Generalized Elastic Scheduling *

Thidapat Chantem and Xiaobo Sharon Hu
Department of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556
{tchantem,shu}@cse.nd.edu

M.D. Lemmon
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556
lemmon@nd.edu

Abstract

The elastic task model [9] is a powerful model for adapting real-time systems in the presence of uncertainty. This paper generalizes the existing elastic scheduling approach in several directions. It reveals that the original task compression algorithm in [9] in fact solves a quadratic programming problem that seeks to minimize the sum of the squared deviation of a task's utilization from initial desired utilization. This finding indicates that the task compression algorithm may be applied to efficiently solve other similar types of problems. In particular, an iterative approach is proposed to solve the task compression problem for real-time tasks with deadlines less than respective periods. Furthermore, a new objective for minimizing the average difference of task periods from desired values is introduced and a closed-form formula is derived for solving the problem without recursion.

1. Introduction

A desirable property of any real-time system is the guarantee that it will perform at least beyond some pre-specified thresholds defined by system designers. This is usually not a concern under normal situations where analysis has been done offline to ensure system performance based on the regular workload. However, in response to an event such as user's input or changing environment, the load of the system may dynamically change in such a way that a temporal overload condition occurs. The challenge, then, is to provide some mechanism to guarantee the minimum performance level under such circumstances.

Many real-time task models have been proposed to extend timing requirements beyond the hard and soft deadlines based on the observation that jobs can be dropped without severely affecting performance ([4], [17]). For example, Ramanathan et al. proposed both online [15] and offline [24] scheduling algorithms that are based on the (m,k) model, which is analyzed in [16]. In this model, up to $k - m$ consecutive jobs are allowed to be dropped in any sliding window of k. Moreover, [28] presented the Dynamic Window-Constrained Scheduling (DWCS) algorithm, which is similar except that the window k is fixed. Further enhancement to these successful models can be found in [23] and [22]. Other frameworks such as the imprecise computation model [12] and reward-based model [2] can be used to capture situations where the quality of service is proportional to the amount of workload completed.

Despite the success of the abovementioned models in alleviating overload situations, it is sometimes more suitable to execute jobs less often instead of dropping them or allocating fewer cycles. For example, limitations on the throughput capacity of ad hoc communication networks [1] make it highly desirable to reduce overall network traffic by having control tasks adaptively adjust their periods in response to the actual activity level of the control application.

The work in [19] was among the first to address this type of requirements. Seto, et al. considered the problem of finding a feasible set of task periods as a non-linear programming problem which seeks to optimize a specific form of control performance measure [25]. In [26], finding all feasible periods of a given set of tasks was studied for the Rate Monotone (RM) scheduling algorithm. Cervin et al. used optimization theory to solve the period selection problem online by adaptively adjusting task periods while optimizing a specific form of control performance [11]. Recently, [5] offered an optimal search algorithm that solves the period selection problem for fixed-priority scheduling schemes. The algorithm may be applicable only during the design phase due to its potentially high complexity. Another interesting framework was introduced in [18] where task periods are adjusted in response to varying computation times.

Buttazzo and his colleagues proposed an elegant yet

*This work is supported in part by NSF under grant number CNS-0410771.

more flexible framework known as the elastic task model [8] where deadline misses are avoided by increasing tasks periods. The work in [10] extends the basic elastic task model to handle cases where the computation time is unknown, [9] incorporated a mechanism to handle resource constraints within the elastic framework, and [7] provided a means to smoothly adjust task execution rate. In addition, [13] uses a control performance metric as a cost function to find an optimal sampling interval for each task.

This paper generalizes the existing elastic scheduling approach in several directions. First, we re-examine the problem of period determination in the elastic task model and show that the task compression algorithm in [9] whose precise nature was not made clear in the original work in fact solves a quadratic programming (QP) problem. The QP problem seeks to minimize the sum of the squared deviation of every task's utilization from its initial utilization, equivalent to minimizing the task set's overall *energy*. Identifying the nature of the optimization problem underlying the task compression algorithm is important in several aspects. For instance, it may suggest other relevant optimization objectives and shed light on determining task periods in the presence of uncertainty for other task models.

Second, we select an alternative objective function, i.e., minimizing the average difference of task periods from their desired minimum periods, as the sampling period is directly related to the performance of some controllers. We show that the solution to this optimization problem can be evaluated with a closed-form formula.

Lastly, the proposed framework is extended to treat cases where task deadlines are less than task periods. Such requirements often arise in control systems to reduce jitters, for example. Moreover, in some situations, it is desirable that tasks finish executing sooner, even if their periods are not up. We formulate the problem of period determination as a constrained optimization problem and propose a heuristic approach based on the the task compression algorithm in [9] to solve the problem. The heuristic is guaranteed to find a feasible solution, if one exists. It is quite efficient and is hence suitable for online period adjustment. Experimental results show that the heuristic is able to find a solution that is close to the global optimal solution.

The remainder of the paper is organized as follows. We begin by reviewing key background materials in Section 2. Section 3 presents the solutions to the period determination problem for tasks with deadlines equal to periods. The optimization approach is then extended in Section 4 to treat the case where task deadlines are less than task periods. Experimental results are presented and discussed in Section 5. Finally, the paper concludes with Section 6.

2. Preliminaries

This section describes the system under consideration as well as important assumptions made throughout the paper. We also briefly review the task compression algorithm used for period selection [9].

2.1. System model

We consider the system where each task τ_i is periodic and is characterized by the following 6-tuple: $(C_i, D_i, T_i, T_{i_{min}}, T_{i_{max}}, e_i)$, for $i = 1, \ldots, N$, where N is the number of tasks in the system, C_i is the worst case execution time of τ_i, D_i is its deadline, and T_i is τ_i's initial period. Furthermore, $T_{i_{min}}$ denotes the most desirable period of τ_i, as specified by control system designers, whereas $T_{i_{max}}$ represents the maximum time interval between two executions of a task that will prevent system performance from falling below some desirable thresholds (e.g., a control system becomes unstable). The elastic coefficient, e_i, represents the resistance of task τ_i to increasing its period in face of changes. The smaller the elastic coefficient of a task, the harder it is to increase that task's period.

Task deadlines are first assumed to be equal to task periods. This requirement will be relaxed in Section 4. All task attributes are real values and are assumed to be known *a priori*. The current utilization of τ_i is $U_i = \frac{C_i}{T_i}$. Similarly, the minimum and maximum utilizations of τ_i is $U_{i_{min}} = \frac{C_i}{T_{i_{max}}}$ and $U_{i_{max}} = \frac{C_i}{T_{i_{min}}}$, respectively.

2.2. Elastic task model

In [9], Buttazzo, et. al. modeled a task as a spring system, where increasing or decreasing a task period is analogous to compressing or decompressing a spring. The elastic coefficient, e_i, introduced above hence has its intuitive meaning of the hardness of the spring. The purpose of increasing task periods is to drive the total utilization of the system down to some desired utilization level, U_d, analogous to a spring system trying to minimize its energy under an external force.

