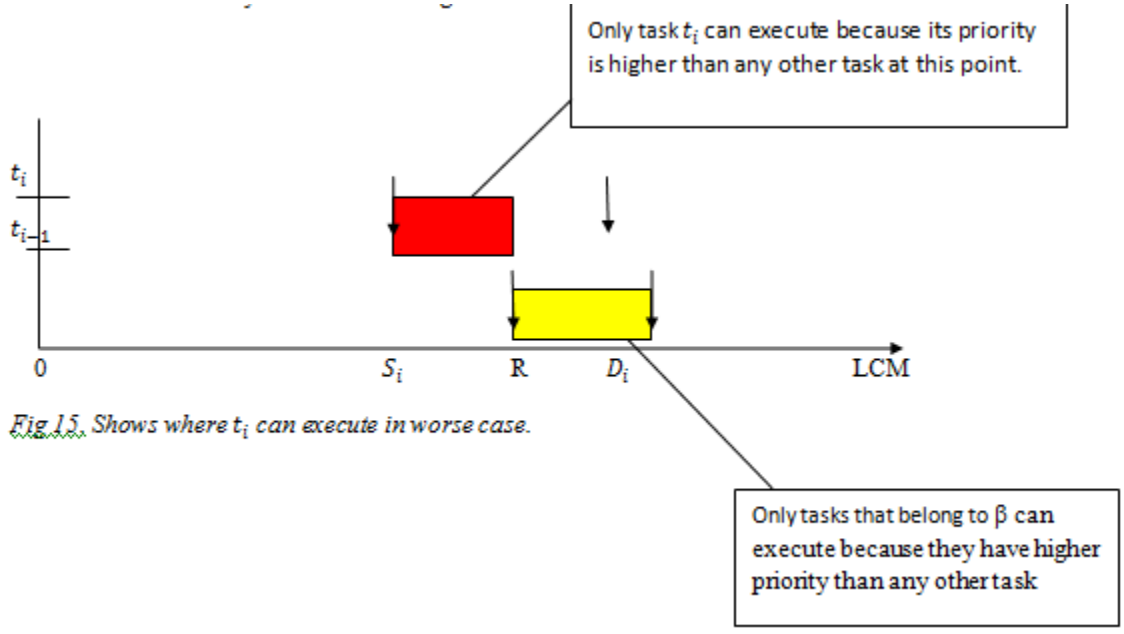

Fig 15. Shows where t_i can execute in worse case.

It can be seen that all task with deadlines before D_i are met, task t_i meets its deadline at R and any computation after R is needed by tasks which have their deadline at D_i or beyond.
Thus within the interval [$R - S_i$, R], a task whose deadline is nearest execute which is similar to the EDF algorithm. Since the EDF has 100% utilization, then t_i must complete its computation with this utilization bound. What is true for t_i is true for all other tasks.

4.6 Optimization techniques:

Normally when given a task set, a simulation to generate a schedule requires that one considers every time unit(decision point) up to the LCM of the task periods (*exhaustive simulation*). This is computationally very expensive. With the dual priority heuristic algorithm developed, the only times at which the *arrivedTask* data structure changes state was at times known as event times as already discussed in session 4.2.1 and 4.2.2 . Thus, instead of taking scheduling decisions at each time point, scheduling decisions were taken only at these event times. With this, the computational time was observed to have drastically reduced compared to the computational time when each time unit is simulated up to LCM. An experiment consisting of 1000 tasks each for task set of 3 to 9 number of tasks was performed to statistically compare the percentage reduction in decision points of the dual priority heuristic algorithm to simulation of each time point up to LCM. A box plot (*a plot that graphically depicts groups of statistical data and shows the 25th, 50th and 75th percentiles of data set as the lower part of a box, the red band in the middle of the box and the upper part of the box respectively*) of the results is shown below.

