applied sciences

MDPI

Article

AUTOSAR Runnable Periods Optimization for DAG-Based Complex Automobile Applications

Daeho Choi [1,†], Tae-Wook Kim [2,†] and Jong-Chan Kim [3,*]

1 Mobility Business Division, SWM.AI, Anyang-si, Gyeonggi-do 14055, Korea; daeho.choi@swm.ai
2 Graduate School of Automotive Engineering, Kookmin University, Seoul 02707, Korea; dsd8135@kookmin.ac.kr
3 Department of Automobile and IT Convergence, Kookmin University, Seoul 02707, Korea
* Correspondence: jongchank@kookmin.ac.kr; Tel.: +82-2-910-4288
† These authors contributed equally to this work.

Received: 13 July 2020; Accepted: 20 August 2020; Published: 23 August 2020

Abstract: When developing an automobile control application, its scheduling parameters as well as the control algorithm itself should be carefully optimized to achieve the best control performance from given computing resources. Moreover, since the wide acceptance of the AUTOSAR standard, where finer-granular scheduling entities (called runnables) rather than the traditional real-time tasks are used, the number of scheduling parameters to be optimized is far greater than the traditional task-based control systems. Hence, due to the vast problem space, it is not feasible to reuse existing time-consuming search-based optimization methods. With this motivation, this paper presents an analytical codesign method for deciding runnable periods that minimize given control cost functions. Our solution approach, based on the Lagrange multiplier method, can find optimized runnable periods in polynomial times due to its analytical nature. Moreover, our evaluation results for synthesized applications with varying complexities show that our method performs significantly better (12% to 59% of control cost reductions) than a state-of-the-art evolutionary algorithm. To the best of our knowledge, this study is one of the first attempts to find runnable periods that maximize a given system's control performance.

Keywords: AUTOSAR; DAG; runnable scheduling; control-scheduling codesign; lagrange multiplier

1. Introduction

AUTOSAR is the de facto standard software architecture for automobile control systems, covering a wide range of applications such as engine management, motor-driven power steering, and advanced driver assistance systems [1–3]. In the AUTOSAR standard, a control system is designed as a set of *software components*, which are the units of software packaging and deployment. Usually, multiple software components are connected and communicate through the AUTOSAR runtime environment (RTE). Each software component is also composed of a set of *runnables*, which are the smallest unit functions for software development and scheduling. Runnables communicate with each other within each software component and across different software components, using asynchronous message passing interfaces provided by the RTE. As a result, a system can be modeled as a directed acyclic graph (DAG) of runnables where data flow from sensors to actuators through the runnables in the DAG.

For runnable executions, each runnable is associated with an event source, which is usually a periodic timer, and runnables with the same periods are grouped into periodic tasks for scheduling on the AUTOSAR real-time operating system (RTOS). Runnable periods should be carefully optimized since they are control knobs for balancing the trade-off between a system's load and control performance. For example, imagine a system with extremely short runnable periods. The system

will then be much too heavily loaded, hence not schedulable since the runnables should execute with extremely high frequencies. On the other hand, the short runnable periods, if realized, can produce a high control performance due to fast data flows and high control frequencies. At the opposite extreme, i.e., a system with extremely long runnable periods, it will be lightly loaded and hence easily schedulable; however, its control performance will be significantly degraded due to slow data flows and low control frequencies. In that sense, we need a method to find optimal runnable periods between those two extreme cases.

However, in the automotive industry, runnable periods are usually decided in an ad-hoc manner with time-consuming trials and errors [4], making it difficult to extract the optimal control performance out of given hardware resources. To cope with this problem, this paper formulates a runnable periods optimization problem for maximizing the control performance of a given system. Our previous work's initial approach was to use a simple combinatorial search method to find real optimal runnable periods [5]. However, our preliminary experiment revealed that even for a small system with a dozen runnables, since the optimization process cannot find solutions in polynomial times, it takes too much time, making it impractical for complex industry applications.

To deal with this scalability problem, our approach is to develop an analytical method that can find near-optimal solutions without time-consuming searches. For that, the first step is to pick an appropriate control performance model as the optimization objective. Among various models in the literature, we chose the linear control cost model from Bini and Cervin [6] that represents a control system's performance as an approximate linear cost function of its control period and delay [6]. This model has been used as a standard tool by many control-scheduling codesign studies [7–10]. The second step is to define the optimization constraint, that is, the schedulability constraint in our problem. Since the AUTOSAR standard assumes a priority-driven scheduling algorithm, we use the Liu and Layland (L&L) utilization bound method [11], which can be used for both the rate monotonic (RM) and the earliest deadline first (EDF) scheduling algorithms. For the explanation purpose, the EDF scheduling algorithm is mostly assumed throughout this paper, and later our method is extended to the RM scheduling algorithm.

Based on the control cost function and the schedulability constraint, our specific problem is to find the runnable periods that minimize the control cost while guaranteeing the schedulability constraint. Since the control cost function is a function of control period and delay, it should be transformed into a function of runnable periods. For that, we carefully investigate how runnable periods affect the temporal behavior, i.e., control period and delay, of a control system, and develop a generalized method for the transformation.

After the transformation, the Lagrange multiplier method is used to find the optimal runnable periods. Note that the Lagrange multiplier is a well-known optimization method for constrained optimization problems. As it provides an analytical method without any problem space search, our method can find the optimal runnable periods regardless of the size and the complexity of a target system. The detailed optimization process is explained in three steps beginning from the most uncomplicated application model to the general DAG model for the explanation purpose. Although our solution cannot find the real optimal solutions due to a heuristic applied during the optimization, our evaluation results for small systems show that the performance loss is marginal compared with the real optimal solutions. Moreover, even for large systems, our method performs better compared with a state-of-the-art optimization method.

This paper's contributions can be summarized as follows:

- We formulate a problem of AUTOSAR runnable periods optimization in the context of control-scheduling codesign, which we consider to be one of the first such attempts;
- For the above problem, we present a Lagrange multiplier-based analytical method for DAG-based AUTOSAR control applications, which can find near-optimal solutions in polynomial times.

The rest of this paper is organized as follows: Next section provides related work. In Section 3, the background is given and the problem is described. Sections 4 and 5 introduce our preliminary

works for limited application models. Then, Section 6 describes our analytical solution for the general DAG model. Section 7 evaluates our method. Finally, Section 8 concludes this paper.

2. Related Work

Periods selection problem. Control-scheduling codesign methods have been developed in the literature to improve a control system's performance through optimizing its scheduling parameters. In this regard, Seto et al.'s seminal work [12] first presented a periods selection problem assuming that the control performance can be expressed as an exponential decay function of sampling periods and that the tasks are scheduled under a dynamic-priority scheduling algorithm. The periods selection problem was extended to fixed-priority systems by finding the finite set of feasible period ranges using a branch and bound-based integer programming method [13]. Later, Bini and Di Natale [14] proposed a faster algorithm that finds a sub-optimal periods assignment, which can be used for task sets of practical size that are not solvable by previous methods due to high computing demands. Du et al. [15] presented an analytical solution using the Lagrange multiplier method and an online algorithm for overloaded situations. Fu et al. [9] developed a heuristic algorithm for multicore processors.

Delay-aware approaches. A common assumption of the above studies regarding the periods selection problem is that the control performance is only affected by sampling rates, i.e., task periods, of a control system. However, delays between sensing and actuation also have significant effects on control performance. With this motivation, Bini and Cervin [6] incorporated each task's delay into their optimization cost function. In their work, to find the optimal periods assignment, cost functions are approximated as linear functions of control period and delay, and the delay is also approximated assuming the fluid model scheduler. Through the approximations, they proposed an analytical solution. Xu et al. [16] extended this approach for systems with harmonic periods.

Periods and deadlines selection problem. Wu et al. [8] formulated an optimization problem for selecting both task periods and deadlines simultaneously for EDF-scheduled systems. They showed that we can upper bound the amount of delays and jitters each task can experience by regulating relative deadlines of tasks. The cost function is assumed to be a nonlinear function of period and deadline of each task. Based on that, a two-step approach was proposed, which first fixes periods and later tries to minimize deadlines exploiting unused resources. Tan et al. [10] proposed an algorithm that simultaneously adjusts periods and deadlines assuming EDF-scheduled linear–quadratic–Gaussian (LQG) controllers. They showed that their algorithm is more robust with various workloads than the previous method. Cha et al. [17] proposed a heuristic algorithm for the periods and deadlines selection problem with arbitrary nonlinear control cost functions for systems scheduled by the RM scheduling algorithm.

Cause-effect chain analysis. The above studies commonly assume independent real-time tasks, where there is no data dependency among tasks and each task is responsible for its dedicated control target plant. To deal with practical automobile control applications composed of tasks with complex dependencies, DAG-based control applications had been studied in the context of cause-effect chain analyses of real-time tasks [18–21]. Even though they are using tasks instead of runnables, their system model is similar to ours. However, they address the opposite direction of our optimization problem, which is to analyze end-to-end delays for a DAG of tasks with given periods. Besides, [22] analyzed end-to-end delays of an engine management system, which is given as a DAG of runnables.

AUTOSAR system optimization. In automobile control systems based on the AUTOSAR standard, each control application is designed as a DAG of fine-granular runnables with more complex data dependencies compared with traditional real-time task-based systems. In this context, Long et al. [23] developed a runnable placement and scheduling method considering the inter-runnable communication overhead in an electronic control unit (ECU). Monot et al. [24] proposed an algorithm for sequencing and scheduling runnables for multicore ECUs. Saidi et al. [25] studied the runnable-to-core mapping problem using the integer linear programming (ILP) technique.

Kehr et al. [26] developed a method for migrating a legacy AUTOSAR application to a multicore processor while minimizing energy consumption.

AUTOSAR runnable scheduling. However, the above studies about AUTOSAR applications commonly assume that runnable periods are given a priori, which is not valid in the industry practice. With this motivation, runnable periods optimization problem was first formulated in our previous research paper by Kim et al. [5], which proposed a combinatorial search method that is useful only for small systems due to its high computing demands. Choi et al. [27] partly solved the scalability problem using an analytical method only for limited application structures. These two papers are precursors to this paper and will be thoroughly explained even in more depth and detail in Sections 4 and 5 for the self-completeness of this paper. This paper then further extends our previous works by presenting a more general solution that applies to arbitrarily-shaped complex DAG-based AUTOSAR applications.