The attractiveness of the elastic task model is its accompanying task compression algorithm, which is quite efficient $\left(O(N^2)\right)$ and can readily be used online. (In fact, the elastic task model and the task compression algorithm have already been implemented in the S.Ha.R.K. kernel [14].) The task compression algorihtm works as follows. If it is possible to drive the system utilization down to U_d without violating any period bounds, the algorithm will return a set of feasible periods $(T_1, T_2, \ldots, T_N)$ that can be used by the system. Tasks whose periods are fixed (if $e_i = 0$ or $T_{i_{min}} = T_{i_{max}}$) are considered inelastic and are treated as special cases. The amount of utilization that each remaining, non-inelastic task should receive is computed based on its elastic coefficient, initial period, and the amount of utilization that must be reduced to achieve U_d. The resultant period of a task τ_i is guaranteed to fall somewhere between $T_{i_{min}}$ and $T_{i_{max}}$.

Throughout this paper, we will assume that the EDF (Earliest Deadline First) scheduling algorithm [20] is used. Furthermore, we will focus our attention on cases where tasks need to either increase or decrease its utilization in response to either internal (e.g., change in sampling rate of one or more tasks in the system) or external (e.g., network traffic) factors.

3. General period selection

Given a particular set of real-time tasks, there may exist numerous sets of feasible periods. It is not difficult to see that different sets of periods would lead to different performance of the resultant system. In general, the period selection problem can be expressed as an optimization problem. That is,

```
optimize:  performance metric
s.t.:      tasks are schedulable
           period bounds are satisfied
```

Below we introduce two specific performance metrics and discuss their implications. We assume that tasks deadlines equal task periods.

3.1. Minimize utilization perturbation

Processor utilization by each task is an important measure for any real-time system. It not only reveals the amount of system resource dedicated to the task but also impacts schedulability. In the elastic task model, one consequence of changing task periods is changing the utilization of tasks. From the stand point of performance preservation, it is desirable to minimize the changes in task utilization. This objective can be captured by the following constrained optimization problem.

$$\text{min:} \quad E(U_1, \cdots, U_N) = \sum_{i=1}^{N} w_i \left(U_{i0} - U_i\right)^2 \quad (1)$$

$$\text{s.t.:} \quad \sum_{i=1}^{N} U_i \leq U_d \quad (2)$$

$$U_i \geq U_{i_{min}} \quad \text{for } i = 1, 2, \cdots, N \quad (3)$$

$$U_i \leq U_{i0} \quad \text{for } i = 1, 2, \cdots, N \quad (4)$$

In the formulation, N is the number of tasks in the system, U_{i0} is the initial utilization of task τ_i and $U_{i0} \geq U_{i_{min}}$, U_i is the utilization of τ_i to be determined, and U_d is the desired total utilization. (U_d is usually set to 1 for EDF scheduling.) Constant w_i (≥ 0) is a weighting factor and reflects the criticality of a task. More critical tasks would have larger w_i's. The first constraint simply states the schedulability condition under EDF. The rest of the constraints bounds the utilization, equivalently bounds the task period by $T_{i_{min}}$ and $T_{i_{max}}$ where $T_{i_{min}} = C_i/U_{i_{max}}$ and $T_{i_{max}} = C_i/U_{i_{min}}$.

Note that for $w_i = 0$, (1) does not change regardless of what U_i value is used. To help satisfying (2), it is natural to simply use $U_i = U_{i_{min}}$. Hence, for the rest of the paper, we will focus on the case where $w_i > 0$ for all $1 \leq i \leq N$.

The problem in (1)–(4) belongs to the category of quadratic programs and can be solved in polynomial time. However, solving such a problem during runtime can be too costly. What makes the above formulation attractive is that its solution is exactly the same as that found by the task compression algorithm in [9]. We introduce a lemma and a theorems to support this argument.

Lemma 1 *Given the constrained optimization problem as specified in (1)–(4) and $\sum_{i=1}^{N} U_{i0} > U_d$, any solution, U_i^*, to the problem must satisfy $\sum_{i=1}^{N} U_i^* = U_d$ and $U_i^* \neq U_{i0}$, for $i = 1, \ldots, N$.*

Proof: We prove the lemma by utilizing the Karush-Kuhn-Tucker (KKT) necessary conditions for the solution to the given problem, which can be written in terms of the Lagrangian function for the problem as

$$J_a(\mathbf{U}, \mu) = \sum_{i=1}^{N} w_i \left(U_{i0} - U_i\right)^2 + \mu_0 \left(\sum_{i=1}^{N} U_i - U_d\right) + \sum_{i=1}^{N} \mu_i \left(U_{i_{min}} - U_i\right) + \sum_{i=1}^{N} \lambda_i \left(U_i - U_{i0}\right) \quad (5)$$

where μ_0, μ_i, and λ_i are Lagrange multipliers, $\mu_0 \geq 0$, $\mu_i \geq 0$, and $\lambda_i \geq 0$, for $i = 1, \ldots, N$. The necessary conditions for the existence of a relative minimum at U_i^* are, for all $i = 1, \ldots, N$,

$$0 = \frac{\partial J_a}{\partial U_i^*} = -2w_i \left(U_{i0} - U_i^*\right) + \mu_0 - \mu_i + \lambda_i \quad (6)$$

$$0 = \mu_0 \left(\sum_{i=1}^{N} U_i^* - U_d\right) \quad (7)$$

$$0 = \mu_i \left(U_{i_{min}} - U_i^*\right) \quad (8)$$

$$0 = \lambda_i \left(U_i^* - U_{i0}\right) \quad (9)$$

Assume that (2) is inactive, i.e., $\mu_0 = 0$ and $\sum_{i=1}^{N} U_i^* < U_d$. Then at least one constraint in (3) or (4) must be active. Suppose the k-th constraint in (3) is active. That is, $U_k^* =$

$U_{k_{min}}$ and $\mu_k \geq 0$. Then, the k-th constraint in (4) must be inactive, i.e., $\lambda_k = 0$. From (6), we obtain

$$\mu_k = -2w_k(U_{k0} - U_{k_{min}}) < 0 \qquad (10)$$

which contradicts the assumption that $\mu_k \geq 0$. Therefore, if any $U_k^* = U_{k_{min}}$, constraint (2) must be active.

