A General and Scalable Method for Optimizing Real-Time Systems with Continuous Variables

Sen Wang, Ryan K. Williams, Haibo Zeng
Bradley Department of Electrical and Computer Engineering, Virginia Tech, USA
{swang666, rywilli1, hbzeng}@vt.edu

***Abstract*—In the optimization of real-time systems, designers often face a challenging problem where the schedulability conditions are non-convex, non-continuous, or lack an analytical form to understand their properties. In this paper, we propose a general and scalable framework for optimizing real-time systems, named Numerical Optimizer with Real-Time Highlight (NORTH). NORTH treats schedulability analysis as a blackbox which may only return true/false results on system schedulability. Built upon the active-set methods from the gradient-based numerical optimization literature, NORTH proposes new methods to manage active constraints to further improve the gradient-based optimizers. We apply the proposed approach to two example problems, one on energy optimization for systems with dynamic voltage and frequency scaling, and the other on the optimization of control performance. Experimental results demonstrate that the proposed framework runs 10^2 to 10^5 times faster than state-of-the-art methods while maintaining similar solution quality.**

I. Introduction

Over the years, the real-time systems community has developed an impressive set of scheduling algorithms and associated schedulability analysis techniques for various applications. However, many of these schedulability analysis techniques are not friendly for optimization purposes, as they often involve non-continuous, non-differentiable functions, e.g., the ceiling function in the classical response time calculation [1], which represents the number of interferences from a higher priority task. Even worse, the analysis may use functions that do not necessarily have an analytical form, such as those based on demand bound functions [2], real-time calculus [3], abstract event model interfaces [4], or timed automata [5], [6].

Another challenge in optimizing real-time systems is their ever-increasing complexity. The functionality of real-time systems such as automotive and unmanned aircraft is growing fast, especially given the recent trend of introducing autonomous features and system-to-system connectivity [7], [8]. For example, modern automotive systems may contain hundreds of software tasks [9]. The supporting hardware and software systems are also becoming more complex, often featuring heterogeneous multicore processors, dedicated hardware accelerators, and domain-specific operating systems.

The existing approaches for optimizing real-time systems do not adequately address the above two challenges as they lack scalability and/or applicability. In particular, the aforementioned first challenge makes it impractical or even impossible to handle schedulability conditions in standard mathematical optimization frameworks. Hence, researchers have explored the possibility to develop customized optimization frameworks that are specially designed for real-time systems and their schedulability analysis (e.g., [10]–[12]). However, they require that the schedulability analysis is *sustainable* with respect to the design variables, including task periods [10] and worst case execution times (WCETs) [11], [12]. Sustainable schedulability analysis means that if a task system is deemed schedulable, then it should remain schedulable with, for example, decreased WCET or increased period [13], [14]. Such a property may not hold [6] or become difficult to prove for sophisticated analysis techniques.

In this paper, we address the above challenges by developing an optimization framework that is generally applicable to real-time systems. Specifically, we treat the schedulability analysis as a *black box* that only answers true/false to the question of system schedulability. Our framework, named Numerical Optimizer with Real-Time Highlight (NORTH), leverages existing numerical optimization methods and tunes them for real-time systems optimization. Compared to other standard optimization frameworks, including Integer Linear Programming (ILP), convex programming, or geometric programming, numerical methods can efficiently solve many large-scale optimization problems without requiring specific forms for the problem constraints or objective function [15]. However, *existing numerical methods do not work well when optimizing real-time systems*. First, gradient-based numerical methods rely on the knowledge of *gradients*, i.e., the vector of partial derivatives. Hence, they cannot be directly applied to non-differentiable or blackbox schedulability constraints. Even if it is possible to use numerical values to estimate the gradient [16], gradient-based methods still suffer from poor solution quality, as indicated by the experiments. On the other hand, gradient-free methods typically run much slower than gradient-based methods and lack the necessary scalability [16].

Recognizing the above issues, we adopt a gradient-based method and further propose a novel technique, termed variable elimination (VE), to significantly improve its solution quality. The critical observation is that different design variables usually have different impacts on the schedulability constraints. Thus, it is possible that changing certain variables may maintain schedulability while improving the objective, and other variables cannot achieve both. VE exploits this difference by fixing the values of the second group of variables (i.e., eliminating them from the rest of the optimization process) and seeking favorable values to the first group of variables.

Contributions. In this paper, we present NORTH, to the best of our knowledge, the first numerical method based framework for optimizing real-time systems. We use two example problems to demonstrate the advantages of our framework. One is the energy minimization based on dynamic voltage and frequency scaling (DVFS), and the other is the optimization of control quality. NORTH has the following features that highlight its advantages over other alternatives:

- *Generality*: NORTH works with any schedulability analysis that may only return true/false results for system schedulability.
- *Scalability*: NORTH is scalable in that the number of times it checks the system schedulability is polynomial to the number of variables. Besides, NORTH can utilize special structures in the problem if available (See more in Theorem 4 and Example 4).
- *Quality*: NORTH adapts classical active-set methods and proposes a new approach, variable elimination, to manage active constraints. It is demonstrated to significantly improve the solution quality for gradient-based optimizers.

II. Related Work

There is a rich set of literature on the optimization methods for real-time systems. According to the categories given by Zhao *et al*. [12], popular methods include: (1) meta-heuristics such as simulated annealing [17] and genetic algorithms [18]; (2) direct formulation of the problem in standard mathematical optimization frameworks such as branch-and-bound [19], ILP [20], [21], and convex programming [22]; (3) problem-specific methods for a given system model and optimization problem, such as those for minimizing energy in systems with DVFS [23]; (4) customized optimization frameworks including [10]–[12], [24]. However, the first category is relatively slow, and often comes with a rather low solution quality. The other categories have limited applicability, as they rely on a particular form or property of the schedulability analysis. For example, ILP requires the schedulability analysis to be formulated as linear functions of the design variables, while [11], [12] requires that the analysis is sustainable.

Numerical optimization methods (NOM) are broadly used in optimization due to their fast speed and generality. Classical examples of NOM include interior point methods (IPM) and active-set methods (ASM) [16], and their more recent extensions to non-convex optimization, such as sequential-quadratic programming, Levenberg-Marquardt [25], [26], Dog-leg [27], Frank-Wolfe [28], and gradient-projection [29].

However, numerical methods cannot be directly applied to schedulability constraints that are non-differentiable or without an analytical form. The first possible solution from the literature is to use numerical approximation of gradients, but it may suffer from significant performance loss because the numerical approximation is not necessarily reliable [16] (e.g., it may become infinite at the non-continuous points). Gradient-free methods, such as model-based and interpolation methods, and the Nelder–Mead method [30], are too slow to handle large industrial problems, and their application to constrained optimization is not as well-studied as gradient-based ones [16]. Finally, a recent trend is to exploit machine learning techniques [31], [32]. However, it usually relies on a large-scale training data set, which makes the design process more time-consuming and less flexible.

We now discuss the related work for the two example problems. For energy management based on Dynamic Voltage and Frequency Scaling (DVFS), various algorithms have been proposed for systems scheduled under Earliest Deadline First (EDF) [22], [33]–[36], Rate Monotonic (RM) [34], [37], etc. A comprehensive review can be found in [23]. Recently, more sophisticated system models have been considered, including mixed-criticality scheduling [38], directed acyclic graph (DAG) models on multi-core [39], and limited-preemptive DAG task models [6]. These models also come with particularly complicated schedulability analysis and may lack the properties that other optimization approaches require. For example, the analysis in [6] is not sustainable.

The other example we consider is the optimization of control performance [40], where the control performance is often modeled as a weighted sum of task periods and response times [11]. Different optimization algorithms have been proposed, such as branch-and-bound [40], [41], genetic algorithm [17], and mixed-integer geometric programming [42]. Also, customized optimization frameworks are proposed in [10], [11]. However, similar to the above, these algorithms are limited to specific system models or require certain properties on the schedulability analysis.

III. System Model

A. Notations

We use light symbols to represent scalars and bold symbols to represent vectors and matrices. Bold symbols with a subscript i represent the i-th element of a vector/function. During optimization iterations, the iteration number is usually denoted as (k), a superscript with parenthesis. For example, $\mathbf{x}_i^{(0)}$ denotes the i-th element of the vector $\mathbf{x}$ at the 0-th iteration. $\|\mathbf{v}\|$ denotes the Euclidean norm for a vector $\mathbf{v}$, $\|\mathbf{v}\|_1$ denotes norm-1, $|\mathrm{S}|$ denotes the size of a set S, $|x|$ could also denote the absolute value of a scalar x. We usually use h to denote the numerical granularity, which is 10^{-5} in experiments. We usually use the letter x to represent variables, $\mathcal{F}(\mathbf{x})$ for objective functions, $g(\mathbf{x})$ for constraints. However, when introducing real applications, we follow the standard notation in the domain to avoid confusion. The gradient is denoted as $\boldsymbol{\nabla}\mathcal{F}(\mathbf{x})$. If $\mathcal{F}(\mathbf{x})$ can be equivalently transformed into a sum-of-items form, yielding an objective:

$$\min_{\mathbf{x}} \sum_i \mathcal{F}_i(\mathbf{x}) \tag{1}$$

then we call each item $\mathcal{F}_i(\mathbf{x})$ a component of the objective function. In this case, the Jacobian matrix is:

$$\mathbf{J}_{ij} = \frac{\partial \mathcal{F}_i(\mathbf{x})}{\mathbf{x}_j} \tag{2}$$

where $\mathbf{J}_{ij}$ is the entry of the Jacobian matrix at the i-th row and the j-th column. $\mathbf{J}^T$ denotes the transpose of $\mathbf{J}$.