3. Background and Problem Description

3.1. System Model

This paper assumes an automobile control application based on the AUTOSAR standard. Figure 1 shows an example system where the application is composed of N software components

$$\{C_1, C_2, \cdots, C_N\}. \tag{1}$$

Each software component C_i is also composed of $|C_i|$ runnables where $|C_i|$ denotes the number of runnables in C_i. Note that a runnable is the smallest unit function in the AUTOSAR standard. As shown in the figure, runnables, denoted by r_is, are connected with directed edges representing data dependencies among them. Thus, the whole system can be thought of as a DAG of runnables without explicitly specifying which software component each runnable belongs to. A DAG G is formally defined as

$$G = (V, E \subset V \times V) \tag{2}$$

where the set of vertices V is a set of n nodes or runnables $\{r_1, r_2, \cdots, r_n\}$ where $n = \sum_{i=1}^{N} |C_i|$ and E represents a set of directed edges or links among them. There exists a directed edge $(r_j, r_k) \in E$ if and only if the runnable r_k has a data dependency on the runnable r_j. Then, each i-th runnable r_i is defined by a tuple

$$r_i = (p_i, e_i) \tag{3}$$

where p_i is its period and e_i is the worst-case execution time. Among the runnables, we assume that r_1, the *sensor runnable*, plays a special role of collecting data from sensors, and r_n, the *actuator runnable*, is responsible for controlling actuators. Thus, G has only one source node r_1 and one sink node r_n. Our system model assumes that, in an ECU, there is only one CPU running a single control application described by G, where the ECU handles only a single control target plant, which is common in the automotive industry's federated architecture [28,29]. Note that e_is are given properties of the system, whereas p_is are controllable parameters. Thus, the runnable periods

$$(p_1, p_2, \cdots, p_n) \tag{4}$$

should be decided before integrating the runnables on the AUTOSAR platform. Once p_is are decided, runnables with the same p_is are grouped and consolidated into RTOS tasks, which are scheduled following the scheduling strategy of the RTOS.

For communications between runnables, an asynchronous sender-receiver communication is used [30]. In this communication method, a sender runnable periodically generates its output in a shared memory buffer with its own period, then a receiver runnable asynchronously reads the data in the memory buffer with its own period. When multiple writes occur to the same memory location

without any reading operation from the receiver, the most recent data are always overwritten in the buffer. For further discussions afterward, we formally introduce the following definitions:

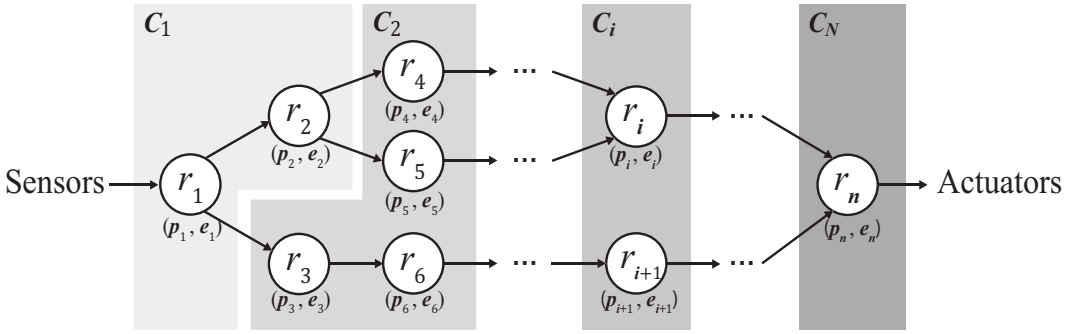


Figure 1. Our directed acyclic graph (DAG)-based system model with N software components and n runnables where each runnable r_i is annotated with its period p_i and worst-case execution time e_i.

Definition 1. (Paths) *For a DAG $G = (V, E)$, there is a finite number of directed paths from the source node r_1 to the sink node r_n. We assume that there are m paths in G, which is denoted by*

$$\mathbb{P}(G) = \{P_1, P_2, \cdots, P_m\}. \tag{5}$$

Then, a path is formally defined as an ordered set of runnables beginning with r_1 and ending with r_n in which all runnables are distinct and every pair of adjacent runnables is joined by a directed edge in E. From now on, for the notational convenience, when we refer to a path $P \in \mathbb{P}(G)$, it can denote the ordered set of runnable indexes $(1 \leq i \leq n)$ as well as the runnables themselves depending on the context.

Definition 2. (Length) *For a path $P \in \mathbb{P}(G)$, its length is defined as*

$$\sum_{i \in P} p_i, \tag{6}$$

which is the sum of runnable periods following a specific path P. When data flow through several paths in parallel, the speed of a data flow is collectively determined by the runnable periods in each path through which the data are flowing, considering our inter-runnable communication method.

Definition 3. (Weight) *For a path $P \in \mathbb{P}(G)$, its weight is defined as*

$$\sum_{i \in P} e_i, \tag{7}$$

which is the sum of runnable execution times following a specific path P. Thus, a path's weight is a representative metric for the amount of computing resource demand of the path.

Definition 4. (Critical Path) *Given a DAG G with its paths $\mathbb{P}(G)$, its critical path is defined by the path found in*

$$\underset{P \in \mathbb{P}(G)}{\operatorname{argmax}} \sum_{i \in P} p_i, \tag{8}$$

which is the path with the longest length. Without loss of generality, we assume that there is only one critical path in each DAG.

Definition 5. (Heaviest Path) *Given a DAG G with its paths $\mathbb{P}(G)$, its heaviest path is defined by the path found in*

$$\underset{P \in \mathbb{P}(G)}{\operatorname{argmax}} \sum_{i \in P} e_i, \tag{9}$$

which is the path maximizing the sum of e_is for the runnables in the path. The intuition behind the heaviest path is that it indicates the path that consumes the maximum amount of CPU time for a single execution of paths. Without loss of generality, we assume that there is only one heaviest path in each DAG.

3.2. Control Performance Model

A control system's performance can be defined in many different aspects. For example, its robustness to external disturbances, control stability, and control error can be such performance metrics. In general, a control system's performance is affected by its timing behavior as well as the control algorithm itself [31]. In this paper, we assume that the control algorithm is given as a fixed system property. Thus, our control performance model is about how the system's temporal properties affect the control performance. More specifically, we consider two distinct temporal properties: control period and end-to-end delay of the target control system.

In the AUTOSAR timing extensions, two latency constraints are defined: (i) data age timing constraint and (ii) reaction time constraint [20,32,33]. The specification states that when an actuator command is periodically produced, its source input (sensing) value's age should be maintained within a specified timing constraint. The reaction time constraint is also considered when an external event, such as pressing a button, should be reacted within a specified timing constraint. In this paper, since we are considering periodic workloads, the data age timing constraint is considered.

Based on the timing model, there are several ways to build a control performance model. One is to measure the resulting performance of the system by artificially controlling the temporal parameters. Simulation tools [34,35] can also be used to predict the control performance when we cannot directly measure the system under investigation. To provide a more general model, Bini and Cervin [6] introduced a linear control cost function as

$$J(T, \Delta) = \alpha T + \beta \Delta \tag{10}$$

where T is the control period, and Δ is the end-to-end delay from the sensors to the actuators. Note that α and β are constants that define the characteristics of the control target plant. Figure 2 shows an example control cost function. The intuition behind it is that if we give control commands to the actuator more often (frequently), its control cost gets smaller, and in the same manner, if we decide the control command with more fresh (recent) sensor data with shorter delays, the cost gets smaller, again. In general, the cost function J can be a nonlinear function of T and Δ, however, it can be approximated as a linear function as in [6–10,16]. In this paper, we use this linear approximate control cost function as the optimization objective.

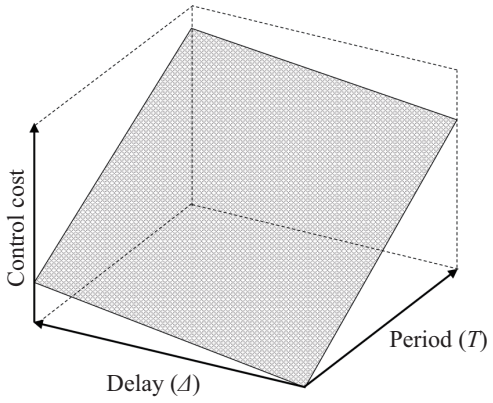


Figure 2. Control cost function, which is a linear function of the system's control period and delay [5].

3.3. Schedulability Constraint

The runnables $\{r_1, r_2, \cdots, r_n\}$ are implemented as RTOS tasks, where runnables with the same period are grouped together and sequentially executed inside a task body when the task is scheduled on a CPU. As most RTOSes only support implicit deadline tasks, their relative deadlines (= periods) should be guaranteed to satisfy runnable-level periodic timing requirements, i.e., p_is. For that, in this paper, we use the L&L utilization bound method, which guarantees the schedulability of a given system if the system utilization is less than or equal to a specific threshold value (i.e., utilization bound) for each scheduling algorithm. For example, the RM scheduling algorithm's utilization bound is roughly 69.3%, and the EDF scheduling algorithm's utilization bound is 100% [11]. We chose to use EDF for its simplicity, where its schedulability condition can be formally expressed as follows:

$$U(p_1, p_2, \cdots, p_n) = \sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1. \tag{11}$$

Although we mainly use the EDF scheduling algorithm throughout this paper, since most scheduling algorithms support the utilization bound method for the schedulability test, we can easily apply our optimization method to other scheduling algorithms like the RM scheduling algorithm. Section 6.5 will deal with this issue in more detail.

3.4. Problem Description

With the system and control performance models and the schedulability constraint presented above, our problem can be defined as follows: With a given AUTOSAR control system composed of DAG-structured runnables $\{r_1, r_2, \cdots, r_n\}$ and a linear control cost function $J(T, \Delta)$ regarding the control target plant, find the optimal runnable periods $(p_1, p_2, \cdots, p_n)$ that minimize the control cost while satisfying the system's schedulability constraint. More formally, our problem is as follows:

$$\begin{aligned} \underset{p_1, p_2, \cdots, p_n}{\text{minimize}} \quad & J(T, \Delta) \\ \text{subject to} \quad & U(p_1, p_1, \cdots, p_n) \leq 1. \end{aligned} \tag{12}$$

In this paper, we try to find the theoretically optimal real numbered runnable periods, without explicitly considering neither the grouping of runnables into predefined periodic tasks nor the scheduling granularity (e.g., integer constraints) of a specific RTOS. However, our solution can be used as a baseline foundation for further practical applications after considering the implementation details imposed by a specific RTOS.

4. Analytical Solution for Linear Path Graphs

4.1. LPG Model

Instead of directly going for a general solution, let us begin by solving our optimization problem for a subset of the DAG model, and later extend the solution step by step towards a generalized one. This section specifically deals with the *linear path graph* (LPG) model, which is for graphs with runnables $\{r_1, r_2, \cdots, r_n\}$ such that the edges are given by $E = \{(r_i, r_{i+1}) | 1 \leq i \leq n-1\}$. Figure 3 shows an example LPG with n runnables and $n-1$ edges between them.

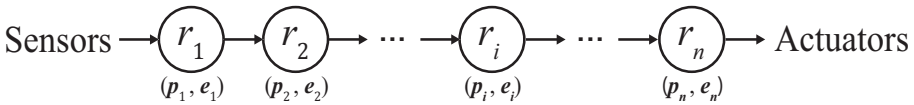


Figure 3. Linear path graph (LPG) model.

4.2. Transformation of Control Cost Function

When solving the optimization problem in the LPG model's scope, the first step is to redefine the control cost function as a function of the free variables of the optimization problem, i.e., runnable periods $(p_1, p_2, \cdots, p_n)$. For the transformation of $J(T, \Delta)$ in Equation (10) into a function of runnable periods, our strategy is to define both T and Δ using only runnable periods considering the LPG model's runnable execution and data flow patterns.