Now assume that some constraints in (4) are active while others are inactive. Suppose $U_h^* = U_{h0}$ (active) and $U_{k_{min}} < U_k < U_{k0}$ (inactive). Then $\mu_h = 0, \lambda_h \geq 0$, and $\mu_k = \lambda_k = 0$. From (6), we have

$$\mu_0 = 2w_h(U_{h0} - U_h^*) + \mu_h - \lambda_h = -\lambda_h \qquad (11)$$

$$\mu_0 = 2w_k(U_{k0} - U_k^*) \qquad (12)$$

Note that (11) and (12) cannot be simultaneously satisfied. Therefore, we can either have all the constraints in (4) be active or all are inactive. If all the constraints in (4) are active, we have $\sum_{i=1}^N U_i^* = \sum_{i=1}^N U_{i0} > U_d$ which constradicts the initial assumption. If all the constraints in (4) are inactive, (12) requires that $\mu_0 > 0$, which contradicts the assumption that constraint (2) is inactive. Therefore, for any solution to the optimization problem, constraint (2) must be active, i.e., $\sum_{i=1}^N U_i^* = U_d$. □

Theorem 1 *Given the constrained optimization problem as specified in (1)–(4),* $\sum_{i=1}^N U_{i0} > U_d$*, and* $\sum_{i=1}^N U_{i_{min}} \leq U_d$*, let* $\widehat{U} = \sum_{U_i^* \neq U_{i_{min}}} U_{i0} + \sum_{U_i^* = U_{i_{min}}} U_{i_{min}}$*. A solution to the problem,* U_i^**, is optimal if and only if*

$$U_i^* = U_{i0} - \frac{\frac{1}{w_i}\left(\widehat{U} - U_d\right)}{\sum_{U_j^* \neq U_{j_{min}}}(1/w_j)} \qquad (13)$$

for $U_i^* > U_{i_{min}}$ *and* $U_i^* = U_{i_{min}}$ *otherwise, and* $\widehat{U} > U_d$*.*

Proof: Consider the KKT conditions given in (6)–(9). From Lemma 1, we know that any solution to the given optimization problem must satisfy (2), i.e., $U_d = \sum_{i=1}^N U_i^*$, and $U_i^* \neq U_{i0}$. Hence, we only need to consider the case where $\lambda_i = 0$, for all $i = 1, \ldots, N$, Suppose that the k-th constraint in (3) is active. We have $U_k^* = U_{k_{min}}$, and

$$\mu_k = \mu_0 - 2w_k\left(U_{k0} - U_{k_{min}}\right), \qquad (14)$$

Otherwise, we have $\mu_k = 0$. By summing up (6) for all i and using the conclusions above,

$$\mu_0 = \frac{2\left(\widehat{U} - U_d\right)}{\sum_{U_i^* \neq U_{i_{min}}}(1/w_i)}. \qquad (15)$$

As long as $\widehat{U} > U_d$, $\mu_0 > 0$, $\mu_i \geq 0$, and constraints in (4) are satisfied. Therefore, a solution, U_i^*, to the optimization problem either satisfies $U_i^* = U_{i_{min}}$ or can be obtained by combining (15) with (6) for $U_i^* > U_{i_{min}}$. (Note that $\mu_i = 0$ when $U_i^* > U_{i_{min}}$.) That is,

$$U_i^* = U_{i0} - \frac{\frac{1}{w_i}\left(\widehat{U} - U_d\right)}{\sum_{U_j^* \neq U_{j_{min}}}(1/w_j)} \qquad (16)$$

Additionally, since the objective function and the inequality constraints in (1)–(4) are convex, the necessary conditions for optimality provided by the KKT conditions also become the sufficient conditions for optimality [21]. Hence, the solution found in Theorem 1 is a global minimum. □

Corollary 1 *Consider a set of N tasks where U_i is the utilization of the ith task. Let U_{i0} denote the initial desired utilization of task τ_i and let $e_i > 0$ be a set of elastic coefficients for $i = 1, \ldots, N$. Let U_d be the desired utilization level and $\sum_{i=1}^N U_{i0} > U_d$. The task utilizations U_i, for $i = 1, \ldots, N$ obtained from the task compression algorithm in [9] minimizes*

$$E(U_1, \ldots, U_N) = \sum_{i=1}^N \frac{1}{e_i}(U_{i0} - U_i)^2$$

subject to the inequality constraints $\sum_{i=1}^N U_i \leq U_d$*,* $U_i \geq U_{i_{min}}$*, and* $U_i \leq U_{i0}$*, for* $i = 1, \ldots, N$*.*

The above corollary has several significant consequences, as it reveals the optimization criterion inherent in the task compression algorithm and illustrates that the task compression algorithm can be used to solve certain convex programming problems.

3.2. Minimize task periods

For some controllers, instead of focusing on utilization, it may be more useful to examine task periods directly, as the sampling period is monotonically decreasing faster than the controller performance.

If the range of possible period values for each task is unbounded, the following constrained optimization problem can be used to minimize the changes in task periods.

$$\text{min:} \qquad J(T_1, \cdots, T_N) = \sum_{i=1}^N w_i(T_i - T_{i0}) \qquad (17)$$

$$\text{s.t.:} \qquad \sum_{i=1}^N \frac{C_i}{T_i} \leq 1 \qquad (18)$$

The constraint in equation (18) is simply the condition assuring that the task set is schedulable under EDF. Let T_i^* denote a locally optimal set of task periods for the above optimization problem. The following theorem provides a closed-form formula to compute the solution to this minimization problem.

Theorem 2 *Given the constrained optimization problem specified in equations (17)–(18), a locally optimal solution is*

$$T_i^* = \sqrt{\frac{C_i}{w_i}} \sum_{k=1}^{N} \sqrt{w_k C_k} \tag{19}$$

Proof: This theorem is proved through a straightforward application of the Kuhn-Tucker necessary conditions.

□

Remark: If we require that $T_i \geq T_{i_{min}}$ (in other words, there is a lower bound on the desired period), the constraint will be satisfied if the choice of w_i satisfies the following inequality:

$$\sqrt{w_i} T_{i_{min}} \leq \sqrt{C_i} \sum_{k=1}^{N} \sqrt{w_k} \sqrt{C_k} \tag{20}$$

The above expression is obtained by simply letting (19) be greater than or equal to $T_{i_{min}}$. The significance of (20) is that it can be used to identify a set of admissible weighting factors whose selection ensures $T_i \geq T_{i_{min}}$. Indirectly, (20) also ensures that the resultant task periods will be greater than zero.

4. Period selection with additional deadline constraints

In this section, we consider the case where task deadlines are less than task periods. This more general model is useful, as there are situations where it is desirable for a task to finish executing early (before its period ends). We again formulate the period selection problem as a constrained optimization problem and propose a novel heuristic based on the task compression algorithm. The algorithm is guaranteed to find a local optimal solution to the problem, if one exists, and is efficient enough for online use.

4.1. Simplify feasibility condition

Baruah et al. considered the case where task deadlines are less than or equal to task periods and derived a sufficient and necessary condition for EDF schedulability [3], which is later improved in [6]. The condition is restated in the following theorem.