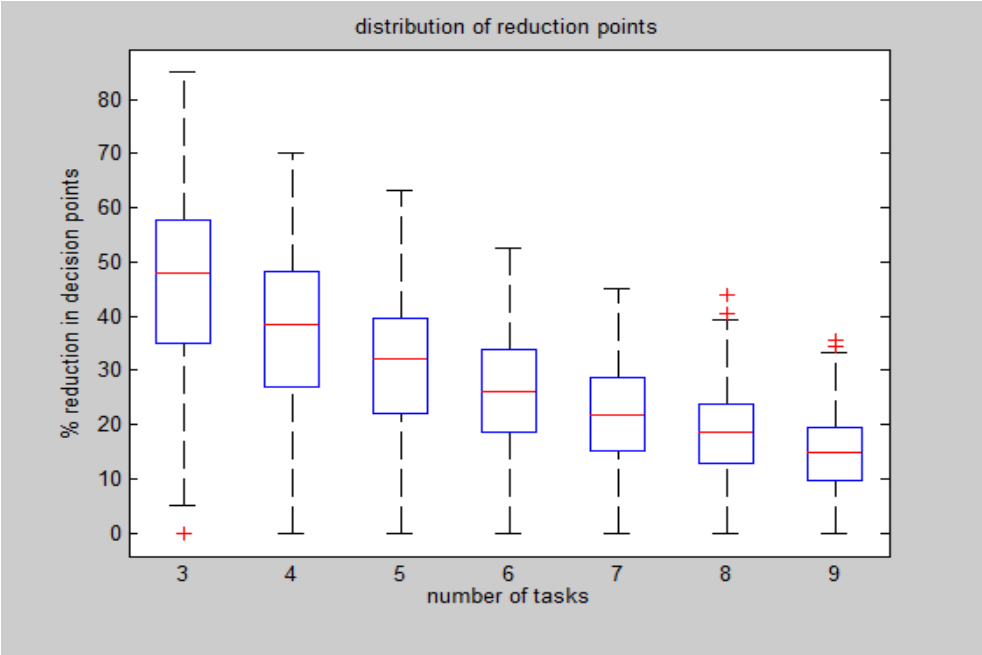

Fig 16. Box plot showing percentage reduction in decision points of the dual priority heuristic algorithm compared to simulating each time unit up to LCM for various task sets. The x-axis represents the number of tasks, which ranges from 3 to 9. The y-axis represents the percentage reduction in decision points (number of points at which scheduling decisions were taken) which ranges from 0 to 100%. For task set consisting of only 3 tasks, the dual priority heuristic algorithm scheduled 25% of the 1000 tasks with less than 35% of the number of decision points needed by an exhaustive simulation. It scheduled 50% with less than about 47% of the points needed by an exhaustive simulation and 75% with less than 58% of the points needed by an exhaustive simulation. Most of the tasks had a reduction ranging 5% to 85% which is indicated by the black dotted lines. However, for one task indicated by the red plus sign, the dual priority heuristic algorithm used the same number of decision points as the exhaustive simulation; as such there was no reduction. For task set consisting of 5 tasks, the reduction range was form 0 to about 65% with 25% of the task having less than 22% reduction, 50% having less than 33% reduction and 75% having about less than 39% reduction. For task set consisting of 9 tasks, most of the task had a reduction range of 0 to about 35%, with 25% of the tasks having less than 10% reduction , 50% of the tasks having less than 15% reduction and 75% having less than 20%. Two task out of the 1000 tasks had reduction of about 39% and 40% indicated by the two red plus signs.

As the number of tasks increases, the percentage reduction decreases, and rightly so, because for the same period range eg 1-20, more task are released, finished and are promoted for task set with higher number of tasks.

4.6 Performance of the heuristic:

To generate the promotion points, an exhaustive search approach will consider T_1*T_2* T_3*…* T_n number of points. This is a huge value especially for task set which have very large integers as their periods. The heuristic algorithm developed in this thesis however considers only $(T_1) + (T_2) + \ldots\ldots(T_n)$ in the worse case which present an enormous reduction in the number of points considered. An experiment was conducted by generating 100 tasks each for task set consisting of 3 to 9 number of tasks. The number of points that an exhaustive search would have considered was determined as well as the amount of points that the heuristic algorithm considered. The percentage reduction in the number of points by the heuristic algorithm compared to the exhaustive search was also determined for each task set and plotted as a box plot below.

