Time-Triggered Scheduling for Nonpreemptive Real-Time DAG Tasks Using 1-Opt Local Search

Sen Wang, *Graduate Student Member, IEEE*, Dong Li, *Graduate Student Member, IEEE*, Shao-Yu Huang, Xuanliang Deng, Ashrarul H. Sifat, Jia-Bin Huang, Changhee Jung, *Senior Member, IEEE*, Ryan Williams, *Member, IEEE*, and Haibo Zeng

***Abstract*—Modern real-time systems often involve numerous computational tasks characterized by intricate dependency relationships. Within these systems, data propagate through cause–effect chains from one task to another, making it imperative to minimize end-to-end latency to ensure system safety and reliability. In this article, we introduce innovative nonpreemptive scheduling techniques designed to reduce the worst-case end-to-end latency and/or time disparity for task sets modeled with directed acyclic graphs (DAGs). This is challenging because of the noncontinuous and nonconvex characteristics of the objective functions, hindering the direct application of standard optimization frameworks. Customized optimization frameworks aiming at achieving optimal solutions may suffer from scalability issues, while general heuristic algorithms often lack theoretical performance guarantees. To address this challenge, we incorporate the "1-opt" concept from the optimization literature (Essentially, 1-opt means that the quality of a solution cannot be improved if only one single variable can be changed) into the design of our algorithm. We propose a novel optimization algorithm that effectively balances the tradeoff between theoretical guarantees and algorithm scalability. By demonstrating its theoretical performance guarantees, we establish that the algorithm produces 1-opt solutions while maintaining polynomial run-time complexity. Through extensive large-scale experiments, we demonstrate that our algorithm can effectively reduce the latency metrics by 20% to 40%, compared to state-of-the-art methods.**

***Index Terms*—End-to-end latency, optimization, real-time system, scheduling, time-triggered scheduling (TTS).**

I. Introduction

ENSURING timeliness, short end-to-end latency, and small data communication time disparity is a paramount consideration across various domains, including control engineering, body electronics, and automotive systems [1]. For example, the RTSS2021 Industry Challenge [2] underscores the importance of bounding worst-case end-to-end latency and time disparity in *nonpreemptive* autonomous driving systems. Nonpreemptive systems are becoming more popular due to the wide adoption of single-instruction-multidata (SIMD) computing architectures such as GPU. Since preemption with GPU usually has a much higher overhead than CPU devices, embedded GPU devices often only provide limited, if any, support for preemption [3].

Scheduling and optimizing systems with respect to data age, reaction time, and time disparity (DARTD)[1] pose significant challenges [1], [4], [5], [6], [7], [8] due to their nonconvex and noncontinuous characteristics. These attributes hinder the application of standard mathematical programming frameworks, such as integer linear programming (ILP) and convex optimization. However, naively employing highly general optimization frameworks like meta-heuristics often lacks theoretical performance guarantees. Conversely, developing customized frameworks targeted at yielding optimal solutions [7] encounters scalability issues, which is particularly important in modern computation systems, where hundreds of computation tasks may exist [9], [10]. To tackle these challenges, we propose a computationally efficient optimization algorithm with some theoretical performance guarantees.

In this article, we leverage the *1-opt* concept, drawn from the optimization literature [11], [12] as a foundation in the development of our optimization algorithm. A solution vector $\boldsymbol{x} \in \mathbb{R}^N$ for an optimization problem is called 1-opt if changing any single component $\boldsymbol{x}_i \in \boldsymbol{x}$ does not result in an improvement beyond the current solution $\boldsymbol{x}$. We refer to algorithms that yield 1-opt solutions as 1-opt algorithms. In contrast to heuristic algorithms, 1-opt algorithms provide stronger theoretical performance guarantees. Moreover, they often demonstrate superior scalability when compared to algorithms aimed at finding optimal solutions.

Nevertheless, constructing 1-opt algorithms for optimizing nonconvex and noncontinuous metrics, such as DARTD, is very challenging. Naively employing brute-force algorithms can result in exponential complexity in worst-case scenarios. To address this, we propose a novel algorithm that employs

Manuscript received 2 August 2024; accepted 3 August 2024. This work was supported by NSF under Grant 1932074. This article was presented at the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Sen Wang and Dong Li contributed equally to this work.) (Corresponding author: Haibo Zeng.)*

The authors are with the Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061 USA, also with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA, and also with the Department of Computer Science, University of Maryland at College Park, College Park, MD 20742 USA (e-mail: swang666@vt.edu; dongli@vt.edu; huan1464@purdue.edu; xuanliang@vt.edu; ashrar7@vt.edu; jbhuang@umd.edu; chjung@purdue.edu; rywilli1@vt.edu; hbzeng@vt.edu).

Digital Object Identifier 10.1109/TCAD.2024.3442985

[1] Given a cause–effect chain, data age measures the maximum duration for which a sensor event influences the computational system, while reaction time measures the maximum latency for the system to first react to a sensor event. Additionally, time disparity quantifies the maximum difference in the generation times of multiple source data from which one task reads input.

a technique to partition the solution space into multiple convex subspaces, allowing for the efficient utilization of linear programming (LP) to minimize DARTD within each subspace. Subsequently, an iterative subroutine efficiently traverses among the subspaces, ensuring that the output is 1-opt. Furthermore, we prove that the solution of each LP is local optimal in nonpreemptive single-core systems. In comparison with simple scheduling heuristics, such as list scheduling [13], scheduling with LP can explore a much larger solution space, leading to enhanced performance. Moreover, the polynomial run-time complexity of solving LP enhances algorithm scalability compared to optimal algorithms that exhibit exponential run-time complexities in the worst case. Finally, to further improve the efficiency of LP, we propose an algorithm capable of efficiently performing nonpreemptive schedulability analysis.

Contributions: Our contributions in this article are as follows.

1) We employ the 1-opt concept in the development of schedule optimization algorithms. To the best of our knowledge, this is the first work to utilize the 1-opt concept in real-time system scheduling problems, and it achieves superior performance compared to state-of-the-art methods.
2) We propose a novel optimization framework designed to minimize worst-case DARTD, which is proven to yield 1-opt solutions with only polynomial run-time complexity.
3) To the best of our knowledge, this is the first work that considers optimizing time disparity with time-triggered scheduling (TTS).
4) Large-scale experiments demonstrate that 1-opt methods achieve 20% to 40% latency reductions and enhanced scalability compared to state-of-the-art techniques.

II. Related Work

As an important indicator of system safety, end-to-end latency has been thoroughly studied. Numerous analyses have delved into cause–effect chains or task sets structured with directed acyclic graphs (DAGs) dependency [1], [4], [5], [7], [8], [14], [15]. These analytical approaches address diverse scenarios, including different scheduling algorithms (e.g., fixed-priority scheduling and earliest deadline first scheduling) and communication protocols (e.g., implicit communication and logical execution time (LET)). Moreover, some studies explore temporal variations across various contexts [16], [17]. Beyond the analysis of end-to-end latency, a considerable body of work focuses on scheduling and the schedulability of DAG task sets [18], [19], [20], [21]. These comprehensive analyses build the foundation for the optimization works performed in this article.

General optimization techniques in real-time systems can be broadly categorized into two categories: 1) heuristic algorithms with general applicability but lacking solution quality guarantees [10], [22] and 2) optimal algorithms built with sophisticated assumptions and problem modeling [7], [23], [24]. However, the latter may encounter scalability issues when facing large-scale optimization problems and the performance may also degrade seriously. Considering the challenge of finding the "perfect" algorithms (optimal and fast) for many real-world problems, algorithm designers often face a tradeoff between solution quality and run-time complexity.

There are many works that optimize the end-to-end latency with different types of variables. Within the LET protocol, many works consider optimizing the time to read/write data, where both optimal [25], [26] and heuristic [27], [28] algorithms have been proposed. Some other works consider implicit communication protocol, primarily concentrating on optimizing task schedules [7]. Besides, there are also works that improve different metrics related to end-to-end latency by performing priority assignments [29], [30].

This article differs from existing literature in proposing to use a new concept, 1-opt, to guide the algorithm design process. We also designed a novel optimization algorithm which is proved to find 1-opt solutions and demonstrated to achieve significantly better performance than the state-of-the-art methods.

III. System Model and Problem Description

In this article, bold fonts are used to represent vectors or sets, while light characters denote scalars or individual elements. The double bars notation $\|\|$ denotes norm-2. During iterations, the kth iteration is denoted by a superscript, such as $\boldsymbol{x}^{(k)}$.

A. System Model

We consider a multirate DAG model $\mathcal{G} = (\boldsymbol{\tau}, \boldsymbol{E})$, in which each task $\tau_i \in \boldsymbol{\tau}$ is represented as a node, and a directed edge $E_k \in \boldsymbol{E}$ from τ_i to τ_j denotes that τ_j reads input from τ_i. The total number of tasks in $\boldsymbol{\tau}$ is denoted as n. Each task releases jobs (i.e., *instances of the task*) periodically with a nominal period. A task τ_i is characterized by a tuple $\{T_i, C_i, D_i\}$, which denotes the period, worst-case execution time (WCET), and the relative deadline, respectively. We assume $D_i \leq T_i$. The kth released job of τ_i is denoted as $J_{i,k}$ and it is released at the time $k \cdot T_i$. The DAG $\mathcal{G}$ is not necessarily fully connected. Without loss of generality, we assume all the tasks are released simultaneously at time 0. However, if there is an offset when all the tasks are initially released, our optimization algorithm can also be applied by modifying the schedulability analysis algorithms and optimization constraints accordingly.

The hyper-period (i.e., the least common multiple of periods of all tasks in $\mathcal{G}$) is denoted as H. Within a hyper-period, each job $J_{i,k}$ starts execution at time $s_{i,k}$ *nonpreemptively* and finishes at $f_{i,k} = s_{i,k} + C_i$. Such a nonpreemptive policy eliminates preemption overhead, which could be large in GPU computation. The total number of jobs within a hyper-period is denoted as N. Potential generalizations into preemptive systems are discussed in Section VIII-B.

