Task Period Selection and Schedulability in Real-Time Systems *

Danbing Seto[1], John P. Lehoczky[2], and Lui Sha[1]
[1] Software Engineering Institute, [2] Department of Statistics
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In many real-time applications, especially those involving computer-controlled systems, the application tasks often have a maximal acceptable latency, and small latency is preferred to large. The interaction between choosing task periods to meet the individual latency requirements and scheduling the resulting task set was investigated in [4] using dynamic priority scheduling methods. In this paper, we present algorithms based on static priority scheduling methods to determine optimal periods for each task in the task set. The solution to the period selection problem optimizes a system-wide performance measure subject to meeting the maximal acceptable latency requirements of each task. This paper also contributes to a new aspect of rate monotonic scheduling, the optimal design of task periods in connection with application related timing specifications and task set schedulability.

1 Introduction

Many real-time computer-controlled applications are developed in two isolated steps: (1) the design of application specific computation modules and (2) the implementation of these modules. In the design stage, the computation modules are designed to fulfill the control objectives of the application, and each module is assigned a period which will guarantee that a certain performance level will be achieved. In the implementation stage, the computation modules are treated as real-time periodic tasks and scheduled using the periods that were determined in the design stage. Task set schedulability is completely determined by the given periods, task execution times, and the scheduling algorithm. These two stages are isolated in the sense that the choice of task period in the design stage is made with no concern for task schedulability requirements in the implementation stage. It is possible that the resulting set of computation modules may not be schedulable. Even they are schedulable, the performance of the overall system may not be optimal in terms of making full use of the underline computing resource.

A better overall system design can be achieved by taking into account the task schedulability at design time. In many real-time applications, there often exists an acceptable range of periods for each computing task, with the largest period still meeting the latency requirements of the task. For example, a real-time control system may have multiple control processes whose sampling frequencies can be chosen from a range of frequencies such that the control objective of each control process is accomplished. Moreover, different choices of frequency will generally result in different task performances as the task executes. By *performance*, we mean a measure of how well a task achieves its objective, or an aggregate measure of how well the overall system meets its overall objective. In practice, the period for each computing task is usually chosen to satisfy certain performance requirements. In choosing the periods, two principles will generally apply: (1) the period of each task will be bounded above by some value corresponding to the maximal permissible latency requirement associated with the task, and (2) the performance of a task is often inversely related to the task's period, i.e., the shorter the period, the better the performance. For the class of tasks for which (1) and (2) apply, it is obvious that task periods should be chosen to be as small as possible, to satisfy the upper bound conditions and to satisfy schedulability conditions. This leads to an optimization problem: for a set of real-time periodic tasks to determine periods for those tasks which will maximize an overall system performance measure subject to satisfying the maximal task periods and task set schedulability requirements. The periods which optimize the performance are called *optimal periods*.

The integration of task schedulability and system performance has been investigated by Locke [2] using best-effort scheduling methods, and by Gerber, Hong and Saksena [1] who focused on distributed systems. An optimal solution to the problem of allocating and scheduling communicating periodic tasks in distributed real-time systems was developed by Peng, Shin and Abdelzaher [3], where the max-

*This research was supported in part by the Office of Naval Research under contracts N00014-92-J-1524 and F19628-95-C-0003, and by the Software Engineering Institute of Carnegie Mellon University.

imum normalized task response time, i.e., the system hazard, is minimized subject to the intercommunication constraints among the tasks to be allocated and scheduled. The present paper is an extension of the paper by Seto, Lehoczky, Sha and Shin [4] who developed an algorithm to determine optimal task periods assuming tasks were scheduled by the earliest deadline first (edf) algorithm. The use of the edf algorithm to schedule periodic tasks with deadlines equal to task periods ensures that 100% schedulable processor utilization can be achieved. Unfortunately, many real-time computer operating systems only support static priority scheduling. Consequently, it is important to obtain a solution to this task period design problem for the optimal static scheduling algorithm, the rate monotonic algorithm (RMA). When RMA is used, task set feasibility becomes an issue because the RMA algorithm only supports full schedulability when the task periods are harmonic. Furthermore, task periods that result in high schedulable utilization may not necessarily also maximize the system performance index. While the optimal RMA periods can be found by inspection in very simple cases, a general algorithm is needed to address this problem.

In this paper, we present a search algorithm for the optimal periods when the tasks are scheduled with RMA. In particular, we first develop an algorithm to search all the possible periods for which the set of tasks are schedulable with RMA. We will show that these periods, although there are infinite many of them, can be specified by a finite set of period ranges. Then, from these period ranges, we identify the set of optimal periods. It is important to note that finding the set of all possible periods for which a task set is RMA schedulable is of independent interest even without any consideration of selecting the optimal periods from that set. Indeed, it could be a necessary step in the development of an integrated design when the system performance characteristics are not known. In addition, it provides the design engineer with the feasible periods that can be used for a variety of purposes, for example, performance trade-off, multiple rate selection, etc. The algorithm for searching the ranges of possible periods for RMA schedulability is developed by formulating the RMA schedulability testing algorithm reported in [5] as an integer programming problem, from which a finite collection of sets of integers can be obtained. These integers represent the number of times each task executes before the task with the lowest priority completes. Based on these sets of integers, the ranges of possible periods can be derived using a *branch and bound* approach . For ease of exposition, we define the following concepts and notation. For a set of tasks $\{\tau_1, .., \tau_p\}$ having upper bounds on their periods given by $[T_{m1}, .., T_{mp}]$, *a set of feasible periods* refers to an array of periods $[T_1, .., T_p]$ which satisfies the condition $T_1 \leq T_{m1}, .., T_p \leq T_{mp}$ and results in all tasks being schedulable using RMA. *A set of ranges of feasible periods* S_p is a set of arrays $[T_1, .., T_p]$ such that all the tasks can be scheduled by RMA with periods chosen from S_p, and it can be expressed as:

$$S_p = \{[T_1, .., T_p] : \ L_j \leq T_j \leq U_j, \ j = 1, .., p\}$$

with L_j and $U_j \ \leq \ T_{mj}$ the lower and the upper limits of the period ranges. In the rest of the paper, all aspects of schedulability are with respect to RMA scheduling.

The paper is organized as follows. In section 2, we present an algorithm to determine the sets of ranges of feasible periods for a given set of tasks. In particular, we consider two cases: the task set is specified with a required priority order, and the task set is specified without any specific required priority order. In section 3, we choose the optimal periods from the sets of ranges of feasible periods such that a performance measure is optimized. Conclusions are given in section 4.

2 Task Periods for RMA Schedulability

In this section, we study the problem of finding all the periods $[T_1, .., T_p]$ for a set of tasks $\{\tau_1, .., \tau_p\}$ with upper bounds on their periods $[T_{m1}, .., T_{mp}]$ such that they can be scheduled with RMA. To achieve this, we need to first determine the priority order for these tasks. According to RMA, a task with a relatively short period should be assigned a relatively high priority. Since the task periods are not given, in principle, all $p!$ permutations of the priority orders should be considered. In the algorithm proposed next, we use a branch and bound approach to search the feasible periods for the given task set tasks. As a result, the number of the possible permutations can be significantly less than $p!$. We first describe an algorithm to determine the periods for a given priority order, and then investigate the periods for all possible priority orders.