Theorem 3 [3] *Given a periodic task set with $D_i \leq T_i$, the task set is schedulable if and only if the following constraint is satisfied $\forall L \in \{kT_i + D_i \leq \min(B_p, H)\}$ and $k \in \mathcal{N}$ (the set of natural numbers including 0), where B_p and H denote the busy period and hyperperiod, respectively,*

$$L \geq \sum_{i=1}^{N} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \tag{21}$$

Based on Theorem 3, the period determination problem can be formulated as follows:

$$\text{min:} \quad E(U_1, \cdots, U_N) = \sum_{i=1}^{N} w_i \left(U_{i0} - U_i\right)^2 \tag{22}$$

$$\text{s.t.:} \quad L \geq \sum_{i=1}^{N} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \tag{23}$$

$$L \in \{kT_i + D_i \leq \min(B_p, H)\}, k \in \mathcal{N} \tag{24}$$

$$U_i \geq U_{i_{min}} \quad \text{for } i = 1, 2, \cdots, N \tag{25}$$

$$U_i \leq U_{i0} \quad \text{for } i = 1, 2, \cdots, N \tag{26}$$

Solving the above constrained optimization problem can be extremely time consuming. Hence, we investigate solving the problem approximately with an efficient algorithm below. An approximate solution is both acceptable and preferred, as a rapid response allows the system to degrade gracefully instead of going into catastropic states caused by some dynamic perturbations. Since verifying the constraint in (21) for all L values is the main source of high complexity, we consider simplifying the feasibility test by using the following stronger schedulability condition,

$$L \quad \geq \quad \sum_{i=1}^{N} \left(\frac{L - D_i}{T_i} + 1 \right) C_i \tag{27}$$

It is not difficult to see that if the inequality in (27) is satisfied then the original inequality in (21) must also be satisfied. What makes (27) an excellent candidate for online use is that the schedulability of a task set can be determined based on a single L value, L^*. Below, we introduce several lemmas and a theorem to support this claim.

For simplicity, we denote the set of all possible values of L by a distinct ordered set $\mathcal{L} = \{L_0, L_1, \ldots\}$ where $L = kT_i + D_i, \forall k \in \mathcal{N}$ and $L \leq \min(B_p, H)$.

Lemma 2 *Given a set Γ of N tasks with $D_i \leq T_i$, let L_j and $L_{j+1} \in \mathcal{L}$ and let $L_j < L_{j+1}$. If the constraint in (27) is satisfied for L_j, then it is satisfied for L_{j+1}.*

Proof: By regrouping the terms in (27), we can rewrite the inequality as follows.

$$L \quad \geq \quad \frac{\sum_{i=1}^{N} C_i - \sum_{i=1}^{N} U_i D_i}{1 - \sum_{i=1}^{N} U_i} \tag{28}$$

Given that L_j satisfies the constraint in (28) and $L_{j+1} > L_j$, it immediately follows that L_{j+1} satisfies the constraint in (28). □

Based on the above lemma, we can conclude that if the constraint in (27) is satisfied for L_j, then it is also satisfied for all $L_k \in \mathcal{L}$, where $L_k > L_j$. It may then seem natural to simply set L^* to be the minimum of all L values in $\mathcal{L}$.

However, such a choice can be extremely pessimistic, often resulting in finding no feasible solutions to the problem. To avoid being too pessimistic, we introduce the next lemma, which identifies useful necessary conditions for any feasible task set. The lemma helps to eliminate pessimistic choices of L^*.

Lemma 3 *Let D_i be the deadline of task τ_i in a given task set Γ. Further, let the tasks in Γ be ordered in a non-decreasing order of deadlines and suppose that D_{min} is unique. Regardless of the choices of periods, any task set that is schedulable must satisfy the following property:*

$$\sum_{i=1}^{j} C_i \quad \leq \quad D_j, \forall j = 1, \ldots, N \tag{29}$$

Proof: We can prove (29) by contradiction. Suppose the task set is schedulable and $\sum_{i=1}^{j} C_i > D_j$ for some j. By D_j, an instance of $\tau_1, \ldots, \tau_j$ must each finish executing. Moreover, The total processor demand at D_j is at least $\sum_{i=1}^{j} C_i$ time units. However, since there are only D_j time units available and $\sum_{i=1}^{j} C_i > D_j$, at least one instance of a task must miss its deadline. This is a contradiction to the assumption that the task set is schedulable. Hence, (29) must be true for all $j = 1, \ldots, N$.

□

We are now ready to introduce two lemmas which form the basis for our selection of L^*.

Lemma 4 *Consider a set Γ of N tasks that satisfy the condition in Lemma 3. Let the tasks in Γ be sorted in a non-decreasing order of deadlines. If $D_1 + T_1 \leq D_2$, and $L^* = D_2$ satisfies the inequality constraint in (27), then the task set is guaranteed to be schedulable.*

Proof: Let $L_h = L^* = D_2$. By Lemma 2, any $L_j \in \mathcal{L}$ with $j > h$ satisfies constraint in (27) and hence satisfies (21). Now consider $j < h$. Since $D_i \geq D_2$ for $i > 2$, L_j can only be equal to $D_1 + kT_1$ for some $k \in \mathcal{N}$. In order for L_j to satisfy (21), noting that $|D_1 + kT_1 - D_i| < T_i$ for $i \geq 2$, we need $D_1 + kT_1 \geq (k+1) \cdot C_1$, which holds true according to Lemma 3. Therefore, for all values of $L \in \mathcal{L}$, (21) is satisfied. □

Lemma 5 *Consider a set Γ of N tasks that satisfy the condition in Lemma 3. Let the tasks in Γ be sorted in a non-decreasing order of deadlines. If $D_1 + T_1 > D_2$, and $L^* = \min_{i=1}^{N}(T_i + D_i)$ satisfies the inequality constraint in (27), then the task set is guaranteed to be schedulable.*

Proof: Let $L_h = L^* = min_{i=1}^{N} (T_i + D_i)$. By Lemma 2, any $L_j \in \mathcal{L}$ with $j > h$ satisfies constraint in (27) and hence satisfies (21). Now consider $j < h$. L_j can only be equal to D_k, $k = 1, \ldots, N$, such that $D_k < L_h$. In order for L_j to satisfy (21), we need $\sum_{i=1}^{j} C_i \leq D_j$, which holds true according to Lemma 3. Therefore, for all values of $L \in \mathcal{L}$, (21) is satisfied.

□

Based on Lemmas 4 and 5, we have a new schedulability test which is stated in the following theorem.

Theorem 4 *Consider a set Γ of N tasks that satisfy the condition in Lemma 3. Let the tasks in Γ be sorted in a non-decreasing order of deadline. A given task set is schedulable if*

$$L^* \quad \geq \quad \sum_{i=1}^{N} \left(\frac{L^* - D_i}{T_i} + 1 \right) C_i \tag{30}$$

where

$$L^* = \begin{cases} D_2 & : \ D_1 + T_1 \leq D_2 \\ \min_{i=1}^{N} (T_i + D_i) & : \ otherwise \end{cases}$$

Proof: From Lemmas 4 and 5, we know that the only L that needs to be checked against the constraint in (27) is D_2 if $D_1 + kT_1 \leq D_2$, k = 0, 1, ..., and $\min_{i=1}^{N}(T_i + D_i)$ otherwise. Moreover, we have proved in Lemma 2 that if (27) is satisfied for some L_j then it is also satisfied for L_{j+1} where $L_j < L_{j+1}$. This, in turns, implies that if (27) is satisfied for L_j then it is also satisfied for all $L_k \in L$, where $L_k > L_j$. Taken together, if the constraint in (27) is satisfied for L^* then the task set must be schedulable.