In a DAG $\mathcal{G}$, tasks with chained reading/writing dependency formulate a cause–effect chain $\mathcal{C} = \{\tau_{p_0} \rightarrow \tau_{p_1} \rightarrow \cdots \rightarrow \tau_{p_k}\}$, which represents a data communication path. The implicit communication protocol [31] is utilized in data communication where each job $J_{i,k}$ reads data at its start time $s_{i,k}$, and writes data at $f_{i,k} = s_{i,k} + C_i$ even if $J_{i,k}$ may finish earlier than its WCET. Multiple cause–effect chains may share tasks, and the set of cause–effect chains is denoted as $\boldsymbol{\mathcal{C}}$.

In scenarios where a single task reads data from the outputs of multiple tasks, we refer to the tasks providing data as the source tasks, and the task that reads these outputs as the sink task. The source tasks and the sink task collectively formulate a "merge" $\mathcal{M}$ (For example, see Example 1). The set containing all merges to be optimized is denoted as $\boldsymbol{\mathcal{M}}$.

The DAG task set is processed by a multiprocessor system. We assume that each job has a known processor assignment before performing the schedule optimization, and we do not consider processor migration during execution. For presentation simplicity, we assume using a homogeneous multiprocessor system. However, the heterogeneous computation can be handled easily by modifying the resource-bound constraint correspondingly after obtaining processor assignments. In experiments, the processor is assigned following the First-Come-First-Serve heuristic, same as Verucchi et al. [7] for a fair comparison. The proposed optimization framework does not optimize processor assignments.

B. General Schedule Optimization Problem Formulation

We consider the schedule optimization problem of time-triggered systems, focusing on reducing the worst-case end-to-end latency and/or time disparity. The optimization variables for our scheduling problem are called a schedule.

Definition 1 (Schedule): Given a DAG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\tau}, \boldsymbol{E})$, a schedule $\boldsymbol{s} \in \mathbb{R}^N$ is a vector of the start time of all jobs of all tasks in $\boldsymbol{\tau}$ within a hyper-period H.

A general schedule optimization problem consists of an objective function and a set of schedulability constraints

$$\underset{\boldsymbol{s}}{\text{Minimize}}\mathcal{F}(\boldsymbol{s}) \tag{1}$$

$$\text{Subject to:}$$

$$\forall i \in \{0, \ldots, n-1\} \quad \forall k \in \{0, \ldots, H/T_i - 1\}$$

$$k \cdot T_i \leq s_{i,k} \leq k \cdot T_i + D_i - C_i \tag{1a}$$

$$\text{ResourceBound}(\boldsymbol{s}) = 0. \tag{1b}$$

Constraint (1a) guarantees every job starts and finishes within its schedulable range. The resource bound (1b) specifies that no computation resources are overloaded (e.g., one CPU core executes more than one job simultaneously). The specific form of (1b) will be introduced later in Section III-E. A schedule $\boldsymbol{s}$ is feasible (or equivalently, schedulable) if it satisfies (1a) and (1b). Given a schedule $\boldsymbol{s}$, the finish time $f_{i,k}$ of each job $J_{i,k}$ in nonpreemptive systems is implicitly decided: $f_{i,k} = s_{i,k} + C_i$.

C. Example Problem—End-to-End Latency Optimization

Each cause–effect chain $\mathcal{C}$ could trigger multiple job chains within a hyper-period. The worst-case data age (reaction time) of a cause–effect chain $\mathcal{C}$ is the length of its longest immediate backward (forward) job chain [5], [6]. These definitions are briefly reviewed below:

Definition 2 (Job Chain [5], [6]): Given a cause–effect chain $\mathcal{C} = \{\tau_{p_0} \to \tau_{p_1} \to \cdots \to \tau_{p_k}\}$, a job chain $\mathcal{C}^J$ is a sequence of jobs $\{J_{p_0,q_0} \to J_{p_1,q_1} \to \cdots \to J_{p_k,q_k}\}$, where

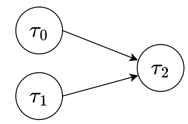

Fig. 1. Example DAG.

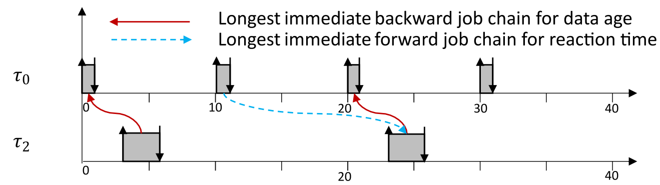

Fig. 2. Longest immediate forward and backward job chains for cause–effect chain $\mathcal{C} = \{\tau_0 \to \tau_2\}$.

J_{p_i,q_i} is the q_i^{th} job of τ_{p_i}, and the data produced by J_{p_i,q_i} is read by $J_{p_{i+1},q_{i+1}}$.

Definition 3 (Length of a Job Chain): The length of a job chain $\mathcal{C}^J = \{J_{p_0,q_0} \to J_{p_1,q_1} \to \cdots \to J_{p_k,q_k}\}$ is the time interval from the start time of J_{p_0,q_0} till the finish time of J_{p_k,q_k}. It is denoted as $L(\mathcal{C}^J) = f_{p_k,q_k} - s_{p_0,q_0}$.

Definition 4 (Immediate Backward (Forward) Job Chain [5], [6]): A job chain $\mathcal{C}^J = \{J_{p_0,q_0} \to J_{p_1,q_1} \to \cdots \to J_{p_k,q_k}\}$ is the immediate backward (forward) chain under schedule $\boldsymbol{s}$ if (2) (3) is satisfied

$$\forall i \in \{1, \ldots, k\}, \quad f_{p_{i-1},q_{i-1}} \leq s_{p_i,q_i} < f_{p_{i-1},(q_{i-1}+1)} \tag{2}$$

$$\forall i \in \{0, \ldots, k-1\}, \quad s_{p_{i+1},(q_{i+1}-1)} < f_{p_i,q_i} \leq s_{p_{i+1},q_{i+1}}. \tag{3}$$

Example 1: Fig. 1 shows a simple DAG with three tasks: $\boldsymbol{\tau} = \{\tau_0, \tau_1, \tau_2\}$ and two edges: $\boldsymbol{E} = \{\tau_0 \to \tau_2, \tau_1 \to \tau_2\}$. The WCET, period, and relative deadline of each task is: $\{C_0 = 1, T_0 = 10, D_0 = 10\}$, $\{C_1 = 2, T_1 = 20, D_1 = 20\}$, $\{C_2 = 3, T_2 = 20, D_2 = 20\}$. The task set is executed on 2 identical processors unless otherwise stated. The hyper-period is 20. The schedule variable contains the start time of $N = 4$ jobs: $\boldsymbol{s} = [s_{0,0}, s_{0,1}, s_{1,0}, s_{2,0}]$.

Suppose we have a schedule $\boldsymbol{s} = [0, 10, 1, 3]$. For the cause–effect chain $\mathcal{C} = \{\tau_0 \to \tau_2\}$, the job chain $\mathcal{C}_0^J = \{J_{0,0} \to J_{2,0}\}$ is both an immediate backward job chain and immediate forward job chain with length $L(\mathcal{C}_0^J) = 6$. $\mathcal{C}_1^J = \{J_{0,1} \to J_{2,1}\}$ is another immediate forward job chain with length $L(\mathcal{C}_1^J) = 16$. Thus, $\max \boldsymbol{DA}_{\mathcal{C}}(\boldsymbol{s}) = 6$, $\max \boldsymbol{RT}_{\mathcal{C}}(\boldsymbol{s}) = 16$. The longest job chains for this scenario are shown in Fig. 2.

Given a schedule $\boldsymbol{s}$, we use $\boldsymbol{DA}_{\mathcal{C}}(\boldsymbol{s})$ ($\boldsymbol{RT}_{\mathcal{C}}(\boldsymbol{s})$) to denote the vector of data age (reaction time) for all job chains of a cause–effect chain $\mathcal{C}$ within a hyper-period.

To summarize, when optimizing the worst-case data age or reaction time, the objective function in (1) becomes

$$\mathcal{F}(\boldsymbol{s}) = \sum_{\mathcal{C} \in \boldsymbol{\mathcal{C}}} \max \boldsymbol{DA}_{\mathcal{C}}(\boldsymbol{s}) \tag{4}$$

or

$$\mathcal{F}(\boldsymbol{s}) = \sum_{\mathcal{C} \in \boldsymbol{\mathcal{C}}} \max \boldsymbol{RT}_{\mathcal{C}}(\boldsymbol{s}). \tag{5}$$

D. Example Problem—Time Disparity Optimization

Similar to a cause–effect chain, a merge $\mathcal{M}$ may have multiple job-level merges.

Definition 5 (Job Merge): A job merge $\mathcal{M}^J$ contains a sink job $J_{j,l}$ and a set of source jobs $\boldsymbol{J}_{j,l}^{Src}$, from which $J_{j,l}$ directly reads data

$$\forall J_{i,k} \in \boldsymbol{J}_{j,l}^{Src}, \quad f_{i,k} \leq s_{j,l} < f_{i,k+1}. \tag{6}$$

Definition 6 (Time Disparity [2], [26]): The time disparity of a job merge $\mathcal{M}^J$, denoted as $TD(\mathcal{M}^J)$, is defined as the difference between the earliest and latest finish times of all source jobs in $\mathcal{M}^J$

$$TD\left(\mathcal{M}^J\right) = \max_{J \in \boldsymbol{J}_{j,l}^{Src}} f_J - \min_{J \in \boldsymbol{J}_{j,l}^{Src}} f_J \tag{7}$$

where f_J represents the finish time of a job J.

Given a schedule $\boldsymbol{s}$, we use $\boldsymbol{TD}_{\mathcal{M}}(\boldsymbol{s})$ to denote the vector of time disparities for all job merges of $\mathcal{M}$ within a hyper-period. When optimizing the worst-case time disparity metric, the objective function in (1) is formulated as follows:

$$\mathcal{F}(\boldsymbol{s}) = \sum_{\mathcal{M} \in \boldsymbol{\mathcal{M}}} \max \boldsymbol{TD}_{\mathcal{M}}(\boldsymbol{s}). \tag{8}$$

Other forms of the objective functions are discussed in Section VIII-A.

Example 2: In Example 1, there is only one merge $\mathcal{M}$ in the DAG with τ_2 as the sink task. The corresponding job merge has $J_{2,0}$ as the sink job and $\{J_{0,0}, J_{1,0}\}$ as the source jobs. The maximum time disparity is $\max \boldsymbol{TD}_{\mathcal{M}}(\boldsymbol{s}) = 3 - 1 = 2$.