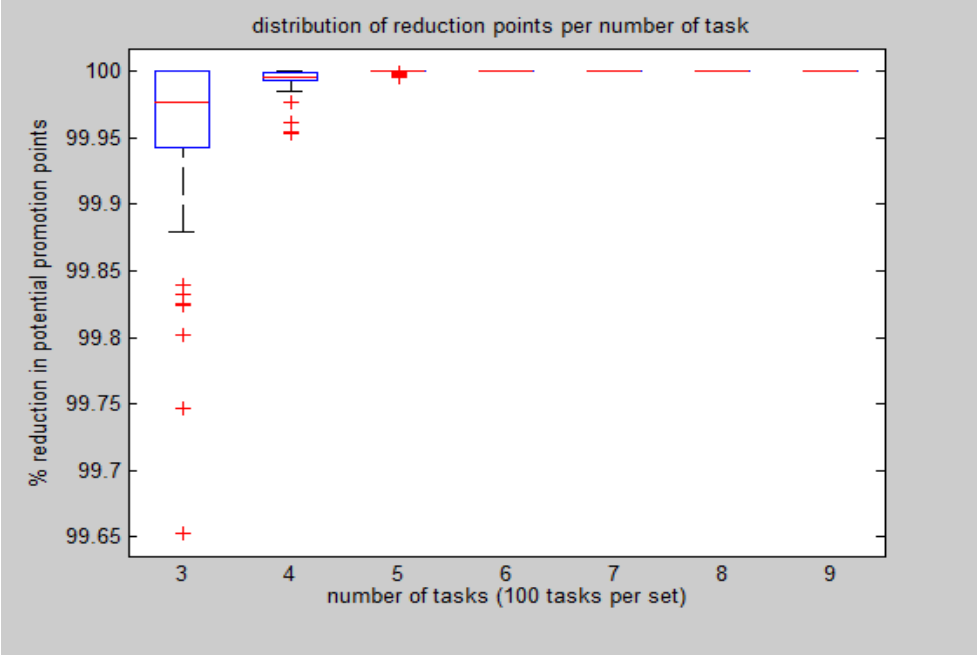

Fig 17. Box plot showing percentage reduction in promotion points compared with exhaustive search approach. The number of tasks is shown on the x-axis and the percentage reduction is on the y-axis. For task set consisting of 3 tasks, most of the 100 tasks had a percentage reduction of between 99.95 to a value very close to 100. However, few of the tasks had other percentage reduction values indicated by the red plus sign. For task set consisting of 4 tasks, the least percentage reduction was about 99.94 indicated by the red plus sign with most of the task achieving almost 100% reduction. For task set of 5 to 9 tasks, the percentage reduction was so huge that the almost all the task achieved a near 100% reduction, this is shown by a single straight red line for these task sets.

Chapter 5

In this chapter, the simulation results as well as the task generation policies are discussed.

5.0Simulations:

The proof shown in session 4.5.2 is based on the assumption that there exist promotion points for all task sets which are not schedulable by RM. In the absence of a formal proof for this assumption, simulation was resorted to where large number of task sets were generated and tested with the algorithm with the hope of finding a single task set (counter example) which was not schedulable by RM and for which promotion points could not also be found. With such a task set, the assumption made in session 4.5.1 and the resulting proof of achieving 100% utilization are defeated.

Since the task generation policy is very important and can greatly affect the experimental result as pointed out by Bini et al in [11], the generation policies for the task sets were chosen carefully. Below is a brief discussion of the various generation techniques.

5.1 Utilization generation policy:

To generate uniformly distributed task utilizations in the range [0, 1], the $UUniFast$ algorithm [11] shown in fig 18 in matlab code was employed. This algorithm of the form $UUniFast(n, U)$ generated n uniformly distributed utilizations values whose sum is equal to U. In [11], Bini and Buttazzo showed that the $UUniFast$ algorithm is the most efficient algorithm with an O (n) complexity for generating uniform distributed task utilizations.

```
function vectU = UUniFast(n, U)
  sumU = U;
  for i=1:n−1,
    nextSumU = sumU.∗ randˆ(1/(n−i));
    vectU(i) = sumU − nextSumU;
    sumU = nextSumU;
  end
vectU(n) = USum;
```

Fig 18. The UUnifast algorithm in Matlab.