Control period T can be formally defined, from a plant's perspective, as a regular time interval between consecutive actuation instances. However, due to the jitter caused by preemption delays among concurrent runnables, the intervals may vary for each actuation instance. Thus, we consider the longest time interval as the worst-case period T. For LPG-based applications, T can be defined as double the actuator runnable r_n's period p_n, which is

$$T = 2p_n. \tag{13}$$

The worst-case scenario happens when a certain instance of r_n is scheduled at the beginning of its period, whereas, in the next instance, r_n is scheduled at the very end of its period. Assuming that actuation commands are emitted at each completion of r_n instances, the time interval between the actuation commands gets to the longest as possible in that particular scenario, which is $2p_n$. It can be argued that e_n should be considered, and the exact worst-case time interval should be $2p_n - e_n$. However, note that since e_n is the worst-case execution time, real execution times can be much smaller than e_n; thus, for simplicity's sake, we do not take e_n into consideration when defining T.

The end-to-end delay Δ is defined as the time taken for new sensor data to go through runnables until arriving at the actuator. According to the data flow architecture of our system model, the sensor runnable r_1 sends out its output to its neighboring runnables with its own period p_1. Then, the neighboring runnables also send out their outputs with their own periods. With these repeated transmissions, new sensor data originating from the source node r_1 gradually propagate through the runnables toward the sink node, i.e., the actuator runnable r_n. After r_n finally receives the updates, it can decide its actuation commands based on the new sensor data. In an LPG-based application where data flow through only a single path from r_1 to r_n, the worst-case end-to-end delay Δ can be calculated as

$$\Delta = 2p_1 + 2p_2 + \cdots + 2p_n = 2\sum_{i=1}^{n} p_i. \tag{14}$$

The worst case happens as in the following: a runnable r_{i-1} emits its output for r_i at a certain time t. Unfortunately, however, r_i begins just right before t, reading the previous (old) output of r_{i-1}. Let us assume that r_i is scheduled at the very beginning of its period at that time. Then, unfortunately again, the next instance of r_i is scheduled at the very end of its period, reading the new data and emitting its output at the end of the period (i.e., $t + 2p_i$). In the above scenario, the time taken for the data to go through r_i is double the r_i's period $2p_i$. Assuming this scenario happening for every runnable in the path, the end-to-end delay Δ becomes double the sum of all the runnable periods as in Equation (14). By combining Equations (13) and (14), the control cost function in Equation (10) is transformed into as follows:

$$J(p_1, p_2, \cdots, p_n) = 2\alpha p_n + 2\beta \sum_{i=1}^{n} p_i. \tag{15}$$

4.3. Finding the Optimal Runnable Periods

For visual understanding of the optimization process, let us pick an example system with only two runnables $\{r_1, r_2\}$. Then, the transformation of the control cost function is illustrated in Figure 4. In the left-hand side, the original control cost function is depicted, which is transformed into a function of p_1 and p_2 as in the right-hand side by Equation (15). Then, Figure 5 illustrates the optimization process where the schedulability constraint and the transformed control cost function are shown upon

the two-dimensional problem space of p_1 and p_2. In the figure, our optimization objective is to find the lowest point in the control cost plane that is inside the green schedulable area. This concept can be generally extended to n-runnable systems in n-dimensional problem spaces. In general, our original optimization problem in Equation (12) can be transformed into the following using the transformed control cost function:

$$\begin{aligned} \underset{p_1,p_2,\cdots,p_n}{\text{minimize}} \quad & J(p_1,p_2,\cdots,p_n) = 2\alpha p_n + 2\beta \sum_{i=1}^{n} p_i \\ \text{subject to} \quad & U(p_1,p_2,\cdots,p_n) = \sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1. \end{aligned} \tag{16}$$

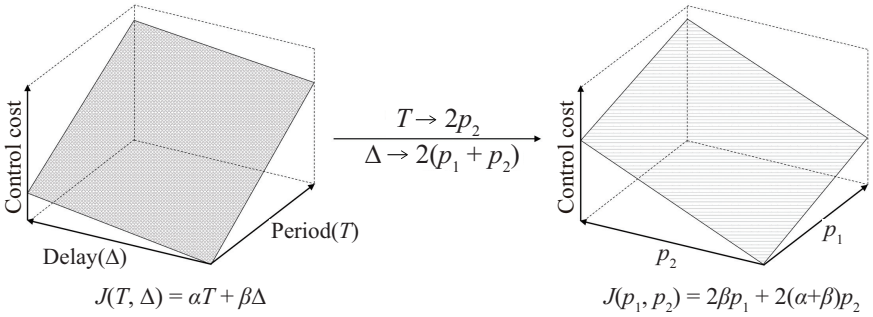


Figure 4. Visually illustrated transformation of control cost function [5].

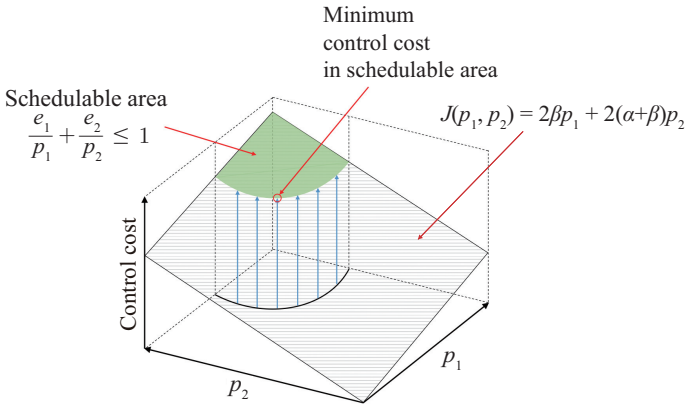


Figure 5. Visually illustrated constrained optimization process [5].

To analytically solve the transformed optimization problem, the Lagranage multiplier method is applied. For the first step, a Lagrange function is formulated as follows:

$$\mathcal{L} = 2\alpha p_n + 2\beta \sum_{i=1}^{n} p_i - \lambda \left(\sum_{i=1}^{n} \frac{e_i}{p_i} - 1 \right). \tag{17}$$

Then, we take the partial derivatives of $\mathcal{L}$ with respect to p_1, p_2, $\cdots$, p_n, and λ, respectively and set them to zeros as follows:

$$\nabla \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial p_1}, \frac{\partial \mathcal{L}}{\partial p_2}, \cdots, \frac{\partial \mathcal{L}}{\partial p_n}, \frac{\partial \mathcal{L}}{\partial \lambda} \right) = 0, \tag{18}$$

which in turn is expanded to the followings:

$$\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial p_1} = 2\beta + \frac{e_1}{p_1^2}\lambda = 0,\\
&\frac{\partial \mathcal{L}}{\partial p_2} = 2\beta + \frac{e_2}{p_2^2}\lambda = 0,\\
&\cdots,\\
&\frac{\partial \mathcal{L}}{\partial p_n} = 2(\alpha+\beta) + \frac{e_n}{p_n^2}\lambda = 0,\\
&\frac{\partial \mathcal{L}}{\partial \lambda} = -\left(\sum_{i=1}^{n}\frac{e_i}{p_i} - 1\right) = 0.
\end{aligned} \tag{19}$$

Then, by isolation λ in the left-hand side of the first in Equation (19), we have

$$\lambda = -\frac{2\beta {p_1}^2}{e_1}, \tag{20}$$

which can be applied to the remaining of Equation (19) except the last one. As a result, p_2, p_3, $\cdots$, and p_n are given in terms of p_1 as in the second to the last of the followings:

$$\begin{aligned}
p_1 &= \sum_{i=1}^{n-1}\sqrt{e_1 e_i} + \sqrt{\frac{(\alpha+\beta)e_1 e_n}{\beta}},\\
p_2 &= p_1\sqrt{\frac{e_2}{e_1}},\\
&\cdots,\\
p_{n-1} &= p_1\sqrt{\frac{e_{n-1}}{e_1}},\\
p_n &= p_1\sqrt{\frac{\beta e_n}{(\alpha+\beta)e_1}}.
\end{aligned} \tag{21}$$

Additionally, p_1 is given as the first of the above by replacing p_2, p_3, $\cdots$, and p_n in the last of Equation (19) with the second to the last of the above. Then, by Equation (21), we can find the real optimal runnable periods for arbitrary LPG-based applications.

5. Analytical Solution for Linear Multipath Graphs

Based on the method for the LPG model explained in Section 4, this section goes one step further to a more complex application model having multiple independent data flows from sensors to actuators.

5.1. LMG Model

When designing automobile control applications, there are cases where a simpler data flow model is preferred instead of using the complex DAG model. The most common such case is when there are several independent parallel data flows from sensors to actuators. In Figure 6, r_1 is the sensor runnable and r_n is the actuator runnable. Between them, there are m paths where each runnable in the middle part $\{r_2, r_3, \cdots, r_{n-1}\}$ belongs to only one specific path among them. To distinguish such a particular application architecture from general DAGs, we specifically call them the *linear multipath graph* (LMG) model. Although the LMG model can be applied to a limited range of applications, it is meaningful since there is an increasing need for integrating independent control algorithms to develop integrated control systems or multi-functional ECUs [36–38]. In such new systems, sensor data propagate through multiple independent paths of runnables to the actuators.

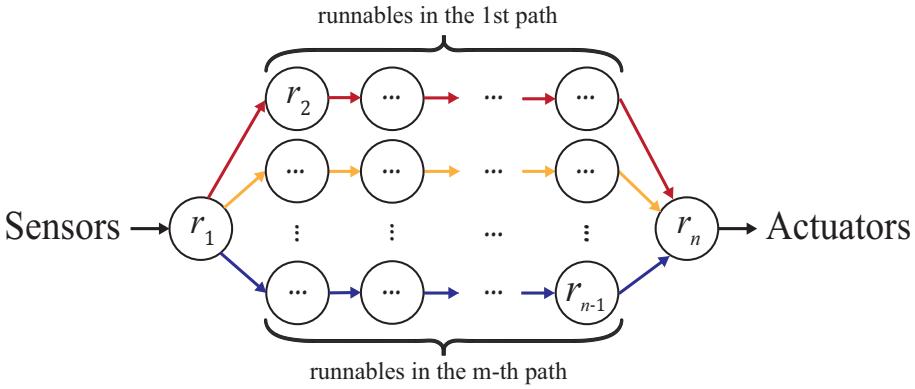


Figure 6. Linear multipath graph (LMG) model with m independent paths identified by edges with different colors [27].