Theorem 1: The objective functions (4), (5), and (8) are all nonconvex.

Proof: We prove it by providing counter-examples. Remember that a function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for all $\boldsymbol{s}^{(1)}$ and $\boldsymbol{s}^{(2)}$ in its domain and $\forall t \in [0, 1]$, we have $f(t\boldsymbol{s}^{(1)} + (1 - t)\boldsymbol{s}^{(2)}) \leq tf(\boldsymbol{s}^{(1)}) + (1-t)f(\boldsymbol{s}^{(2)})$. We now give a counterexample for reaction time, and the counterexamples for data age and time disparity are similar. In Example 1, consider $\boldsymbol{s}^{(1)} = [0, 10, 1, 3]$ with a reaction time 16, $\boldsymbol{s}^{(2)} = [0, 10, 1, 11]$ whose reaction time is 14. If we define $t = 0.5$, then $\boldsymbol{s}^{(t)} = t\boldsymbol{s}^{(1)} + (1 - t)\boldsymbol{s}^{(2)} = [0, 10, 1, 7]$, but the reaction time of $\boldsymbol{s}^{(t)}$ is 20, which violates the property required by convex functions. ■

E. Resource Bound Constraint—Interval Overlapping Test

In a nonpreemptive system, the interval overlapping Test (IO Test) analyzes whether processors are overloaded (one processor executes multiple jobs in parallel) for a given schedule $\boldsymbol{s}$. In this case, each job $J_{i,k}$ can be modeled as an interval $[s_{i,k}, f_{i,k}]$ that starts execution at $s_{i,k}$ and finishes at $f_{i,k} = s_{i,k} + C_i$. Inspired by the demand bound function [32], we propose an efficient nonpreemptive schedulability analysis for optimization. Intuitively speaking, there are no overloaded processors if any two job intervals mapped to the same processor do not overlap.

Theorem 2 (IO Test): In nonpreemptive systems, there are no overloaded processors if the following inequality holds for any two jobs $J_{i,k}$ and $J_{j,l}$ assigned to the same processor:

$$\text{if } f_{j,l} \geq s_{i,k}, \text{ then } f_{j,l} - s_{i,k} \geq C_i + C_j. \tag{9}$$

Proof: Prove by contradiction. If there are overloaded processors, by definition, there must be two job execution intervals overlapping with each other. Let us denote the job with a larger finish time as $J_{j,l}$, the other job as $J_{i,k}$, then we have

$$f_{j,l} - s_{i,k} < C_i + C_j. \tag{10}$$

This contradicts the IO test assumption above. ■

Theorem 3: Given a set of job intervals $\boldsymbol{I} = \{[s_{i,k}, f_{j,l}]\}$ sorted based on its start time $s_{i,k}$ in increasing order, no intervals overlap with each other if any two adjacent job intervals do not overlap with each other.

Proof: Skipped. It can be proved easily by contradiction. ■

Since a schedule will repeat in every hyper-period, the IO test only needs to consider all jobs within a hyper-period. Within partitioned scheduling, each processor has to be tested separately. The time complexity of the IO test is $O(N\log(N))$.

Example 3: Let us continue with the task set in Example 1. Suppose we only have one processor and have a schedule $\boldsymbol{s} = [0, 10, 1, 3]$. If without sorting, the IO test requires verifying whether the following six pairs of intervals overlap:

$$\begin{aligned}
&\{[s_{0,0}, f_{0,0}], [s_{0,1}, f_{0,1}]\} \quad \{[s_{0,0}, f_{0,0}], [s_{1,0}, f_{1,0}]\} \\
&\{[s_{0,0}, f_{0,0}], [s_{2,0}, f_{2,0}]\} \quad \{[s_{0,1}, f_{0,1}], [s_{1,0}, f_{1,0}]\} \\
&\{[s_{0,1}, f_{0,1}], [s_{2,0}, f_{2,0}]\} \quad \{[s_{1,0}, f_{1,0}], [s_{2,0}, f_{2,0}]\}.
\end{aligned}$$

With sorting, only the following three pairs require verification:

$$\begin{aligned}
&\{[s_{0,0}, f_{0,0}], [s_{1,0}, f_{1,0}]\} \quad \{[s_{1,0}, f_{1,0}], [s_{2,0}, f_{2,0}]\} \\
&\{[s_{2,0}, f_{2,0}], [s_{0,1}, f_{0,1}]\}.
\end{aligned}$$

If there is no overlap, then the IO test states that the processor is not overloaded.

Now, we can give the complete form of the resource bound (1b) in nonpreemptive systems

$$\text{ResourceBound}(\boldsymbol{s}) = \begin{cases} 0, & \text{if } \boldsymbol{s} \text{ passes IO test} \\ 1, & \text{otherwise.} \end{cases} \tag{11}$$

F. Model Assumptions

Assumption 1: The start time of each job could take continuous value.

Although the computer time is integer multiples of CPU cycles, the very high-CPU run-time frequency (MHz or GHz) means that rounding a float-point number into its adjacent integers only incurs a small precision loss in timing metrics, if the jobs' relative reading/writing time order remains the same.

Assumption 2: A feasible schedule [a solution that satisfies (1a) and (1b)] is available to start the iterative algorithms introduced next.

Normally, Assumption 2 can be easily satisfied with simple list schedulers [7]. This article focuses on optimizing the timing metrics rather than finding a schedulable schedule, although such an extension is possible (see Section VIII-C).

G. Challenges

Solving the optimization problem (1) for DARTD is difficult because the objective function follows a *nonlinear, nonmonotonic, nonconvex, and noncontinuous* relationship with the variables (see Theorem 1 and its proof). Therefore, most popular optimization frameworks cannot be directly utilized except ILP. However, ILP requires introducing many extra binary variables and could suffer from bad algorithm scalability.

IV. Job Order and Scheduling

The proposed optimization framework that solves the problem (1) is built upon the concept of the job order, which specifies the jobs' reading/writing relationships and simplifies the problem into a set of LP problems.

A. Job Order

Definition 7 (Job Scheduling Time): The job scheduling time of a job $J_{i,k}$ is denoted as $\mathcal{T}_{i,k}$, which could be either the start time (denoted as $\mathcal{T}^s_{i,k}$, called *scheduling start time*) or the finish time (denoted as $\mathcal{T}^f_{i,k}$, called *scheduling finish time*) of $J_{i,k}$.

Since we adopt the implicit communication protocol and nonpreemptive scheduling, a job $J_{i,k}$'s reading time is its start time, and its writing time is its finish time.

Example 4: In Example 1, consider a schedule $\boldsymbol{s} = [0, 10, 1, 3]$. The job $J_{0,0}$ has two scheduling times: 1) scheduling start time $\mathcal{T}^s_{0,0} = 0$ and 2) scheduling finish time $\mathcal{T}^f_{0,0} = 1$.

Definition 8 (Job Order): Given a set of jobs $\boldsymbol{J}$, a job order $\mathcal{O}$ of $\boldsymbol{J}$ is an ordered list containing all job scheduling times (both start and finish) of all the jobs in $\boldsymbol{J}$. The job scheduling times are ordered in nondecreasing order.

For notation convenience, we use $\mathcal{O}(i)$ to denote the ith job scheduling time in the job order $\mathcal{O}$. For any two job scheduling times $\mathcal{T}_{i,k}, \mathcal{T}_{j,l} \in \mathcal{O}$, if $\mathcal{T}_{i,k}$ has a smaller index than $\mathcal{T}_{j,l}$ in $\mathcal{O}$, denoted as $\mathcal{T}_{i,k} \prec \mathcal{T}_{j,l}$, then that means $\mathcal{T}_{i,k}$ happens earlier than or at the same time as $\mathcal{T}_{j,l}$.

Example 5: Consider the task set in Example 1. There are four jobs within a hyper-period. For a schedule $\boldsymbol{s} = [s_{0,0}, s_{0,1}, s_{1,0}, s_{2,0}] = [0, 10, 1, 3]$, its job order is $\mathcal{O} = \{\mathcal{T}^s_{0,0}, \mathcal{T}^f_{0,0}, \mathcal{T}^s_{1,0}, \mathcal{T}^f_{1,0}, \mathcal{T}^s_{2,0}, \mathcal{T}^f_{2,0}, \mathcal{T}^s_{0,1}, \mathcal{T}^f_{0,1}\}$. We also give two examples for indexing: 1) $\mathcal{O}(0) = \mathcal{T}^s_{0,0}$ and 2) $\mathcal{O}(3) = \mathcal{T}^f_{1,0}$.

A job order $\mathcal{O}$ implies a set of linear constraints on the schedule $\boldsymbol{s}$ of the optimization problem (1)

$$\forall\ i < j,\ \mathrm{Time}(\mathcal{O}(i)) \le \mathit{Time}(\mathcal{O}(j)) \tag{12}$$

where $\mathit{Time}(\mathcal{T}_{i,k})$ denotes the time that $\mathcal{T}_{i,k}$ happens. If $\mathcal{T}_{i,k}$ is a scheduling start time, $\mathit{Time}(\mathcal{T}_{i,k}) = s_{i,k}$, otherwise, $\mathit{Time}(\mathcal{T}_{i,k}) = s_{i,k} + C_i$.

B. Scheduling With Job Order

Finding a schedule that satisfies a given job order $\mathcal{O}$ is equivalent to solving the problem (1) with extra linear constraints given by (12). Here, we provide the job order scheduling problem for $\mathcal{O}$

$$\underset{\boldsymbol{s}}{\text{Minimize}}\ \mathcal{F}(\boldsymbol{s}) \tag{13}$$

Subject to:

$$\forall i \in \{0, \ldots, n-1\} \quad \forall k \in \{0, \ldots, H/T_i - 1\}$$
$$k \cdot T_i \le s_{i,k} \le k \cdot T_i + D_i - C_i \tag{13a}$$
$$\text{ResourceBound}(\boldsymbol{s}) = 0 \tag{13b}$$
$$\forall i \in \{0, \ldots, 2N-2\},\ \mathit{Time}(\mathcal{O}(i)) \le \mathit{Time}(\mathcal{O}(i+1)) \tag{13c}$$

where the objective function $\mathcal{F}(\boldsymbol{s})$ could be, for example, data age (4), reaction time (5), or time disparity (8).