In many situations, the response time r_i of a task τ_i may depend on some variables, e.g., $\mathbf{x}$, then $r_i(\mathbf{x})$ denotes a function which returns the response time of τ_i.

B. Problem Formulation

Generally speaking, real-time system design can be mathematically described as an optimization problem:

$$\min_{\mathbf{x}} \mathcal{F}(\mathbf{x}) \tag{3}$$

$$\textit{s.b.t.} \quad \text{Sched}(\mathbf{x}) = 0 \tag{4}$$

$$\text{lb}_i \leq \mathbf{x}_i \leq \text{ub}_i \tag{5}$$

where $\mathbf{x} \in \mathbb{R}^N$ represents the design choices, such as the execution times or periods of tasks, lb_i (ub_i) denotes the lower (upper) bound of $\mathbf{x}_i$, and $\mathcal{F}(\mathbf{x}) : \mathbb{R}^N \rightarrow \mathbf{R}$ describes the design objective, which could be overall energy consumption, or some customized objective related to design choices. If possible, we represent $\mathcal{F}(\mathbf{x})$ using a least-square form to enable faster convergence speed from numerical optimizers. Finally, of central importance is the schedulability analysis constraint:

$$\text{Sched}(\mathbf{x}) = \begin{cases} 0, & \text{system is schedulable} \\ 1, & \text{otherwise} \end{cases} \tag{6}$$

which *only* returns binary results, i.e., schedulable or not. This is a challenging but realistic assumption in many situations. For example, analysis methods based on demand bound functions [2] or timed automata may fall into this category.

Assumption 1. We assume a feasible initial solution is available to begin optimization iterations. This initial solution can usually be easily acquired. For example, in the case of DVFS, the maximum CPU frequency can be a good initial solution.

Assumption 2. We assume the optimization variables $\mathbf{x}$ in problem (3) are continuous in their domain, but the constraints or objective functions may be non-differentiable with respect to these variables. Examples are task WCETs, periods, or deadlines.

Relaxation of the two assumptions above will be discussed in Section VII.

Challenges. The major challenge comes from the schedulability constraint (4), which could only return 0/1 and thus lacks gradient information. Furthermore, the objective function and constraints could be non-convex, non-differentiable, or even black-box functions. Therefore, most state-of-the-art nonlinear programming (NLP) methods, which are proposed for smooth problems, cannot be directly applied or may suffer from serious performance loss.

In addition, performing schedulability analysis is expensive in many cases because it could have pseudo-polynomial or even exponential computation costs. Therefore, to finish optimization within a reasonable time, minimizing the number of times to call schedulability analysis is desired.

C. Application: Energy Optimization

The first example application is energy optimization based on DVFS [23], one of the most popular optimization strategies used in real-time embedded systems. By sacrificing part of the response time of running tasks at a lower frequency, DVFS can achieve significant power consumption improvements. To be more specific, we consider a task set of N periodic tasks scheduled by a given scheduling method. This problem can be formulated as a least-square optimization problem by minimizing the energy consumption $E_i(\mathbf{f})$ of each task τ_i with respect to the run-time frequency $\mathbf{f}$ of all the tasks:

$$\min_{\mathbf{f}} \sum_{i=0}^{N-1} (\sqrt{E_i(\mathbf{f})})^2 \tag{7}$$

$$\textit{s.b.t.} \quad \text{Sched}(\mathbf{f}) = 0 \tag{8}$$

$$\text{lb}_i \leq \mathbf{f}_i \leq \text{ub}_i, i \in [0, N-1] \tag{9}$$

with the energy function $E_i(\mathbf{f})$ for the task τ_i approximated as:

$$E_i(\mathbf{f}) = \frac{H}{T_i}(\beta + \alpha \mathbf{f}_i^{\gamma}) \times c_i \tag{10}$$

where H is the hyper-period (i.e., the least common multiple of the task set's periods) and T_i is task τ_i's period. The second term in the above equation is a power model commonly adopted in the literature, which considers both static and dynamic power consumption [43], [44]. The parameters ($\alpha = 1.76\text{Watts}/\text{GHz}^3$, $\gamma = 3$, $\beta = 0.5$) are highly accurate on real platforms [43], [45]. The execution time c_i of each task τ_i can be obtained from the run-time frequency $\mathbf{f}_i$ by a frequency model adopted from [22], [23]:

$$c_i = c_i^{\text{fix}} + \frac{c_i^{\text{var}}}{\mathbf{f}_i} \tag{11}$$

which considers both speed-independent (c_i^{fix}) and speed-dependent (c_i^{var}) operations when executing the computational tasks.

The schedulability analysis of the above formulation could be performed by various methods. For convenience in comparing to baseline methods, we will first consider the classical response time analysis (RTA) model for uni-processor, fixed-priority, preemptive platforms [46]:

$$r_i = c_i + \sum_{j \in \text{hp}(i)} \lceil \frac{r_i}{T_j} \rceil c_j \tag{12}$$

where $\text{hp}(i)$ denotes the tasks with higher priority than the task τ_i. In this case, let D_i denote the deadline of τ_i, and we have

$$\text{Sched}(\mathbf{f}) = \begin{cases} 0, & \forall i, r_i(\mathbf{f}) \leq D_i \\ 1, & \text{otherwise} \end{cases} \tag{13}$$

Importantly, our framework supports much more complicated schedulability analysis methods than Equation (12). For example, model verification methods such as those proposed by Nasri *et al.* [6] for limited-preemptive DAG task models are also used in our experiments. Such methods usually have advantages in real systems by providing less pessimism than

many analytic methods. However, state-of-art methods such as [11] cannot be applied because [6]'s analysis is not sustainable.

D. Application: Control Quality Optimization

Our second application example is concerned with optimizing the control performance of a task set [11], [40]. Following the problem description in [11], we use a least-square formulation to approximate the control cost, which is a weighted sum of task period $\mathbf{T}_i$ and response time $r_i(\mathbf{T})$:

$$\min_{\mathbf{T}} \sum_{i=0}^{N-1} (\sqrt{\alpha_i \mathbf{T}_i + \beta_i r_i(\mathbf{T})})^2 \tag{14}$$

$$s.b.t. \quad \text{Sched}(\mathbf{T}) = 0 \tag{15}$$

$$\text{lb}_i \leq \mathbf{T}_i \leq \text{ub}_i, i \in [0, N-1] \tag{16}$$

where the variables $\mathbf{T}_i$ are the task τ_i's periods, α_i and β_i are given control weight parameters, and $r_i(\mathbf{T})$ is the task τ_i's response time. Similar to energy optimization, the schedulability analysis can be posed in various forms. In our experiments, we use the RTA model given by (12), which is the same as [11] for easy comparison.

E. Concepts from Numerical Optimization

Definition 1 (Feasible solution)**.** *A solution $\boldsymbol{x}$ for the optimization problem (3) is feasible if it satisfies all the constraints.*

Definition 2 (Differentiable point)**.** *In this paper, a point $\boldsymbol{x}$ is called (non-)differentiable if the objective function $\mathcal{F}(\boldsymbol{x})$ is (not) differentiable at $\boldsymbol{x}$.*

Definition 3 (Descent vector)**.** *A vector $\boldsymbol{\Delta}$ is called a descent vector for function $\mathcal{F}(\boldsymbol{x})$ at $\boldsymbol{x}$ if*

$$\mathcal{F}(\boldsymbol{x} + \boldsymbol{\Delta}) < \mathcal{F}(\boldsymbol{x}) \tag{17}$$

Definition 4 (Descent direction)**.** *$\boldsymbol{\Delta}$ is a descent direction if $\exists \alpha > 0$ such that $\alpha\boldsymbol{\Delta}$ is a descent vector.*

We also clarify several terms in numerical optimization [16]:

Continuous variable: A variable is considered continuous if it can take any floating-point values within its domain.

Active/inactive constraints: At point $\mathbf{x}$, a constraint $g(\mathbf{x}) \leq 0$ is called *active* if it holds with an "equal" relationship, i.e., $g(\mathbf{x}) = 0$ at $\mathbf{x}$. All the equality constraints are active by definition. An inequality constraint that holds with strict "larger/smaller than" at $\mathbf{x}$ is inactive, e.g., $g(\mathbf{x}) < 0$.

Active-set framework (ASM): Essentially, in each iteration to solve a constrained optimization problem, the active-set framework solves a simplified problem that only has equality constraints (active constraints). This is based on the assumption that inactive constraints will not influence a local update if the update is small enough. Famous active-set algorithms include the simplex method, sequential-quadratic programming, etc.

Trust-region methods (TRM): Essentially, TRM defines a region around the current variable $\mathbf{x}^{(k)}$ to search for a local update, and then updates the region radius based on optimization results. The search process is usually based on solving an approximate model of the original problem.