5.2 Period generation policy:

The randi function of Matlab [12] was used to generate n periods in the range [1-max]. The randi function is a discrete uniform distribution that puts equal weight on the integers from 1 to max. The function used has the form randi (max, 1, n).

5.3 Worse case execution time generation policy (WCET):

Having generated n utilizations and periods, the worse case execution times C_i were calculated by multiplying the periods with the utilization and the result rounded to the nearest integer. The new utilization value was calculated as $U = \sum \frac{C_i}{t_i}$ and if U >1, then that task set was ignored.

5.4 Experiments conducted:

Having generated the tasks with the generation policies above, various experiments were conducted on them with the dual priority heuristic algorithm to determine if a counter example could be found. In all the cases, task set with the number of tasks n ranging from 3 to 9 and period ranging from 1 to 50 was used. The above range for the number of tasks n and period were chosen because it was found that for task set with larger n and higher periods, it was computationally too expensive (it took too much time) for the laboratory computers (core 2 duo , 2.4 GHz) to perform the simulation. For each task set, 1000 tasks were generated and tested with the algorithm. Below are detailed descriptions of the various experiments conducted and the results obtained.

5.4.1 Effect of number of task:

From session *2.1.6.2* Eq 1, as the number of tasks increases, the task set utilization that can be guaranteed by RM reduces, converging to 69.3%. With the dual priority heuristic algorithm aiming to achieve a task set utilization of 100% implies that increasing the number of tasks while keeping the task set utilization set at 100% should not fail to schedule any task set. With this, the tasks that were generated were tested with RM and the dual priority heuristic algorithm and the success rate (percentage of the task set that were successfully schedulable by the algorithm) was plotted against the number of tasks n as shown in the graph below.

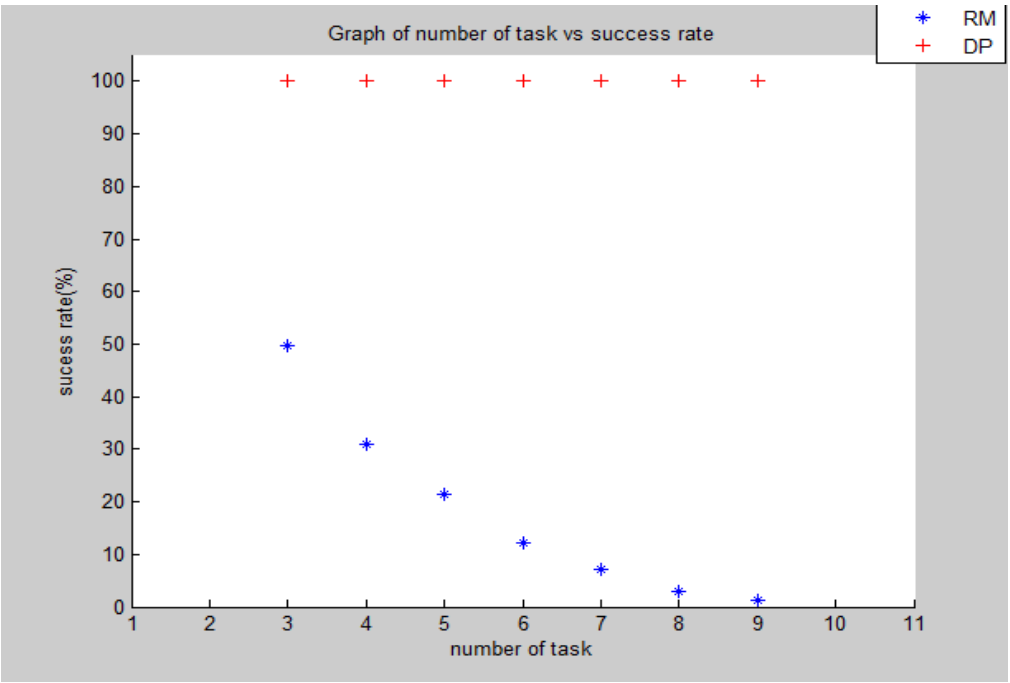

Fig 19 . Graph showing the effect of number of tasks on the dual priority algorithm. The x-axis represents the number of tasks per set n, which ranges from 3 to 9 and the y-axis represent the success rate (percentage of the number of task sets that are successfully scheduled). The blue markers represent the success rate obtained for RM for the various numbers of tasks and the red marker represent the results obtained for the dual priority heuristic algorithm.