Theorem 4: The constraints from a job order $\mathcal{O}$ simplify the problem (13) into a convex problem, specifically, an LP problem, when the optimization objective is DARTD.

Proof: Given a job order $\mathcal{O}$, the relative start/finish relationship of any two jobs is known, therefore all the job chains and job merges are decided. Then $\boldsymbol{DA}(\boldsymbol{s})$ and $\boldsymbol{RT}(\boldsymbol{s})$ become linear functions (lengths of all job chains in Definition 3). The $\boldsymbol{TD}(\boldsymbol{s})$ can also be similarly transformed into linear functions following [26]. Constraints (13a) and (13c) are evidently linear functions. As for the computational resource bounds (13b) from the IO test (9), since the given job order $\mathcal{O}$ already specifies the relative order of all the job scheduling times, (9) becomes linear inequalities. Therefore, problem (13) is an LP problem. ■

Next, we use $\pi^*(\mathcal{O})$ to denote the optimal schedule for the problem (13). Note that the $\pi^*(\mathcal{O})$ depends on the specific forms of objective functions and constraints.

Definition 9 (Optimal Job Order Schedule): The optimal job order schedule, $\boldsymbol{s}^* = \pi^*(\mathcal{O}) = \mathrm{argmin}_{\boldsymbol{s}} \mathcal{F}(\boldsymbol{s})$, is the optimal solution of the optimization problem (13).

Example 6: In Example (1), consider a job order: $\mathcal{O} = \{\mathcal{T}^s_{0,0}, \mathcal{T}^f_{0,0}, \mathcal{T}^s_{1,0}, \mathcal{T}^f_{1,0}, \mathcal{T}^s_{2,0}, \mathcal{T}^f_{2,0}, \mathcal{T}^s_{0,1}, \mathcal{T}^f_{0,1}\}$, where we assume $J_{0,0}$ and $J_{1,0}$ are assigned to one processor $\mathcal{P}_0$, while $J_{2,0}$ and $J_{0,1}$ are assigned to another processor $\mathcal{P}_1$. Next, consider optimizing the reaction time of a cause–effect chain $\mathcal{C} = \{\tau_0 \rightarrow \tau_2\}$. The problem (13) can be transformed into an LP problem as follows:

$$\underset{\boldsymbol{s}}{\text{Minimize}}\ \max\{f_{2,0} - s_{0,0}, f_{2,1} - s_{0,1}\} \tag{14}$$

Subject to:

$$f_{0,0} = s_{0,0} + C_0,\ f_{0,1} = s_{0,1} + C_0 \tag{14a}$$
$$f_{1,0} = s_{1,0} + C_1,\ f_{2,0} = s_{2,0} + C_2 \tag{14b}$$
$$f_{2,1} = s_{2,0} + H + C_2 \tag{14c}$$
$$0 \le s_{0,0} \le D_0 - C_0,\ T_0 \le s_{0,1} \le T_0 + D_0 - C_0 \tag{14d}$$
$$0 \le s_{1,0} \le D_1 - C_1,\ 0 \le s_{2,0} \le D_2 - C_2 \tag{14e}$$
$$f_{1,0} - s_{0,0} \ge C_0 + C_1,\ f_{0,1} - s_{2,0} \ge C_0 + C_2 \tag{14f}$$
$$s_{0,0} \le s_{0,0} + C_0 \le s_{1,0} \le s_{1,0} + C_1 \le s_{2,0} \tag{14g}$$
$$s_{2,0} \le s_{2,0} + C_2 \le s_{0,1} \le s_{0,1} + C_0. \tag{14h}$$

The objective function (14) considers the length of two job chains initiated by $J_{0,0}$ and $J_{0,1}$ within a hyper-period. Constraints (14a)–(14c) are due to the nonpreemptive scheduling. Constraints (14d) and (14e) are schedulability constraints. Inequalities (14f) are the resource bound (13b). There are only two IO-test constraints because jobs assigned to different processors can overlap. Constraints (14g) and (14h) posed by the given job order.

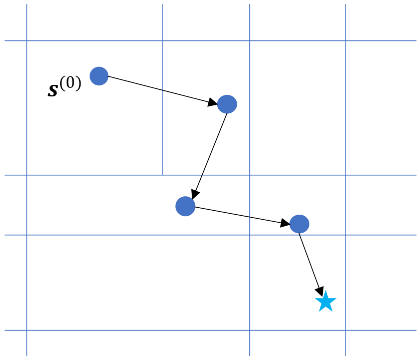

Fig. 3. TOM intuition. The solution space is divided into multiple "subspaces," and the optimal solution within each subspace can be found efficiently by solving an LP problem. This process is visualized above: each job order defines a convex subspace (because all the constraints are linear after specifying a job order) and is informally visualized as a grid in the figure above. The optimal solution within each grid is denoted as a solid circle. The original optimization problem, which needs to explore the whole solution space, is simplified into evaluating only the optimal solutions within each subspace.

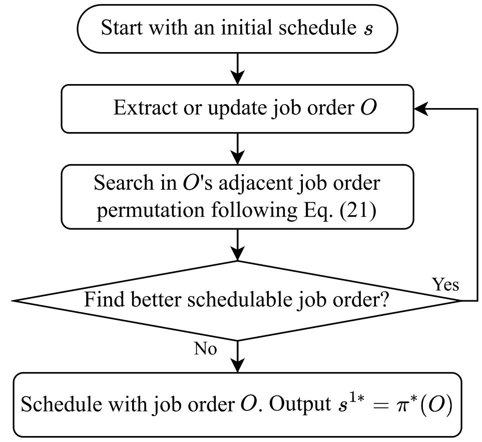

Fig. 4. Main optimization framework. We begin with an initial feasible solution $\boldsymbol{s}$ and its job order $\mathcal{O}$. Then in each iteration, we search for a better job order in $\mathcal{O}$'s adjacent job order permutation $\boldsymbol{\mathcal{B}}(\mathcal{O})$ and update the best job order found yet. Eventually, the iteration will terminate at a 1-opt solution.

Definition 10 (Schedulable Job Order): A job order $\mathcal{O}$ is schedulable if there exists a schedulable schedule $\boldsymbol{s}$ that also satisfies the job order (13c).

V. Two-Stage Optimization Scheduling

Although finding the optimal schedule given a job order is simple and efficient, enumerating all the possible job orders naively requires high-computation costs. Therefore, we propose an iterative algorithm, two-stage optimization method (TOM), to search for better job orders. TOM is proven to find 1-opt solutions.

A. Optimization Concepts Review

Definition 11 (Global Optimality): A solution $\boldsymbol{s}^*$ for the problem (1) is global optimal if there is no other feasible solutions $\boldsymbol{s}$ such that $\mathcal{F}(\boldsymbol{s}) < \mathcal{F}(\boldsymbol{s}^*)$.

Definition 12 (Local Optimality): A solution $\boldsymbol{s}^*$ for the problem (1) is local optimal if there exists a small number $\delta > 0$, such that there is no other feasible solutions $\boldsymbol{s} \in \boldsymbol{\mathcal{B}}(\boldsymbol{s}^*)$ where $\mathcal{F}(\boldsymbol{s}) < \mathcal{F}(\boldsymbol{s}^*)$, $\boldsymbol{\mathcal{B}}(\boldsymbol{s}^*) = \{\boldsymbol{s} \mid \|\boldsymbol{s} - \boldsymbol{s}^*\| \le \delta\}$.

Definition 13 (1-opt, [11], [12]): A solution $\boldsymbol{s}^{1*}$ for the problem (1) is *1-opt* if "the objective value at $\boldsymbol{s}^{1*}$ does not improve by changing a single coordinate," i.e., $\mathcal{F}(\boldsymbol{s}^{1*}) \le \mathcal{F}(\boldsymbol{s}^{1*} + \mathbf{e}_i c)$ for arbitrary unit vector $\mathbf{e}_i = \{0, \dots, 1, \dots, 0\}$ and $c \neq 0$.

Although a global optimal solution is also local optimal and 1-opt, local optimal and 1-opt solutions are not inclusive of each other. In many real-time system problems, achieving global optimal or even local optimal solutions within reasonable time limits is difficult. In these cases, 1-opt provides a better tradeoff between optimality and run-time complexity.

B. Two-Stage Optimization Method

Due to the nonconvex and noncontinuous nature of problem (1), straightforward optimization algorithms necessitate an infinite number of objective function evaluations to verify whether a solution is 1-opt. However, the concept of job order significantly simplifies the problem (1) and allows us to verify whether a solution is 1-opt with only polynomial time complexity. Therefore, we propose a two-stage optimization method (TOM). Fig. 4 shows an overview of TOM. Starting from an initial feasible schedule, the first stage searches for better job orders based on an iterative algorithm, while the second stage finds the optimal schedule by solving problem (13) for each job order to evaluate.

C. Theorems on 1-opt Conditions

Definition 14 (Adjacent Schedule Permutation): The adjacent schedule permutation $\boldsymbol{\mathcal{B}}(\boldsymbol{s})$ of a schedule $\boldsymbol{s}$ is a set of schedules, where each schedule $\boldsymbol{\mathcal{B}}(\boldsymbol{s})_l$ differs from $\boldsymbol{s}$ by only one job's start time.

Definition 15 (Adjacent Job Order Permutation): Adjacent job order permutation $\boldsymbol{\mathcal{B}}(\mathcal{O})$ of a job order $\mathcal{O}$ is a finite set of distinct job orders. For each job order $\boldsymbol{\mathcal{B}}(\mathcal{O})_l$, there is one and only one job $J_{i,k}$ that the position of its scheduling start time $\mathcal{T}^s_{i,k}$, or its scheduling finish time $\mathcal{T}^f_{i,k}$, or both, are different from those in $\mathcal{O}$. The relative order of all the other jobs' scheduling time in $\mathcal{O}$ and $\boldsymbol{\mathcal{B}}(\mathcal{O})_l$ remain the same.

