Uncertainty Theory Based Partitioning for Cyber-Physical Systems with Uncertain Reliability Analysis

SI CHEN, GUOQI XIE, and RENFA LI, Hunan University, China
KEQIN LI, State University of New York, USA

Reasonable partitioning is a critical issue for **cyber-physical system (CPS)** design. Traditional CPS partitioning methods run in a determined context and depend on the parameter pre-estimations, but they ignore the uncertainty of parameters and hardly consider reliability. The state-of-the-art work proposed an uncertainty theory based CPS partitioning method, which includes parameter uncertainty and reliability analysis, but it only considers linear uncertainty distributions for variables and ignores the uncertainty of reliability. In this paper, we propose an uncertainty theory based CPS partitioning method with uncertain reliability analysis. We convert the uncertain objective and constraint into determined forms; such conversion methods can be applied to all forms of uncertain variables, not just for linear. By applying uncertain reliability analysis in the uncertainty model, we for the first time include the uncertainty of reliability into the CPS partitioning, where the reliability enhancement algorithm is proposed. We study the performance of the reliability obtained through uncertain reliability analysis, and experimental results show that the system reliability with uncertainty does not change significantly with the growth of task module numbers.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems;**

Additional Key Words and Phrases: Cyber-physical system (CPS) partitioning, uncertainty theory, uncertain reliability analysis

ACM Reference format:
Si Chen, Guoqi Xie, Renfa Li, and Keqin Li. 2021. Uncertainty Theory Based Partitioning for Cyber-Physical Systems with Uncertain Reliability Analysis. *ACM Trans. Des. Autom. Electron. Syst.* 27, 3, Article 23 (November 2021), 19 pages.
https://doi.org/10.1145/3490177

This work was supported in part by the National Natural Science Foundation of China under Grants 61972139, 61932010, and 62133014, and the Fundamental Research Funds for the Central Universities, Hunan University, China.

Authors' addresses: S. Chen and R. Li, Key Laboratory for Embedded and Cyber-Physical systems of Hunan Province, College of Computer Science and Electronic Engineering, Hunan University, Lushan Road (S), Changsha 410082, Hunan, China; emails: {chensi_2018, lirenfa}@hnu.edu.cn; G. Xie (corresponding author), Key Laboratory for Embedded and Cyber-Physical systems of Hunan Province, College of Computer Science and Electronic Engineering, Hunan University, Lushan Road (S), Changsha 410082, Hunan and Research and Innovation Center for Convergence of Automobile and Cyberspace, Research Institute of Hunan University in Chongqing, No. 188 Huanan Road, Chongqing 401120, Chongqing, China; email: xgqman@hnu.edu.cn; K. Li, Department of Computer Science, State University of New York, 1 Hawk Drive, New Paltz, New York 12561, USA; email: lik@newpaltz.edu.

1084-4309/2021/11-ART23 $15.00
https://doi.org/10.1145/3490177

1 INTRODUCTION

1.1 Background

CPSs have two core technical elements (software and hardware) and are applied to our daily life, such as Smart Home and Intelligent Transport Systems; these systems are becoming increasingly complicated with the development of Industry 4.0 [14]. Co-design of hardware and software is crucial for many fields [9, 24, 29] as well as for CPS. In general, hardware spends more resources than software, but runs faster. It is a critical issue to reasonably use software modules and hardware modules (i.e., reasonable partitioning) to meet the specific CPS design requirement. The CPS partitioning problem is a key step in hardware software co-design, where critical tasks are implemented in hardware while non-critical tasks are implemented in software; various CPS partitioning methods have proposed many algorithms (exact algorithms [17, 20, 27], heuristic algorithms [7, 8, 10], etc.), and have made good tradeoffs among many performance metrics (time [30], power [19], cost [28], etc.).

Traditional CPS partitioning methods run in a determined context, where the parameters that affect performances are deterministic; these methods depend on the parameter pre-estimations at design time. However, these estimations cannot be completely accurate during the design stage; that is, parameters are uncertain. Unfortunately, traditional CPS partitioning methods ignore the uncertainty of parameters, resulting in the partitioning result not actually meeting the specific design requirement. Furthermore, reliability is a very important performance metric for safety-critical cyber-physical systems, but traditional CPS partitioning methods hardly consider reliability.

1.2 Motivation

In order to solve the uncertainty in CPS partitioning, uncertainty theory is introduced [12, 13, 26]. Uncertainty theory is a branch of mathematics for modeling uncertainty of imprecise quantities [16], which is a common phenomenon in reality. In cyber-physical system design, these parameters which cannot be accurately pre-estimated are regarded as imprecise quantities.

The state-of-the-art work [12] proposed an uncertainty theory based CPS partitioning method, which includes parameter uncertainty and reliability analysis, but it has the following insufficiencies.

(1) The method considers linear uncertainty distributions for variables; the method of solving uncertainty described by the model is applicable to linear uncertain variables. However, other uncertain variables, such as linear, normal, zigzag, and lognormal, are needed by CPS to depict different situations, and it is not enough to elaborate on the linear case.
(2) The method considers the uncertainty of time and cost via uncertainty theory, while ignoring the uncertainty of reliability. However, reliability is uncertain and its uncertainty needs to be dealt with.

1.3 Contributions

To solve the above insufficiencies, we propose an uncertainty theory based CPS partitioning method with uncertain reliability analysis [4]. We use uncertain variables to formulate cost, time, and reliability; we convert them to the forms of uncertainty distributions that are certain. Our main contributions are described below.

(1) We convert the uncertain objective and constraint into determined forms; such conversion methods can be applied to various forms of uncertain variables, not just for linear. Furthermore, the methods can be generalized to other combinatorial optimization problems with uncertainty.