Linearization: According to the Taylor expansion, a function $f(\mathbf{x})$ can be locally approximated by a linear function with its first-order gradient:

$$f(\mathbf{x}) \approx f(\mathbf{x}^{(0)}) + \nabla f(\mathbf{x}^{(0)})(\mathbf{x} - \mathbf{x}^{(0)}) \tag{18}$$

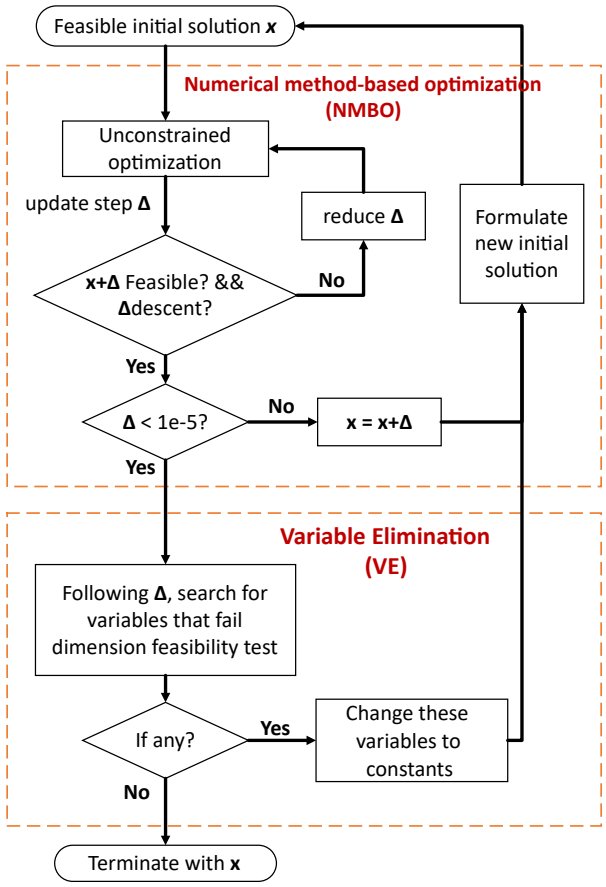


Figure 1: Framework overview: NORTH and its two components: NMBO and VE. During iterations, beginning with a feasible solution $\mathbf{x}^{(k)}$, we formulate an unconstrained problem and use classical optimizers to find an update direction $\boldsymbol{\Delta}$. If $\boldsymbol{\Delta}$ leads into an infeasible region, then we will decrease $\boldsymbol{\Delta}$ until it becomes small enough so that $\mathbf{x}^{(k+1)}$ is feasible. After it, we check whether there exist close active constraints. If so, we find their dependent variables and transform these variables to constants at $\mathbf{x}^{(k+1)}$ to continue the future iterations. The algorithm terminates when there are no variables to eliminate.

IV. Numerical Method-Based Optimization

A. Motivation

In this section, we introduce the first component of NORTH: Numerical Method-Based Optimization (NMBO) to attack the challenges above by avoiding the gradient evaluation of the schedulability constraints. NMBO is inspired by active-set methods (ASM) and trust-region methods (TRM) from numerical optimization. ASM is suitable because it allows us to drop the inactive constraints during iterations, therefore simplifying many iterations where the variables $\mathbf{x}^{(k)}$ are not

close to becoming infeasible. If $\mathbf{x}^{(k)} + \mathbf{\Delta}$ becomes infeasible, we will keep decreasing its update step $\mathbf{\Delta}$ until $\mathbf{x}^{(k)} + \mathbf{\Delta}$ becomes feasible or stop if $\mathbf{\Delta}$ becomes small enough. In this way, $\mathbf{x}^{(k)}$ will finally stop at either a stationary point or somewhere near the feasibility region boundary.

NMBO is similar to but different from the popular numerical methods such as the trust-region method (TRM) [16] because NMBO works with the schedulability constraint (4): At first, NMBO utilizes the idea of ASM to avoid evaluating constraint gradients because the gradients are usually not differentiable. Secondly, NMBO adds an extra step to examine constraint violation after applying the update step $\mathbf{\Delta}$ to guarantee the feasibility of (4).

B. Methodology

In this section, we describe NMBO, which combines the active-set and the trust-region algorithms to optimize with the schedulability constraint (4). In each iteration, we begin with a feasible solution and perform optimization without constraints. After obtaining an update step, we verify whether this new update step is feasible and descent. If so, the update step $\mathbf{\Delta}$ is accepted; otherwise, $\mathbf{\Delta}$ is decreased based on classical trust-region or line search methods [16]. This way, NMBO will terminate at either static points or feasible region boundaries.

Unconstrained optimization could be provided by any methods. For example, in our experiments, we use the popular Levenberg-Marquardt algorithm [25], [26] (LM) as the unconstrained optimizer. LM finds update steps as follows:

$$(\mathbf{J}^{(k)T}\mathbf{J}^{(k)} + \lambda \operatorname{diag}(\mathbf{J}^{(k)T}\mathbf{J}^{(k)}))\mathbf{\Delta} = -\mathbf{J}^{(k)T}\boldsymbol{\mathcal{F}}(\mathbf{x}^{(k)}) \quad (19)$$

where $\mathbf{J}^{(k)}$ is the Jacobian matrix after linearizing the objective function:

$$\boldsymbol{\mathcal{F}}(\mathbf{x}^{(k)}) = \left[\mathcal{F}_0(\mathbf{x}^{(k)}), .., \mathcal{F}_m(\mathbf{x}^{(k)})\right]^T \quad (20)$$

The i-th row of $\mathbf{J}$ is given by:

$$\mathbf{J}_{i,:} = \boldsymbol{\nabla}\mathcal{F}_i(\mathbf{x}) \quad (21)$$

$$\boldsymbol{\nabla}\mathcal{F}_i(\mathbf{x}) = \left[\frac{\partial \mathcal{F}_i(\mathbf{x})}{\partial x_1}, ..., \frac{\partial \mathcal{F}_i(\mathbf{x})}{\partial x_N}\right] \quad (22)$$

The length of $\mathbf{\Delta}$ can be indirectly managed by controlling the damping parameter λ in Equation (19). For example, increasing λ usually decreases the update step $\mathbf{\Delta}$.

C. Numerical Gradient

If the objective function $\mathcal{F}_i(\mathbf{x})$ is not differentiable at $\mathbf{x}^{(k)}$, or if it is difficult to derive analytical gradients (e.g., $\mathcal{F}_i(\mathbf{x})$ could be the response time given by a black-box schedulability analysis), the numerical gradient can be utilized. The numerical gradient for a function $f(\mathbf{x})$ is estimated as follows:

$$\frac{\partial f}{\partial x_i} = \frac{f(x_1, .., x_i + h, .., x_N) - f(x_1, .., x_i - h, .., x_N)}{2h} \quad (23)$$

The step size h can be chosen by considering both approximation accuracy (smaller is better) and round-off error (cannot be too small) [16]. We use 10^{-5} in the experiments. Alternative gradient estimation methods, such as stochastic methods, may also be used. Although numerical gradient (23) is convenient, we make the following critical observations:

Observation 1. *Applying numerical gradient (23) at non-differentiable points may return a misleading gradient.*

Justification. The numerical gradient (23) at a non-differentiable point could point in any directions, including some non-descent directions. Furthermore, if the step size h is small, the norm of the numerical gradient could become very large, and optimizers will perform worse.

Observation 2. *Classical gradient-based constrained programming methods may return infeasible results if the constraints are not differentiable.*

Justification. Gradient-based methods usually build a local approximation model for constraints in each iteration. If the constraints are not differentiable, the numerical gradient could be misleading, and such an approximation model cannot be reliable, therefore, may return unsafe results.

Despite being a gradient-based optimizer, NMBO solves the issues above by building local approximation models only for objective functions but not for constraints. After obtaining a descent vector, NMBO checks whether it leads to a feasible solution and only accepts it if it is safe.

D. Termination Conditions for NMBO

NMBO stops if the update steps $\mathbf{\Delta}$ or the relative difference

$$\delta_{\text{rel}} = \frac{\mathcal{F}(\mathbf{x}^{(k+1)}) - \mathcal{F}(\mathbf{x}^{(k)})}{\mathcal{F}(\mathbf{x}^{(k)})} \quad (24)$$

is small enough, e.g., $\delta_{\text{rel}} \leq 10^{-5}$, or the number of iterations exceeds a given bound (e.g., 1000). Although the latter is theoretically possible, it is not observed in our experiments.

Example 1. We use a simplified energy minimization problem as an example throughout this paper to demonstrate how to execute each step. To make the examples more intuitive, we use the execution time $\mathbf{c}_i$ rather than run-time frequency $\mathbf{f}_i$ as variables.

$$\min_{\mathbf{c}_1,\ \mathbf{c}_2} \ 64\mathbf{c}_1^{-2} + \mathbf{c}_2^{-2} \quad (25)$$

$$s.b.t. \ \text{Sched}(\mathbf{c}_1, \mathbf{c}_2) = 0 \quad (26)$$

$$4 \leq \mathbf{c}_1 \leq 10 \quad (27)$$

$$1 \leq \mathbf{c}_2 \leq 10 \quad (28)$$

where the variables are the execution time of each task. Task sets are described in Table I, the schedulability constraint is given by Equation (13).

Table I: Example task set for elimination

	Old computation time	Period	Deadline	Priority
Task 1	4	10	6	High
Task 2	1	40	40	Low

We assume a feasible initial solution is given at $\mathbf{c}^{(0)} = (4, 1)$, and use LM (Equation (19)) as the unconstrained optimizer. The minimum numerical granularity is 10^{-5}, initial λ from (19) is 1, then we have:

$$\mathbf{J}^{(0)} = \begin{bmatrix} -64 \times 2/4^3 & 0 \\ 0 & -2/1^3 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix} \tag{29}$$

$$\mathbf{c}^{(1)} = \mathbf{c}^{(0)} + \boldsymbol{\Delta} = \begin{bmatrix} 5 & 1.25 \end{bmatrix} \tag{30}$$

Such iterations will continue until $\mathbf{c}^{(k)}$ becomes very close to the schedulability boundary, e.g., $(5.999, 1.499)$. The unconstrained optimization will stop at this point because it cannot find a feasible update step with very big λ, e.g., 10^{12}.