Example 7: Following Example 1, let us consider a job order $\mathcal{O} = \{\mathcal{T}^s_{0,0}, \mathcal{T}^f_{0,0}, \mathcal{T}^s_{1,0}, \mathcal{T}^f_{1,0}, \mathcal{T}^s_{2,0}, \mathcal{T}^f_{2,0}, \mathcal{T}^s_{0,1}, \mathcal{T}^f_{0,1}\}$. As an example, $\boldsymbol{\mathcal{B}}(\mathcal{O})$ could include an job order such as $\{\mathcal{T}^s_{0,0}, \mathcal{T}^f_{0,0}, \mathcal{T}^s_{2,0}, \mathcal{T}^f_{2,0}, \mathcal{T}^s_{1,0}, \mathcal{T}^f_{1,0}, \mathcal{T}^s_{0,1}, \mathcal{T}^f_{0,1}\}$ by moving $J_{1,0}$ to the end of $J_{2,0}$. An alternative adjacent job order could be $\{\mathcal{T}^s_{0,0}, \mathcal{T}^s_{1,0}, \mathcal{T}^f_{0,0}, \mathcal{T}^f_{1,0}, \mathcal{T}^s_{2,0}, \mathcal{T}^f_{2,0}, \mathcal{T}^s_{0,1}, \mathcal{T}^f_{0,1}\}$ where $\mathcal{T}^f_{0,0}$ is moved to the back of $\mathcal{T}^s_{1,0}$, which means $J_{1,0}$ will start execution before $J_{0,0}$ finishes. It is schedulable if there is more than 1 processor.

Theorem 5: Consider a schedule $\boldsymbol{s}^{1*}$ and its job order $\mathcal{O}^{1*}$. $\boldsymbol{s}^{1*}$ is a 1-opt solution for the optimization problem (1) if it satisfies the following conditions:

$$\mathcal{O}^{1*} = \underset{\mathcal{O} \in \boldsymbol{\mathcal{B}}(\mathcal{O}^{1*}) \cap \Omega}{\operatorname{argmin}} \mathcal{F}\big(\pi^*(\mathcal{O})\big) \tag{15}$$

$$\boldsymbol{s}^{1*} = \pi^*\big(\mathcal{O}^{1*}\big) \tag{16}$$

where $\pi^*(\mathcal{O})$ denotes the optimal schedule obtained by solving the problem (13) for $\mathcal{O}$ and Ω denotes the set of schedulable job orders following Definition 10.

Proof: Consider an arbitrary solution $\hat{\boldsymbol{s}}$ which differs from $\boldsymbol{s}^{1*}$ by only one job's start time, and denote the job order of $\hat{\boldsymbol{s}}$ as $\hat{\mathcal{O}}$. In the case, we can introduce a function $\pi(\cdot)$ which obtains the schedule $\hat{\boldsymbol{s}} = \pi(\hat{\mathcal{O}})$. $\pi(\cdot)$ is possibly different from $\pi^*(\cdot)$ in Definition 9. Following Definition 15, we know $\hat{\mathcal{O}} \in \boldsymbol{\mathcal{B}}(\mathcal{O}^{1*})$, and therefore

$$\mathcal{F}(\hat{\boldsymbol{s}}) = \mathcal{F}\left(\pi\left(\hat{\mathcal{O}}\right)\right) \geq \mathcal{F}\left(\pi^*\left(\mathcal{O}^{1*}\right)\right) = \mathcal{F}\left(\boldsymbol{s}^{1*}\right). \tag{17}$$

Therefore, $\boldsymbol{s}^{1*}$ is 1-opt. ∎

Example 8: Let us continue with Example 1 and consider the reaction time optimization problem of a chain $\mathcal{C} = \{\tau_0 \rightarrow \tau_2\}$. A 1-opt schedule could be $\boldsymbol{s}^{1*} = [s_{0,0}, s_{0,1}, s_{1,0}, s_{2,0}] = [9, 10, 18, 11]$. This solution is 1-opt because there is no better feasible solution if only changing one job's start time while leaving the other three jobs' start times unchanged.

Lemma 1: If there are six variables which satisfy $a_1 + c_1 \leq b_1$, $b_2 + c_2 \leq a_2$, then $\max(|a_1 - a_2|, |b_1 - b_2|) \geq \min(c_1, c_2)$.

Proof: Prove by contradiction. Assume $\max(|a_1 - a_2|, |b_1 - b_2|) < \min(c_1, c_2)$, then we have

$$a_2 - a_1 < c_1, \; b_1 - b_2 < c_2. \tag{18}$$

Combine with the theorem assumptions, we can derive

$$a_2 < a_1 + c_1 \leq b_1, \; b_1 < b_2 + c_2 \leq a_2. \tag{19}$$

The two inequalities above conflict with each other, therefore the lemma is proven. ∎

Theorem 6: Assume each job has a nonzero execution time and is executed in single-core systems nonpreemptively. Any schedule $\boldsymbol{s}$ obtained by solving the LP problem (13) is local optimal.

Proof: Prove by contradiction. Assume $\boldsymbol{s}$ is not a local optimal solution. This implies the existence of another feasible solution $\boldsymbol{s}^*$ such that $\mathcal{F}(\boldsymbol{s}^*) < \mathcal{F}(\boldsymbol{s})$, where $\|\boldsymbol{s} - \boldsymbol{s}^*\| < \delta$, and $\delta > 0$ is a very small number. Denote the job order of $\boldsymbol{s}$ and $\boldsymbol{s}^*$ as $\mathcal{O}$ and $\mathcal{O}^*$, respectively. Then we must have $\mathcal{O}^* \neq \mathcal{O}$ because $\boldsymbol{s}$ is optimal for the problem (13) given the job order $\mathcal{O}$.

Since we are considering a nonpreemptive single-core platform, no jobs can run in parallel. Furthermore, since the job orders are different, there must exist at least two jobs $J_{i,k}$ and $J_{j,l}$, whose relative execution order is different. Without loss of generality, assume $J_{i,k}$ runs earlier than $J_{j,l}$ in $\mathcal{O}$, and $J_{j,l}$ runs earlier in $\mathcal{O}^*$. Mathematically speaking, that means

$$s_{i,k} + C_i \leq s_{j,l}, \; s^*_{j,l} + C_j \leq s^*_{i,k}. \tag{20}$$

Based on Lemma 1, we have $\max(|s_{i,k} - s^*_{i,k}|, |s_{j,l} - s^*_{j,l}|) \geq \min(C_1, C_2)$. Therefore, $\|\boldsymbol{s} - \boldsymbol{s}^*\| \geq \max(|s_{i,k} - s^*_{i,k}|, |s_{j,l} - s^*_{j,l}|) \geq \min(C_1, C_2) > \delta$, which causes a contradiction. Therefore, the theorem is proved. ∎

Thus, the 1-opt schedule $\boldsymbol{s}^{1*}$ from Theorem 5 for a nonpreemptive single-core system is also local optimal.

D. Optimization Algorithm Toward 1-opt Schedules

Following Theorem 5, we can design a simple algorithm to search for better job orders iteratively. The algorithm will update the job order following (21) and terminate when the iterations converges, i.e., $\mathcal{O}^{(k+1)} = \mathcal{O}^{(k)}$:

$$\mathcal{O}^{(k+1)} = \underset{\mathcal{O} \in \boldsymbol{\mathcal{B}}(\mathcal{O}^{(k)}) \cap \Omega}{\operatorname{argmin}} \; \mathcal{F}\big(\pi^*(\mathcal{O})\big) \tag{21}$$

where $\pi^*(\mathcal{O})$ is the optimal job order schedule of $\mathcal{O}$ and Ω denotes the set of schedulable job orders following Definition 10.

Theorem 7: An iterative algorithm that updates the job order variables following (21) will terminate after a finite number of iterations, and the solution found is 1-opt.

Proof: The iterative algorithm will terminate after a finite number of iterations because a new iteration is initiated only after finding a feasible, better solution in previous iterations. Considering that the optimal objective function value is positive, the algorithm is guaranteed to terminate after a finite number of iterations. When the algorithm terminates, the two conditions in Theorem 5 are both satisfied and therefore the solution is 1-opt. ∎

VI. Enhancing TOM—Strategies for Improved Performance and Efficiency

A. Skipping Unschedulable Job Orders

Although the feasibility of a job order can be analyzed by solving the LP problem in problem (13), the average runtime complexity is $O(N^{2.5})$ [33]. Therefore, we propose the following lightweight lemma to quickly examine whether a job order is schedulable with $O(N)$ complexity. These lemmas are necessary, but not sufficient, conditions of schedulability:

Lemma 2: Given a job order $\mathcal{O}$, if there exists one job $J_{i,k}$ whose scheduling finish time $\mathcal{T}^f_{i,k}$ precedes its scheduling start time $\mathcal{T}^s_{i,k}$, then $\mathcal{O}$ is not schedulable.

Lemma 3: Given a job order $\mathcal{O}$, if the maximum number of concurrent jobs exceeds the total number of processors, then $\mathcal{O}$ is not schedulable.

Proofs of these lemmas are straightforward as they breach either (13a) or (13b).

B. More Relaxed Constraints in LP

The solution quality of an optimization problem could become better if its constraints are relaxed. In problem (13), although we cannot relax (13a) and (13b) (hard schedulability constraints), we can relax the job order (13c) because it is only necessary to maintain the relative order of jobs that influence the objective functions (because not all the tasks contribute to the cause–effect chains or merges) to guarantee that the objective functions can be equivalently transformed into linear functions.