2.1 Task Periods for RMA Schedulability with A Priority Order

Problem Statement: Given a set of periodic tasks $\{\tau_1, .., \tau_p\}$ with execution times $C_1, .., C_p$ and period upper bounds $T_{m1}, .., T_{mp}$, suppose the set of tasks is schedulable with the maximum periods $T_{m1}, .., T_{mp}$. Find all the possible periods $T_1, .., T_p$, satisfying $T_1 \leq \cdot\cdot \leq T_p$, and $T_1 \leq T_{m1}, .., T_p \leq T_{mp}$, such that all the tasks are schedulable.

We start with the schedulability conditions for a set of tasks with fixed periods $T_1 \leq \cdot\cdot \leq T_p$, and use a schedulability testing algorithm reported in [5]. According to RMA, for any task $\tau_k, \ 1 \leq k \leq p$, the tasks $\tau_1, .., \tau_{k-1}$ have

higher priorities than τ_k, and at time t, the total cumulative demand on CPU time by these k tasks is given by

$$w_k(t) = \sum_{j=1}^{k} C_j \left\lceil \frac{t}{T_j} \right\rceil$$

The conditions for a set of tasks to be schedulable are summarized in the following lemma.

Lemma 2.1 A set of tasks $\{\tau_1, .., \tau_p\}$ with execution times $C_1, .., C_p$ and periods $T_1 \leq \cdot\cdot \leq T_p$ is schedulable if and only if for each task $k,\ k = 1, .., p$, the iterative sequence

$$t_0^k = \sum_{j=1}^{k} C_j, \quad t_1^k = w_k(t_0^k), \quad t_2^k = w_k(t_1^k), \ \cdot\cdot$$

converges to a t_c^k, i.e.,$t_c^k = w_k(t_c^k)$ and $t_c^k \leq T_k$.

Note that for each i, $t_i^k = \sum_{j=1}^{k} n_j C_j$ where $n_1, .., n_k$ are integers. Thus the schedulability conditions can be reformulated as follows.

Proposition 2.1 A set of tasks $\{\tau_1, .., \tau_p\}$ with execution times $C_1, .., C_p$ and periods $T_1 \leq \cdot \leq T_p$ is schedulable if and only if for each task $k,\ k = 1, .., p$, there exists a set of integers $[n_1, .., n_k]$ with $n_k = 1$ such that

$$\sum_{j=1}^{k} n_j C_j \leq n_i T_i,\ i = 1, .., k \qquad (1)$$

Proof: For sufficiency, we show that the set of tasks is schedulable if conditions in (1) are satisfied. For each task τ_k, it is clear that the integer $n_j,\ j = 1, .., k-1$, in (1) is the number of times that task τ_k is preempted by task τ_j. Then conditions in (1) imply that task τ_k will complete at time $t_c^k = \sum_{j=1}^{k} n_j C_j$ with n_j preemptions by tasks $\tau_j,\ j = 1, .., k-1$, and $t_c^k \leq T_k$. Therefore, task τ_k is schedulable. As this holds for all $k = 1, .., p$, the task set is schedulable.

For necessity, we prove that the schedulability of the task set implies condition (1). For each task τ_k in the task set, the schedulability of τ_k indicates that the completion time of τ_k is given by $t_c^k = \sum_{j=1}^{k} n_j C_j$ with $n_k = 1$, and $t_c^k \leq T_k$. Thus, before its completion, task τ_k will be preempted n_j times by the higher priority tasks $\tau_j,\ j = 1, .., k-1$. Since each task $\tau_i,\ i = 1, .., k$, will execute n_i times before τ_k completes, we conclude that $t_c^k \leq n_i T_i,\ i = 1, .., k$, which are exactly the equations given in (1).

Remark 2.1 Proposition 2.1 is equivalent to Lemma 2.1 in terms of task schedulability. It represents the schedulability conditions in terms of the existence of a set of integers. This formulation provides the basis for an integer programming algorithm for searching possible periods which guarantee task schedulability.

Remark 2.2 The set of integers which satisfies the conditions in (1) for one task may not be unique, i.e., for a given set of periods $[T_1, .., T_k]$ with $T_1 \leq \cdot\cdot \leq T_k$, there could be multiple sets of integers satisfying the conditions in (1). Among those sets of integers, only the one which gives the smallest completion time for task k, i.e., $t_c^k = \sum_{j=1}^{k} n_j C_j$, has a real meaning in terms of task execution. As we described in the proof of Proposition 2.1, the integers in that set indicate how many times tasks $\tau_1, .., \tau_k$ will execute before task τ_k completes.

Example 2.1 To illustrate Proposition 2.1, we determine the schedulability of a set of tasks $\{\tau_1,\ \tau_2,\ \tau_3\}$ with the following two cases

C_i	1	5	7
T_i	4	10	29

and

C_i	1	5	7
T_i	4	12	29

where $C_i,\ T_i,\ i = 1, 2, 3$, are the tasks' execution times and periods, respectively. For τ_3 to be schedulable, we check the existence of a set of integers $[n_1,\ n_2,\ 1]$ such that conditions in (1) are satisfied. Let $t_c = n_1 C_1 + n_2 C_2 + C_3$ and $B = \min\{n_1 T_1,\ n_2 T_2, T_3\}$. Then the conditions in (1) can be written as $t_c \leq B$. In the first case, we obtain the following results:

n_1	n_2	t_c	B
≥ 1	1	≥ 13	≤ 10
2,3	2	19, 20	8, 12
≥ 4	2	≥ 21	≤ 20
3,4,5,6,7	3	25, 26, 27, 28, 29	12, 16, 20, 24, 28
≥ 8	3	≥ 30	29
≥ 4	≥ 4	29	≤ 28

Apparently, none of the sets of integers will satisfy the condition $t_c \leq B$, and hence, the task set with the given execution times and periods is not schedulable. For the second case, we obtain:

n_1	n_2	t_c	B
≥ 1	1	≥ 13	≤ 12
2, 3, 4, 5	2	19, 20, 21, 22	8, 12, 16, 20
6	2	23	24
7	2	24	24
≥ 8	2	≥ 25	24
3, 4, 5, 6, 7	3	25, 26, 27, 28, 29	12, 16, 20, 24, 28
≥ 8	3	≥ 30	29
≥ 4	≥ 4	≥ 31	29

In this case, the sets of integers $[6, 2, 1]$ and $[7, 2, 1]$ satisfy the condition $t_c \leq B$, which indicates that task τ_3 is schedulable. Similarly, for τ_2 schedulability, we obtain

$$n_1 = 2, 3, 4, 5, 6, \text{ or } 7, \text{ and } n_2 = 1.$$

Since the schedulability of τ_1 is guaranteed by $C_1 < T_1$, we conclude that all the tasks are schedulable. According to Remark 2.2, only the sets of integers $[6, 2, 1]$ and $[7, 2, 1]$ have real meaning in terms of task execution.