□

The above theorem paves the way to finding a simpler constrained optimization problem formulation for the purpose of period determination. We present the actual problem formulation in the following subsection.

4.2. Minimize utilization perturbation with deadline constraints

By using Theorem 4, we can express the period determination problem where task deadlines are less than task periods as a constrained optimization problem similar to that in (1)–(4). Letting $r_i = L - D_i$, (27) can be rewritten as

$$\sum_{i=1}^{N} r_i U_i \leq L - \sum_{i=1}^{N} C_i. \tag{31}$$

Then the period determination problem where task deadlines are less than task periods can be formulated as

$$\text{min:} \qquad E(U_1, \cdots, U_N) = \sum_{i=1}^{N} w_i \left(U_{i0} - U_i\right)^2 \tag{32}$$

$$\text{s.t.:} \quad \sum_{i=1}^{N} r_i U_i \leq L - \sum_{i=1}^{N} C_i \tag{33}$$

$$L = \begin{cases} D_2 & : \ D_1 + \frac{C_1}{U_1} \leq D_2 \\ \min(\frac{C_i}{U_i} + D_i) & : \ otherwise \end{cases} \tag{34}$$

$$U_i \geq U_{i_{min}} \quad \text{for } i = 1, 2, \cdots, N \tag{35}$$

$$U_i \leq U_{i0} \quad \text{for } i = 1, 2, \cdots, N \tag{36}$$

Note that the above constrained optimization problem would have exactly the same format as the QP problem in (1)–(4) if L and r_i can be treated as constants.

Solving the above optimization problem with a general optimization software can still be rather inefficient in terms of both processor time and memory usage. Consequently, we leverage the task compression algorithm to tackle the challenge of solving the problem efficiently.

Consider the case where $D_1 + \frac{C_1}{U_1} \leq D_2$. According to Lemma 4, we only need to check $L^* = D_2$ for schedulability, which indeed leads to a constant L value in (33). It follows that we can solve the optimization problem efficiently by using the following theorem.

Theorem 5 *Given the constrained optimization problem as specified in (32)–(36), for $L = D_2$, $\sum_{i=1}^{N} r_i U_{i0} > L - \sum_{i=1}^{N} C_i$, and $U_{1_{min}} \leq U_1^* < U_{10}$, a solution, U_i^*, is optimal if and only if*

$$U_i^* = \begin{cases} \frac{D_2 - \sum_{j=1}^{N} C_j - \sum_{j=3}^{N} r_j U_{j0}}{D_2 - D_1} & : \ i = 1 \\ U_{i0} & : \ otherwise \end{cases}$$

for $D_2 > \sum_{j=1}^{N} C_j + \sum_{j=3}^{N} r_j U_{j0}$.

Proof: Let $L_d = L - \sum_{i=1}^{N} C_i$. The KKT conditions for the solution to the optimization problem in (32)–(36) can be written as follows:

$$0 = -2w_i \left(U_{i0} - U_i^*\right) + r_i \mu_0 - \mu_i + \lambda_i \tag{37}$$

$$0 = \mu_0 \left(\sum_{j=1}^{N} r_j U_j^* - L_d \right) \tag{38}$$

$$0 = \mu_i \left(U_{i_{min}} - U_i^*\right) \tag{39}$$

$$0 = \lambda_i \left(U_i^* - U_{i0}\right) \tag{40}$$

for $i = 1, \cdots, N$, where μ_0, μ_i's and λ_i's are Lagrange multipliers, $\mu_0 \geq 0$, $\mu_i \geq 0$, and $\lambda_i \geq 0$ for $i = 1, \cdots, N$.

Consider first those tasks with $D_k = D_2$. Then $r_k = L - D_k = 0$. Now (37) reduces to

$$\mu_k - \lambda_k = -2w_k \left(U_{k0} - U_k^*\right) \tag{41}$$

Assume that $U_{k_{min}} < U_k^* < U_{k0}$. In order to satisfy (39) and (40), we must have $\mu_k = \lambda_k = 0$, which contradicts (41). Now assume that $U_k^* = U_{k_{min}}$. Then to satisfy (40), we need $\lambda_k = 0$. However, this leads to $\mu_k < 0$ from (41), which violates the KKT conditions. Therefore, for those tasks with $D_k = D_2$, $U_k^* = U_{k0}$. (It can be readily proved that such a solution indeed satisfies the KKT conditions.)

Similarly, consider next those tasks with $D_h > D_2$. Then, $r_h = L - D_h < 0$. Now (37) becomes

$$-2w_h \left(U_{h0} - U_h^*\right) = \mu_h - \lambda_h - r_h \mu_0 \tag{42}$$

In order to satisfy (39) and (40), we must have $\mu_h = \lambda_h = 0$, which will cause (41) to become

$$-2w_h(U_{h0} - U_h^*) = D_h - D_2. \tag{43}$$

This is clearly a contradiction, since $D_h > D_2$ and $U_{h0} - U_h^* \geq 0$. Now, assume that $U_h^* = U_{h_{min}}$. Then, to satisfy (40), we need $\lambda_h = 0$. However, this leads to $\mu_h < 0$ in (42), which violates the KKT conditions. Therefore, for any task with $D_h > D_2$, $U_h^* = U_{h0}$.

For $i = 1$, since $r_1 > 0$, it can be readily shown that there exist $\mu_0 \geq 0$, $\mu_1 \geq 0$, and $\lambda_1 \geq 0$ that satisfy (37)–(40) if $U_{1_{min}} \leq U_1^* < U_{10}$ (otherwise, no feasible solution exists). By replacing $U_i^* = U_{i0}$ for $i \geq 2$ in (36), we obtain the value of U_1^* exactly as defined in Theorem 5. (It can be proved that (33) must be active using similar method as in Lemma 1).

We have shown that the values of U_i^* as defined in Theorem 5 satisfy the KKT conditions and form a feasible solution to the problem under consideration. Since the constrained optimization problem is convex, it follows that this feasible solution is also an optimal one [21]. $\square$

The above theorem immediately leads to an efficient algorithm to solve the optimization problem in (32)–(36) when $D_1 + T_1 \leq D_2$. Moreover, Theorem 5 also implies that if $U_1 < U_{1_{min}}$ then the task set is infeasible.

Let us now consider the case where $D_1 + T_1 > D_2$. According to Lemma 5, one needs to check whether $L^* = \min_{i=1}^{N} (T_i + D_i)$ satisfies (27) to determine feasibility. Since the value of L^* may change as T_i changes, the constraint in (33) is no longer linear and can greatly increase the complexity of the optimization problem. Since our aim is to have an efficient online algorithm, we propose a heuristic to solve the problem. The following theorem forms the basis for our heuristic approach.