It can be seen that more tasks miss their deadlines for the rate monotonic when the number of task per set increases, but all tasks meet their deadlines for the dual priority heuristic algorithm, as such its success rate remains at 100%. A simple reason being that, as the number of tasks per set increases, the deviation of the task utilization from the Liu Layland bound in session *2.1.6.2* increases and as such the probability that a task set misses it deadline becomes high. This is however just a statistical result.

5.4.2 Effect of increasing Utilization:

The experiment conducted above varied the number of task while keeping the task set utilization constant at 100%. However , it might be the case that there exist a counter example with task set utilization less than 100% but still not schedulable under RM. To find such a task set, the task set utilization was varied from 69% to 100% at steps of 1% for the task sets generated in session 5.4. Since RM guarantees a task set utilization up to 69.3%, then the probability of finding a counter example in the range chosen if there exist one is higher. The task sets that were generated were tested with RM and the dual priority heuristic algorithm and the success rate (percentage of the task set that were successfully schedulable by the algorithm) was plotted against the task set utilization as shown in the graph below. The results for task set with 3, 6 and 9 number of tasks

were shown in the graph just for clarity (showing all the results for all the task sets made the graph very unreadable).

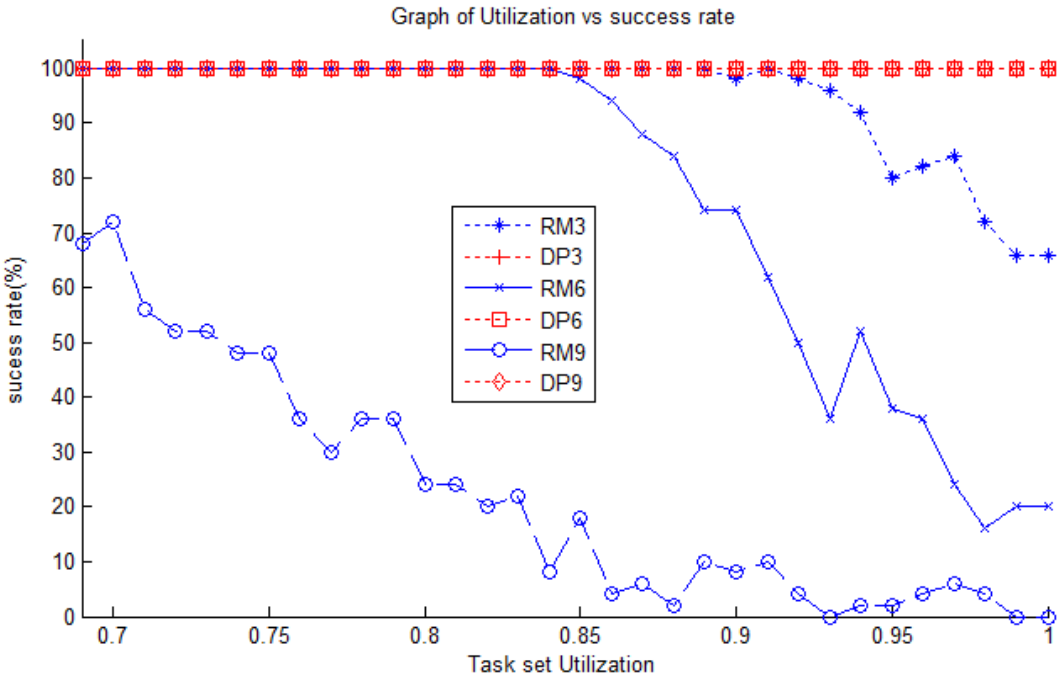

Fig 20. Graph showing the effect of increasing task utilization on the dual priority algorithm. The x-axis represent the task set utilization ranging from 0.69(69%) to 1 (100%). The y-axis represents the success rate (percentage of the number of task sets that are successfully scheduled). The blue dotted lines represent the result for RM for task set with number of tasks equal to 3. The blue straight lines represent the result for RM for task set with number of tasks equal to 6 and the blue lines represent the result for RM for task set with number of tasks equal to 9. The red dotted lines represent the result of the dual priority heuristic algorithm for task sets with number of task equal to 3,6 and 9.