We now consider the schedulability problem with changeable periods by investigating the relation between the sets of integers and the ranges of feasible periods. Suppose a set of tasks is given as described in the Problem Statement. It is easy to see that the sets of integers in (1) will be infinite if the periods can be freely chosen, which implies that the algorithms for searching the sets of feasible periods may not terminate. However, when the periods are constrained to lie below their upper bounds, then the number of possible sets of integers becomes finite, and this makes possible an integer programming approach. In fact, the sets of eligible integers for τ_k to be schedulable must satisfy the following conditions:

$$n_k \equiv 1,\ \sum_{j=1}^{k} n_j C_j \leq n_i T_{mi},\ \ i = 1, .., k. \qquad (2)$$

and they are possibly not unique. It is worth emphasizing that the multiplicity of the set of integers described here is different from the case discussed in Remark 2.2 where a set of periods is fixed and given. Here different sets of integers may result in different sets of ranges of periods as described in below. Our objective is to identify all the possible ranges of periods for the tasks to be schedulable. Since the task set is assumed to be schedulable with the maximum periods, conditions in (2) for $k = 1$ are always satisfied. Suppose there exist R_k sets of integers satisfying (2) for task τ_k, and let N_k^r be the rth set among them. Then we write $N_k^r = [n_1^{kr}, ..., n_k^{kr}],\ r = 1, .., R_k$, and rewrite (2) in an equivalent form:

$$n_k^{kr} \equiv 1,\ \ D_k^r \leq \min\{n_i^{kr} T_{mi},\ i = 1, .., k\} \qquad (3)$$

where $D_k^r = \sum_{j=1}^{k} n_j^{kr} C_j$ is the total demand for CPU time for task τ_k to complete, with the number of preemptions n_j^{kr}, caused by $\tau_j,\ j = 1, .., k$. According to the relations in (1), the set of ranges of periods corresponding to the set of integers N_k^r is given by:

$$S_k^r = \{[T_1, .., T_k] :\ \frac{D_k^r}{n_i^{kr}} \leq T_i \leq T_{mi},\ i = 1, .., k\} \quad (4)$$

Apparently, when R_k has the trivial value zero, neither the task τ_k nor the task set is schedulable.

The above analysis provides the basis for developing an integer programming algorithm to search for all the sets of feasible periods. In fact, for each task $\tau_k,\ k = 2, .., p$, sets of integers $N_k^r,\ r = 1, .., R_k$, can be found from (3). Corresponding to each N_k^r, a set of ranges of periods S_k^r can be derived as in (4), which contains all the possible periods with which task τ_k is guaranteed to complete before its deadline, and it will be preempted n_j^{kr} times by task $\tau_j,\ j = 1, .., k-1$, during its execution. To make the whole task set schedulable, the intersections of the sets of ranges of periods derived for different tasks needs to be considered. Specifically, with R_k sets of ranges of periods for each task $\tau_k,\ k \geq 2$, there would be a total of $\prod_{k=2}^{p} R_k$ different intersections, which can be written as

$$I_{j_2 \cdots j_p} = \bigcap_{k=2}^{p} S_k^{j_k},\ j_i = 1, .., R_i,\ i = 2, .., p \qquad (5)$$

Since the period ranges given in (4) only affect the periods of the tasks involved, the intersection of $S_{k_1}^{r_{k_1}}$ with $S_{k_2}^{r_{k_2}},\ k_1 < k_2$, only intersects the first k_1 ranges between these two sets and leaves the remaining $k_2 - k_1$ ranges from $S_k^{r_{k_2}}$ unchanged. Eq. (5) gives the complete sets of ranges of feasible periods.

Example 2.2 To demonstrate the procedure to obtain the set of ranges of feasible periods, we consider the second case of Example 2.1, and we use the given set of periods as the upper bounds of the periods, i.e., $[T_{m1}, T_{m2}, T_{m3}] = [4, 12, 29]$. As calculated earlier, we found that task τ_3 and τ_2 are schedulable with the upper bounds of periods and obtained the sets of integers:

$$N_3^1 = [6, 2, 1],\ \ N_3^2 = [7, 2, 1],$$
$$N_2^r = [r + 1, 1],\ r = 1, .., 6$$

These integers correspond to the ranges of periods for τ_3 to be schedulable:

With N_3^1 :

$$3.83 \leq T_1 \leq 4$$
$$11.5 \leq T_2 \leq 12$$
$$23 \leq T_3 \leq 29$$

With N_3^2 :

$$3.43 \leq T_1 \leq 4$$
$$T_2 = 12$$
$$24 \leq T_3 \leq 29$$

and for τ_2 to be schedulable:

With N_2^1 :

$$3.5 \leq T_1 \leq 4$$
$$7 \leq T_2 \leq 12$$

With N_2^2 :

$$2.67 \leq T_1 \leq 4$$
$$8 \leq T_2 \leq 12$$

With N_2^3 :

$$2.25 \leq T_1 \leq 4$$
$$9 \leq T_2 \leq 12$$

With N_2^4 :

$$2.0 \leq T_1 \leq 4$$
$$10 \leq T_2 \leq 12$$

With N_2^5 :

$$1.83 \leq T_1 \leq 4$$
$$11 \leq T_2 \leq 12$$

With N_2^6 :

$$1.71 \leq T_1 \leq 4$$
$$12 \leq T_2 \leq 12$$

Intersecting each of the sets of ranges of periods for τ_3 with the sets for τ_2, we obtain the final sets of ranges of feasible periods:

$$3.83 \leq T_1 \leq 4 \qquad 3.43 \leq T_1 \leq 4$$
$$11.5 \leq T_2 \leq 12 \qquad T_2 = 12$$
$$23 \leq T_3 \leq 29 \qquad 24 \leq T_3 \leq 29$$

In deriving these final sets, we have excluded any set that can be contained in these two sets. This will be elaborated on later.

We now develop an integer programming algorithm for finding all the feasible periods for a given set of tasks with a fixed priority order. As shown earlier, deriving the sets of integers from (3) and the corresponding ranges of periods from (4) for each task's schedulability is fairly straightforward, but constructing the intersections as in (5) could be computationally intensive. To reduce the computation, we apply the branch and bound approach in the construction. Specifically, starting from task p, we consider all the intersections of $S_p^{j_p}$ with $S_{p-1}^{j_{p-1}}$ for all $j_p = 1, .., R_p$, $j_{p-1} = 1, .., R_{p-1}$. Among the resulting sets of ranges of periods, some of them may be contained by the others. Suppose A and B are two sets from these intersections, and $A \subset B$. As further intersections are taken, for instance, $A \cap S_{p-2}^{j_{p-2}}$ and $B \cap S_{p-2}^{j_{p-2}}$, it is clear that $(A \cap S_{p-2}^{j_{p-2}}) \subset (B \cap S_{p-2}^{j_{p-2}})$. Namely, any further intersection starting from A will be contained in the same intersection starting with B. Therefore, all further intersections starting from A can be ignored once A is discovered to be contained in another set. This may reduce computation significantly. A complete algorithm is described in below.

Algorithm 2.1 Searching for All Feasible Periods with A Fixed Priority Order $T_1 \leq \cdot\cdot \leq T_p$

Step 1: Determine the sets of integers. For each task k, $k = 2, .., p$, we search for the sets of integers $N_k^r = [n_1^{kr}, .., n_{k-1}^{kr}, 1]$, $r = 1, .., R_k$, such that (3) is satisfied. Since $T_1 \leq \cdot\cdot \leq T_p$, only $n_1^{kr} \geq \cdot\cdot n_{k-1}^{kr} \geq 1$ needs to be considered.

Step 2: Derive the ranges of periods. For each N_k^r obtained from Step 1, we compute the set of ranges of periods S_k^r from (4). In addition, we define $k \times 2$ matrices $B_k^r, r = 1, .., k$, to represent the lower limits of the ranges of periods in S_k^r, $L_j^{kr} = D_k^r / n_j^{kr}$, $j = 1, .., R_k$, and the upper limits $T_{m1}, .., T_{mk}$. Namely,

$$B_k^r = \begin{bmatrix} L_1^{kr}, ..., L_k^{kr} \\ T_{m1}, ..., T_{mk} \end{bmatrix}^T$$

Therefore, each set of the ranges of periods, S_k^j, can be specified by the corresponding matrix B_k^r.