5.2. Transformation of Control Cost Function

For the optimization, the objective function $J(T, \Delta)$ in Equation (10) should be transformed into a function of runnable periods $(p_1, p_2, \cdots, p_n)$. For that, in the same way in Section 4.2, the period T is transformed into as follows:

$$T = 2p_n. \tag{22}$$

For the end-to-end delay Δ, however, we cannot simply reuse the method in Section 4.2 since we have multiple paths with possibly different lengths. Thus, Δ should be defined as the length of the longest path among them to represent the worst-case end-to-end delay. More specifically, let us remind that P_i denotes the i-th path of m independent paths $\{P_1, P_2, \cdots, P_m\}$. Then, Δ is defined as follows:

$$\Delta = \max_{1 \le i \le m} \left(\sum_{j \in P_i} 2p_j \right). \tag{23}$$

Thus, our original optimization problem is transformed into as follows:

$$\begin{aligned} \underset{p_1, p_2, \cdots, p_n}{\text{minimize}} \quad & J(p_1, p_2, \cdots, p_n) = 2\alpha p_n + \beta \max_{1 \le i \le m} \left(\sum_{j \in P_i} 2p_j \right) \\ \text{subject to} \quad & U(p_1, p_2, \cdots, p_n) = \sum_{i=1}^{n} \frac{e_i}{p_i} \le 1, \end{aligned} \tag{24}$$

where unlike the LPG model, the max operator introduces nonlinearity making it difficult to develop an analytical solution. Fortunately, however, due to the LMG model's workload characteristics, we can simplify the problem by defining an equilibrium state, which is to be obtained to find the optimal runnable periods in the LMG model. The equilibrium state of an LMG can be defined as follows:

Definition 6. (Equilibrium state) *For a set of m paths of a given LMG G, which is denoted by* $\mathbb{P}(G) = \{P_1, P_2, \cdots, P_m\}$*, G is in its equilibrium state if and only if*

$$\sum_{i \in P_1} p_i = \sum_{i \in P_2} p_i = \cdots = \sum_{i \in P_m} p_i. \tag{25}$$

Theorem 1. (Equilibrium state theorem) *If G is an LMG with its optimal runnable periods, G is always in its equilibrium state.*

Proof of Theorem 1. If G is with its optimal runnable periods while not in the equilibrium state, we can increase runnable periods that do not belong to the critical path, without affecting the end-to-end

delay. Then, the increased runnable periods will make a lower system utilization, which can be used to further decrease the end-to-end delay by shortening runnable periods in the critical path. Thus, we can conclude that G is not with the optimal runnable periods, which is a contradiction. □

By the equilibrium state theorem, we can narrow down the problem space without sacrificing the optimality by excluding non-equilibrium states from the problem space. To express the equilibrium state more efficiently, Δ is re-expressed by breaking it into three parts as

$$\Delta = 2p_1 + \max_{1 \le i \le m} \left(\sum_{j \in \hat{P}_i} 2p_j \right) + 2p_n \tag{26}$$

with a helper notation $\hat{P}_i = P_i - \{1, n\}$. Then, to explicitly express the equilibrium state, we define a new notation p_* as in the following:

$$p_* = \sum_{i \in \hat{P}_1} p_i = \sum_{i \in \hat{P}_2} p_i = \cdots = \sum_{i \in \hat{P}_m} p_i, \tag{27}$$

which is an aggregate variable representing the path length of the middle part in the equilibrium state. Then, by using p_*, Δ can be re-expressed from Equations (26) and (27) as follows:

$$\Delta = 2(p_1 + p_* + p_n). \tag{28}$$

Finally, the control cost function $J(T, \Delta)$ from Equation (10) is rewritten as a function of (p_1, p_*, p_n) by Equations (22) and (28) as in the following:

$$J(p_1, p_*, p_n) = 2\alpha p_n + 2\beta(p_1 + p_* + p_n). \tag{29}$$

With this transformed control cost function $J(p_1, p_*, p_n)$, our original problem of n runnable periods is transformed into a problem of three free variables (p_1, p_*, p_n). Then, once they are decided, p_* is distributed to runnables along each path. For that, we use a heuristic that runnables with larger e_is are assigned with longer p_is. More specifically, we assign p_is strictly proportional to e_is. Following this assignment rule, runnable periods p_js for each $\hat{P}_i$ are decided as follows:

$$\forall i \in [1 \,..\, m] \; \forall j \in \hat{P}_i : \; p_j = \frac{e_j}{\sum_{k \in \hat{P}_i} e_k} p_*. \tag{30}$$

5.3. Transformation of Schedulability Constraint Function

This subsection transforms the schedulability constraint function in Equation (11) to a function of (p_1, p_*, p_n). First, the original function $U(p_1, p_2, \cdots, p_n)$ is re-expressed by breaking it into three parts, and the middle part is arranged by grouping the runnables by the paths they belong to. The new expression can be simply comprehended as the sum of m per-path sums of utilizations as in the following:

$$\begin{aligned} U(p_1, p_2, \cdots, p_n) &= \frac{e_1}{p_1} + \left(\frac{e_2}{p_2} + \cdots + \frac{e_{n-1}}{p_{n-1}} \right) + \frac{e_n}{p_n} \\ &= \frac{e_1}{p_1} + \sum_{i=1}^{m} \left(\sum_{j \in \hat{P}_i} \frac{e_j}{p_j} \right) + \frac{e_n}{p_n}. \end{aligned} \tag{31}$$

Then, to transform each i-th per-path utilization sum into a function of p_*, Equation (30) is applied to eliminate p_j as in the following:

$$\begin{aligned}\sum_{j\in\hat{P}_i}\frac{e_j}{p_j} &= \sum_{j\in\hat{P}_i}\frac{e_j}{\frac{e_j}{\sum_{k\in\hat{P}_i}e_k}p_*} = \sum_{j\in\hat{P}_i}\frac{\sum_{k\in\hat{P}_i}e_k}{p_*}\\ &= \left|\hat{P}_i\right|\frac{\sum_{k\in\hat{P}_i}e_k}{p_*} = \frac{\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right|e_k}{p_*}\end{aligned} \tag{32}$$

where $\left|\hat{P}_i\right|$ denotes the number of elements in the ordered set $\hat{P}_i$. Finally, our utilization constraint is transformed into as follows:

$$U(p_1,p_*,p_n) = \frac{e_1}{p_1} + \frac{\sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right|e_k}{p_*} + \frac{e_n}{p_n} \le 1. \tag{33}$$

5.4. Finding the Optimal Runnable Periods

After the transformation of the control cost function and the schedulability constraint function, our runnable periods optimization problem for the LMG model can be formulated with the three free variables (p_1, p_*, p_n) as follows:

$$\begin{aligned}\underset{p_1,p_*,p_n}{\text{minimize}} \quad & J(p_1,p_*,p_n) = 2\alpha p_n + 2\beta(p_1+p_*+p_n)\\ \text{subject to} \quad & U(p_1,p_*,p_n) = \frac{e_1}{p_1} + \frac{\sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right|e_k}{p_*} + \frac{e_n}{p_n} \le 1.\end{aligned} \tag{34}$$

To solve the optimization problem, a Lagrange function is formulated as follows:

$$\mathcal{L} = 2\alpha p_n + 2\beta(p_1+p_*+p_n) - \lambda\left(\frac{e_1}{p_1} + \frac{\sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right|e_k}{p_*} + \frac{e_n}{p_n} - 1\right). \tag{35}$$

Then, we take the partial derivatives of $\mathcal{L}$ with respect to p_1, p_*, p_n, and λ, respectively and set them to zeros as follows:

$$\nabla\mathcal{L} = \left(\frac{\partial\mathcal{L}}{\partial p_1}, \frac{\partial\mathcal{L}}{\partial p_*}, \frac{\partial\mathcal{L}}{\partial p_n}, \frac{\partial\mathcal{L}}{\partial\lambda}\right) = 0, \tag{36}$$

which in turn is expanded to the followings:

$$\begin{aligned}
\frac{\partial L}{\partial p_1} &= 2\beta + \frac{e_1}{p_1^2}\lambda = 0,\\
\frac{\partial L}{\partial p_*} &= 2\beta + \frac{\sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right| e_k}{p_*^2}\lambda = 0,\\
\frac{\partial L}{\partial p_n} &= 2(\alpha+\beta) + \frac{e_n}{p_n^2}\lambda = 0,\\
\frac{\partial L}{\partial \lambda} &= -\left(\frac{e_1}{p_1} + \frac{\sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right| e_k}{p_*} + \frac{e_n}{p_n} - 1\right) = 0.
\end{aligned} \tag{37}$$

Then, the first, second, and third of Equation (37) are rearranged by isolating λ in each left-hand side as follows:

$$\begin{aligned}
\lambda &= -2\beta\,\frac{p_1^2}{e_1},\\
\lambda &= -2\beta\,\frac{p_*^2}{\sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right| e_k},\\
\lambda &= -2(\alpha+\beta)\,\frac{p_n^2}{e_n}.
\end{aligned} \tag{38}$$

Then, by combining the first and second of Equation (38), we have the following:

$$2\beta\,\frac{p_1^2}{e_1} = 2\beta\,\frac{p_*^2}{\sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right| e_k} \Longrightarrow \frac{1}{p_*} = \frac{1}{p_1}\sqrt{\frac{e_1}{\sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right| e_k}}. \tag{39}$$

By combining the first and third of Equation (38), we have the following:

$$2\beta\,\frac{p_1^2}{e_1} = 2(\alpha+\beta)\,\frac{p_n^2}{e_n} \Longrightarrow \frac{1}{p_n} = \frac{1}{p_1}\sqrt{\frac{\alpha+\beta}{\beta}}\sqrt{\frac{e_1}{e_n}}. \tag{40}$$

By replacing $\frac{1}{p_*}$ and $\frac{1}{p_n}$ in the last of Equation (37) with the findings in Equations (39) and (40), we have the following:

$$\frac{1}{p_1}\left(e_1 + \sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right| e_k\sqrt{\frac{e_1}{\sum_{i=1}^{m}\sum_{k\in\hat{P}_i}\left|\hat{P}_i\right| e_k}} + e_n\sqrt{\frac{(\alpha+\beta)\,e_1}{\beta e_n}}\right) = 1. \tag{41}$$

Finally, from Equations (39)–(41), we have the following solution:

$$p_1 = e_1 + \sqrt{e_1 \sum_{i=1}^{m} \sum_{k \in \hat{P}_i} \left|\hat{P}_i\right| e_k} + \sqrt{\frac{(\alpha+\beta) e_1 e_n}{\beta}}$$

$$p_* = p_1 \sqrt{\frac{\sum_{i=1}^{m} \sum_{k \in \hat{P}_i} \left|\hat{P}_i\right| e_k}{e_1}} \tag{42}$$

$$p_n = p_1 \sqrt{\frac{\beta e_n}{(\alpha+\beta)\, e_1}}.$$

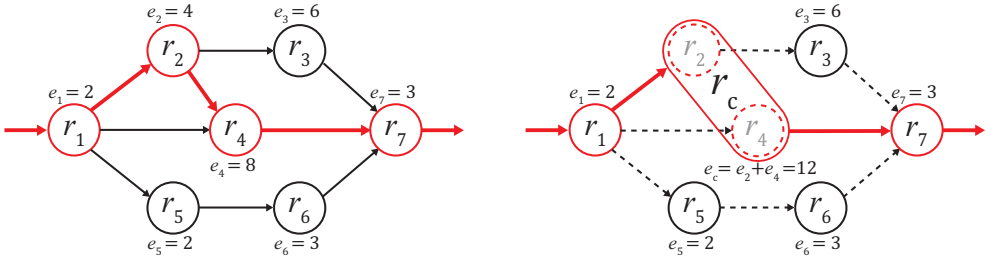
We have one remaining step of deciding $(p_2, p_3, \cdots, p_{n-1})$. For that, we distribute p_* to runnables in each path in proportion to their e_is as in Equation (30). It is also worth noting that even with the equilibrium state theorem, we cannot find the real optimal solutions since we lose the optimality while distributing p_* with a heuristic. Nevertheless, we can find high-quality solutions close to the real optimal runnable periods. Interested readers are referred to our previous work [27].