Observation 3. *If schedulability analysis could utilize a safe "warm start", then the schedulability results from previous iterations may speed up iterative methods such as NMBO.*

Justification. The "incremental" nature of iterative algorithms is helpful in some schedulability analyses. For example, consider the response time analysis model by Equation (12). After increasing the execution time $\mathbf{c}_i$ of the task τ_i to $\mathbf{c}_i + \boldsymbol{\Delta}_i$ and $\boldsymbol{\Delta}_i > 0$, $r_i(\mathbf{c}_i)$ from the last iteration could be a warm start to begin the fixed-point iterations in Equation (12) to obtain $r_i(\mathbf{c}_i + \boldsymbol{\Delta}_i)$. More related work can be found in Davis *et al.* [47].

V. Variable Elimination

A. Motivation and Concepts

As shown in Fig. 2, an infeasible descent vector $\boldsymbol{\Delta}$ could contain a feasible direction. Therefore, if we only optimize a portion of variables that will not violate constraints soon, we may continue optimization after NMBO terminates. We call this simple idea *variable elimination* (VE).

Therefore, it is important to know when iterations reach the feasible region's boundary to enable VE, which is not easy because Equation (4) only returns 0/1. Early work such as [48], [49] usually assumes a specific schedulability analysis, and thus is unsuitable in our case. The classical definition of active constraints introduced in Section III-E is also not suitable because it is defined for continuous functions. Therefore, we extend the classical definition for active constraints in the context of real-time system design:

Definition 5 (Active schedulability constraint)**.** *The schedulability constraint given by Equation (4) becomes an active constraint at a point $\boldsymbol{x}$ if*

$$Sched(\boldsymbol{x}) = 0 \tag{31}$$

$$\exists \boldsymbol{\delta}, \|\boldsymbol{\delta}\| \leq h, Sched(\boldsymbol{x} + \boldsymbol{\delta}) = 1 \tag{32}$$

where h is the minimum numerical granularity.

In other words, if a point $\mathbf{x}$ has an active schedulability constraint, it means that $\mathbf{x}$ is schedulable but very close to becoming unschedulable. When the optimizer terminates, some inequality or schedulability constraints may become active constraints, preventing classical optimizers from moving forward. A simple solution is removing some variables that could cause constraint violation. Finding these variables will be easier with the following definitions:

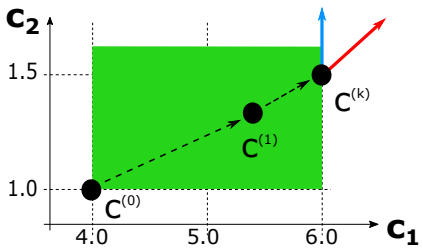


Figure 2: Variable elimination motivating example. We consider Example 1 which terminates at $(5.999, 1.499)$. The red arrow shows the update step $\boldsymbol{\Delta}$ from classical unconstrained optimizers. It will make $\mathbf{x}$ infeasible because task 1 will miss its deadline. However, if we only update $\mathbf{x}_2$ following $\boldsymbol{\Delta}$ while leaving $\mathbf{x}_1$ unchanged, we can obtain a feasible descent step, as shown by the blue arrow.

Definition 6 (Dimension feasibility test)**.** *A solution $\boldsymbol{x} \in \mathbb{R}^N$ for optimization problem (3) passes dimension-j feasibility test of length d along the direction $\boldsymbol{\Delta}$ if $\boldsymbol{x} \oplus (\frac{\boldsymbol{\Delta}_j}{|\boldsymbol{\Delta}_j|} d, j)$ is feasible, with $j \in [0, N-1]$.*

where the $\oplus$ operation for arbitray vectors $\mathbf{x}$ and $\mathbf{y}$ in this paper is defined as follows:

$$\mathbf{y} = \mathbf{x} \oplus (d, j) \;\Leftrightarrow\; \mathbf{y}_i = \begin{cases} \mathbf{x}_i, & i \neq j \\ \mathbf{x}_i + d & i = j \end{cases} \tag{33}$$

In other words, any $\mathbf{x}_i$ that does not pass the dimension feasibility test along the descent direction $\boldsymbol{\Delta}$, given by unconstrained optimizers, of length $d \geq h$ should be eliminated, where h is minimum numerical granularity. After $\mathbf{x}_i$ is eliminated at $\mathbf{x}^{(k)}$, it will remain unchanged as $\mathbf{x}_i^{(k)}$ in the future. d can be estimated following the first paragraph in Section V-D.

Example 2. In this example, we continue with Example 1 and show how to perform the Dimension Feasibility test. We will first consider a small elimination tolerance $d = 10^{-5}$. In this case, $\mathbf{c}$ passes dimension-1 (with $\mathbf{c}_1$) and dimension-2 (with $\mathbf{c}_2$) feasibility test because both $(5.999 + 10^{-5}, 1.499)$ and $(5.999, 1.499 + 10^{-5})$ are schedulable. However, if we consider a larger elimination tolerance, e.g., $d = 0.1$, $\mathbf{c}$ will fail the dimension-1 feasibility test (with $\mathbf{c}_1$) but still pass the dimension-2 feasibility test (with $\mathbf{c}_2$). How to select the proper elimination tolerance will be introduced later.

B. Performance Influence of Variable Elimination

Definition 7 (Strict descent step)**.** *An update step $\boldsymbol{\Delta} \in \mathbb{R}^N$ at $\boldsymbol{x}$ is called a strict descent step if*

$$\forall i \in [0, N-1],\ \mathcal{F}(\boldsymbol{x} \oplus (\boldsymbol{\Delta}_i, i)) \leq \mathcal{F}(\boldsymbol{x}) \tag{34}$$

In many real-time systems, the design metric is posed at the task level, and each task provides one or a set of variables individually. Two examples can be found in Section III-C

and III-D. Such decoupled systems are beneficial because the descent step returned by gradient-based optimizers is usually a strict descent step. The following theorem summarizes this observation using the objective function given by (3):

Theorem 1. *If $\mathcal{F}(\boldsymbol{x})$ is differentiable at $\boldsymbol{x}$, then there exists $\alpha > 0$ such that $\boldsymbol{\Delta} = -\alpha \boldsymbol{\nabla} \mathcal{F}$ is a strict descent step.*

Proof.

$$\boldsymbol{\nabla} \mathcal{F} = \left[\nabla \mathcal{F}_0(x_0), ..., \boldsymbol{\nabla} \mathcal{F}_0(x_{N-1})\right] \tag{35}$$

According to the Taylor expansion, for $\forall i \in [0, N-1]$, there exists a small $\alpha > 0$ such that:

$$\mathcal{F}(\mathbf{x} \oplus (\boldsymbol{\Delta}_i, i)) = \mathcal{F}(\mathbf{x}) - \alpha (\boldsymbol{\nabla} \mathcal{F}_i(\mathbf{x}_i))^2 \leq \mathcal{F}(\mathbf{x}) \tag{36}$$

■

Now we introduce more notations for the following theorems. At a feasible point $\mathbf{x} \in \mathbb{R}^N$, we denote $\boldsymbol{\Delta} \in \mathbb{R}^N$ as a descent, but not necessarily feasible, step returned by an unconstrained optimizer, such as Equation (19). Furthermore, we denote $\mathcal{S}$ as the collection of the dimension indexes that pass the dimension feasibility test with length $d \geq h$ along the direction $\boldsymbol{\Delta}$, where h is the minimum numerical granularity, and $\bar{\mathcal{S}}$ for those that fail the test. The update step $\mathbf{v}_i^{\mathcal{D}} \in \mathbb{R}^N$ for the dimension-i feasibility test can be described as:

$$\mathbf{v}_{ij}^{\mathcal{D}} = \begin{cases} \boldsymbol{\Delta}_j & j = i \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

Theorem 2. *If $\boldsymbol{\Delta}$ is a strict descent step, and the number of indexes that pass the dimension feasibility test is $|\mathcal{S}| = q > 0$, then all the q vectors $\boldsymbol{v}_i^{\mathcal{D}}$ that pass the dimension feasibility test provide feasible descent update directions at $\boldsymbol{x}$.*

Proof. Results of Definition 7 and Theorem 1. ■

Theorem 3. *If both the objective function $\mathcal{F}(\boldsymbol{x})$ and all the constraints (4) and (5) are locally differentiable at $\boldsymbol{x}^{(k)}$, and the number of indexes that pass dimension feasibility test $|\mathcal{S}| = q > 0$, then there exists $\bar{d} < d$ that is small enough such that $\boldsymbol{x}^{(k)} + \hat{\boldsymbol{\Delta}}$ is a feasible solution, where*

$$\hat{\boldsymbol{\Delta}} \in \{\sum_i^q \alpha_i \boldsymbol{v}_i^{\mathcal{D}} \mid \|\hat{\boldsymbol{\Delta}}\|_1 \leq \bar{d},\ \alpha_i \geq 0\} \tag{38}$$

$$\mathcal{F}(\boldsymbol{x}^{(k)} + \hat{\boldsymbol{\Delta}}) \leq \mathcal{F}(\boldsymbol{x}^{(k)}) \tag{39}$$

In Equation (38), norm-1 is used such that each dimension of $\boldsymbol{\Delta}$ is no larger than d.

Proof. Similar to Theorem 1, the feasibility and descent property can be proven using the Taylor expansion to constraints and objective function, respectively. ■

Observation 4. *Although the objective functions and constraints in real-time systems are not differentiable functions in $\mathbb{R}^N$, they are differentiable at many feasible points.*

This is because a function of continuous variables, such as the response time (12), usually only has a limited number of non-differentiable points but infinite differentiable points.