Step 3: Intersect the sets of ranges of periods. In this step, we construct a sequence of intersections from $S_p^{j_p}$ to $S_2^{j_2}$, i.e.,

$$(\cdot\cdot((S_p^{j_p} \cap S_{p-1}^{j_{p-1}}) \cap S_{p-2}^{j_{p-2}}) \cap \cdot\cdot \cap S_2^{j_2}).$$

Let $\mathcal{I}$ be a set of $p \times 2$ matrices that contain the limits of the resulting intersections, i.e.,

$$\mathcal{I} = \{\{I_{\alpha\beta}^r\},\ r = 1, .., N\}$$

where $\alpha = 1, .., p$, $\beta = 1, 2$, $I_{\alpha 1}^r$ and $I_{\alpha 2}^r$ are the lower and upper limits of the range of period T_α in the kth intersection, respectively, and N is the number of intersections specified in $\mathcal{I}$. Starting with $S_p^{j_p}$, we initialize $\mathcal{I}$ to contain all $B_p^{j_p}$, $j_p = 1, .., R_p$. Then we intersect each of the intersections specified by the corresponding element in $\mathcal{I}$ with each set of ranges of periods for task τ_k, specified by the corresponding matrix $B_k^{j_k}$, $k = p-1, .., 2$. In particular, the intersection of the set specified by $\{I_{\alpha\beta}^r\}$ with $S_k^{j_k}$ can be specified by a $p \times 2$ matrix given by

$$\begin{bmatrix} \max\{I_{11}^r, L_1^{kj_k}\} & T_{m1} \\ \vdots & \vdots \\ \max\{I_{k1}^r, L_k^{kj_k}\} & T_{mk} \\ I_{(k+1)1}^r & T_{m(k+1)} \\ \vdots & \vdots \\ I_{p1}^r & T_{mp} \end{bmatrix}$$

For ease of exposition, we will refer the above matrix manipulation as $\mathcal{I} \cap B_k^{j_k}$. After all the intersections of $\mathcal{I}$ with $B_k^{j_k}$, $j_k = 1, .., R_k$, have been taken, we obtain a new set $\mathcal{I}$ with $N \times R_k$ intersections. Among these intersections, we delete those that can be contained by the others. For example, if $\{I_{\alpha\beta}^{r_1}\}$ and $\{I_{\alpha\beta}^{r_2}\}$ are two elements in the newly derived $\mathcal{I}$, we say $\{I_{\alpha\beta}^{r_1}\}$ is contained in $\{I_{\alpha\beta}^{r_2}\}$ if for all $\alpha = 1, .., p$, $\{I_{\alpha 1}^{r_1}\} \geq \{I_{\alpha 1}^{r_2}\}$, and $\{I_{\alpha\beta}^{r_1}\}$ will be eliminated from the set $\mathcal{I}$. Then the resulting $\mathcal{I}$ will have a reduced number of elements and this set will be used to intersect further with $B_{k-1}^{j_{k-1}}$. Repeat this step until all $S_2^{j_2}$, $j_2 = 1, .., R_2$, have been intersected with $\mathcal{I}$. Then the final set $\mathcal{I}$ specifies all the sets of ranges of feasible periods for the order of $T_1 \leq \cdot\cdot \leq T_p$.

Remark 2.3 While the sets of feasible periods obtained from Algorithm 2.1 contains all the periods for which the set of tasks is schedulable, the sets of ranges of feasible periods specified in the final set $\mathcal{I}$ may overlap. This implies that there exists some set of periods which can be simultaneously obtained from different sets of integers. In other words, there will be multiple sets of integers satisfying (3) with the same set of periods as discussed in Remark 2.2. For example, consider a set of tasks $\{\tau_1, \tau_2\}$ with execution times $C = [10, 20]$. From (4), we obtain the ranges of periods summarized in the following table, for $n_1 = 1, 2, 3$.

$n_1 = 1$	$n_1 = 2$	$n_1 = 3$
$T_1 \geq 30$	$20 \leq T_1$	$16.66 \leq T_1$
$T_2 \geq 30$	$T_2 \geq 40$	$T_3 \geq 50$

Apparently, the sets of ranges of periods obtained from the sets of integers $[1, 1]$ and $[2, 1]$ overlap in the ranges $30 \leq T_1$ and $T_2 \geq 40$. Therefore, any periods chosen from these overlapped ranges will have two sets of integers which both satisfy the conditions in (1). Although it would be nice to have exclusive sets of ranges of feasible periods, it is not necessary as far as task schedulability is concerned. As discussed in Remark 2.2, once the set of periods is determined, the set of integers, i.e., the actual

47.00	50.00	47.50	50.00	47.00	50.00	50.00	50.00
80.00	80.00	78.33	80.00	78.33	80.00	62.50	80.00
80.00	100.00	95.00	100.00	78.33	100.00	83.33	100.00
117.50	166.67	117.50	166.67	125.00	166.67	125.00	166.67
235.00	250.00	235.00	250.00	235.00	250.00	250.00	250.00

Figure 1: Elements of set $\mathcal{I}$ for Example 2.3. Each block is a matrix in $\mathcal{I}$.

number of execution of each task, is fixed with the one yielding the smallest completion time of the task with the longest period.

Example 2.3 Consider a set of tasks $\{\tau_1, .., \tau_5\}$ with computation time $C_1, .., C_5$ and the upper bounds of period $T_{m1}, .., T_{m5}$ given in the following table. Find all the feasible periods with the priority order $T_1 \leq \cdot\cdot \leq T_5$.

	τ_1	τ_2	τ_3	τ_4	τ_5
C_i (ms)	10	15	20	25	30
T_{mi} (ms)	50	80	100	166.67	250

Following Algorithm 2.1, we find that the number of sets of integers for each task to be schedulable are $R_5 = 2,\ R_4 = 10,\ R_3 = 9,\ R_2 = 6$, and these sets of integers would result in $6 \cdot 9 \cdot 10 \cdot 2 = 1080$ possible sets of ranges of feasible periods. By using the branch and bound approach, most of the intermediate ranges will be eliminated at an earlier stage and the final number of sets of ranges of feasible periods will be reduced to 4. The final ranges of feasible periods specified in $\mathcal{I}$ are given in Fig. 1.

Example 2.4 Consider a set of tasks $\{\tau_1, .., \tau_5\}$ with real-time data given in the following table. The period of task τ_5 is required to be fixed as given. Find all the feasible periods $[T_1, .., T_4]$ with the priority order $T_1 \leq \cdot\cdot \leq T_5$.

	τ_1	τ_2	τ_3	τ_4	τ_5
C_i (ms)	10	10	10	10	5
T_{mi} (ms)	50	55.56	66.67	100	100

Again, the number of sets of integers for each task to be schedulable is obtained from Algorithm 2.1 as $R_5 = 9,\ R_4 = 10,\ R_3 = 5,\ R_2 = 4$, and there are totally 12 sets of ranges of feasible periods specified in set $\mathcal{I}$, and they are listed in Fig. 2.

While both examples are scheduled with the fixed priority order the same as the order of the maximum periods, the numbers of final sets of ranges of feasible periods are different significantly. Such differences are a consequence of the utilization levels that each set of tasks can have. For the first example, the allowable utilization is constrained in the range [0.8575, 1],while the second example has a larger range [0.63, 1]. Therefore, there is a wider choice of periods in the second example than the first one. The difference on the allowable utilization will have a large impact when the tasks are scheduled without any prescribed priority order, as will be addressed in the subsequent section.