Theorem 6 *Given the constrained optimization problem as specified in (32)–(36), for a fixed value of L (where $L = \min\{\frac{C_i}{U_i} + D_i\}, \forall i = 1, \ldots, N$) and $\sum_{i=1}^{N} r_i U_{i0} > L - \sum_{i=1}^{N} C_i$, let*

$$R = \sum_{U_j^* \neq U_{j_{min}}} - \sum_{U_j^* = U_{j0}} \frac{r_j^2}{w_j}$$

$$\overline{V} = \sum_{U_j^* \neq U_{j_{min}}} r_j U_{j0} - (L - \sum_{i=1}^{N} C_j) + \sum_{U_j^* = U_{j_{min}}} r_j U_j$$

a solution, U_i^*, is optimal if and only if

$$U_i^* = U_{i0} - \frac{r_i}{w_i R} \cdot \overline{V} \quad (44)$$

for $r_i > 0$ and $U_i^* = U_{i0}$, for $r_i \leq 0$, and $0 < \frac{\overline{V}}{R} \leq \frac{w_i}{r_i}(U_{i0} - U_{i_{min}})$.

Proof: Consider the KKT conditions given in (37)–(40). Using a method similar to that in Lemma 1, it can readily be proved that the constraint in (33) must be active. Hence, we know that any solution to the given optimization problem must satisfy (33), i.e., $L_d = L - \sum_{i=1}^{N} C_i = \sum_{i=1}^{N} r_i U_i$.

Consider the case where $r_k \leq 0$. Assume that both μ_k and λ_k are inactive (i.e., $U_{k0} < U_k^* < U_{k_{min}}$). Then, (37) becomes

$$2w_k(U_{k0} - U_k^*) = r_k \mu_0. \quad (45)$$

However, since $U_k^* < U_{k0}$ and $r_k \leq 0$, the above equation is a contradiction.

Now, assume that $\mu_k > 0$ but $\lambda_k = 0$. Then (37) gives

$$2w_k(U_{k0} - U_k^*) + \mu_k = r_k \mu_0, \quad (46)$$

which contradicts the assumption that $\mu_k > 0$. Hence, for r_k must be greater than 0 and, consequently, $U_k^* = U_{k0}$.

For the case where $r_k > 0$, we proceed as follows. Suppose that the k-th constraint in (39) is active. That is, $U_k^* = U_{k_{min}}$, $\mu_k > 0$, and $\lambda_k = 0$. Then, from (37),

$$\mu_k = r_k \mu_0 - 2w_k(U_{k0} - U_{k_{min}}) \quad (47)$$

and $\mu_k = 0$ otherwise. Similarly, if the k-th constraint in (40) is active, then $U_k^* = U_{k0}$, $\lambda_k > 0$, $\mu_k = 0$ and

$$\lambda_k = r_k \mu_0 \quad (48)$$

and $\lambda_k = 0$ otherwise. Multiplying (37) by r_i, summing it up for all i, and using the conclusions above, we have

$$\mu_0 = \frac{2\overline{V}}{R} \quad (49)$$

as long as $0 < \frac{\overline{V}}{R} \leq \frac{w_i}{r_i}(U_{i0} - U_{i_{min}})$. Therefore, a solution to the optimization problem either satisfies $U_i^* = U_{i0}$ for $r_i \leq 0$ or can be obtained by combining (49) with (37) for $U_{i_{min}} < U_i^* < U_{i0}$ (and $\mu_i = \lambda_i = 0$). That is,

$$U_i^* = U_{i0} - \frac{r_i \overline{V}}{w_i R} \quad (50)$$

□

Our proposed algorithm aims to find a feasible solution to the optimization problem defined in (22)–(25) in an efficient manner. The algorithm adopts an iterative approach. During iteration h, a set of periods $T_i(h)$ is found and the algorithm checks to see whether the constraint in (27) is satisfied. If this is the case and if $T_i(h)$ also minimize the objective function till now, then the algorithm keeps $T_i(h)$ as the current best solution to the problem. The iterative process will terminate when certain stopping criterion is met (to be discussed later). The solution thus found may not be optimal but it is guaranteed to be schedulable by the EDF policy. The detailed algorithm is given in Figure 1.

In each iteration h, we fix the value of L as either $L(h) = D_2$ if $T_1(h-1) + D_1 \leq D_2$ or $L(h) = \min_{i=1}^{N}(T_i(h-1) + D_i)$ otherwise. For any task τ_i whose $r_i = L(h) - D_i \leq 0$, its period is immediately set to T_{i0}. For any task τ_i whose $r_i > 0$, its utilization, $U_i(h)$, can be determined using Theorem 5 or Theorem 6, respectively. In the case of $L(h) = \min_{i=1}^{N}(T_i(h-1) + D_i)$, as shown in Figure 1, U_i is obtained by using a slightly modified task compression algorithm. (To save space, we omit the modified task compression algorithm.) The following modifications were made to the original task compression algorithm: (i) the inputs to the task compression algorithm are task set Γ and $L(h)$, instead of Γ and U_d, and (ii) the equation $U_i = U_{i0} - (U_{v0} - U_d + U_f)\, E_i/E_v$ in the original algorithm is replaced by (44). For the case where $L(h) = D_2$, Theorem 5 is applied straightforwardly.

To determine convergence, a user-defined parameter, Δ, is included as a stopping criterion; if the difference between U_i found in the current iteration and U_i found in the last iteration is smaller than Δ for all i, the algorithm terminates and returns the best set of periods it has encountered. To handle the case where task periods do not converge to some fixed values (or when it may take too long for the solution to converge), the algorithm uses another user-defined parameter, *maxIter*, to limit the maximum number of iterations.

An additional challenge is how to assign the initial value of L. We propose to set the initial value of L to $\min_{i=1}^{N}(T_{i_{max}} + D_i)$. In this way, if the task set is found to be infeasible, then the algorithm immediately exits since the task set cannot be made schedulable without violating the given period bounds. The following lemma serves to support our choice of the initial value as well as the iterative approach.

Lemma 6 *Let T_i for $1 \leq i \leq N$ be a set of periods that satisfies the constraint in (21). Then the set of $T_i' \geq T_i$ also satisfy the constraint in (21).*

Proof: The left-hand side of (21) can be thought of as representing the amount of available processing power during the interval $[0, L]$, whereas the right-hand side of (21) represents the processor demand during that same inter-

Algorithm Task_compress_deadline($\Gamma, \Delta, maxIter$) {
 $sumC = 0$;
 for each ($\tau_i \in \Gamma$) {
  $sumC = sumC + C_i$;
  if ($sumC > D_i$)
   return NULL; // no feasible solution exists
 }

 $bestObjF = \infty$;
 for each ($\tau_i \in \Gamma$) {
  $prevT_i = T_{i_{min}}$;
  $currT_i = T_{i_{max}}$;
 }
 for ($h = 0, h < maxIter, h = h + 1$**)** {
  if ($D_1 + currT_1 \leq D_2$)
   $L = D_2$**;**
  else
   $L = \min_{i=1}^{N}(currT_i + D_i)$**;**

  $r_i = L - D_1$**;**
  if ($r_1 \leq 0$) {
   $T_i = T_{i0}$;
   $e_i = 0$;
   $prevT_i = T_{i0}$;
   $currT_i = T_{i0}$;
  }