C. Non-differentiable Objective Function

When NMBO terminates at a non-differentiable point $\mathbf{x}^{(k)}$, we can find and eliminate the non-differentiable components $\mathbf{x}_i$, because their numerical gradient is not reliable and may prevent optimizers from finding a feasible descent direction. If a guarantee of performance improvement is desired without the assumptions above, e.g., differentiable and decoupled objective functions/constraints at $\mathbf{x}^{(k)}$, the dimension feasibility test can be replaced with:

Definition 8 (Dimension feasibility descent test)**.** *A solution $\boldsymbol{x} \in \mathbb{R}^N$ for optimization problem (3) passes dimension-j feasibility descent test of length d along the direction $\boldsymbol{\Delta}$ if $\boldsymbol{x} \oplus (\frac{\boldsymbol{\Delta}_j}{|\boldsymbol{\Delta}_j|} d, j)$ is feasible, where $j \in [0, N-1]$ and*

$$\mathcal{F}(\boldsymbol{x} \oplus (\frac{\boldsymbol{\Delta}_j}{|\boldsymbol{\Delta}_j|} d, j)) \leq \mathcal{F}(\boldsymbol{x}) \tag{40}$$

Coordinate descent [50] can then be used for future iterations. It is useful for situations such as when non-sustainable schedulability analysis is observed.

D. How to Eliminate Variables

Following Theorem 3, we can possibly improve $\mathbf{x}^{(k)}$ by transforming $\mathbf{x}_{\bar{\mathcal{S}}}$ into constant values at $\mathbf{x}^{(k)}$, where $\bar{\mathcal{S}}$ denotes those indexes that do not pass the dimension feasibility (descent) tests. The complete procedure is given by Algorithm 1. In reality, estimating proper d is difficult. Therefore, we design an adaptive strategy: first try small values (e.g., minimum numerical granularity) and keep increasing d (e.g., 1.5x in each iteration) until at least one variable is eliminated.

Algorithm 1: Variable elimination

Input: $\mathbf{x}^{(k)}$, descent step $\boldsymbol{\Delta}$, elimination tolerance d, un-eliminated indexes set $\mathcal{U}$
Output: eliminated variable indexes

```
for (i in U) do
    x_i^(k) = x_i^(k) + d * Δ_i/|Δ_i|
    if (AnyConstraintIsViolated(x_i^(k)) OR
     IsNonDifferentiable(x_i^(k))) then
        return i
    end
    x_i^(k) = x_i^(k) - d * Δ_i/|Δ_i|  // Recover x_i^(k)
end
```

Example 3. We continue with Example 1 to illustrate the adaptive strategy in selecting d. To find out which variable to eliminate at $(5.999, 1.499)$, we can try to increase $\mathbf{c}_1$ or $\mathbf{c}_2$ by d separately and see whether there would be constraints being violated. We may first try $d = 10^{-5}$, but notice that no constraint violation is found. Then we slowly increase d (e.g., by 1.5x each time) until $d = 10^{-5} \times 1.5^{12} > 0.001$, and notice that $\mathbf{c}$ fails the dimension-1 feasibility test (with $\mathbf{c}_1$) while still passing the dimension-2 feasibility test (with $\mathbf{c}_2$). Therefore, we eliminate $\mathbf{c}_1$. After it, $\mathbf{c}_1$ participates in

the following iterations as a constant variable 5.999, and $\mathbf{c}_2$ becomes the only variable to optimize.

Although Algorithm 1 requires examining each uneliminated variable once in each loop, this process can be sped up for many schedulability analyses.

Theorem 4. *During variable elimination, if an variable $\boldsymbol{x}_i$ is eliminated at a feasible point $\boldsymbol{x}$, i.e.:*

$$Sched(.., x_j, .., x_i + d\frac{\boldsymbol{\Delta}_i}{|\boldsymbol{\Delta}_i|}, ..) = 1 \tag{41}$$

and $\boldsymbol{x}_j$ has bigger influence than $\boldsymbol{x}_i$ on schedulability, i.e.,

$$Sched(.., x_j + d\frac{\boldsymbol{\Delta}_j}{|\boldsymbol{\Delta}_j|}, .., x_i, ..) \geq Sched(.., x_j, .., x_i + d\frac{\boldsymbol{\Delta}_i}{|\boldsymbol{\Delta}_i|}, ..) \tag{42}$$

and $\boldsymbol{\Delta}_i \boldsymbol{\Delta}_j \geq 0$, then $\boldsymbol{x}_j$ also satisfies elimination conditions and should be eliminated.

Proof. From the assumptions, $\mathbf{x}_j$ satisfies the elimination conditions from Equation (41) and (42), therefore should be eliminated. ■

Although analyzing whether some variables have a bigger influence than others is difficult for black-box schedulability analysis, there are many cases where it can be done, and Theorem 4 becomes useful. In preemptive systems, if execution times are variables, the utility test for the earliest deadline first (tasks with lower period have bigger influence) and rate monotonic [51], or Equation (12) (priority decides influence) all satisfy the assumption above. In some DAG schedulability analyses such as [52], [53] under multi-processor platforms, tasks in the critical path may have a bigger influence than other tasks, together with other mild assumptions.

Example 4. We use the schedulability analysis given by Equation (12) as an example to illustrate Theorem 4. The variables are the worst-case execution time $\mathbf{c}_i \neq 0$ of each task. In this case, tasks with higher priority have bigger influence. We denote τ_k as any one task that has higher priority than τ_i, $r_i(\mathbf{c}_i, \mathbf{c}_k)$ as the response time of τ_i that depends on $\mathbf{c}_i$ and $\mathbf{c}_k$, and assume $\mathbf{c}$ does not pass the dimension-i feasibility test and will be eliminated, i.e., there exists $\Delta > 0$ such that

$$r_i(\mathbf{c}_i, \mathbf{c}_k) = \mathbf{c}_i + \sum_{j \in \text{hp}(i)} \lceil \frac{r_i(\mathbf{c}_i, \mathbf{c}_k)}{T_j} \rceil \mathbf{c}_j \leq D_i \tag{43}$$

$$r_i(\mathbf{c}_i + \Delta, \mathbf{c}_k) = \mathbf{c}_i + \Delta + \sum_{j \in \text{hp}(i)} \lceil \frac{r_i(\mathbf{c}_i + \Delta, \mathbf{c}_k)}{T_j} \rceil \mathbf{c}_j > D_i \tag{44}$$

Then, $\mathbf{c}_k$ from task τ_k will also be eliminated together with $\mathbf{c}_i$ because:

$$\begin{aligned} r_i(\mathbf{c}_i, \mathbf{c}_k + \Delta) &= \mathbf{c}_i + \sum_{j \in \text{hp}(i), j \neq k} \lceil \frac{r_i(\mathbf{c}_i, \mathbf{c}_k + \Delta)}{T_j} \rceil \mathbf{c}_j \\ &\quad + \lceil \frac{r_i(\mathbf{c}_i, \mathbf{c}_k + \Delta)}{T_k} \rceil (\mathbf{c}_k + \Delta) \\ &\geq r_i(\mathbf{c}_i + \Delta, \mathbf{c}_k) > D_i \end{aligned} \tag{45}$$

The reason why the last "$\geq$" sign holds can be seen by considering the process of fixed-point iterations. When $r_i(\mathbf{c}_i + \Delta, \mathbf{c}_k)$ and $r_i(\mathbf{c}_i, \mathbf{c}_k + \Delta)$ begin with the same value such as $r_i(\mathbf{c}_i, \mathbf{c}_k)$, $r_i(\mathbf{c}_i, \mathbf{c}_k + \Delta)$ will result into a value that is not less than $r_i(\mathbf{c}_i, \mathbf{c}_k + \Delta)$ because:

$$\lceil \frac{r_i(\mathbf{c}_i, \mathbf{c}_k)}{T_k} \rceil \Delta \geq \Delta \tag{46}$$

Therefore, when iterations terminate, we will have $r_i(\mathbf{c}_i, \mathbf{c}_k + \Delta) \geq r_i(\mathbf{c}_i + \Delta, \mathbf{c}_k)$.

Theorem 5. *Results returned by NORTH are always feasible.*

Proof. The input to VE is always feasible because NMBO only accepts feasible updates. The output of VE is also feasible because VE does not change the values of variables. When performing future iterations, all the constraints remain the same as the original constraints. Therefore solutions from NORTH are always feasible. ■

E. Termination Condition

There are two possible termination conditions. If the inner trust-region optimizer stops at a differentiable point from gradient-based optimizers, that means we reach a stationary point, and the overall algorithm will terminate. However, in most cases, after the inner algorithm terminates, VE will find new variables to eliminate, and possibly a feasible descent direction to continue iterations. The overall framework will terminate if all the variables are eliminated.

Theorem 6. *The total number of iterations of NORTH is no larger than the number of variables.*

Proof. The adaptive elimination tolerance introduced before could guarantee that at least one variable will be eliminated every time after NMBO returns. Since there are only a finite number of variables, the framework will always terminate after eliminating all the variables. ■

Example 5. We continue with Example 3 to explain when NORTH terminates. After eliminating $\mathbf{c}_1$ at 5.999, $\mathbf{c}_2$ will be the only variable to optimize. NMBO will soon terminate at the schedulability boundary when $\mathbf{c}_2 = 15.89$, where the task τ_2's worst-case response time $r_2 = 39.89 < 40$. The algorithm does not terminate at $\mathbf{c}_2 = 15.99$ because the relative error difference between the last two iterations in NMBO is already smaller than 10^{-5} when $\mathbf{c}_2 = 15.89$.