27.50	50.00	32.50	50.00	37.50	50.00	28.33	50.00
55.00	55.56	32.50	55.56	37.50	55.56	42.50	55.56
55.00	66.67	65.00	66.67	37.50	66.67	42.50	66.67
55.00	100.00	65.00	100.00	75.00	100.00	85.00	100.00
100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
23.75	50.00	31.67	50.00	42.50	50.00	40.00	50.00
47.50	55.56	31.67	55.56	42.50	55.56	47.50	55.56
47.50	66.67	47.50	66.67	42.50	66.67	47.50	66.67
95.00	100.00	95.00	100.00	42.50	100.00	47.50	100.00
100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
31.67	50.00	31.67	50.00	35.00	50.00	31.67	50.00
50.00	55.56	47.50	55.56	47.50	55.56	47.50	55.56
50.00	66.67	60.00	66.67	47.50	66.67	47.50	66.67
50.00	100.00	60.00	100.00	70.00	100.00	80.00	100.00
100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

Figure 2: Elements of set $\mathcal{I}$ for Example 2.4. Each block is a matrix in $\mathcal{I}$.

2.2 Task Periods for RMA Schedulability with No Prescribed Priority Order

We now move on to investigate the schedulability of a given set of tasks without a prescribed priority order of task execution. Again, we are searching for the possible periods for all the tasks to be schedulable.

Problem Statement: Given a set of periodic tasks $\{\tau_1, .., \tau_p\}$ with execution time $C_1, .., C_p$ and upper bounds of periods $T_{m1}, .., T_{mp}$. Suppose the set of tasks is schedulable with the maximum periods $T_{m1}, .., T_{mp}$. Find all the possible periods $T_1, .., T_p$ satisfying the condition $T_1 \leq T_{m1}, .., T_p \leq T_{mp}$ so that all the tasks are schedulable. Without loss of generality, we assume $T_{m1} \leq \cdot\cdot \leq T_{mp}$.

As a direct use of Algorithm 2.1, one could arrange the periods in $p!$ different priority orders, and run through the algorithm for each one of them to find out the ranges of periods. The resulting period ranges will cover all the periods, arranged in all orders, such that the set of tasks is schedulable. Undoubtedly, this will involve a huge amount of computation, and it is not always necessary. In most cases, a large number of the permutations of the priority order may result in the ranges of feasible periods which can be contained by the ones obtained from other permutations. Therefore, one can use the branch and bound approach to eliminate some of the permutations in the search. A *branch* for the current problem is defined as a particular order of task priority, represented by the order of the periods as $T_{i_1} \leq \cdot\cdot \leq T_{i_p}$, where $1 \leq i_j \leq p,\ \forall j = 1, .., p$, and $i_1, .., i_p$ are distinguished. By a *priority level*, we mean

that a task at this level will have the ith highest priority among all the tasks. To apply the branch and bound approach, we make the following observations.

1. Suppose $r < p$ tasks are chosen to run last with a fixed priority order $T_{i_{p-r+1}} \leq \cdot\cdot \leq T_{i_p}$, and the remaining tasks have an unrestricted priority order. Then the condition in (3) can be modified to generate all the sets of integers with all possible priority orders of the $p - r$ tasks such that the ranges of periods derived from the integers to provide all the possible periods for those r task to be schedulable. Specifically, for task τ_{i_k} to be schedulable, $i_k \in \{i_{p-r+1}, .., i_p\}$, the set of integers $n_{i_1}, .., n_{i_k}$ must satisfy the following condition:

$$\begin{cases} \sum_{j=1}^{k} n_{i_j} C_{i_j} \leq \min\{n_i T_{mi},\ \forall i \in \{i_1, ..i_k\}\} \\ n_{i_{p-r+1}} \leq \cdot\cdot \leq n_{i_k} \equiv 1 \end{cases} \tag{6}$$

   From these sets of integers, we obtain all the possible ranges of periods for tasks $\tau_{i_1}, .., \tau_{i_k}$ as in (4) such that τ_{i_k} is schedulable given its priority in the fixed priority order. Furthermore, by running through this procedure for all $i_k \in \{i_{p-r+1}, .., i_p\}$, and taking intersections of the resulting sets of ranges of periods from task to task as described in Algorithm 2.1, we obtain all the possible sets of ranges of periods for all tasks that guarantee the schedulability of those r tasks as the last to run with the fixed priority order. We will call the problem of finding all the possible ranges of periods for this set of tasks the *partially-fixed-order problem*. The case of only one tasks being chosen as the last to run is also a partially-fixed-order problem since the order of that task is fixed.

2. Again, assume the same as in the first observation. Let $\sigma =< j_{p-r+1}, .., j_p >$ represent a permutation of the fixed priority order $T_{p-r+1} \leq \cdot\cdot \leq T_p$, and $\mathcal{I}_\sigma$ be the set of $p \times 2$ matrices that contains the limits of all the possible ranges of periods obtained from solving the partially-fixed-order problem with the fixed priority specified by σ. Then $\mathcal{I}_\sigma$ specifies a set of ranges of periods that contain all the sets of ranges of feasible periods obtained from all the possible priority orders with last r tasks fixed as given.

To summarize, we conclude that, if the set of ranges of periods specified by $\mathcal{I}_\sigma$ turns out to be contained in some set of ranges of feasible periods that have been considered, then no more permutation of $< *, .., *, \sigma >$ needs to be investigated, where $< *, .., * >$ represents all permutations of period for tasks $\tau_{i_1}, .., \tau_{i_{p-r}}$ and $< *, .., *, \sigma >=< *, .., *, j_{p-r+1}, .., j_p >$ if $\sigma =< j_{p-r+1}, .., j_p >$. In this case, all the permutations $< *, .., *, \sigma >$ are considered to be checked in the sense of generating feasible periods.