6. Generalized Analytical Method for Directed Acyclic Graphs

This section generalizes the previously explained methods for the LPG model and the LMG model to the general DAG model. Both methods are not usable for a general DAG-based application for their limited applicability. In particular, since our method for the LMG model assumes that there is no such runnable that belongs to different paths at the same time, it is not applicable to DAGs with at least one such runnable. If we forcibly try it, Equation (30) may yield two different, hence, conflicting results for such runnables. Thus, we need a separate method for the DAG model.

6.1. DAG Model and Its Challenge

The DAG model is already explained in Section 3.1. Hence this subsection just highlights how it is different from the LPG model and the LMG model and presents a challenge that does not exist in the previous models. As noted earlier, there is only one path in the LPG model, making it easy to define the system's end-to-end delay. In the LMG model, even though there are multiple paths, we can use the equilibrium state theorem to simply represent them together by their identical path length in the middle part, denoted by p_*. Figure 7a shows a simple DAG that is not an LPG nor an LMG. In the figure, note that r_4 belongs to the following two different paths: $< r_1, r_2, r_4, r_7 >$ and $< r_1, r_4, r_7 >$. As a runnable period cannot be zero, the former is always longer than the latter. Thus, unlike the LMG model where we can always make an equilibrium state, we cannot always make an equilibrium state in the DAG model.



(a) Original DAG (b) Transformed DAG

Figure 7. An example of DAG explaining the concept of the critical path with r_c.

6.2. Transformation of Control Cost Function

With the above challenge, let us transform the control cost function to a function of runnable periods. For the period T, we can use the same method as for the LPG model and the LMG model since it is only concerned with the actuator runnable r_n. Thus, T is transformed into as follows:

$$T = 2p_n. \tag{43}$$

Unfortunately, however, when transforming the end-to-end delay Δ, we cannot simply reuse the method for the LMG model in Section 5.2 since we cannot be sure that an equilibrium state can be made. To handle this challenge, let us begin with the general definition of Δ as in the following, assuming m paths $\{P_1, P_2, \cdots, P_m\}$:

$$\Delta = \max_{1 \le i \le m} \left(\sum_{j \in P_i} 2p_j \right). \tag{44}$$

For example, in Figure 7a, there are four paths, $P_1 =< 1, 2, 3, 7 >$, $P_2 =< 1, 2, 4, 7 >$, $P_3 =< 1, 4, 7 >$, and $P_4 =< 1, 5, 6, 7 >$. Among them, it is apparent that P_3 cannot be the critical path since it is always shorter than P_2. However, among P_1, P_2, and P_4, we cannot be sure which is the longest since all of them can be the critical path according to how we decide p_is.

To overcome this challenge, we propose to employ a heuristic with a clear rule regarding which path should be the critical path. For that, with given e_is, we employ a heuristic described by the following:

$$p_i \propto e_i, \text{ for } 2 \le i \le n-1. \tag{45}$$

Certainly, the most important benefit from making p_is simply proportional to e_is is that we can simply decide the critical path based on the path weights (See Definition 3) such that we can choose the heaviest path (See Definition 5) as the critical path. For example, in Figure 7a, the weights of paths, P_1 to P_4, are calculated as 15, 17, 13, and 10, respectively. Then, P_2, specified by the red color, turns out to be the heaviest path. By the heuristic, it is used as the critical path. The intuition behind this heuristic is that we give longer periods to runnables with longer execution times to evenly distribute the system utilization across runnables, eliminating possible bottlenecks. Note that p_1 and p_n are excluded in Equation (45) as they have no effect on deciding the critical path.

For further explanations, we introduce a new notation r_c, which is defined as the set of runnables in the critical path excluding r_1 and r_n. Figure 7b shows $r_c = \{r_2, r_4\}$. Then, let us think as if r_c is a virtual *composite runnable* combining its member runnables just like the ellipse covering r_2 and r_4 in the figure. Then, the critical path can be thought of as a three-runnable ordered set $< r_1, r_c, r_n >$. For r_c, let us also define e_c and p_c as in the followings:

$$e_c = \sum_{i \in r_c} e_i \tag{46}$$

and

$$p_c = \sum_{i \in r_c} p_i. \tag{47}$$

Based on the above notations, Δ can be defined as follows:

$$\Delta = 2(p_1 + p_c + p_n), \tag{48}$$

which makes the control cost function as follows with Equation (43):

$$J(p_1, p_c, p_n) = 2\alpha p_n + 2\beta(p_1 + p_c + p_n). \tag{49}$$

Then, regarding how to derive p_2, p_3, $\cdots$, and p_{n-1} from p_c, we use the following assignment rule following Equation (45):

$$p_i = \frac{e_i}{e_c} p_c \text{ for } 2 \leq i \leq n-1. \tag{50}$$

Under the above assignment rule, the followings are ensured:

- p_is are always proportional to e_is;
- The length of the critical path excluding r_1 and r_n is equal to p_c;
- The length of any other path is always shorter than p_c.

As an example, in Figure 7b, we can find that $e_c = e_2 + e_4 = 12$ and $p_c = p_2 + p_4$. Once p_c is decided, each p_i can be derived as in the followings according to Equation (50):

$$p_2 = \frac{4}{12} p_c,\ p_3 = \frac{6}{e_c} p_c,\ p_4 = \frac{8}{12} p_c,\ p_5 = \frac{2}{12} p_c, \text{ and } p_6 = \frac{3}{12} p_c. \tag{51}$$

6.3. Transformation of Schedulability Constraint Function

The schedulability constraint in Equation (11) uses a function of n runnable periods. Thus, it is transformed into a function of the three free variables (p_1, p_c, p_n), following the rule in Equation (50) as follows:

$$\begin{aligned} U(p_1, p_2, \cdots, p_n) &= \frac{e_1}{p_1} + \left(\frac{e_2}{p_2} + \cdots + \frac{e_{n-1}}{p_{n-1}} \right) + \frac{e_n}{p_n} = \\ &= \frac{e_1}{p_1} + \left(\frac{e_2}{\frac{e_2}{e_c} p_c} + \cdots + \frac{e_{n-1}}{\frac{e_{n-1}}{e_c} p_c} \right) + \frac{e_n}{p_n} \\ &= \frac{e_1}{p_1} + (n-2)\frac{e_c}{p_c} + \frac{e_n}{p_n}. \end{aligned} \tag{52}$$

Now, the utilization function $U(p_1, p_2, \cdots, p_n)$ can be replaced by a function of (p_1, p_c, p_n) as in the following:

$$U(p_1, p_c, p_n) = \frac{e_1}{p_1} + (n-2)\frac{e_c}{p_c} + \frac{e_n}{p_n}. \tag{53}$$

6.4. Finding the Optimal Runnable Periods

Based on the control cost function in Equation (49) and the utilization function in Equation (53), our runnable periods optimization problem for the DAG model can be formulated with the three free variables (p_1, p_c, p_n) as follows:

$$\begin{aligned} \underset{p_1, p_c, p_n}{\text{minimize}} \quad & J(p_1, p_c, p_n) = 2\alpha p_n + 2\beta(p_1 + p_c + p_n) \\ \text{subject to} \quad & U(p_1, p_c, p_n) = \frac{e_1}{p_1} + (n-2)\frac{e_c}{p_c} + \frac{e_n}{p_n} \leq 1. \end{aligned} \tag{54}$$

To solve the optimization problem, a Lagrange function is formulated as follows:

$$\begin{aligned} \mathcal{L} &= J(p_1, p_c, p_n) - \lambda(U(p_1, p_c, p_n) - 1) \\ &= 2\alpha p_n + 2\beta(p_1 + p_c + p_n) - \lambda \left(\frac{e_1}{p_1} + (n-2)\frac{e_c}{p_c} + \frac{e_n}{p_n} - 1 \right). \end{aligned} \tag{55}$$

Then, we take the partial derivatives of $\mathcal{L}$ with respect to p_1, p_c, p_n, and λ, respectively and set them to zeros as follows:

$$\nabla \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial p_1}, \frac{\partial \mathcal{L}}{\partial p_c}, \frac{\partial \mathcal{L}}{\partial p_n}, \frac{\partial \mathcal{L}}{\partial \lambda} \right) = 0, \tag{56}$$

which in turn is expanded to the followings:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial p_1} &= 2\beta + \frac{e_1}{p_1^2}\lambda = 0, \\
\frac{\partial \mathcal{L}}{\partial p_c} &= 2\beta + (n-2)\frac{e_c}{p_c^2}\lambda = 0, \\
\frac{\partial \mathcal{L}}{\partial p_n} &= 2(\alpha+\beta) + \frac{e_n}{p_n^2}\lambda = 0, \\
\frac{\partial \mathcal{L}}{\partial \lambda} &= -\left(\frac{e_1}{p_1} + (n-2)\frac{e_c}{p_c} + \frac{e_n}{p_n} - 1\right) = 0.
\end{aligned} \tag{57}$$

Then, the first, second, and third of Equation (57) are rearranged by isolating λ in each left-hand side as follows:

$$\begin{aligned}
\lambda &= -2\beta\,\frac{p_1^2}{e_1}, \\
\lambda &= -2\beta\,\frac{p_c^2}{(n-2)e_c}, \\
\lambda &= -2(\alpha+\beta)\,\frac{p_n^2}{e_n}.
\end{aligned} \tag{58}$$

Then, by combining the first and second of Equation (58), we have the following:

$$2\beta\,\frac{p_1^2}{e_1} = 2\beta\,\frac{p_c^2}{(n-2)e_c} \Longrightarrow \frac{1}{p_c} = \frac{1}{p_1}\sqrt{\frac{e_1}{(n-2)e_c}}. \tag{59}$$

By combining the first and third of Equation (58), we have the following:

$$2\beta\,\frac{p_1^2}{e_1} = 2(\alpha+\beta)\,\frac{p_n^2}{e_n} \Longrightarrow \frac{1}{p_n} = \frac{1}{p_1}\sqrt{\frac{(\alpha+\beta)e_1}{\beta e_n}}. \tag{60}$$

By replacing $\frac{1}{p_c}$ and $\frac{1}{p_n}$ in the last of Equation (57) with the findings in Equations (59) and (60), we have the following:

$$\frac{1}{p_1}\left(e_1 + (n-2)e_c\sqrt{\frac{e_1}{(n-2)e_c}} + e_n\sqrt{\frac{(\alpha+\beta)e_1}{\beta e_n}}\right) = 1. \tag{61}$$

Finally, from Equations (59)–(61), we have the following solution:

$$\begin{aligned}
p_1 &= e_1 + \sqrt{(n-2)e_1e_c} + \sqrt{\frac{(\alpha+\beta)e_1e_n}{\beta}} \\
p_c &= p_1\sqrt{\frac{(n-2)e_c}{e_1}} \\
p_n &= p_1\sqrt{\frac{\beta e_n}{(\alpha+\beta)e_1}}.
\end{aligned} \tag{62}$$

After finding the optimal (p_1, p_c, p_n), the remaining runnable periods $(p_2, p_3, \cdots, p_{n-1})$ should be decided, too. For that, the assignment rule in Equation (50) is used.