Example 9: Continue with Example 1, given a job order $\mathcal{O} = \{\mathcal{T}^s_{0,0}, \mathcal{T}^f_{0,0}, \mathcal{T}^s_{1,0}, \mathcal{T}^f_{1,0}, \mathcal{T}^s_{0,1}, \mathcal{T}^f_{0,1}, \mathcal{T}^s_{2,0}, \mathcal{T}^f_{2,0}\}$, suppose we only have one processor and want to optimize the reaction time of the cause–effect chain $\mathcal{C} = \{\tau_0 \rightarrow \tau_2\}$. In this case, the optimal schedule $\pi^*(\mathcal{O}) = [s_{0,0}, s_{0,1}, s_{1,0}, s_{2,0}] = [7, 10, 8, 11]$, the worst-case reaction time is 7 from the job

Algorithm 1: Simple Job Order Scheduler

Input: Job order $\mathcal{O}$
Output: Schedule $\boldsymbol{s}$

$t = 0$ `// Record current time`
for *each* $\mathcal{T}_i$ *in* $\mathcal{O}$ **do**
  $J_i = GetJob(\mathcal{T}_i)$
  if $\mathcal{T}_i$ *is job scheduling start time* **then**
    $t = \max(t, J_i.release\_time, NextProcessorAvailableTime()\ )$
    $s_i = t$
  else
    if $s_i + C_i \leq t$ **then**
      $t = s_i + C_i, f_i = s_i + C_i$
    else
      return 0 `//` $\mathcal{O}$ `is unschedulable`
    end
  end
end
return $\boldsymbol{s}$

Algorithm 2: Single Iteration of TOM

Input: Job order $\mathcal{O}^{(k)}$, job set $\boldsymbol{J}$ containing all jobs in a hyper-period
Output: $\mathcal{O}^{(k+1)}$

1 $\mathcal{O}^{tmp} = \mathcal{O}^{(k)}$
2 **for** *each job* J_i *in* $\boldsymbol{J}$ **do**
3   **for** *each job order* $\mathcal{O}$ *in* $\boldsymbol{\mathcal{B}}^{J_i}(\mathcal{O}^{tmp})$ **do**
4   **if** $\mathcal{F}(\pi^*(\mathcal{O})) < \mathcal{F}(\pi^*(\mathcal{O}^{tmp}))$ **then**
5       $\mathcal{O}^{tmp} = \mathcal{O}$
6   **end**
7   **end**
8 **end**
9 $\mathcal{O}^{(k+1)} = \mathcal{O}^{tmp}$
10 **return** $\mathcal{O}^{(k+1)}$

chain $\{J_{0,0} \to J_{2,0}\}$. Since $J_{1,0}$ does not influence the length of the cause–effect chain $\mathcal{C} = \{\tau_0 \to \tau_2\}$, only enforcing the relative job order among $\{J_{0,0}, J_{0,1}, J_{2,0}\}$ is enough to transform the objective function (5) into linear functions. Then the optimal schedule with relaxed constraints become $\boldsymbol{s}^{\text{relaxed}} = [s_{0,0}, s_{0,1}, s_{1,0}, s_{2,0}] = [9, 10, 0, 11]$. The worst-case reaction time is reduced to 5.

C. Simple Job Order Scheduler

In cases when the run-time complexity becomes a major performance bottle-neck, we can use a heuristic scheduling algorithm with $O(N)$ complexity to replace solving the LP problem (13) that usually requires $O(N^{2.5})$ time complexity [33]. The simple job order scheduler adopts a First-In-First-Out scheduling policy. A job becomes ready for execution after satisfying two conditions: 1) its release time has passed and 2) its previous job scheduling time has happened. Algorithm 1 shows the pseudocode of the simple job order scheduler in a simulation environment.

Example 10: Continue with Example 9, consider the same job order $\mathcal{O} = \{\mathcal{T}^s_{0,0}, \mathcal{T}^f_{0,0}, \mathcal{T}^s_{1,0}, \mathcal{T}^f_{1,0}, \mathcal{T}^s_{0,1}, \mathcal{T}^f_{0,1}, \mathcal{T}^s_{2,0}, \mathcal{T}^f_{2,0}\}$. If there is only one computation core, the schedule obtained from the simple order scheduler is $[s_{0,0}, s_{0,1}, s_{1,0}, s_{2,0}] = [0, 10, 1, 11]$. In case of two cores, the schedule is $[s_{0,0}, s_{0,1}, s_{1,0}, s_{2,0}] = [0, 10, 0, 10]$.

Despite its fast speed, the simple job order scheduler suffers from two major disadvantages: 1) nonexact schedulability analysis and 2) nonoptimal schedule without any theoretical guarantee. It is only encouraged to use if solving the problem (13) iteratively suffers from a big time-out issue.

VII. Implementation Details

A. Initial Solution Estimation

In the experiments, we use a simple list-scheduling method [13] to obtain an initial schedule. If multiple jobs become ready, jobs with the least finish time will be dispatched first. The processor assignments are decided based on a simple First-Come-First-Serve strategy. In practice, other methods can also be used to obtain a feasible initial schedule.

B. Faster Implementation Within Time Limits

TOM is implemented slightly differently from (21) for faster run-time efficiency. When searching for an optimal job order $\mathcal{O}^{(k)*}$ within $\boldsymbol{\mathcal{B}}(\mathcal{O}^{(k)})$, we immediately accept a new job order $\mathcal{O}$ if it improves $\mathcal{O}^{(k)}$. Algorithm 2 shows the pseudocode of one single iteration. In line 3, $\boldsymbol{\mathcal{B}}^{J_i}(\mathcal{O}^{tmp})$ denotes the adjacent job order permutation of $\mathcal{O}^{tmp}$ by only changing the index of J_i's job scheduling time. $\mathcal{O}^{tmp}$ will be updated if a better job order is found. Following Theorem 7, Algorithm 2 also finds 1-opt solutions after algorithm termination.

C. When to Assign Processor

A simple first-come-first-serve (FCFS) policy is used for processor assignment for each job. In experiments, we utilize the simple job order scheduler (Section VI-C) to generate the processor assignment before evaluating a job order (i.e., solving problem (13)). After obtaining the processor assignments, we formulate the resource-bound constraints for problem (13).

D. Worst-Case Complexity Analysis

The overall algorithm's complexity depends on the complexity of each iteration and the total number of iterations. In the experiments, TOM usually terminates in less than 10 iterations. Following (21), the cost of each iteration depends on the number of job orders to search and the cost to evaluate a single job order [problem (13)]. In the worst case, the total number of adjacent job order permutations could be $O(N^3)$. However, techniques from Section VI-A can greatly reduce the possible permutations. Evaluating a single job order has two steps: 1) obtaining a schedule and 2) then evaluating the objective function. The former could be as fast as $O(N)$ if a simple job order scheduler is used. In terms of solving the linear program, the complexity could increase to $O(N^{2.5})$ in average case [33] (in reality, since problem (13) is very sparse, the real run-time speed should be much faster than $O(N^{2.5})$).

Finally, evaluating the objective function given a schedule requires $O(N^2)$ complexity in worst cases.

Overall, the worst-case complexity in one iteration is $O(N^3 \cdot (N^{2.5} + N^2))$ if an optimal job order scheduler [solving problem (13)] is used. However, most experiments finish optimizing task sets of thousands of jobs within 1000 s, which suggests the average time complexity to be $O(N^4)$.

VIII. Extensions and Limitations

This section briefly discusses several possible extensions and leaves the experiment verification to future works.

A. Alternative Objective Functions

Apart from the objective functions shown in Sections III-C and III-D, TOM also supports other forms of objective functions, such as linear combination of DARTD. Besides, TOM can also optimize nonlinear functions of different timing metrics (such as jitters of end-to-end latency) and solve them with nonLP methods [10], [34], though without the 1-opt or local-optimal guarantee anymore.

B. Extension For Preemptive Scheduling

While the TOM framework is designed for nonpreemptive TTS systems, it can be extended to work with preemptive systems. First, similar to the start time variables, an extra set of finish time variables has to be incorporated into the optimization problem formulation. The schedulability analysis constraints (Section III-E) have to be replaced with the demand bound function used in [32]. The concept of job order remains the same because it already incorporates the finish time.

C. Finding Feasible Initial Schedules

The TOM optimization framework can also be utilized to find feasible schedules. This section briefly discusses the theoretical foundations. Since feasibility is a binary metric that is not friendly for optimization, we utilize "tardiness" as the optimization objective function (similar to [10]). The feasibility optimization problem is formulated as follows:

$$\underset{\boldsymbol{s}}{\text{Minimize}} \sum_{i=0}^{n-1} \sum_{k=0}^{H/T_i-1} \text{Barrier}\big(kT_i + D_i - C_i - s_{i,k}\big) \tag{22}$$

$$\text{Barrier}(x) = \begin{cases} 0, & x \geq 0 \\ -x, & x < 0 \end{cases} \tag{22a}$$

Subject to:

$$\forall i \in \{0, \ldots, n-1\} \quad \forall k \in \{0, \ldots, H/T_i - 1\}, kT_i \leq s_{i,k} \tag{22b}$$

$$\text{ResourceBound}(\boldsymbol{s}) = 0. \tag{22c}$$

Theorem 8: If a solution $\boldsymbol{s}$ can reduce the objective function in problem (22) into 0 while also being feasible for problem (22), then $\boldsymbol{s}$ is a schedulable schedule.

Proof: If the objective function is reduced to 0, no jobs violate the deadline constraints. Combined with the job release (22b) and processor overloading (22c), the schedule $\boldsymbol{s}$ is schedulable by definition. ■

Theorem 9: List scheduling can always provide a feasible initial solution to problem (22).

Proof: The schedule found by list scheduling is always feasible for problem (22) because a job is dispatched for execution whenever there is an idle processor [satisfying (22c)] after the job is released in (22b). ■

Theorem 10: The problem (22) can be equivalently transformed into an LP problem after adding an extra set of job order constraints [the inequality (13c)].

Proof: Following Theorem 4, we only need to prove that the objective function (22) can be transformed into linear functions. This can be easily done by introducing an artificial variable $z_{i,k}$ for each term following [26]. After that, the objective function becomes

$$\underset{\boldsymbol{s}}{\text{Minimize}} \sum_{i=0}^{n-1} \sum_{k=0}^{H/T_i-1} z_{i,k} \tag{23}$$

with extra linear constraints

$$\forall i \in \{0, \ldots, n-1\} \quad \forall k \in \{0, \ldots, H/T_i - 1\}$$
$$z_{i,k} \geq 0 \;\; \& \;\; z_{i,k} \geq -1 \cdot \big(kT_i + D_i - C_i - s_{i,k}\big). \tag{24}$$

Since both the objective functions and the constraints are linear functions after transformation, the theorem is proved. ■

The theorems above show that TOM can also solve the feasibility problem (22). It is also guaranteed to perform better than simple scheduling heuristics, such as list scheduling, because TOM utilizes them as initial solutions.

D. Limitations

Compared with global optimality, 1-opt provides a weaker form of theoretical guarantee. However, in general cases, obtaining global optimal solutions requires significantly higher computation costs. Therefore, given the same computation costs, 1-opt could potentially achieve better performance, as shown in our experiments.