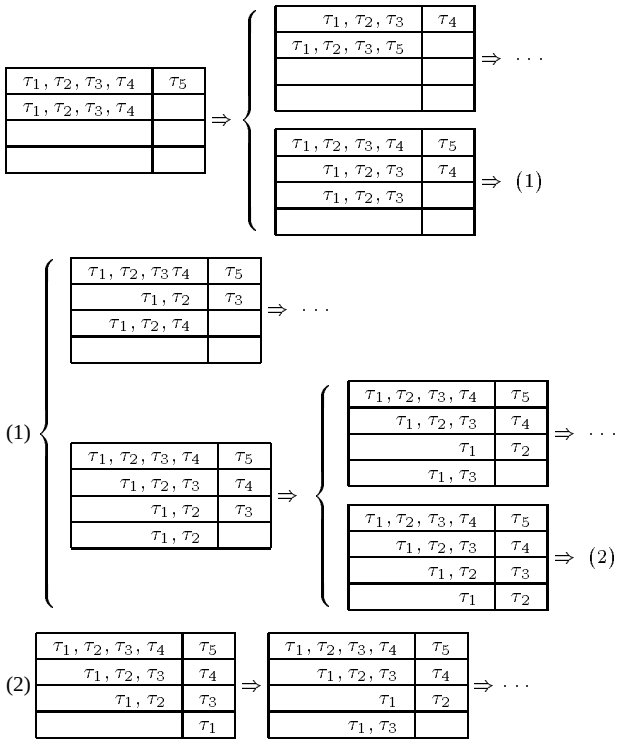

Figure 3: Some of the steps from the algorithm

We now develop an integer programming algorithm with the branch and bound approach utilizing the properties described above. Let $\mathcal{I}$ be the set of $p \times 2$ matrices that contains the limits of ranges of feasible periods. Starting with a fixed order of period, say the order same as the maximum periods, i.e., $\sigma =< 2, .., p >$, we derive a set of ranges of feasible periods specified by $\mathcal{I}_\sigma$ and initialize $\mathcal{I} = \mathcal{I}_\sigma$. For each priority level, we associate with it a list of candidate tasks to be scheduled with this priority. Starting from $\sigma =< 2, .., p >$, we go through a sequence of permutations of priority and for each permutation, we derive the set $\mathcal{I}_\sigma$. If some elements in $\mathcal{I}_\sigma$ can not be contained by any element in $\mathcal{I}$, we add them in $\mathcal{I}_\sigma$ to $\mathcal{I}$ if σ is a permutation of $p-1$ tasks; otherwise, we proceed with a new permutation $< \gamma, \sigma >$, where γ is the index of a candidate task that can run just before the tasks ordered by σ. τ_γ will be withdrawn from the list of candidates. A new element in $\mathcal{I}_\sigma$ is the one that can not be contained by any element in $\mathcal{I}$. If $\mathcal{I}_\sigma$ does not contain any new elements, i.e., all the elements in $\mathcal{I}_\sigma$ can be contained by some elements in $\mathcal{I}$, there is no need to pursue any further check with permutations $< *, .., *, \sigma >$, and we change the permutation $\sigma =< j_{p-r+1}, .., j_p >$ by replacing j_{p-r+1} with the index of a task from the list of candidates. Such a task will be removed from the list of candidates once it is placed in the fixed order. If no more candidate task are in the list, we replace j_{p-r+2} with a new one and proceed with $\sigma =< j_{p-r+2}, .., j_p >$. This procedure is repeated until all the permutation are considered to be checked in the sense of generating feasible periods.

To illustrate some of the steps in this algorithm, we consider a task set with five tasks. Construct a block with two columns: the left column lists the candidate tasks at each of the priority levels 2,..,5, and the right column gives the permutations of priority to be checked, i.e., σ. The lower the task is in the block, the higher is its priority. When the algorithm proceeds, the content in the block changes as shown in Fig. 3. To start, we pick τ_5 as the last to run and set $\sigma =< 5 >$. Then tasks $\tau_1, .., \tau_4$ will be the candidates at the priority levels 4 and 5. Derive $\mathcal{I}_\sigma$ and compare it with set $\mathcal{I}$ which is initialized as $\mathcal{I}_{<2,..,5>}$. If $\mathcal{I}_\sigma$ has no new element, a new task is chosen from the list of candidate tasks at priority level 5, say τ_4, and removed from the list. The list of candidate tasks at priority level 4 is then modified by taking out τ_4 and adding τ_5. The algorithm continues with $\mathcal{I}_{<4>}$. If $\mathcal{I}_\sigma$ contains some new elements, a task with higher priority is picked from the list of candidate tasks at priority level 4, say τ_4, and removed from the list. A new list of candidate tasks at priority level 3 is established by using the list at the priority level 4, and we move on with $\sigma =< 4, 5 >$. Suppose both $\mathcal{I}_{<4,5>}$ and $\mathcal{I}_{<3,4,5>}$ will have some new elements. Then $\mathcal{I}_{<2,3,4,5>}$ will be derived and all its elements will be contained by some elements in $\mathcal{I}$. This is the path indicated by $\cdot\cdot \Rightarrow (1) \Rightarrow (2) \Rightarrow \cdot\cdot$. To proceed after $\sigma =< 2, 3, 4, 5 >$, we replace τ_2 by τ_1 as the second task to run, and derive $\mathcal{I}_{<1,3,4,5>}$. If $\mathcal{I}_{<1,3,4,5>}$ contains some new elements, add them to $\mathcal{I}$. Since there is no more candidate at priority level 2, we pick a new task, say τ_3, as the 3rd task to run, and proceed in the similar manner as earlier. A formal description of the algorithm is given as below.

Algorithm 2.2 Searching for All Feasible Periods with All the Possible Priority Orders:

Variables

$To\_Be\_Checked$: a p element array, and each element $To\_Be\_Checked[i],\ i = 2, .., p$, contains a list of the candidate tasks at priority level i.

γ_i : the highest index of the task among the tasks contained in $To\_Be\_Checked[i]$.

r : the number of tasks involved in σ.

Init:

- Derive $\mathcal{I}_{<2,..,p>}$ and initialize $\mathcal{I} = \mathcal{I}_{<2,..,p>}$;
- $To\_Be\_Checked[i] = \{\tau_k,\ k = 1, .., p-1\}$, $i = p,\ p-1$;
- $\sigma =< p >$ and $r = 1$.

# of Orders Checked	σ	# of New Matrices in I_σ	# of Matrices obtained in I
1	<5>	1	4
2	<4,5>	0	4
3	<3,5>	0	4
4	<2,5>	0	4
5	<15>	0	4
6	<4>	0	4
7	<3>	0	4
8	<2>	0	4
9	<1>	0	4

Figure 4: The ranges of feasible periods for the set of tasks in Example 2.3 with all possible priority orders.

Step 1:

- Solve the partially-fixed-order problem for $\mathcal{I}_\sigma$.
- If $\mathcal{I}_\sigma$ has no new element, go to Step 2;
- Otherwise
  If $r = p - 1$, add the new elements to $\mathcal{I}$;
  If $r < p - 1$, $r = r + 1$;
  go to Step 2.

Step 2:

- $v = p - r + 1$;
- If $To\_Be\_Checked[v]$ is empty and $r = 1$
  the search is over. $\mathcal{I}$ specifies all the ranges of feasible periods with all the priority orders;
- If $To\_Be\_Checked[v]$ is empty and $r > 1$
  $r = r - 1$ and empty $To\_Be\_Checked[v-1]$;
  go back to Step 2;
- If $To\_Be\_Checked[v]$ is not empty
  If $r < p - 1$
  $To\_Be\_Checked[v-1]$
  $= \{To\_Be\_Checked[v] - \{\tau_{\gamma_v}\},\ \tau_{j_v}\}$;
  $\sigma =< \gamma_v, j_{v+1}, .., j_p >$;
  eliminate γ_v from $To\_Be\_Checked[v]$;
  go to Step 1.

We now apply Algorithm 2.2 in Examples 2.3 and 2.4 to find all the sets of ranges of feasible periods with all the possible orders of priority. Results are shown in Fig. 5. Due to the limitation of space, we only present the number of sets of ranges of periods at various steps as the algorithm proceeds without showing the actual ranges. In the first example, we initialize $\mathcal{I} = \mathcal{I}_{<2,..,5>}$. Starting with $\sigma =< 5 >$ and $r = 1$, we see that $\mathcal{I}_\sigma$ has one new element and that leads to the check of the new permutation $\sigma =< 4, 5 >$. Since $\mathcal{I}_{<4,5>}$ does not give any new element, $\sigma =< 3, 5 >$ is considered afterwards. As none of $\sigma =< 3, 5 >$, $< 2, 5 >$ and $< 1, 5 >$ generate any new element in $\mathcal{I}_\sigma$, all the branches $< *, .., *, 5 >$ are checked, and the task at priority level 5 is replaced. Further calculation shows that none of $\mathcal{I}_\sigma$, $\sigma =< 4 >, .., < 1 >$, have any new element, and therefore, the algorithm terminates. To conclude, we see that there are four sets of ranges of feasible periods, all of them obtained from $\mathcal{I}_{<2,..,5>}$, and only 9 out of 120 (5!) permutations are checked.