6.5. Applying Our Method to Other Scheduling Algorithms

Thus far, we assumed the EDF scheduling algorithm for the underlying RTOS scheduling. However, other scheduling algorithms such as RM are also widely used in the automotive industry.

With this motivation, this subsection explains how we can apply our method to different scheduling algorithms. Fortunately, most real-time scheduling algorithms provide a schedulability analysis method based on the L&L utilization bound, where if the system utilization is less than or equal to a specific threshold value called a utilization bound, denoted by U_B, the system is guaranteed to be schedulable. As noted earlier, U_B for EDF is 100%, whereas U_B for RM is 69.3%. Then, the schedulability condition is formally expressed as follows:

$$U(p_1, p_2, \cdots, p_n) = \sum_{i=1}^{n} \frac{e_i}{p_i} \leq U_B. \tag{63}$$

Then, our optimization problem is slightly changed from Equation (54) to the following using U_B in the schedulability constraint:

$$\begin{aligned} \underset{p_1, p_c, p_n}{\text{minimize}} \quad & J(p_1, p_c, p_n) = 2\alpha p_n + 2\beta(p_1 + p_c + p_n) \\ \text{subject to} \quad & U(p_1, p_c, p_n) = \frac{e_1}{p_1} + (n-2)\frac{e_c}{p_c} + \frac{e_n}{p_n} \leq U_B. \end{aligned} \tag{64}$$

Then, its Lagrange function is also modified as follows:

$$\begin{aligned} \mathcal{L} &= J(p_1, p_c, p_n) - \lambda(U(p_1, p_c, p_n) - U_B) \\ &= 2\alpha p_n + 2\beta(p_1 + p_c + p_n) - \lambda\left(\frac{e_1}{p_1} + (n-2)\frac{e_c}{p_c} + \frac{e_n}{p_n} - U_B\right). \end{aligned} \tag{65}$$

Solving the above Lagrange function yields the following solution:

$$\begin{aligned} p_1 &= \frac{e_1 + \sqrt{(n-2)e_1 e_c} + \sqrt{\frac{(\alpha+\beta)e_1 e_n}{\beta}}}{U_B} \\ p_c &= p_1\sqrt{\frac{(n-2)e_c}{e_1}} \\ p_n &= p_1\sqrt{\frac{\beta e_n}{(\alpha+\beta)e_1}}. \end{aligned} \tag{66}$$

From the above, $(p_2, p_3, \cdots, p_{n-1})$ are decided by the assignment rule in Equation (50). Note that we can apply our method to any scheduling algorithm whose schedulability analysis can be conducted by the L&L utilization bound method.

6.6. Algorithm

Algorithm 1 shows a complete procedure for finding optimal runnable periods for a given DAG by our analytical method. As inputs, the algorithm accepts (i) a list of worst-case execution times for n runnables, (ii) a list of m paths in the DAG, (iii) α and β of a given control cost function, and (iv) a utilization bound U_B. The algorithm just needs a list of paths instead of the entire structure of the DAG. Thus, the algorithm itself does not consider generating paths from a DAG. As an output, the algorithm returns a list of optimal runnable periods. Note that the algorithm's computational complexity is just $\mathcal{O}(n \times m)$, which is caused when finding the heaviest path in line 3. This polynomial time complexity makes our analytical method practical for use with large systems.

Algorithm 1: Find optimal runnable periods for a DAG

Input: $\mathbb{E} =< e_1, e_2, \cdots, e_n >$: list of runnable execution times
Input: $\mathbb{P} =< P_1, P_2, \cdots, P_m >$: list of paths
Input: (α, β): coefficients of a control cost function
Input: U_B: utilization bound
Output: $< p_1, p_2, \cdots, p_n >$: optimal runnable periods
Function `FindOptimalRunnablePeriods`($\mathbb{E}$, $\mathbb{P}$, α, β, U_B)**:**

1 $n \leftarrow |\mathbb{E}|$ `/* n: number of runnables */`
2 $m \leftarrow |\mathbb{P}|$ `/* m: number of paths */`
3 $e_c \leftarrow \max_{P \in \mathbb{P}} \left(\sum_{i \in P} e_i \right) - (e_1 + e_n)$
4 $p_1 \leftarrow \dfrac{e_1 + \sqrt{(n-2)e_1 e_c} + \sqrt{\dfrac{(\alpha+\beta)e_1 e_n}{\beta}}}{U_B}$
5 $p_c \leftarrow p_1 \sqrt{\dfrac{(n-2)e_c}{e_1}}$
6 $p_n \leftarrow p_1 \sqrt{\dfrac{\beta e_n}{(\alpha+\beta)e_1}}$
7 **for** $i \leftarrow 2$ **to** $n-1$ **do**
8 $\quad p_i \leftarrow \dfrac{e_i}{e_c} p_c$
9 **return** $< p_1, p_2, \cdots, p_n >$

6.7. Applying Our Method to Conventional Task-Based Systems

Many control applications, but for the automotive industry, are still designed as a set of periodic real-time tasks. Thus, it can be beneficial if we can apply our runnable periods optimization method to such traditional control systems. For that, we first classify them into two different categories. The first is for systems with independent tasks with multiple target plants [12,14,15] and the second is for systems composed of periodic tasks with DAG-based data dependencies [18–20].

Note that the applications in the second category have a strong resemblance to our assumed system model. If we simply assume one-to-one mappings from runnables to tasks, our method for the runnable periods optimization can be applied to systems with periodic tasks without much modifications. However, the applications in the first category cannot make use of our method due to their disagreeing application model with ours. However, for those applications, traditional control-scheduling codesign methods [9,12–15] can be used instead.

7. Evaluation

This section specifically evaluates our optimization method for the general DAG model. Readers interested in the evaluation results for the LPG and LMG models are referred to [27]. More specifically, we evaluate our optimization method by answering the following questions:

- Q1: Is our analytical method able to find near-optimal runnable periods?
- Q2: Is it practical to find real optimal solutions by the exhaustive search method?
- Q3: Is our method practically competitive when optimizing large systems?

7.1. Evaluation Method

For the evaluation, we have to consider the following: (i) workload synthesis, (ii) control cost functions, (iii) optimization algorithms, and (iv) performance metrics. In the remainder of this subsection, the above topics are discussed to explain our evaluation method.

Workload synthesis. As representative AUTOSAR workloads, a total of nine DAGs are artificially synthesized. Among them, the first six DAGs in Figure 8 are relatively small ones with four to six runnables, whereas the remaining three DAGs in Figure 9 are with a relatively large number of runnables ranging from 12 to 25. Note that the DAGs are manually generated, however, understand that the resulting DAGs are purely random without any unfair bias. The small DAGs are used to test the optimality of our method since we can find the real optimal solutions for those small DAGs in Figure 8. On the other hand, we cannot find the real optimal solutions for the large DAGs in Figure 9 due to the vast problem space. However, although we cannot evaluate the optimality of our method with the large DAGs, they are still useful when evaluating our method in comparison to other heuristic optimization methods. Each DAG in the figures is labeled by a notation (nR, mL) representing its size and complexity, where nR denotes n runnables and mL denotes m links (or edges) between them. Basically, with larger n and m, DAGs become more and more complex. For example, the DAG in Figure 8a is labeled by $(4R, 5L)$, which has four runnables and five links, and the DAG in Figure 9c, labeled by $(25R, 34L)$, has 25 runnables and 34 links. For each DAG, we generate 100 sets of random runnable execution times uniformly distributed in the range of [20 ms, 150 ms].

Control cost functions. As our optimization objective, we use a linear control cost function as in Equation (10), which is a function of the control period (T) and the end-to-end delay (Δ). As a representative control cost function, we use the following as our default control cost function, unless otherwise stated:

$$J(T, \Delta) = 0.01T + 0.01\Delta. \tag{67}$$

Note the above control cost function has two coefficients $\alpha = 0.01$ and $\beta = 0.01$. Here, however, the relative ratio of α and β is more important than their absolute values since the ratio represents the control cost function's relative sensitivity to the control period and the end-to-end delay. By using the same values for α and β in our default control cost function, the control cost is equally sensitive to the control period and the end-to-end delay. To represent other scenarios with varying relative sensitivities, we also use varying αs and βs in the range of [0.01, 0.05].

Optimization algorithms. To evaluate the optimization performance of our method, for the comparison purpose, we specifically consider the following three optimization methods:

- *OUR:* Near-optimal solutions found by our analytical optimization method;
- *EXH:* Real optimal solutions found by the exhaustive combinatorial search method;
- *PSO:* Solutions optimized by the particle swarm optimization (PSO) method [39].

More specifically, the *EXH* method searches through the discrete integer problem space within [1 ms, 1000 ms] for each runnable period. We compare our method with the *EXH* method to evaluate the optimality of our optimization method with small DAGs to answer the question Q1. To evaluate the optimization performance with large DAGs as an answer to the question Q3, we compare our method with the *PSO* method.

Performance metrics. We mainly use two optimization performance metrics: (i) absolute control costs and (ii) normalized control costs. Absolute control costs are the raw control cost values resulting from an optimization process, whereas normalized control costs are used to compare our method with another algorithm, i.e., the *EXH* method and the *PSO* method. A normalized control cost is defined as the relative ratio of our resulting control cost by letting another method's result as 100%. Besides, to evaluate the practicality of each optimization method, the optimization times are measured with respect to varying application complexities. For the optimization, we use a workstation with an Intel i7-9700k CPU with 64 GB RAM (Dell, Round Rock, TX, USA)

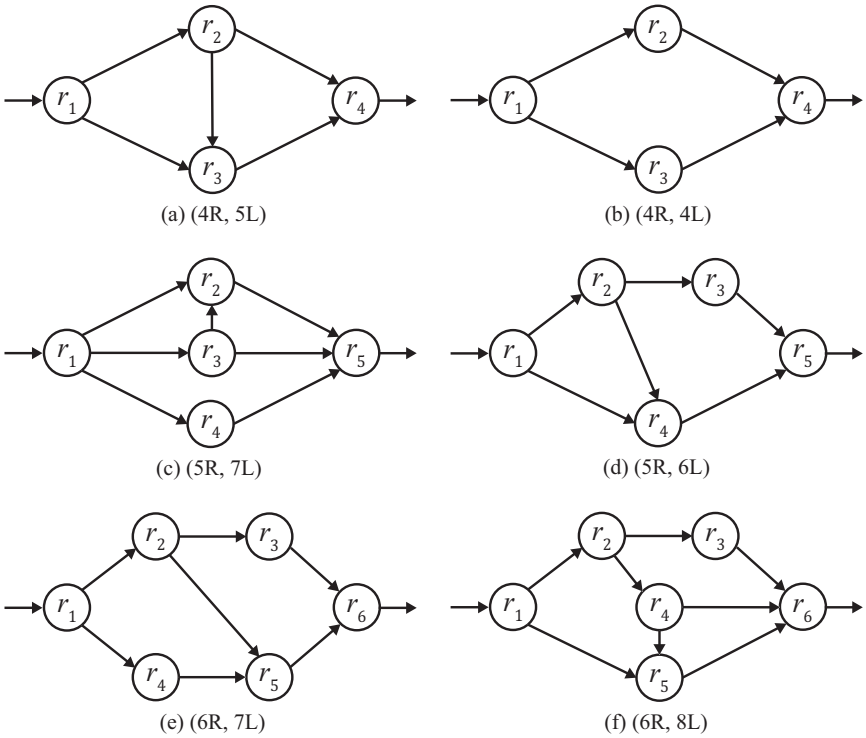


Figure 8. Small DAGs with varying number of runnables (denoted by nR) and links (denoted by mL).