At this time, VE will try to find variables to eliminate again. The old elimination tolerance $10^{-5} \times 1.5^{12}$ from Example 3 is inherited there, which cannot trigger eliminating any variables. Therefore, VE will try to increase the elimination tolerance d further until $d = 10^{-5} \times 1.5^{23} > 0.11$, where $\mathbf{c}_2$ will violate the schedulability constraint and be eliminated. After eliminating $\mathbf{c}_2$, no variables are left. Therefore, NORTH cannot make any progress further and terminates at $(5.999, 15.89)$. The final solution differs from but is close to the optimal solution $(6, 16)$.

F. Applications of Variable Elimination

VE could only bring performance improvements to non-local/global optimal solutions when these solutions locate near the boundary of feasible regions. In real-time systems where non-convex or black-box schedulability analyses are considered, VE will likely bring significant performance improvements because the optimal solution often lies at the schedulability boundary, and VE allows each variable to explore the schedulability boundary without interfering with each other.

A different way to view VE is that variables usually have different sensitivity to schedulability constraints. Correspondingly, we need to differentiate these variables (by dimension feasibility test) and make distinct adjustments (by variable elimination). An example is that each task provides one or a set of optimization variables and has its own deadline constraint. In this case, it may be rare that all the tasks meet their deadline simultaneously during iterations. However, VE could optimize each task separately until these tasks cannot be optimized further.

G. Limitations of Variable Elimination

It is possible that NORTH fails to find a feasible descent direction $\mathbf{\Delta}^{(k)} \in \mathbb{R}^N$ after NMBO terminates at a non-optimal point $\mathbf{x}^{(k)} \in \mathbb{R}^N$. Therefore, VE is utilized to search for a feasible descent direction in a smaller space $\mathbb{R}^{N-n}$, where n is the number of variables that have a fixed value in subsequent iterations (i.e., they were "eliminated" in the previous iterations). This means that VE sacrifices the potential performance improvements from these eliminated variables in favor of higher computation efficiency.

There are possibly other alternatives to variable elimination. For example, projecting the gradient along the boundaries of the schedulable region may be a better way to find feasible descent directions if the boundaries can be linearized. Such an idea is explored in classical gradient projection (GP) algorithms (described in [16, Chapter 16]). However, the high computation cost of the projection process in high-dimension space limits the application of GP primarily to constraints of simple forms. We leave the study of this enhancement to future work.

VI. Run-time complexity estimation

From the main framework Fig 1, the overall run time can be analyzed as follows:

$$\text{Cost} = N_{\text{Elimi}} \times (\text{Cost}_{\text{TR}} + \text{Cost}_{\text{Elimi}}) \tag{47}$$

where N_{Elimi} describes the number of elimination loops, Cost_{TR} describes the cost of the inner trust-region optimizer, and $\text{Cost}_{\text{Elimi}}$ describes the cost of detecting elimination variables.

For each of these items above, we have

$$N_{\text{Elimi}} \leq N \tag{48}$$

where N is the number of variables from Theorem 6.

$$\text{Cost}_{\text{TR}} \leq N_{\text{TR}} \times (\text{Cost}_{\text{sched}} + O(N^3)) \tag{49}$$

where $\text{Cost}_{\text{sched}}$ denotes the cost for schedulability analyses, the number of trust-region iterations N_{TR} is decided by the distance between the initial solution $\mathcal{F}(\mathbf{x}^{(0)})$ and the final solution $\mathcal{F}^*_{\mathbf{lb}}$:

$$N_{\text{TR}} \leq \frac{\|\mathcal{F}(\mathbf{x}^{(0)}) - \mathcal{F}^*_{\mathbf{lb}}\|}{\varepsilon} \tag{50}$$

and convergence speed of the unconstrained optimizer. Many gradient-based optimizers usually have super-linear or quadratic convergence speed [16], which are very fast! The last item in Equation (49) denotes the worst-case matrix-related computation. Finally, we have

$$\text{Cost}_{\text{Elimi}} \leq N \times \text{Cost}_{\text{sched}} \tag{51}$$

VII. Application and generalizations

In the previous sections, we use two assumptions 1 and 2 to develop our framework. In this section, we discuss how these two assumptions may be relaxed.

A. Finding initial solutions

Finding feasible initial solutions can follow a strategy that assigns variable values that are most "friendly" toward schedulability, e.g., the shortest execution time/longest period. Such a strategy is optimal in schedulability if sustainable schedulability analyses are used, e.g., the schedulability analysis based on (12) and (13).

However, it is possible to start with infeasible initial solutions if the penalty of violating schedulability and other constraints can be quantified. For example, in many real-time systems, tardiness (the extra time that tasks need to finish execution after passing deadlines) can be given by schedulability analyses. In this case, we can introduce a barrier function as follows:

$$w\text{Barrier}(D_i - r_i) = \begin{cases} 0 & D_i \geq r_i \\ -w(D_i - r_i) & D_i < r_i \end{cases} \tag{52}$$

where D_i and r_i are the deadline and worst-case response time of the task τ_i, and w is a punishment coefficient that is much larger than the possible values of the objective function. During the optimization process, the barrier function is expected to first reduce to zero such that all $r_i \leq D_i$ (i.e., the solution becomes schedulable). Then the rest of the optimization process would be similar to the case that starts with a feasible initial solution.

B. Optimizing Discrete Variables

NORTH may directly handle discrete variables in situations such as when an appropriate rounding strategy is available. For example, suppose all the constraints are monotonic (e.g., schedulability analysis is sustainable). In that case, floating-point variables can be rounded into adjacent integer values following a simple strategy: If a smaller value for the variable implies no worse system schedulability, we may round it down to ensure the schedulability; otherwise, it is rounded up.

Also, it may be possible to leverage the idea of the coordinate-descent algorithm [50] by optimizing continuous

and discrete variables separately. For example, for the co-optimization of task priorities and periods, we may use simple heuristics such as rate-monotonic policy for priority assignment and let NORTH handle the continuous variables such as task periods.

C. Limitations

NORTH may not work well if no useful gradient information can be obtained from the objective function, in which case gradient-based optimizers cannot work. For example, consider (14) as the control objective function and Equation (12) for response time calculation. If $\alpha_i \mathbf{T}_i$ is removed from the objective function, then the gradient of the objective function would be either 0 or undefined, and gradient-based optimizers will get stuck at the initial solution. Another typical example is binary variables, where no gradient can be evaluated. Since NORTH relies on the calculation of gradient, it also suffers from the common limitations, for example, it cannot guarantee to find global optimal solutions for general nonlinear optimization problems.

VIII. Experiments

The proposed framework is implemented using a numerical optimization library GTSAM [54] in C++, which is widely used in the robotics area, such as Simultaneous Localization and Mapping (SLAM) [15], [55] and motion planning [56], [57]. All the experiments are conducted on a standard desktop environment (Intel Core i7-11700 CPU, 16 GB Memory). The code is released[1].

Interior-point method (IPM), one of the most powerful constrained optimization methods [16], is used as a classical gradient-based method for comparison. The implementation is provided by a widely-applied optimization library IPOPT [58], [59], which also incorporates sophisticated heuristic optimization tricks. When working with IPOPT, the numerical gradient from (23) is used for non-differentiable functions. All the baseline methods are summarized as follows:

- Zhao20, the optimization framework proposed in [11]. In their experiments, the energy function is weighted by 10^9, and we keep this weight in all of our experiments. It finds the optimal solution if it terminates before time-out.
- MIGP, a mixed-integer geometric programming method to solve the example application problems. After formulating a MIGP, it is solved by the geometric programming solver gpposy [60] with BnB solver in YALMIP [61].
- IPM, implemented with IFOPT and IPOPT. If IPM cannot find a feasible solution, the initial solution is used for performance evaluation.
- NMBO_LM, NMBO method introduced in Section IV where LM (19) is the unconstrained optimizer.
- NORTH_LM, the optimization framework proposed in this paper, where LM (19) is the unconstrained optimizer.
- SA, simulated annealing, a standard general optimizer. We only compare with it when there are no good baseline methods. The cooling rate is 0.99, the temperature is 10^5, and the iteration limit is 10^6.

All the methods have the same time limit of 600 seconds.

A. Energy Optimization based on FTP Model

The same experiment settings as [11] are used in our experiments, where a simplified energy function is used:

$$E_i(\mathbf{x}) = \frac{H}{T_i}(\alpha \mathbf{f}_i^3) \times (\frac{c_i^{\text{org}}}{\mathbf{f}_i}) \tag{53}$$

where c_i^{org} denotes the WCETs for $\mathbf{f}_i = 1$. The task sets were generated randomly: system utilization was selected within the range [0.5, 0.9], period T_i was generated following log-uniform distribution within the range [100, 100000], and utilization of each task was generated using Uunifast [47]. The optimization variables were only allowed to be at most twice their original values, which means the clock rate can be decreased as low as half the base clock rate. The deadlines were the same as the task periods, and priorities were assigned according to Rate Monotonic (RM) policy. Schedulability analysis is given by Equation (12) and (13).