# of Orders Checked	σ	# of New Matrices in $\mathcal{I}_\sigma$	# of Matrices obtained in $\mathcal{I}$
1	<5>	10	13
2	<4,5>	15	13
3	<3,4,5>	7	13
4	<2,3,4,5>	0	13
5	<1,3,4,5>	7	20
6	<2,4,5>	7	20
7	<3,2,4,5>	1	21
8	<1,2,4,5>	6	27
9	<1,4,5>	1	27
10	<3,1,4,5>	1	28
11	<2,1,4,5>	0	28
12	<3,5>	2	28
13	<4,3,5>	0	28
14	<2,3,5>	2	28
15	<4,2,3,5>	0	28
16	<1,2,3,5>	2	30
17	<1,3,5>	0	30
18	<2,5>	0	30
19	<1,5>	0	30
20	<4>	21	30
21	<5,4>	15	30
22	<3,5,4>	8	30
23	<2,3,5,4>	6	36
24	<1,3,5,4>	2	38
25	<2,5,4>	5	38
26	<3,2,5,4>	3	41
27	<1,2,5,4>	2	43
28	<1,5,4>	2	43
29	<3,1,5,4>	2	45
30	<2,1,5,4>	0	45
31	<3,4>	16	45
32	<5,3,4>	1	45
33	<2,5,3,4>	1	46
34	<1,5,3,4>	0	46
35	<2,3,4>	10	46
36	<5,2,3,4>	3	49
37	<1,2,3,4>	7	56
38	<1,3,4>	5	56
39	<5,1,3,4>	3	59
40	<2,1,3,4>	2	61
41	<2,4,>	4	61
42	<5,2,4>	0	61
43	<3,2,4>	0	61
44	<1,2,4>	4	61
45	<5,1,2,4>	2	63
46	<3,1,2,4>	2	65
47	<1,4>	0	65
48	<3>	0	65
49	<2>	0	65
50	<1>	0	65

Figure 5: The ranges of feasible periods for the set of tasks in Example 2.4 with all possible priority orders.

For the second example, we assume that the period of task τ_5 can be changed as well. Again, we initialize $\mathcal{I} = \mathcal{I}_{<2,..,5>}$ with 13 sets of ranges of feasible periods. Following the steps of the algorithm, we find that all of $\mathcal{I}_\sigma$, $\sigma =< 5 >$, $< 4, 5 >$ and $< 3, 4, 5 >$, contain some new elements, and that leads to the check of $\mathcal{I}_{<2,3,4,5>}$, which, of course, will not have any new element. Replacing τ_2 by τ_1, the candidate at priority level 2, we derive $\mathcal{I}_{<1,3,4,5>}$ and get 7 new elements. These new elements are then added to $\mathcal{I}$. It is worth emphasizing that the new elements from $\mathcal{I}_\sigma$ are added to $\mathcal{I}$ only when σ is a permutation of $p-1$ tasks, which implies that all the tasks are schedulable with the priority order specified by σ. As σ involves less than $p-1$ tasks, only those tasks being specified have guaranteed schedulability, and as a result, the ranges of periods specified in $\mathcal{I}_\sigma$ may not all be feasible. To conclude the example, we notice that the permutations $\sigma =< *, ..*, 5 >$ and $\sigma =< *, .., *, 4 >$ both generate many new elements, while $\sigma =< *, ..*, 3 >$, $< *, .., *, 2 >$ and $< *, .., *, 1 >$ give none. This can be explained intuitively by taking a look at the period bounds. In fact, the upper bounds of T_1, T_2 and T_3 are almost one half of the bounds for T_4 and T_5, and therefore, using one of T_1, T_2 or T_3 as the lower priority task would constrain the ranges of feasible periods much more than making T_4 or T_5 as the lower priority tasks. In the end, there are 65 sets of ranges of feasible periods derived, and 50 permutations have been checked, which is 42% of 5!. Comparing to the result from Example 2.3, where the number of permutations checked is only 7.5% of 5!, we conclude that the more the utilization is constrained, the more effective is the proposed algorithm.

3 The Optimal Periods for RMA Schedulability

In this section, we address the issue of finding the periods for a set of tasks such that the given performance measure of the overall system is optimized and the task set is schedulable using RMA. In particular, we will focus on the class of systems in which the performance index for each task can be described by a monotonically increasing function of task period, which we will call a cost function. It is desirable to have a lower cost for each task. Such classes of systems have been discussed in [4] where the tasks are control law computations and the cost functions are defined as the difference on a performance index evaluated using continuous and digitized control laws. A precise description of the problem is given by the following:

Problem Statement: Given a set of periodic tasks $\tau_1, .., \tau_p$ with execution times $C_1, .., C_p$, upper bounds of periods $T_{m1}, .., T_{mp}$, and a cost function for the overall system, $J(T_1, .., T_p)$, with $\partial J/\partial T_i \geq 0$, $\forall 0 < T_i \leq T_{mi}$. Suppose the set of tasks is schedulable with the maximum periods. Let $T = [T_1, .., T_p]$. Find the set of optimal periods T^* such that $J(T)$ is minimized and all the tasks are schedulable.

The optimization problem can be solved after all the feasible periods have been found using the search algorithm proposed in the last section. Since all the sets of ranges of feasible periods are specified by matrices in $\mathcal{I}$, all the feasible periods are given by

$$I^i_{j1} \leq T_j \leq T_{mj}, \quad j = 1, .., p, \tag{7}$$

where $i = 1, .., N$ and N is the total number of matrices in $\mathcal{I}$. Then the optimal solution to the periods is obtained as:

$$T^* = \min_{T \in \{L_i,\ i=1,..,M\}} \{J(T)\}, \quad L_i = [I^i_{11}, .., I^i_{p1}].$$

While the optimal solution can be obtained by checking all the possible periods which guarantee RMA schedulability, it is not necessary to do so if finding the optimal periods is the sole objective. In fact, the search algorithm presented in last section can be easily modified to provide the set of optimal periods. Suppose a set of tasks is given without any prescribed priority order. In last section, we

have discovered that for any permutation of priority σ, the sets of ranges of periods specified by $\mathcal{I}_\sigma$ will contain all the sets of ranges of periods corresponding to permutations $< *, .., *, \sigma >$. Since only the lower limits of the periods in each set are of interest as far as minimizing $J(T)$ is concerned, none of the sets of ranges of periods would possibly result in an optimal solution if the ones specified by $\mathcal{I}_\sigma$ do not. Therefore, if none of the $J(T)$ evaluated with the sets of the lower limits of periods specified by $\mathcal{I}_\sigma$ has a smaller value than some $J(T)$ already obtained, the whole branch specified by permutations $< *, .., *, \sigma >$ can be eliminated from further consideration. Let L_σ^{min} be the set of lower limits of periods specified by $\mathcal{I}_\sigma$ which gives the smallest value of $J(T)$ among all the sets of periods specified in $\mathcal{I}_\sigma$. By taking into account the evaluation of $J(T)$, we modify the algorithm given in last section as follows:

Algorithm 3.1 Searching for the Optimal Feasible Periods:

Init:

- Derive $\mathcal{I}_{<2,..,p>}$ and evaluate $J(T)$ to obtain $L^{min}_{<2,..,p>}$;
- Initialize $T^* = L^{min}_{<2,..,p>}$;
- $To\_Be\_Checked[i] = \{\tau_k,\ k = 1, .., i-1\}$, $\forall i = p, p-1$.
- $\sigma =< p >$ and $r = 1$.