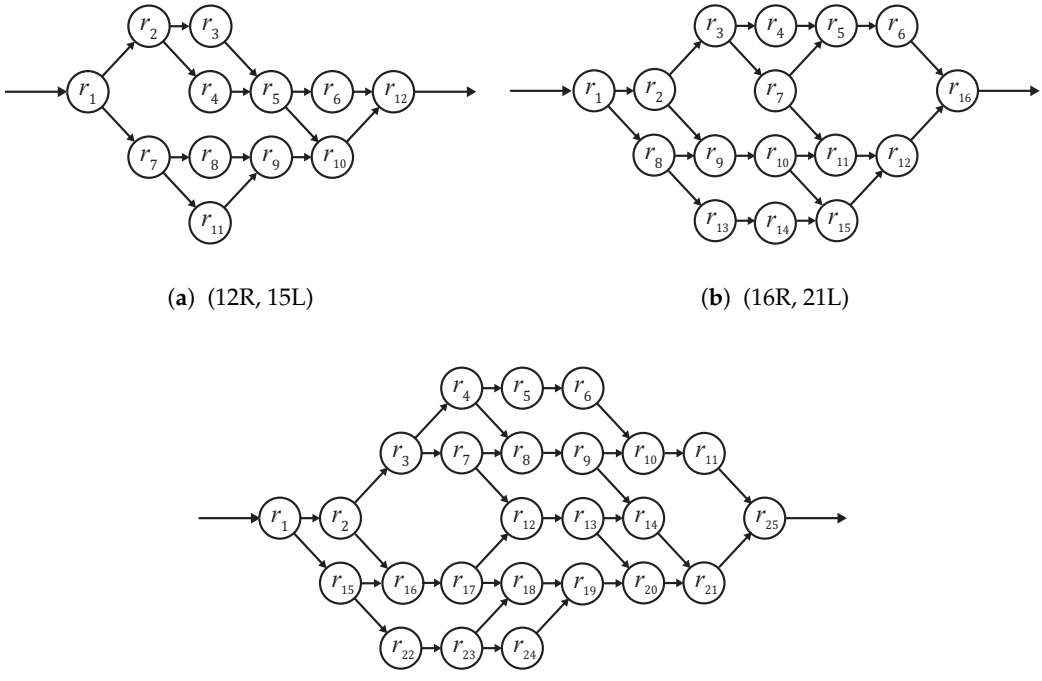


Figure 9. Large DAGs used for evaluating the practicality of our method.

7.2. Evaluation Results and Discussion

Q1: Is our analytical method able to find near-optimal runnable periods? For the six DAGs from Figure 8a–f, their normalized average control costs by the *EXH* method and our method are compared in Figure 10. In the figure, our method shows near-optimal control costs with marginal performance losses compared with the *EXH* method. The minimum control cost increase is just 1.1% and the

maximum is 12.3%. One interesting point is that DAG (a) shows a significantly better performance than the other DAGs. That is because our heuristic for selecting the critical path is always valid in this particular DAG shape. Note that, in DAG (a), $< r_1, r_2, r_3, r_4 >$ will be correctly chosen as the critical path by our heuristic regardless of their execution times since every other path is a subset of it. On the other hand, in other DAGs, there are multiple choices for the critical path. Nonetheless, our method must bet on a certain path based on the given execution times. However, as shown in the figure, the performance losses from the real optimal solutions are marginal.

Thus far, we have assumed the EDF scheduling algorithm. However, in the automotive industry, the RM scheduling algorithm is also widely used for scheduling real-time tasks. In this regard, Figure 11 compares the EDF case where the utilization bound is 100% and the RM case where the utilization bound is 69.3%. More specifically, Figure 11a compares their normalized average control costs across the six DAGs, each of which represents the relative optimization performance compared with the *EXH* method. The figure shows that our method provides not much different optimization results across the two scheduling algorithms. Figure 11b compares their absolute average control costs where EDF shows a significantly lower average control cost since it can efficiently schedule workloads with its higher utilization bound.

To investigate how varying control cost functions affect the optimization performance, Figures 12a,b show normalized average control costs with varying αs and βs, respectively. As shown in the figures, our method provides a consistent optimization performance with varying control cost functions that represent various sensitivities to the control period and the end-to-end delay.

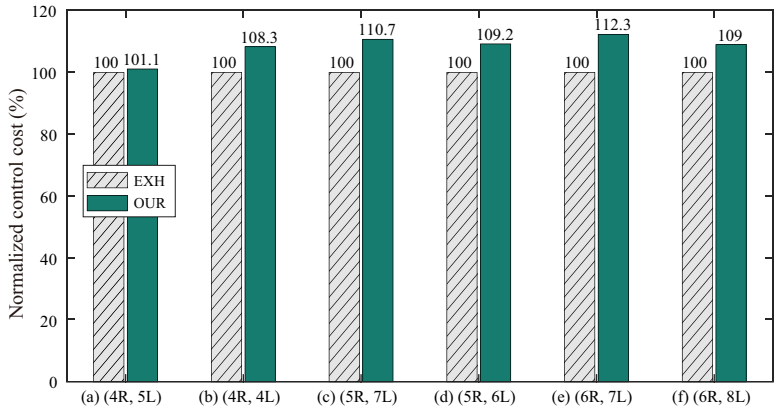
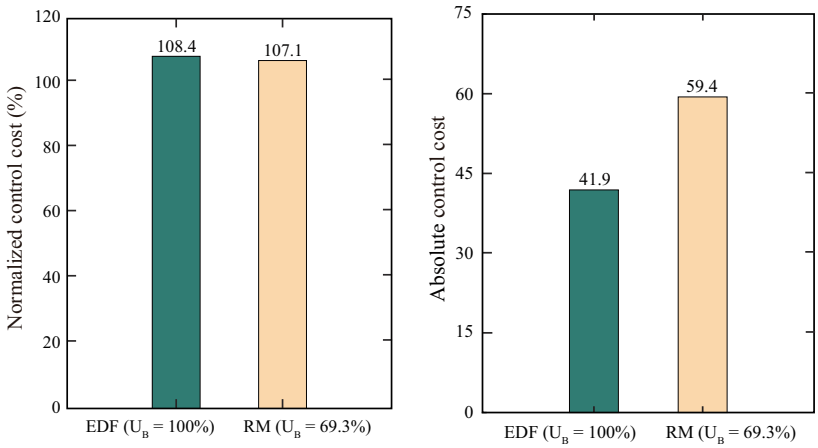
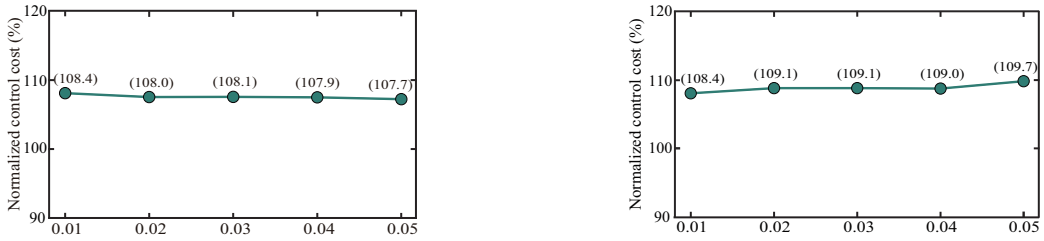


Figure 10. Normalized average control costs of our method compared with the *EXH* method.



(**a**) Normalized average control costs. (**b**) Absolute average control costs.

Figure 11. Comparison of average control costs with the earliest deadline first (EDF) and rate monotonic (RM) scheduling algorithms.



(**a**) With varying αs. (**b**) With varying βs.

Figure 12. Normalized average control costs with varying control cost functions.

Q2: Is it practical to find real optimal solutions by the exhaustive search method? By looking at Q1's results, one can argue that if we can use the *EXH* method to find real optimal runnable periods, why not just use the *EXH* method instead of our method? However, this argument does not hold since the *EXH* method is not usable for large DAGs due to the vast search space when $n > 6$. To answer the question, we evaluate our optimization method in terms of the required time for the optimization process. Table 1 shows the required optimization times for our method and the *EXH* method with varying number of runnables. The numbers inside parentheses are projected numbers. As our method finds the runnable periods by an analytical method, it shows negligible computational complexities as predicted in Algorithm 1. With the *EXH* method, we can find optimal runnable periods in about one month when the numbers of runnables is seven. However, after seven runnables, it is not feasible to use the *EXH* method since it takes more than a year even for small and medium-size systems. Thus, due to the scalability problem, the *EXH* method cannot be used for practical industrial applications, whereas our analytical method can be used for large industry applications.

Table 1. Required optimization times.

Number of Runnables	OUR	EXH
4	0.012 ms	1 s
5	0.016 ms	90 s
6	0.022 ms	3 h
7	0.025 ms	21 days
8	0.031 ms	(350 days)
9	0.036 ms	(3.4 years)
10	0.042 ms	(3065 years)

Q3: Is our method practically competitive when optimizing large systems? From Q1 and Q2, we showed that (i) our analytical method works well with small systems and (ii) real optimal solutions cannot be found for large systems. One remaining question is whether we can apply our analytical method to large systems with sufficient optimization performance. To answer this question, we use the three large DAGs with n = 12, 16, and 25, as in Figure 9. For the comparison, we additionally use an evolutionary metaheuristic algorithm known as particle swarm optimization (denoted by *PSO*) that searches an unknown vast problem space efficiently with swarm intelligence. To implement the *PSO* method, we use the PySwarms library [40]. Figure 13 shows the normalized average control costs of our method compared with the *PSO* method, letting the resulting control costs by the *PSO* method as 100%. As shown in the figure, our method shows significantly better results than the *PSO* method, especially for the largest DAG case. We notice the decreasing trend of normalized average control costs with the increasing complexity of DAGs. With the above results, we can claim that our analytical method performs better than the traditional evolutionary optimization method.

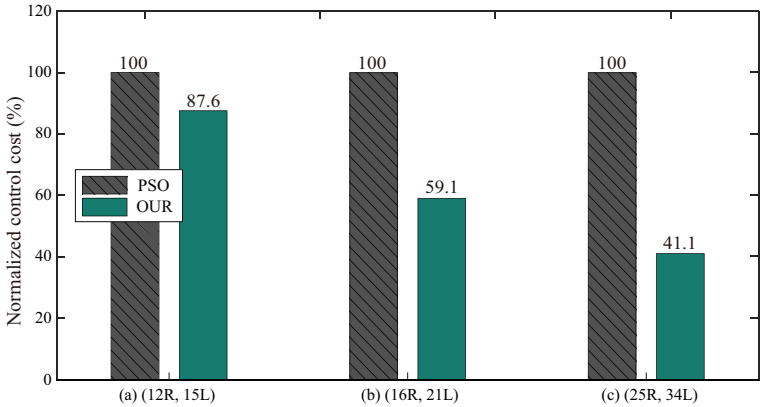


Figure 13. Normalized average control costs of our method for large DAGs compared with the *PSO* method.