TOM's computation cost depends on the number of jobs within a hyper-period. Therefore, there could be a higher computation cost in nonharmonic task sets. However, in realistic TTS systems [35], there cannot be too many jobs within a hyper-period because that would incur a high overhead in task management and scheduling. Therefore, it is expected that the computation cost associated with TOM should be reasonably low in real-world systems.

IX. Experiment

The proposed framework was implemented in C++ and tested on a computing cluster (AMD EPYC 7702 CPU). We consider the following methods in experiments.

1) *List Scheduling [13]:* Whenever there are available processors, it dispatches the ready job with the least finish time for execution.
2) *Simulated Annealing [36]:* A general heuristic method for optimization problems. The initial temperature is $1e8$, and the cooling rate is 0.99, which encourages the algorithm to explore the solution space. The initial

schedule is obtained from the list scheduling, the same as TOM.

3) *Verucchi20 [7]:* It was proposed to minimize the worst-case data age and reaction time in multirate DAG. The code implementation is adopted from their official release repository. If it does not run time out, its solution quality is close to the optimal solutions. To the best of our knowledge, it is also the most recent state-of-the-art work that considers a similar problem setting.
4) *TOM:* The optimization framework proposed in this article. When solving problem (13), CPLEX [37] is used to find optimal solutions.
5) *TOM_SimpleScheduler:* Similar to TOM, except that the simple job order scheduler (Section VI-C) instead of LP is used when obtaining a schedule from a job order.
6) *TOM_Extended:* Similar to TOM, except that we also enabled the relaxations on the LP problem's constraints, which is introduced in Section VI-B.

If one method runs time-out without a feasible solution, we use the results of list scheduling during the result analysis.

A. Task Set Generation and Results

The simulated DAG task sets are generated following a real-world automotive benchmark [9], all the tasks' periods are randomly generated from a limited set {1, 2, 5, 10, 20, 50, 100, 200, 1000}, with relative probability distribution: {3, 2, 2, 25, 25, 3, 20, 1, 4}. The overall task set's utilization is set to 0.9 m, where m is the number of cores available, 4 in our experiments. Each task's WCET is generated by UUnifast [38] while following the multicore adaptation implementation in [10]. Each task's relative deadline is the same as its period. Task sets generated in this way usually have hundreds or *thousands* of jobs to schedule.

Task dependencies are generated randomly following He et al. [39]. After generating individual tasks, we go through each pair of tasks and randomly add an edge from one task to another with a given probability, 0.9 in our experiments (smaller probabilities are usually insufficient to generate many cause–effect chains in the DAG). The number of tasks in a task set ranges from 5 to 20. Cause-effect chains are generated as the paths between random pairs of tasks using the shortest path algorithm in Boost Graph Library [40]. Task merges are generated by randomly selecting a sink task and then collecting all source tasks on which the sink task directly depends.

For a task set with n tasks, there are n to $2n$ random cause–effect chains and $\lfloor 0.25n \rfloor$ to n random task merges. The maximum number of source tasks in a merge varies from 2 to 9 following ROS [16]. The lengths and activation patterns of the cause–effect chains adhere to distributions outlined in Tables VI and VII of the automotive benchmark [9]. To meet distribution criteria, we initially generate plenty of task sets, evaluate the likelihood for each task set, and then sample 1000 random task sets weighted by the likelihood for each given number of tasks. All task sets are schedulable under the list scheduling method. The run-time limit for scheduling one task set is 1000 s per method.

We tested the performance of each method in optimizing DARTD separately. The experiment results are reported in Fig. 5. All performance gaps are compared against the list scheduling method

$$\frac{\mathcal{F}_{\text{method}} - \mathcal{F}_{\text{List\_Scheduling}}}{\mathcal{F}_{\text{List\_Scheduling}}} \times 100\%. \tag{25}$$

B. Result Analysis and Discussion

Overall, TOM and its extensions significantly outperform other methods in various experiments. Next, we provide a more detailed analysis of different aspects.

1) Comparison With Baseline Methods: Compared with other baseline methods, the performance improvements of TOM and TOM_Extended are not obvious when the number of tasks is small ($n = 5$). This is because the solution space is very small and most methods can find good solutions. However, as the number of tasks increases, Verucchi20 quickly reaches time limits and can barely find schedulable schedules or schedules with low-end-to-end latency. Simulated annealing always starts its iteration with a feasible schedule. However, due to its inefficient solution space exploration techniques, it usually requires a long time to find a good solution, which often exceeds the given time limit and therefore cannot show much performance improvement. In contrast, guided by 1-opt, TOM and TOM_Extended are able to explore the solution space efficiently while still maintaining good solution quality. These experiment results show the benefits of both 1-opt optimality and the proposed TOM optimization algorithms.

2) TOM Versus TOM_SimpleScheduler: The performance improvements of TOM against TOM_SimpleScheduler show the benefits of the LP formulation. Compared with simple heuristics, such as list scheduling, LP explores a larger solution space, can find nonwork-conserving schedules, and thus achieves better solution quality. The disadvantage of the LP approach is the higher computation cost. To compensate for the extra computation costs, many heuristics are proposed in this article without sacrificing the theoretical guarantee, such as using fast necessary conditions to filter unschedulable job orders (Section VI-A), exploring the sparse structure in implementation (the resource bound constraints are sparse linear constraints). However, TOM_SimpleScheduler could still be an option in situations with many tasks/jobs.

3) TOM Versus TOM_Extended: The performance improvements of TOM_Extended against TOM show the effectiveness of the heuristics (Section VI-B) to further improve upon 1-opt while maintaining a similar run-time speed. Since the results obtained from both TOM_Extended and TOM are 1-opt (if not running time-out), it implies that there are potentially many 1-opt solution candidates with varying solution qualities in the whole solution space. If applicable, utilizing heuristics to further improve upon 1-opt solutions is beneficial.

4) Time-Out Issue: It is possible that TOM does not finish iterations before running time out. In these cases, TOM degrades into heuristic algorithms without a theoretical guarantee. However, the trend in Fig. 5 shows that running time-out does not seriously degrade the solution quality even

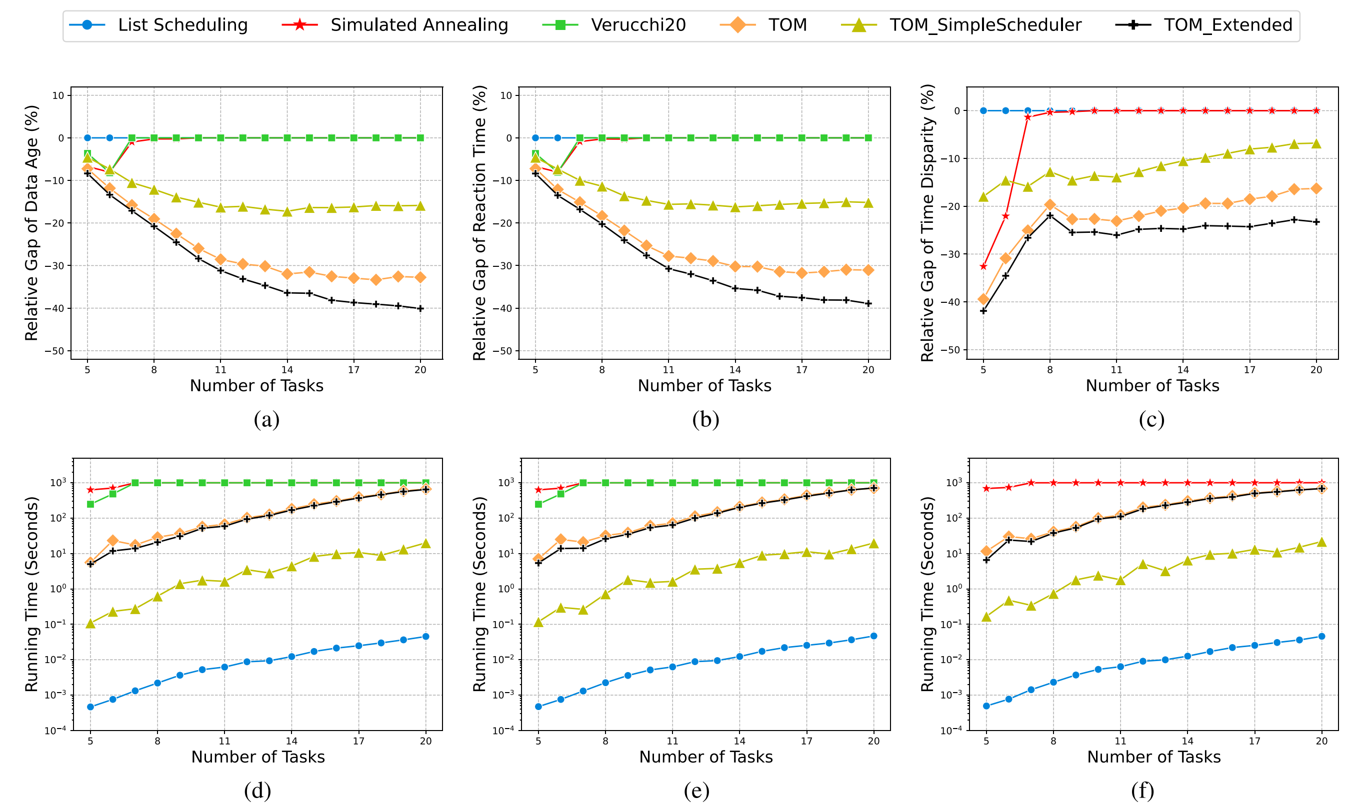

Fig. 5. Performance gap and running time for optimizing end-to-end latency and time disparity on synthetic task sets. (a) Data age performance. (b) Reaction time performance. (c) Time disparity performance. (d) Data age running time. (e) Reaction time running time. (f) Time disparity running time.

though more than 30% cases running time out when $n = 20$ (around 4000 jobs per task set). We expect TOM to work reasonably well for task sets with less than 10^4 jobs if the time limit is 1000 s. Optimizing larger task sets, such as those with 10^5 jobs, would require a much longer time limit.

5) Data Age Versus Reaction Time: Experiments show that data age and reaction time optimization have similar results. Furthermore, reducing one metric usually reduces the other, which is broadly consistent with the findings in [14]. This observation may improve the algorithm efficiency in cases where both data age and reaction time need to be optimized: we may just consider only one metric in the objective function and leave the other out.