Step 1:

- Solve the partially-fixed-order problem for $\mathcal{I}_\sigma$.
- Evaluate $J(T)$ to find L_σ^{min};
- If $L_\sigma^{min} \geq T^*$, got to Step 2;
- Otherwise
  If $r = p-1,\ T^* = L_\sigma^{min}$ and go to Step 2;
  If $r < p-1, r = r+1$ and go to Step 2.

Step 2:

- $v = p - r + 1$;
- If $To\_Be\_Checked[v]$ is empty and $r = 1$
  the search is over and
  T^* contains the set of optimal periods;
- If $To\_Be\_Checked[v]$ is empty and $r > 1$
  $r = r - 1$ and empty $To\_Be\_Checked[v-1]$;
  go back to Step 2;
- If $To\_Be\_Checked[v]$ is not empty
  If $r < p - 1$
  $To\_Be\_Checked[v-1]$
  $= \{To\_Be\_Checked[v] - \{\tau_{\gamma_v}\},\ \tau_{j_v}\}$;
  $\sigma =< \gamma_v, j_{v+1}, .., j_p >$;
  eliminate γ_v from $To\_Be\_Checked[v]$;
  go to Step 1.

Again, Algorithm 3.1 is illustrated by the examples presented earlier. As before, we reassign the tasks' indices such that $T_{m1} \leq \cdot\cdot \leq T_{mp}$. We assume both examples have a cost function given by $J(T) = \sum_{i=1}^{5} \alpha_i e^{-\beta_i/T_i}$ with parameters for Example 2.3

α_i	0.67	1.33	2.0	2.67	3.33
β_i	0.3	0.4	0.5	0.6	0.7

for Example 2.4

α_i	1	2	5	3	0
β_i	0.1	0.3	0.5	0.7	0

Results of Example 2.3

# of Orders Checked	σ	# of Matrices obtained in I_σ	Optimal Cost
1	<5>	2	0.19965
2	<4,5>	4	0.19965
3	<3,5>	1	0.19965
4	<2,5>	1	0.19965
5	<1,5>	1	0.19965
6	<4>	1	0.19965
7	<3>	1	0.19965
8	<2>	1	0.19965
9	<1>	1	0.19965

Final Result:
Optimal cost: 0.19965
Optimal Periods: [47, 80, 80, 117.5, 235]

Results of Example 2.4

# of Orders Checked	σ	# of Matrices obtained in I_σ	Optimal Cost
1	<5>	18	0.02048
2	<4>	34	0.02048
3	<3>	4	0.02048
4	<2>	3	0.02048
5	<1>	2	0.02048

Final Result:
Optimal cost: 0.02048
Optimal Periods: [23.75, 47.5, 47.5, 95, 100]

Figure 6: Illustration of finding the optimal feasible periods for a set of task with no particular priority order.

As can be seen in Fig. 6, where the results are shown, the search terminated much faster than the search for feasible periods, when the check of minimum value of $J(T)$, evaluated with each set of ranges of periods, is used for branching. This will be the case only when the true optimal value of $J(T)$ is found, but not confirmed, at an earlier stage. In both examples, the minimum value of $J(T)$ was obtained from the range of feasible periods specified by $\mathcal{I}_{1\cdot\cdot 5}$, the set derived first.

4 Conclusion

In this paper, we studied the issue of choosing the periods for a given set of real-time periodic tasks such that the performance of the system is optimized in the sense

that a given cost function is minimized, and all the tasks are schedulable with RMA. The problem is solved by finding all the sets of periods with which the set of tasks is schedulable with RMA, and then identifying the set of periods, among all the possible sets, that minimizes the cost function. To find the possible periods that render the RMA schedulability for the task set, the testing algorithm reported in [5] is formulated as an integer programming problem. By solving this integer programming problem with the branch and bound approach, all the possible periods for RMA schedulability are obtained, described by a finite number of sets of ranges of feasible periods. An algorithm to find all the feasible periods with a given order of priority is developed first, and based on that, an algorithm for searching for all the feasible periods with all the possible priority orders is presented. The results obtained from the latter algorithm cover all the sets of periods that make the task set schedulable with RMA. From the derived sets of ranges of feasible periods, the optimal periods are determined by evaluating the cost function with all the lower limits of these ranges. Moreover, by incorporating the evaluation of the cost function in the search algorithm, the optimal periods can be obtained without actually deriving all the possible ranges of periods.

The contributions of the paper are three-fold. First, it addresses the issue of RMA schedulability from an alternative perspective, namely, reformulating the schedulability conditions as the existence of a set of integers. Such a formulation transforms task schedulability into an integer programming problem, which enables the use of a class of branch and bound search algorithms.

Second, finding all the feasible periods and making them available in the design stage of an application would reduce the gap between design and implementation. By presenting the computing resource constraint as a range of feasible periods for the tasks in the task set, the designer could carry out the task design in a systematic manner, and the resulting task set will be guaranteed to be schedulable. For example, in a control application, the control algorithm can be designed subject to the constraint on sampling frequency, given as the ranges of feasible frequencies.

Third, using the optimal periods for the tasks would make a better use of the limited resource with respect to some given performance measure. Raising the efficiency of resource usage is an important issue in practice. In some applications, the number of computing devices is limited due to certain space, weight and power constraints, and therefore, adding more facilities may not be a solution to accommodate the increase of tasks' frequencies as required for performance improvement. In other cases, if the computing device is one element of a mass produced system (for example computer controllers in automobiles), then the cost of the computing device may be a major concern. In such applications, using a high speed computing device to allow for high task sampling frequencies and ease of schedulability may not be an acceptable solution because of its cost impact on the final product. In both situations, it would be desirable to determine the best performance that the system can achieve with the given available computing resource, and how to make the system improve its performance by achieving a higher level utilization of the available resource. The approaches presented in this paper provide a solution to these concerns when RMA is used for scheduling.

The efficiency of the search algorithm proposed in this paper depends on applications, as can be seen from the examples. They are efficient when the utilization is constrained. The number of the task, on the other hand, does not play a significant role here, opposed to what one would expect. In the case of a more restrictive utilization being given, the algorithms will rapidly finish their search no matter how many tasks are involved. On the other hand, when the system contains a large number of tasks and they have a low level utilization of the computing resource, it would take a long time for the algorithm to complete the search, and the result obtained, for instance the sets of ranges of feasible periods, could be a large set of different possibilities. Therefore, we conclude that the algorithms developed in this paper are efficient for reasonably structured engineering applications.

References

[1] Gerber, R., Hong, S. and Saksena, M., "Guaranteeing end-to-end timing constraints by calibrating Intermediate Processes," *Proceedings of the IEEE Real-Time Systems Symposium*, December, 1994.

[2] Locke, C. D., "Best-effort decision making for real-time scheduling." Ph.D. Dissertation, Computer Science Department, Carnegie Mellon University, 1986.

[3] D.-T. Peng, K. G. Shin, and T. K. Abdelzaher, "Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems," *IEEE Transactions on Software Engineering*, Vol. 23, No. 12, December, 1997.

[4] Seto, D., J. P. Lehoczky, L. Sha, and K. G. Shin, "On Task Schedulability in Real-Time Control System," in the *Proceedings of 17th Real-Time Systems Symposium*, pp. 13-21, December, 1996.

[5] Sha, L., Rajkumar, R. and Sathaye, S. S., "Generalized rate-monotonic scheduling theory: A framework for developing real-time systems," *Proceedings of the IEEE*, Vol. 82, No. 1, January, 1994.