It can be seen that as the utilization increases with the number of task per set, more tasks missed their deadlines for the rate monotonic scheduling algorithm while for the dual priority heuristic algorithm no task missed its deadline for the same reason as increasing the number of task

5.5 Conclusion and limitations of experiments:

The aim of the experiments conducted was to find a task set (counter example) what could not be scheduled by the dual priority heuristic algorithm. If one counter example was found, it would have signified that the dual priority heuristic algorithm cannot guarantee 100% utilization. From the results presented for the various experiments, no counter example was found from the large task set generated. This is however a statistical result so cannot be used as the sole proof that the dual priority heuristic algorithm achieves 100% utilization, but it goes a long way to serves as the basis upon which further research which may lead to the formal proof can be carried out.

Chapter 6

Future work and conclusion is discussed in this chapter

6.0 Future work and Conclusion:

This thesis has focused mainly on the development of the dual priority heuristic algorithm which uses rate monotonic priority ordering in both lower and higher priority bands. It has also focused on providing a proof which shows that this algorithm achieves 100% utilization. Extensive simulations have also been performed on the developed algorithm in the hope of finding a counter example and the results obtained suggest otherwise.

What is left to do as a future work is the formal proof that the dual priority algorithm can always find promotion points when a task set is not schedulable by the rate monotonic scheduling algorithm .Burns in [8] provided a proof for the dual priority algorithm for just two tasks but trying to generalize for more tasks has proven to be difficult.

In developing the algorithm, unrealistic assumptions were made in section 3.1. Future work could relax these assumptions and more simulations performed to observe the performance of the algorithm. Also practical implementation issues could be a subject of future work.

References:

[1] Arezou Mohammadi and Selim G. Akl "Scheduling Algorithms for Real-Time Systems" Technical Report No. 2005-499, Queens University Canada, 2005.

[2] C. L. Liu and J. W. Layland, *"Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment,"* Journal of the ACM, Volume 20, Number 1, pp. 46-61, 1973

[3] A. Burns and A. Wellings "Real-time systems and programming languages" Forth edition

[4] M. Joseph, *"Real-time Systems: Specification, Verification and Analysis,"* Prentice Hall, 1996.

[5] J. Goossens and P. Richard "Overview of real-time scheduling problems" Euro Workshop on Project Management and Scheduling 2004.

[6] A. Burns and A. Wellings "Dual Priority Assignment: A Practical Method For Increasing Processor Utilization".

[7] R. Davis and A. Wellings "Dual priority scheduling "

[8] A. Burns "Dual Priority Scheduling: Is the Processor Utilization bound 100%?"

[9] R. Davis "Dual Priority Scheduling: A Means of Providing Flexibility in Hard Real-time Systems."

[10] M.G. Harbour, M.H. Klein and J.P. Lehoczky, *Fixed Priority Scheduling of Periodic Tasks with varying Execution Priority*, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, USA (1991).

[11] E. Bini and G. Buttazzo. 'Measuring the performance of schedulability tests'. *Journal of Real-Time Systems*, 30(1-2):129–154, 2005

[12] http://www.mathworks.com/help/techdoc/ref/randi.html

[13] Audsley, et al. "Applying new scheduling theory to static priority pre-emptive scheduling

[14] Katharina Wennerström "Environment for Brake by Wire System Development"

[15] Krishna C.M and Shin K.G Real-time systems . Tata McGraw-Hill, 1997.

[16] T.P. Baker and Alan Shaw " The Cyclic Executive Model and Ada"