C. Time Disparity Optimization Result

Although the overall results on time-disparity optimization are good, Fig. 5(c) shows that the performance seems to become worse when the number of tasks increases from 5 to 8. This is mainly due to the nature of the problem itself, rather than the limitations of the optimizers. For example, consider two merges where one merge has 2 source tasks and 1 sink task, and another merge has the same sink task, the same 2 source tasks, and 2 more extra source tasks. In this case, the maximum source time disparity of the second merge could never become smaller than the first merge. In practice, adding more source tasks does not necessarily make the list scheduling perform worse after reaching certain limits, but it does make the optimization more difficult, and limits the performance improvements even for global optimal solutions.

X. Conclusion

In this article, we investigate a multirate DAG scheduling problem to reduce the worst-case end-to-end latency and/or time disparity metrics. Given the potentially vast number of variables within the solution space, we advocate for guiding the scheduling design with 1-opt. Our optimization algorithm introduces a novel technique called *job order* to partition the solution space into multiple convex subspaces. This partitioning strategy allows utilizing LP to minimize DARTD within each subspace. Building upon this partition, our algorithm iteratively traverses among the subspaces, ensuring that the output is 1-opt. In contrast to alternative optimization algorithms, such as meta-heuristics algorithms lacking any theoretical performance guarantees, or optimal algorithms that may require exponential run-time complexity, the 1-opt algorithm balances the tradeoff between theoretical performance guarantee and run-time complexity. We rigorously prove that our optimization algorithm achieves 1-opt solutions while maintaining polynomial run-time complexity. Further optimization heuristics are also proposed to improve the algorithm's performance and efficiency without compromising the 1-opt solution guarantee. Experimental results indicate significant improvements over state-of-the-art methods in both performance and computational efficiency.

References

[1] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, “A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics,” in *Proc. IEEE Real-Time Syst. Symp.*, 2009, pp. 1–8.

[2] “2021 RTSS industry challenge.” PerceptIn. 2021. [Online]. Available: http://2021.rtss.org/industry-session/

[3] I. S. Olmedo, N. Capodieci, J. L. Martinez, A. Marongiu, and M. Bertogna, “Dissecting the CUDA scheduling hierarchy: A performance and predictability perspective,” in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp.*, 2020, pp. 213–225.

[4] J. Abdullah, G. Dai, and W. Yi, “Worst-case cause–effect reaction latency in systems with non-blocking communication,” in *Proc. Design, Autom. Test Europe Conf. Exhibit.*, 2019, pp. 1625–1630.

[5] M. Günzel, K.-H. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J.-J. Chen, “Timing analysis of asynchronized distributed cause–effect chains,” in *Proc. IEEE 27th Real-Time Embed. Technol. Appl. Symp.*, 2021, pp. 40–52.

[6] M. Dürr, G. von der Brüggen, K.-H. Chen, and J.-J. Chen, “End-to-end timing analysis of sporadic cause–effect chains in distributed systems,” *ACM Trans. Embed. Comput. Syst.*, vol. 18, pp. 1–24, Oct. 2019.

[7] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, “Latency-aware generation of single-rate DAGs from multi-rate task sets,” in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp.*, 2020, pp. 226–238.

[8] T. Klaus, M. Becker, W. Schröder-Preikschat, and P. Ulbrich, “Constrained data-age with job-level dependencies: How to reconcile tight bounds and overheads,” in *Proc. IEEE 27th Real-Time Embed. Technol. Appl. Symp.*, 2021, pp. 66–79.

[9] S. Kramer, D. Ziegenbein, and A. Hamann, “Real world automotive benchmarks for free,” in *Proc. Int. Workshop Anal. Tools Methodol. Embed. Real-Time Syst.*, 2015, pp. 1–6.

[10] S. Wang, R. K. Williams, and H. Zeng, “A general and scalable method for optimizing real-time systems with continuous variables,” in *Proc. IEEE 29th Real-Time Embed. Technol. Appl. Symp.*, 2023, pp. 119–132.

[11] J. Park and S. Boyd, “A semidefinite programming method for integer convex quadratic minimization,” *Optim. Lett.*, vol. 12, pp. 499–518, May 2018.

[12] K. Khurshid, S. Irteza, A. A. Khan, and S. Shah, “Application of heuristic (1-opt local search) and metaheuristic (ant colony optimization) algorithms for symbol detection in MIMO systems,” *Commun. Netw.*, vol. 03, pp. 200–209, Nov. 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:44194350

[13] H. R. Topcuoglu, S. Hariri, and M. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[14] M. Günzel, H. Teper, K.-H. Chen, G. von der Brüggen, and J.-J. Chen, “On the equivalence of maximum reaction time and maximum data age for cause–effect chains,” in *Proc. Euromicro Conf. Real-Time Syst.*, 2023, pp. 1–22. [Online]. Available: https://api.semanticscholar.org/CorpusID:259318171

[15] J. Martinez, I. Sañudo, and M. Bertogna, “End-to-end latency characterization of task communication models for automotive systems,” *Real-Time Syst.*, vol. 56, pp. 315–347, Jul. 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219935505

[16] R. Li, N. Guan, X. Jiang, Z. Guo, Z. Dong, and M. Lv, “Worst-case time disparity analysis of message synchronization in ROS,” in *Proc. IEEE Real-Time Syst. Symp.*, 2022, pp. 40–52.

[17] X. Jiang, X. Luo, N. Guan, Z. Dong, S. Liu, and W. Yi, “Analysis and optimization of worst-case time disparity in cause–effect chains,” in *Proc. Design, Autom. Test Europe*, 2023, pp. 1–6.

[18] S. Baruah, M. Bertogna, G. Buttazzo, S. Baruah, M. Bertogna, and G. Buttazzo, “The sporadic DAG tasks model,” in *Multiprocessor Scheduling for Real-Time Systems*. Cham, Switzerland: Springer, 2015, pp. 191–204.

[19] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, “Response-time analysis of conditional DAG tasks in multiprocessor systems,” in *Proc. Euromicro Conf. Real-Time Syst.*, 2015, pp. 211–221.

[20] J. Li, K. Agrawal, C. Lu, and C. D. Gill, “Analysis of global EDF for parallel tasks,” in *Proc. Euromicro Conf. Real-Time Syst.*, 2013, pp. 3–13.

[21] J. Fonseca, G. Nelissen, and V. Nélis, “Improved response time analysis of sporadic DAG tasks for global FP scheduling,” in *Proc. Int. Conf. Real-Time Netw. Syst.*, 2017, pp. 28–37.

[22] K. Tindell, A. Burns, and A. Wellings, “Allocating hard real-time tasks: An NP-hard problem made easy,” *Real-Time Syst.*, vol. 4, pp. 145–165, 1992. [Online]. Available: https://link.springer.com/article/10.1007/BF00365407

[23] Y. Zhao and H. Zeng, “The virtual deadline based optimization algorithm for priority assignment in fixed-priority scheduling,” in *Proc. IEEE Real-Time Syst. Symp.*, 2017, pp. 116–127.

[24] Y. Zhao, R. Zhou, and H. Zeng, “An optimization framework for real-time systems with sustainable schedulability analysis,” in *Proc. IEEE Real-Time Syst. Symp.*, 2020, pp. 333–344.

[25] J. Martinez, I. Sañudo, and M. Bertogna, “Analytical characterization of end-to-end communication delays with logical execution time,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2244–2254, Nov. 2018.

[26] S. Wang et al., “Optimizing logical execution time model for both determinism and low latency,” in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp.*, 2024, pp. 135–148.

[27] C. Bradatsch, F. Kluge, and T. Ungerer, “Data age diminution in the logical execution time model,” in *Proc. Int. Conf. Archit. Comput. Syst.*, 2016, pp. 173–184.

[28] L. Maia and G. Fohler, “Reducing end-to-end latencies of multi-rate cause–effect chains in safety critical embedded systems,” in *Proc. 12th Eur. Congr. Embed. Real Time Softw. Syst. (ERTS)*, 2024, pp. 1–10.

[29] Y. Tang, X. Jiang, N. Guan, D. Ji, X. Luo, and W. Yi, “Comparing communication paradigms in cause–effect chains,” *IEEE Trans. Comput.*, vol. 72, no. 1, pp. 82–96, Jan. 2023.

[30] F. Paladino, A. Biondi, E. Bini, and P. Pazzaglia, “Optimizing per-core priorities to minimize end-to-end latencies,” in *Proc. 36th Euromicro Conf. Real-Time Syst. (ECRTS)*, 2024, pp. 1–25. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2024.6

[31] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, “Communication centric design in complex automotive embedded systems,” in *Proc. Euromicro Conf. Real-Time Syst.*, 2017, pp. 1–20.

[32] S. Baruah, “Scheduling DAGs when processor assignments are specified,” in *Proc. Int. Conf. Real-Time Netw. Syst.*, 2020, pp. 111–116.

[33] M. B. Cohen, Y. T. Lee, and Z. Song, “Solving linear programs in the current matrix multiplication time,” *J. ACM*, vol. 68, no. 1, pp. 1–39, 2021.

[34] J. Nocedal and S. J. Wright, “Nonlinear equations,” in *Numerical Optimization*. New York, NY, USA: Springer, 2006, pp. 270–302.

[35] A. Minaeva and Z. Hanzálek, “Survey on periodic scheduling for time-triggered hard real-time systems,” *ACM Comput. Surveys*, vol. 54, pp. 1–32, Mar. 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:233354222

[36] P. J. Van Laarhoven, E. H. Aarts, P. J. van Laarhoven, and E. H. Aarts, *Simulated Annealing*. Dordrecht, The Netherlands: Springer, 1987.

[37] I. I. Cplex, “V12.1: User’s manual for CPLEX,” *Int. Bus. Mach. Corp.*, vol. 46, no. 53, p. 157, 2009.

[38] E. Bini and G. Buttazzo, “Measuring the performance of schedulability tests,” *Real-Time Syst.*, vol. 30, pp. 129–154, May 2005.

[39] Q. He, M. Lv, and N. Guan, “Response time bounds for DAG tasks with arbitrary intra-task priority assignment,” in *Proc. Euromicro Conf. Real-Time Syst.*, 2021, pp. 1–21.

[40] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual* (C++-depth). Boston, MA, USA: Addison-Wesley, 2001.