The performance results were shown in Fig. 3a, where the y-axis showed the relative gap between the baseline methods E_{baseline} against Zhao *et al.* [11]:

$$\frac{E_{\text{baseline}} - E_{\text{Zhao20}}}{E_{\text{Zhao20}}} \times 100 \tag{54}$$

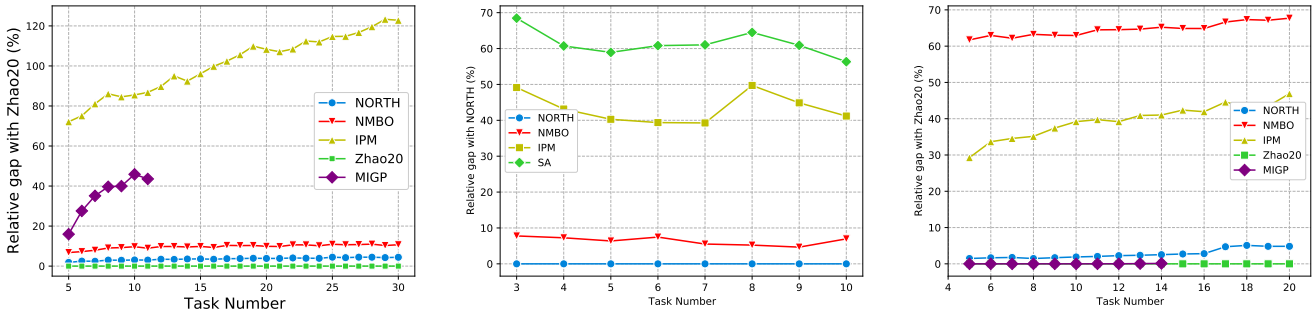
The average run-time speed is shown in Fig 3d.

B. Energy Optimization for DAG Model

A more complicated, but realistic schedulability analysis proposed by Nasri *et al.* [6] was also used with DVFS. It targets non-preemptive directed acyclic graph (DAG) applications under multicore systems. It is not sustainable with respect to tasks' worst-case execution time. The schedulability analysis cannot be modeled by either MIGP or Zhao20 (not sustainable); thus, we used simulated annealing in addition to NMBO and IPM as baseline methods.

Simulated task sets were generated following real-world automotive benchmark [9]. Each task set was composed of several periodic DAG tasks, and each DAG had several nodes, the number of which followed a uniform distribution between 1 and 20. The dependency relationships between each node inside a single DAG were generated following [62], where a directed edge was added from one node to another if a random number was smaller than a parallel factor, 0.2 in our experiments. The execution time of each node was generated using Uunifast [47] given the DAG task's utilization, while each DAG task's utilization was generated using a modified Uunifast algorithm. We modified Uunifast to cut each task's maximum utilization to be no larger than 100%; otherwise, the task sets would be un-schedulable following [6]. All the task sets were executed on a homogeneous 4-core computing platform. The task sets' period parameters were randomly selected according to an automotive task set benchmark [9] that is commonly used in literature to generate simulated task

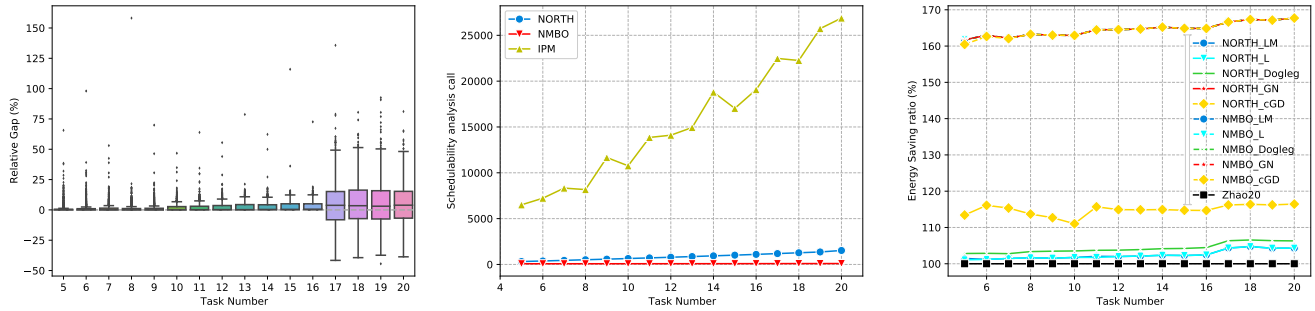
[1] https://github.com/zephyr06/RT_System_Design_Numerical_Methods



(a) DVFS performance subject to (12) (b) DVFS performance subject to [6] (c) Control performance optimization

(d) DVFS log run-time subject to (12) (e) DVFS log run-time subject to [6] (f) Control optimization log run-time

Figure 3: The upper and lower figures in each column show the performance and speed of the three experiments, respectively. Compared with state-of-art method Zhao20 [11], NLP_Elim_approx performed only 1% worse while running 10^2 to 10^5 times faster. The performance gap among different NLP methods validates the success of the proposed framework and variable elimination. The run-time report shows great scalability of NLP methods, which optimized large-scale task sets(e.g., $N = 80$) in less than 1 second.



(a) Box plot between NORTH and Zhao20 (b) RTA calling times (c) Different optimizers in NORTH

Figure 4: More results in the control performance application. (a) shows performance distribution between NORTH and [11]. (b) shows average RTA calling times. NORTH has an almost linear increasing rate. (c) shows the performance of 5 classical numerical optimizers in NMBO and NORTH.

sets [63], [64]. Considering that the response time analysis based on [6] is much more time-consuming than the RTA model (12), the task periods were randomly selected within a sub-set: $\{1, 2, 5, 10, 20, 50, 100\}$. For each given task number, we generated 105 task sets, among which 15 tasks were generated for each overall utilization from 0.1×4 to 0.9×4. During optimization, we increased the relative error tolerance to 10^{-3} and the initial elimination tolerance to 10.

The energy optimization was performed at the node level in each DAG task. That means there were 100 nodes on average to optimize when the task number was 10, 200 nodes at most. The results were reported in Fig 3b, and Fig 3e. The y-axis in Fig 3b showed the relative performance gap of the

energy saving ratio, which is the energy consumption ratio after optimization over the initial energy consumption.

C. Control Optimization

The problem of control performance optimization is described by Equations (14)-(16). This problem is more challenging than the energy optimization problem for numerical methods because the objective function involves non-differentiable response time analyses. The task sets were generated similarly as [11], [40]. They consisted of periodic tasks τ_i scheduled by RM on a uni-processor platform and had implicit deadlines. In the objective function (14), α_i was randomly generated in the range [1, 1000], β_i was generated in the range[1, 10000], while the worst-case execution time c_i of the task τ_i was from [1, 100]. The task periods were bounded by $T_i \leq 5\sum_{i=1}^{n} c_i$. Fig 3c, 3f and 4a respectively illustrated the average optimized control performance, the runtime of the optimization procedures, and the boxplot comparing NORTH and [11].

D. Discussions

The three experiments above showed that NORTH could maintain very similar performance (around $1\sim3\%$) as state-of-the-art methods while running $10^2 \sim 10^5$ times faster in sample applications. Compared with the classical optimization method IPM, NORTH showed significant improvements in performance and speed. Such advantages are primarily because the non-differentiable nature of schedulability constraint (4) prevents IPM from converging to a feasible solution. Finally, Fig 3d and 3b show that NORTH could optimize a larger range of real-time systems than [11], such as large-scale systems and systems with non-sustainable schedulability requirements. Next, we use the control experiment as an example to discuss some further common observations.

1) Performance statistics: Although NORTH had worse average-case performance than state-of-the-art methods [11] in small task sets, it showed great promise to outperform [11] in larger task sets given a limited time budget. Fig 4a showed the distribution of the relative gap between the proposed method and [11] in the control optimization. When $N \geq 17$, the proposed algorithm showed better performance in almost half the cases because of [11]'s time-out issue, even though [11] could possibly return the global optimal otherwise. As such, it can be expected that NORTH may perform even better than [11] as N becomes larger.

2) Computation cost related to schedulability analysis: Fig 4b showed the average time of calling schedulability analysis in each experiment, which is the most time-consuming part in NORTH. Some methods were not analyzed because their major cost includes other expensive components, such as a branch-and-bound searching procedure. From the figures, IPM basically followed a quadratic relationship with respect to the task number. In contrast, NORTH followed an almost linear relationship because it avoided the numerical gradient of the schedulability analysis. This indicated promising scalability properties of NORTH.

3) Improving classical numerical optimizers: As an optimization framework, NORTH improves upon various unconstrained optimizers. Fig 4c showed the performance comparison on five widely-applied unconstrained optimizers (Levenberg [25], Levenberg-Marquardt [26], Dog-leg [27], Gauss-Newton, and conjugated gradient descent) and their application with NORTH. Although different optimizers may influence NORTH's performance, NORTH achieved robust and reasonably good performance overall except GN, which cannot find update steps within the feasible region.

4) Obtaining feasible initial solutions: In all the experiments, we followed a simple strategy that used the shortest execution time and longest period possible as the initial solution. Such a strategy is optimal in the first and third experiments because the schedulability analysis is sustainable. However, in the second experiment where a non-sustainable schedulability analysis was used this strategy may fail to find a feasible initial solution. Therefore, we measured the chance that such a strategy provided feasible initial solutions. The settings were the same as in Section VIII-B, except that each task set only contains 3 DAG (10 nodes per DAG on average) to allow the examination of system schedulability. There is a time limit of 100s in finding feasible exeuction times. The task set will be discarded from the statistics if time-out. Overall, among 4500 random task sets, 3343 were schedulable, and our strategy finds a feasible initial solution in 96% of the schedulable cases. The distribution under different per-core utilization is shown in Table II.

Table II: Proportion of random DAG task sets that are schedulable by the shortest execution time subject to [6]

U(%)	10	20	30	40	50	60	70	80	90
Proportion (%)	100	100	100	100	97	93	88	72	47

IX. Conclusion

This paper proposes NORTH, a general and scalable optimization framework for real-time systems based on numerical methods. NORTH is designed to optimize with black-box schedulability constraints based on the idea of the classical active-set methods (ASM). However, NORTH differs from ASM in identifying/managing active constraints and maintaining feasibility for the schedulability constraints. We use two examples to demonstrate the advantages of our framework: one is the minimization of energy consumption, and the other is the optimization of control performance. Extensive experiments suggest that the framework may achieve very similar solution quality as state-of-the-art methods while running 10^2 to 10^5 times faster.