  $objF = cns = 0$;
  for each ($\tau_i \in \Gamma$) {
   $objF = objF + \frac{1}{e_i}(U_{i0} - C_i/currT_i)^2$;
   $cns = cns + \left(\frac{L-D_i}{currT_i} + 1\right) C_i$;
  }
  if ($cns > L$) **and** ($h = 0$)
   return NULL; // no feasible solution can be found
  if ($cns \leq L$) **and** ($objF < bestObjF$) {
   $bestObjF = objF$;
   for each ($\tau_i \in \Gamma$)
    $bestT_i = currT_i$**;**
  }
  for each ($\tau_i \in \Gamma$) {
   $deltaT_i = |currT_i - prevT_i|$;
   $prevT_i = currT_i$;
  }
  if $deltaT_i \leq \Delta$**, break; //achieved convergence**
  if ($D_1 + currT_1 \leq D_2$)
   Compute currT following Theorem 5;
  else
   $currT =$ **Mod_Task_compress(**Γ, L**);**
 }
 return bestT;
}

Figure 1. Period selection alg. for $D_i < T_i$

val. Since increasing task periods result in increasing slacks and since we assume that the computation times are constant, the right-hand side of (21) will remain unchanged, whereas the left-hand side of (21) will become larger. Let $L = \min_{i=1}^{N}(T_i + D_i)$ and $L' = \min_{i=1}^{N}(T'_i + D_i)$. As (21) is satisfied for L and $L' \geq L$, (21) is also satisfied for L'.

□

The above lemma has two significant consequences. First, if our algorithm cannot find a feasible solution when setting $L(0) = \min_{i=1}^{N}(T_{i_{max}} + D_i)$, it is not fruitful to continue with the algorithm as any smaller T_i's would not satisfy the constraints in (21). Second, to guarantee feasibility, it is advantageous to use larger T_i's. However, larger T_i's tends to increase the objective function. Therefore, by setting $L(h) = \min_{i=1}^{N}(T_{i_{h-1}} + D_i)$, we hope to achieve a good balance.

The following theorem states the correctness and complexity of our proposed algorithm.

Theorem 7 *Given a set of N periodic tasks with deadlines less than periods, the algorithm in Figure 1 takes $O(N^2 \cdot maxIter)$ time to output a set of task periods. Moreover, if the solution set is non-empty, the task set with the new periods is guaranteed to be schedulable by the EDF policy.*

Proof: In [9], Buttazzo et al. proved that the task compression algorithm takes $O(N^2)$ time. Since the changes made to said algorithm does not affect its complexity, **Mod_Task_compress** will also take $O(N^2)$ time. In addition, since the modified task compression algorithm constitutes the most expensive step in the main for- loop controlled by the user-defined parameter $maxIter$, the worst-case running time of the proposed heuristic is $O(N^2 \cdot maxIter)$.

Lastly, the non-empty solution set will contain feasible periods, since it is found using either Theorem 5 or Theorem 6.

□

Finally, with sufficiently small *maxIter*, the time complexity makes the proposed algorithm suitable for online period adjustments. In the next section, we will provide some guidance on how to adjust such user-defined parameters based on experimental results.

5. Experimental results

Here, we present some results to verify the claims made in the previous sections.

5.1. General period selection

To demonstrate that the task compression algorithm solves the optimization problem in(1)–(4), we reuse the task set provided in the experimental results section of [9] (reproduced below in Table 1). The task compression algorithm was written in `C++`, while `MatLab` was used to obtain the results for the constrained optimization problems and the closed-form expression. In this experiment, all tasks start at time 0 with an initial period of 100 time units and the task set is schedulable under EDF. Assume that, at time 10000, τ_1 needs to reduce its period to 33 time units, perhaps due to some changes in system dynamics not experienced by other tasks. To allow for τ_1 to change its period,

Table 1. Task set parameters used

Task	C_i	T_{i0}	$T_{i_{min}}$	$T_{i_{max}}$	e_i
τ_1	24	100	30	500	1
τ_2	24	100	30	500	1
τ_3	24	100	30	500	1.5
τ_4	24	100	30	500	2

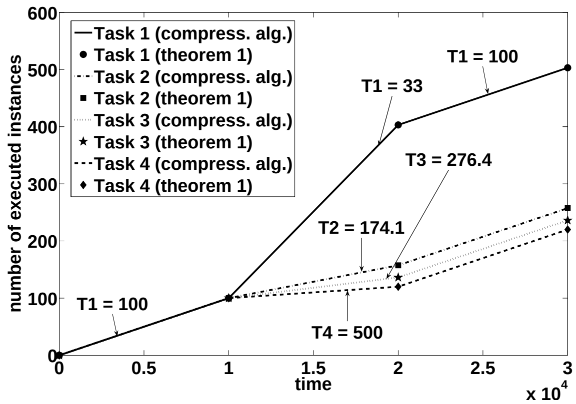

Figure 2. Utilization perturbation example

the period of tasks τ_2, τ_3, and τ_4 must increase for the system to remain schedulable. At time 20000, τ_1 goes back to its original period. Figures 2 shows the cumulative number of executed instances for each task as its period changes over time. First of all, the data verifies that the results obtained from the task compression algorithm and those from Theorem 1 match perfectly.

Furthermore, it can be seen from the graph that the number of executed instances of a task is inversely proportional to its elastic coefficient. Recall that the weight of a task is the inverse of its elastic coefficient. Although τ_2, τ_3, and τ_4 all have the same computation time, initial period, and period range, τ_2 is determined to have the smallest (e.g., best) sampling period because of its weight. On the other hands, τ_4 has the largest sampling period because it is considered to be of least importance. As before, Figure 3 shows the cumulative number of executed instances for each task as a function of time as obtained using the closed-form formula from Theorem 2. For this case, task periods are determined solely based on elastic coefficients since we assume no bound on the periods. For example, the periods of τ_1 and τ_2 were determined to be equal, as τ_1 and τ_2 both have an elastic coefficient of 1. On the other hands, the period of τ_3 is determined to be between that of τ_1 and τ_4 since e_3 is between e_1 and e_4. It is worth noting that the results from (17)–(18) matched those shown in this figure (the graph only shows the results from Theorem 2 for clarity). Hence, this finding also serves as a verification to our claim that the expression from Theorem 2 will give the same solution as the constrained optimization problem in (17)–(18). In addition, the graphs in Figures 2 and 3 show that the results from different constrained optimization problems are not comparable to each other and that there is no "best" solution for all situations. System designers must make the decision of selecting the most useful objective function given a specific system and its constraints.