8. Conclusions

This paper formulates a runnable periods optimization problem for AUTOSAR control applications and provides an analytical solution based on the Lagrange multiplier method. Our method can find near-optimal solutions that maximize a given system's control performance regardless of the size and complexity of the application. Since the complexity of automobile control applications is rapidly growing due to the recent development of various advanced driver assistance systems and autonomous driving applications, it is no longer feasible to use traditional ad-hoc methods or time-consuming search-based optimization algorithms. Due to the analytical nature of our proposed runnable periods optimization method, we consider that our solution can be readily used in the automotive industry when designing their complex industry-scale AUTOSAR control applications.

Although our method provides a promising solution for optimizing complex applications, our method is only usable when the control cost is given or approximated as a linear function. As the approximation can induce overestimated control costs, we plan to extend our optimization method to nonlinear control cost functions in our future work.

Author Contributions: Conceptualization, J.-C.K.; Data curation, D.C. and T.-W.K.; Methodology, D.C.; Software, D.C. and J.-C.K.; Supervision, J.-C.K.; Visualization, T.-W.K.; Writing—original draft, T.-W.K.; Writing—review & editing, J.-C.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported partly by the Ministry of Land, Infrastructure, and Transport (MOLIT), Korea, through the Transportation Logistics Development Program (20TLRP-B147674-03, Development of Operation Technology for V2X Truck Platooning) and partly by the National Research Foundation (NRF) grant funded by the Korea government (MSIT; Ministry of Science and ICT) (No. 2017R1C1B5018374) and partly by Institute for Information and communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (2014-0-00065, Resilient Cyber-Physical Systems Research).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

RTE	Runtime Environment
DAG	Directed Acyclic Graph
RTOS	Real-Time Operating System
L&L	Liu and Layland
RM	Rate Monotonic
EDF	Earliest Deadline First
LQG	Linear Quadratic Gaussian

ECU	Electronic Control Unit
ILP	Integer Linear Programming
LPG	Linear Path Graph
LMG	Linear Multipath Graph
PSO	Particle Swarm Optimization

References

1. Kehr, S.; Panić, M.; Quiñones, E.; Böddeker, B.; Sandoval, J.B.; Abella, J.; Cazorla, F.J.; Schäfer, G. Supertask: Maximizing runnable-level parallelism in AUTOSAR applications. In Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 25–30.
2. Kim, J.W.; Lee, K.J.; Ahn, H.S. Development of software component architecture for motor-driven power steering control system using AUTOSAR methodology. In Proceedings of the 2015 15th International Conference on Control, Automation and Systems (ICCAS), Busan, Korea, 13–16 October 2015; pp. 1995–1998.
3. Park, J.; Choi, B.W. Design and implementation procedure for an advanced driver assistance system based on an open source AUTOSAR. *Electronics* **2019**, *8*, 1025. [CrossRef]
4. AUTOSAR. Recommended Methods and Practices for Timing Analysis and Design within the AUTOSAR Development Process. AUTOSAR Classic Platform 4.3.1. Available online: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TR_TimingAnalysis.pdf (accessed on 13 July 2020).
5. Kim, T.W.; Lee, G.M.; Kim, J.C. AUTOSAR Runnable Scheduling for Optimal Tradeoff between Control Performance and CPU Utilization. In Proceedings of the 2018 18th International Conference on Control, Automation and Systems (ICCAS), PyeongChang, Korea, 17–20 October 2018; pp. 602–605.
6. Bini, E.; Cervin, A. Delay-aware period assignment in control systems. In Proceedings of the 2008 Real-Time Systems Symposium, Barcelona, Spain, 30 November–3 December 2008; pp. 291–300.
7. Wu, Y.; Bini, E.; Buttazzo, G. A framework for designing embedded real-time controllers. In Proceedings of the 2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Kaohisung, Taiwan, 25–27 August 2008; pp. 303–311.
8. Wu, Y.; Buttazzo, G.; Bini, E.; Cervin, A. Parameter selection for real-time controllers in resource-constrained systems. *IEEE Trans. Ind. Inform.* **2010**, *6*, 610–620. [CrossRef]
9. Fu, H.; Liu, J.; Han, Z.; Shao, Z. A heuristic task periods selection algorithm for real-time control systems on a multi-core processor. *IEEE Access* **2017**, *5*, 24819–24829. [CrossRef]
10. Tan, L.; Du, C.; Dong, Y. Control-performance-driven period and deadline selection for cyber-physical systems. In Proceedings of the 2015 10th Asian Control Conference (ASCC), Kota Kinabalu, Malaysia, 31 May–3 June 2015; pp. 1–6.
11. Liu, C.L.; Layland, J.W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **1973**, *20*, 46–61. [CrossRef]
12. Seto, D.; Lehoczky, J.P.; Sha, L.; Shin, K.G. On task schedulability in real-time control systems. In Proceedings of the 17th IEEE Real-Time Systems Symposium, Washington, DC, USA, 4–6 December 1996; pp. 13–21.
13. Seto, D.; Lehoczky, J.P.; Sha, L. Task period selection and schedulability in real-time systems. In Proceedings of the 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279), Madrid, Spain, 2–4 December 1998; pp. 188–198.
14. Enrico Bini, M.D.N. Optimal task rate selection in fixed priority systems. In Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05), Miami, FL, USA, 6–8 December 2005; pp. 399–409.
15. Du, C.; Tan, L.; Dong, Y. Period selection for integrated controller tasks in cyber-physical systems. *Chin. J. Aeronaut.* **2015**, *28*, 894–902. [CrossRef]
16. Xu, Y.; Cervin, A.; Årzén, K.E. Harmonic scheduling and control co-design. In Proceedings of the 2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Daegu, Korea, 17–19 August 2016; pp. 182–187.
17. Cha, H.J.; Jeong, W.H.; Kim, J.C. Control-scheduling codesign exploiting trade-off between task periods and deadlines. *Mob. Inf. Syst.* **2016**, *2016*, 3414816. [CrossRef]

18. Davare, A.; Zhu, Q.; Di Natale, M.; Pinello, C.; Kanajan, S.; Sangiovanni-Vincentelli, A. Period optimization for hard real-time distributed automotive systems. In Proceedings of the 44th annual Design Automation Conference, San Diego, CA, USA, 4–8 June 2007; pp. 278–283.
19. Feiertag, N.; Richter, K.; Nordlander, J.; Jonsson, J. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In Proceedings of the IEEE Real-Time Systems Symposium, Washington, DC, USA, 30 November–3 December 2009.
20. Schlatow, J.; Mostl, M.; Tobuschat, S.; Ishigooka, T.; Ernst, R. Data-age analysis and optimisation for cause-effect chains in automotive control systems. In Proceedings of the 2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES), Graz, Austria, 6–8 June 2018; pp. 1–9.
21. Dürr, M.; Brüggen, G.V.D.; Chen, K.H.; Chen, J.J. End-to-End Timing Analysis of Sporadic Cause-Effect Chains in Distributed Systems. *ACM Trans. Embed. Comput. Syst.* **2019**, *18*, 1–24. [CrossRef]
22. Choi, J.; Kang, D.; Ha, S. End-to-end latency analysis of cause-effect chains in an engine management system. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 1195–1198.
23. Long, R.; Li, H.; Peng, W.; Zhang, Y.; Zhao, M. An approach to optimize intra-ecu communication based on mapping of autosar runnable entities. In Proceedings of the 2009 International Conference on Embedded Software and Systems, Hangzhou, China, 25–27 May 2009; pp. 138–143.
24. Monot, A.; Navet, N.; Bavoux, B.; Simonot-Lion, F. Multisource software on multicore automotive ECUs—Combining runnable sequencing with task scheduling. *IEEE Trans. Ind. Electron.* **2012**, *59*, 3934–3942. [CrossRef]
25. Saidi, S.E.; Cotard, S.; Chaaban, K.; Marteil, K. An ILP approach for mapping autosar runnables on multi-core architectures. In Proceedings of the 2015 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, Amsterdam, The Netherlands, 21 January 2015; p. 6.
26. Kehr, S.; Quiñones, E.; Langen, D.; Böddeker, B.; Schäfer, G. Parcus: Energy-aware and robust parallelization of AUTOSAR legacy applications. In Proceedings of the 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Pittsburg, PA, USA, 18–21 April 2017; pp. 343–352.
27. Choi, D.; Lee, G.M.; Jeon, W.; Kim, J.C. Control Performance-aware AUTOSAR Runnable Scheduling. *Trans. Korean Soc. Automot. Eng.* **2020**, *28*, 9–17. [CrossRef]
28. Obermaisser, R.; El Salloum, C.; Huber, B.; Kopetz, H. From a federated to an integrated automotive architecture. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2009**, *28*, 956–965. [CrossRef]
29. Di Natale, M.; Sangiovanni-Vincentelli, A.L. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proc. IEEE* **2010**, *98*, 603–620. [CrossRef]
30. AUTOSAR. Virtual Function Bus. AUTOSAR Classic Platform 4.3.1. Available online: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_VFB.pdf (accessed on 13 July 2020).
31. Cervin, A.; Henriksson, D.; Lincoln, B.; Eker, J.; Arzen, K.E. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control. Syst. Mag.* **2003**, *23*, 16–30.
32. AUTOSAR. Specification of Timing Extensions. AUTOSAR Classic Platform 4.3.1. Available online: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_TimingExtensions.pdf (accessed on 13 July 2020).
33. Peraldi-Frati, M.A.; Blom, H.; Karlsson, D.; Kuntz, S. Timing modeling with autosar-current state and future directions. In Proceedings of the 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 12–16 March 2012; pp. 805–809.
34. Lincoln, B.; Cervin, A. Jitterbug: A tool for analysis of real-time control performance. In Proceedings of the 41st IEEE Conference on Decision and Control, Las Vegas, NV, USA, 10–13 December 2002; Volume 2, pp. 1319–1324.
35. Henriksson, D.; Cervin, A.; Årzén, K.E. TrueTime: Simulation of control loops under shared computer resources. *IFAC Proc. Vol.* **2002**, *35*, 417–422. [CrossRef]
36. Yu, F.; Li, D.F.; Crolla, D. Integrated vehicle dynamics control—State-of-the art review. In Proceedings of the 2008 IEEE Vehicle Power and Propulsion Conference, Harbin, China, 3–5 September 2008; pp. 1–6.
37. Ono, E.; Hattori, Y.; Muragishi, Y.; Koibuchi, K. Vehicle dynamics integrated control for four-wheel-distributed steering and four-wheel-distributed traction/braking systems. *Veh. Syst. Dyn.* **2006**, *44*, 139–151. [CrossRef]
38. Akiyama, S.; Tashiro, T. Integrated Vehicle Control System. U.S. Patent 7,047,117, 16 May 2006.

39. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Munich, Germany, 17–19 September 1995; Volume 4, pp. 1942–1948.
40. Miranda, L.J.V. PySwarms a Research Toolkit for Particle Swarm Optimization in Python. Available online: https://pyswarms.readthedocs.io/en/latest/index.html (accessed on 13 July 2020).