X. Acknowledgment

This work is partially supported by NSF Grants No. 1812963 and 1932074.

REFERENCES

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software engineering journal*, vol. 8, no. 5, pp. 284–292, 1993.

[2] S. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," *[1990] Proceedings 11th Real-Time Systems Symposium*, pp. 182–190, 1990.

[3] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, pp. 101–104 vol.4, 2000.

[4] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis–the symta/s approach," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[5] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, pp. 134–152, 1997.

[6] M. Nasri, G. Nelissen, and B. B. Brandenburg, "Response-time analysis of limited-preemptive parallel dag tasks under global scheduling," in *ECRTS*, 2019.

[7] L. Heintzman, A. Hashimoto, N. Abaid, and R. K. Williams, "Anticipatory planning and dynamic lost person models for Human-Robot search and rescue," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8252–8258, ieeexplore.ieee.org, May 2021.

[8] PerceptIn, "2021 rtss industry challenge." http://2021.rtss.org/industry-session/, 2021.

[9] A. H. Simon Kramer, Dirk Ziegenbein, "Real world automotive benchmarks for free," 2015.

[10] Y. Zhao, V. Gala, and H. Zeng, "A unified framework for period and priority optimization in distributed hard real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2188–2199, 2018.

[11] Y. Zhao, R. Zhou, and H. Zeng, "An optimization framework for real-time systems with sustainable schedulability analysis," *2020 IEEE Real-Time Systems Symposium (RTSS)*, pp. 333–344, 2020.

[12] Y. Zhao, R. Zhou, and H. Zeng, "Design optimization for real-time systems with sustainable schedulability analysis," *Real-Time Systems*, vol. 58, no. 3, pp. 275–312, 2022.

[13] S. Baruah and A. Burns, "Sustainable scheduling analysis," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pp. 159–168, IEEE, 2006.

[14] A. Burns and S. Baruah, "Sustainability in real-time scheduling," *J. Comput. Sci. Eng.*, vol. 2, pp. 74–97, 2008.

[15] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Found. Trends Robotics*, vol. 6, pp. 1–139, 2017.

[16] Nocedal and J. Wright, *Numerical Optimization, 2nd edition*. Springer New York, NY, 2006.

[17] M. Shin and M. Sunwoo, "Optimal period and priority assignment for a networked control system scheduled by a fixed priority scheduling system," *International Journal of Automotive Technology*, vol. 8, pp. 39–48, 2007.

[18] K. Tindell, A. Burns, and A. Wellings, "Allocating hard real-time tasks: An np-hard problem made easy," *Real-Time Systems*, vol. 4, pp. 145–165, 2004.

[19] J. Jonsson and K. Shin, "A parametrized branch-and-bound strategy for scheduling precedence-constrained tasks on a multiprocessor system," *Proceedings of the 1997 International Conference on Parallel Processing (Cat. No.97TB100162)*, pp. 158–165, 1997.

[20] M. Natale, L. Guo, H. Zeng, and A. Sangiovanni-Vincentelli, "Synthesis of multitask implementations of simulink models with minimum delays," *IEEE Transactions on Industrial Informatics*, vol. 6, pp. 637–651, 2010.

[21] H. Zeng and M. Di Natale, "Efficient implementation of autosar components with minimal memory usage," in *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pp. 130–137, IEEE, 2012.

[22] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pp. 313–322, 2006.

[23] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Trans. Embed. Comput. Syst.*, vol. 15, pp. 7:1–7:34, 2016.

[24] Y. Zhao and H. Zeng, "The virtual deadline based optimization algorithm for priority assignment in fixed-priority scheduling," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, pp. 116–127, IEEE, 2017.

[25] K. Levenberg, "A method for the solution of certain non – linear problems in least squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.

[26] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of The Society for Industrial and Applied Mathematics*, vol. 11, pp. 431–441, 1963.

[27] M. POWELL, "A new algorithm for unconstrained optimization," in *Nonlinear Programming* (J. Rosen, O. Mangasarian, and K. Ritter, eds.), pp. 31–65, Academic Press, 1970.

[28] S. J. Reddi, S. Sra, B. Póczos, and A. Smola, "Stochastic frank-wolfe methods for nonconvex optimization," *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1244–1251, 2016.

[29] M. V. Balashov, B. Polyak, and A. A. Tremba, "Gradient projection and conditional gradient methods for constrained nonconvex minimization," *Numerical Functional Analysis and Optimization*, vol. 41, pp. 822 – 849, 2019.

[30] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Comput. J.*, vol. 7, pp. 308–313, 1965.

[31] S. Lee, H. Baek, H. Woo, K. G. Shin, and J. Lee, "Ml for rt: Priority assignment using machine learning," *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 118–130, 2021.

[32] Z. Bo, Y. Qiao, C. Leng, H. Wang, C. Guo, and S. Zhang, "Developing real-time scheduling policy by deep reinforcement learning," *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 131–142, 2021.

[33] F. F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pp. 374–382, 1995.

[34] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001.

[35] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pp. 52–62, 2003.

[36] C.-H. Lee and K. Shin, "On-line dynamic voltage scaling for hard real-time systems using the edf algorithm," *25th IEEE International Real-Time Systems Symposium*, pp. 319–335, 2004.

[37] E. Bini, G. Buttazzo, and G. Lipari, "Minimizing cpu energy in real-time systems with discrete speed management," *ACM Trans. Embed. Comput. Syst.*, vol. 8, pp. 31:1–31:23, 2009.

[38] A. Bhuiyan, F. Reghenzani, W. Fornaciari, and Z. Guo, "Optimizing energy in non-preemptive mixed-criticality scheduling by exploiting probabilistic information," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, pp. 3906–3917, 2020.

[39] A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, N. Guan, and Z. Guo, "Energy-efficient parallel real-time scheduling on clustered multi-core," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, pp. 2097–2111, 2020.

[40] G. M. Mancuso, E. Bini, and G. Pannocchia, "Optimal priority assignment to control tasks," *ACM Trans. Embed. Comput. Syst.*, vol. 13, pp. 161:1–161:17, 2014.

[41] E. Bini and M. D. Natale, "Optimal task rate selection in fixed priority systems," *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, pp. 11 pp.–409, 2005.

[42] A. Davare, Q. Zhu, M. D. Natale, C. Pinello, S. Kanajan, and A. L. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," *2007 44th ACM/IEEE Design Automation Conference*, pp. 278–283, 2007.

[43] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-efficient real-time scheduling of dags on clustered multi-core platforms," *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 156–168, 2019.

[44] S. Pagani and J.-J. Chen, "Energy efficient task partitioning based on the single frequency approximation scheme," *2013 IEEE 34th Real-Time Systems Symposium*, pp. 308–318, 2013.

[45] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient dvfs scheduling for mixed-criticality systems," *2014 International Conference on Embedded Software (EMSOFT)*, pp. 1–10, 2014.

[46] M. Joseph and P. K. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, pp. 390–395, 1986.
[47] R. I. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *IEEE Transactions on Computers*, vol. 57, pp. 1261–1276, 2008.
[48] F. Dorin, P. Richard, M. Richard, and J. Goossens, "Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities," *Real-Time Systems*, vol. 46, pp. 305–331, 2010.
[49] P. B. Betoret, I. Ripoll, and A. Crespo, "Minimum deadline calculation for periodic real-time tasks in dynamic priority systems," *IEEE Transactions on Computers*, vol. 57, pp. 96–109, 2008.
[50] J. H. Friedman, T. J. Hastie, H. Hofling, and R. Tibshirani, "Pathwise coordinate optimization," *The Annals of Applied Statistics*, vol. 1, pp. 302–332, 2007.
[51] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, pp. 46–61, 1973.
[52] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," *2012 IEEE 33rd Real-Time Systems Symposium*, pp. 63–72, 2012.
[53] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.
[54] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," in *Factor Graphs and GTSAM: A Hands-on Introduction*, 2012.
[55] M. Kaess, A. Ranganathan, and F. Dellaert, "isam: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, pp. 1365–1378, 2008.
[56] M. Mukadam, J. Dong, F. Dellaert, and B. Boots, "Steap: simultaneous trajectory estimation and planning," *Autonomous Robots*, pp. 1–20, 2018.
[57] S. Wang, J. Chen, X. Deng, S. A. Hutchinson, and F. Dellaert, "Robot calligraphy using pseudospectral optimal control in conjunction with a novel dynamic brush model," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6696–6703, 2020.
[58] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006.
[59] A. W. Winkler, "Ifopt - A modern, light-weight, Eigen-based C++ interface to Nonlinear Programming solvers Ipopt and Snopt.," 2018.
[60] A. Mutapcic, K. Koh, S. Kim, and S. Boyd, "Ggplab version 1.00: a matlab toolbox for geometric programming," 2006.
[61] J. Lofberg, "Yalmip : a toolbox for modeling and optimization in matlab," *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*, pp. 284–289, 2004.
[62] Q. He, M. Lv, and N. Guan, "Response time bounds for dag tasks with arbitrary intra-task priority assignment," in *ECRTS*, 2021.
[63] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate dags from multi-rate task sets," *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 226–238, 2020.
[64] S. Bozhko, G. von der Bruggen, and B. B. Brandenburg, "Monte carlo response-time analysis," *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021.