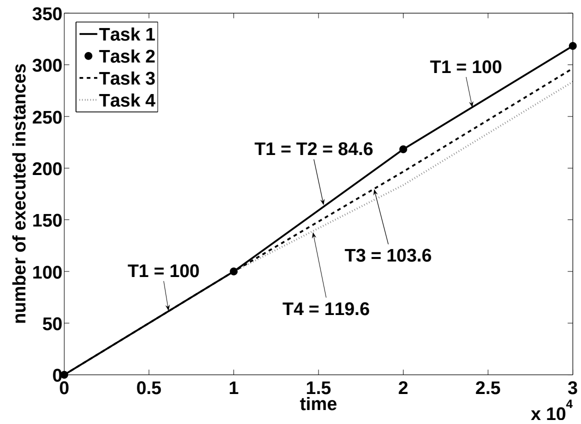

Figure 3. Period perturbation example

5.2. Period selection with additional deadline constraints

To approximate the goodness of the algorithm proposed in Section 4 as compared to the optimal solution, we performed an experiment consisting of 50 randomly generated task sets. Each set contained 5 tasks and was initially unschedulable. As before, task computation times and deadlines were kept constant.

The proposed algorithm was written in `C++`, whereas the optimization problem from (32)–(34) was solved using `Loqo`, a non-linear solver. The proposed algorithm found a convergent set of periods for 43 out of 50 sets. The user-defined parameters *maxIter* and Δ were set to 200 and 1×10^{-10}, respectively. (The parameter Δ was set to such a small value to show that task periods found by the heuristic indeed converged to some fixed values.) On the other hand, `Loqo` only found an optimal solution to 16 sets provided that a maximum of 500 iterations were run. We did not allow the solver to run longer, as, according to [27], it is unlikely that an optimal solution will be found after the $100th$ iteration. Table 2 shows the difference in total utilization obtained from using the proposed algorithm and that from `Loqo` for the task sets that were solved optimally by `Loqo`.

As can be seen by Table 2, 11 task sets are directly comparable. In other words, 11 task sets were optimally solved by `Loqo` and a convergent set of periods were found by the algorithm for these same sets. Amongst them, 7 sets have the same solutions, characterized by a difference of less than 0.01% in total utilization. The other 4 task sets were found to have different solutions where the maximum discrepancy was less than 8%. We suspect that such dis-

Table 2. Difference in total utilization

Benchmark	Iterations needed (Heuristic)	Total utilization difference (%)
2	3	0.000210
7	39	0.000054
10	200+	8.761857
12	2	0.000023
20	2	0.005683
25	21	5.657413
28	2	0.000099
29	200+	5.201867
32	17	0.059623
33	200+	3.088146
37	200+	0.925411
40	32	0.154522
42	200+	1.421277
44	49	2.745394
46	4	7.517813
48	2	0.000011

crepancy was a result of the algorithm finding a local minimizer. Interestingly, for the taks sets that the algorithm could not find a convergent set of periods, the maximum difference in utilization was less than 9%.

Based on the above results, it seems that the convergent solution set found by the proposed algorithm is in general on par with the optimal solution. In sereral instances, the heuristic successfully returns a set of feasible solutions while it was not possible to solve the optimization problem directly. Since the complexity of the heuritic is lower, the proposed algorithm is preferable for online period adjustment. Lastly, the experiment suggests that the maximum number of iterations, *maxIter*, need not be greater than 100. On the other hands, Δ can be set to be in the order of time granularity used by the operating system.

6. Conclusion and future work

In this paper, we created a general framework where the elastic task model can be treated as a special case. The framework allows for trade-offs to be viewed as optimizataion problems and for formulating a problem in a systematic way, making it easier to develop efficient algorithms to optimize a specific performance measure. As shown in the paper, the task compression algorithm can not only solve a QP problem, but also be used as a powerful component of a heuristic.

Since the algorithm presented in Section 4 is best-effort, it would be interesting to study whether there exists a way to select the value of L at every iteration such that the solution found will always be optimal. Finally, we plan on exploring different classes of objective functions and constraints that may be even harder to solve. The case for aperiodic tasks may be worth investigating since they impose a different type of constraints such as response times.

7. Acknowledgment

The authors would like to thank Bren Mochocki for providing his tool for generating task sets used to assess the heuristic algorithm.

References

[1] P. Antsaklis and J. B. (editors). Special issue on networked control systems. *Trans. on Automatic Control*, 49:1421–1423, 2004.

[2] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez. Optimal reward-based scheduling for periodic real-time tasks. *RTSS*, pp. 79–89, 1999.

[3] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, 1990.

[4] G. Bernat, A. Burns, and A. Llamosí. Weakly hard real-time systems. *Trans. on Computers*, 50(4):308–321, 2001.

[5] E. Bini and M. D. Natale. Optimal task rate selection in fixed priority systems. *RTSS*, pp. 399–409, 2005.

[6] G. Buttazzo. *Hard real time computing systems: predictable scheduling algorithms and applications*. Springer, 2005.

[7] G. Buttazzo and L. Abeni. Smooth rate adaptation through impedance control. *ECRTS*, pp. 3–10, 2002.

[8] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. *RTSS*, pp. 286–295, 1998.

[9] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *Trans. on Computers*, 51(3):289–302, 2002.

[10] M. Caccamo, G. Buttazzo, and L. Sha. Elastic feedback control. *ECRTS*, pp. 121–128, 2000.

[11] A. Cervin, J. Eker, B. Bernhardsson, and K.-E.Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1):25–53, 2002.

[12] J.-Y. Chung, J. W. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *Trans. on Computers*, 39(9):1156–1175, 1990.

[13] J. Eker, P. Hagander, and K.-E. Årzén. A feedback scheduler for real-time controller tasks. *Control Engineering Practice*, 8(1369–1378):12, 2000.

[14] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo. A new kernel approach for modular real-time systems development. *ECRTS*, pp. 199–208, 2001.

[15] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *Trans. on Computers*, 44(12):1443–1451, 1995.

[16] M. Hamdaoui and P. Ramanathan. Evaluating dynamic failure probability for streams with (m,k)-firm deadlines. *Trans. on Computers*, 46(12):1325–1337, 1997.

[17] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. *RTSS*, pp. 110–117, 1995.

[18] X. Koutsoukos, R. Tekumalla, B. Natarajan, and C. Lu. Hybrid supervisory utilization control of real-time systems. *RTAS*, pp. 12–21, 2005.

[19] T.-W. Kuo and A. Mok. Load adjustment in adaptive real-time systems. *RTSS*, pp. 160–171, 1991.

[20] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[21] D. Luenberger. *Lienar and Nonlinear Programming*. Addison-Wesley Publishing Co., Reading, MA, 1989.

[22] A. Mok and W. Wang. Window-constraint real-time periodic task scheduling. *RTSS*, pp. 15–24, 2001.

[23] G. Quan and X. Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. *RTSS*, pp. 79–98, 2000.

[24] P. Ramanathan. Overload management in real-time control applications using (m,k)-firm guarantee. *Trans. on Parallel and Distributed Systems*, 10(6):549–559, 1999.

[25] D. Seto, J. Lehoczky, and L. Sha. Task period selection and schedulability in real-time systems. *RTSS*, pp. 188–199, 1998.

[26] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. *RTSS*, pp. 13–21, 1996.

[27] R. Vanderbei. Loqo. <http://www.princeton.edu/ rvdb/>.

[28] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. *RTSS*, pp. 239–248, 2000.