(2) By applying uncertain reliability analysis in the uncertainty model, we for the first time include the uncertainty of reliability into the CPS partitioning, where the reliability enhancement algorithm based on the custom genetic algorithm is proposed, aiming at enhancing the system reliability while satisfying time and cost constraints.
(3) We study the performance of the reliability obtained through uncertain reliability analysis, which includes the uncertainty of reliability and is different from the reliability studied in other literature. Experimental results show that the system reliability with uncertainty does not change significantly with the growth of task module numbers.

The rest of this paper is structured as follows. Section 2 shows related work. Section 3 briefly introduces theoretical foundations for this paper, including uncertainty theory and uncertain reliability analysis. Section 4 presents our uncertainty model. Section 5 gives experimental results, and Section 6 concludes the paper.

2 RELATED WORK

CPSs depend on the integration of physical and embedded systems with communication and IT systems [22]; these systems are closely related to numerous domains such as Smart Cities [23] and Internet of Things [5]. Many pieces of research have paid attention to the performances of CPS. Jiang et al. [11] bridged the model-driven development and verification to improve the dependability of CPS. Song et al. [21] focused on the security and privacy of CPS and discussed security and privacy in many application scenarios. Rashid et al.[18] looked to the survivability of CPS and introduced a method to determine the survivability of the simulation models at run time. In this paper, we are mainly concerned with the uncertainty of CPS. CPSs have two core technical elements: hardware and software; co-design of hardware and software is crucial for CPS. The CPS partitioning problem aims at conducting a reasonable partitioning for hardware implementation and software implementation of CPS, which is a key step in CPS co-design as well as a typical problem involving uncertainty.

It has been known for a long time that the CPS partitioning problem is NP-hard [3]; traditional CPS partitioning methods have made many efforts to solve it. Ouyang et al. [17] proposed an integer linear programming based optimal method to solve the CPS partitioning problem for small inputs. Shi et al.[20] presented a dynamic programming algorithm to solve the multiple-choice CPS partitioning of the tree-shaped task graph, aiming at obtaining the exact solutions for small scale instances. Wu et al.[27] put forward a hybrid branch-and-bound approach to enhance the problem-solving capacity of the CPS partitioning problem. These exact algorithms work well with small inputs but tend to be slow for large systems. To be more efficient, heuristics algorithms, such as genetic algorithm [8], greedy algorithm [7], and tabu search [10], are widely used.

All these traditional methods above do not consider the uncertainty of parameters, but there exists uncertainty when performing partitioning tasks. Albuquerque et al.[1] considered uncertainty in system-level partitioning, but they focus not on the uncertainty of system parameters, but rather on the implementation uncertainty of a developer. Wang et al.[26] paid attention to the uncertainty of system parameters and proposed an uncertain CPS partitioning method. However, they only consider the uncertainty of some performance metrics, such as the time, cost, and resources of the system, while not concerning the reliability performance that is particularly significant in a safety-critical system [25].

In 2019, Jiang et al.[12] for the first time introduced reliability in the uncertain CPS partitioning method. In that work, the system reliability is computed under the condition that each module's reliability is deterministic. That is, it only considers the uncertainty of time and cost but ignores the uncertainty of reliability. In our work, we try to consider reliability in an uncertain way, by applying uncertain reliability analysis in the CPS partitioning uncertainty model.

3 THEORETICAL FOUNDATIONS

Considering that we deal with uncertainty based on uncertainty theory and uncertain reliability analysis, this section presents some basic definitions of uncertainty theory and briefly introduces uncertain reliability analysis.

3.1 Uncertainty Theory

Uncertainty theory was founded by Liu in 2007, which is motivated to model imprecise quantities that behave neither like randomness nor like fuzziness [16]. There are three fundamental concepts in uncertainty theory [16]. The first one is the uncertain measure which is used to measure the belief degree of an uncertain event. The second one is the uncertain variable which is used to represent imprecise quantities. The third one is the uncertainty distribution which is used to describe uncertain variables in an incomplete but easy-to-use way. Following notions are used when defining the above three concepts: let Γ be a nonempty set; a collection $\mathcal{L}$ of subsets of Γ is called a σ-algebra; each element Λ in the σ-algebra $\mathcal{L}$ is called an event; each event Λ is assigned with a number $\mathcal{M}\{\Lambda\}$ which indicates the belief degree that the event Λ will occur.

Definition 1 (Uncertain measure [16]). Uncertain measure is a function from $\mathcal{L}$ to [0,1]. The set function $\mathcal{M}$ is called an uncertain measure if it satisfies normality, monotonicity, self-duality, and countable subadditivity axioms.

- $\mathcal{M}\{\Lambda\} = 1$ for the universal set Γ (*normality*)
- $\mathcal{M}\{\Lambda_1\} \leq \mathcal{M}\{\Lambda_2\}$ whenever $\Lambda_1 \subset \Lambda_2$ (*monotonicity*)
- $\mathcal{M}\{\Lambda\} + \mathcal{M}\{\Lambda^c\} = 1$ for any event Λ (*self-duality*)
- $\mathcal{M}\{\cup_{i=1}^{\infty}\Lambda_i\} \leq \sum_{i=1}^{\infty}\mathcal{M}\{\Lambda_i\}$ for every countable sequence of events $\{\Lambda_i\}$ (*countable subadditivity*)

Definition 2 (Uncertain variable [16]). If $\mathcal{M}$ is an uncertain measure, the triplet $(\Gamma, \mathcal{L}, \mathcal{M})$ is called an uncertainty space. An uncertain variable is a measurable function ξ from an uncertainty space $(\Gamma, \mathcal{L}, \mathcal{M})$ to the set of real numbers.

Definition 3 (Uncertainty distribution [16]). The uncertainty distribution $\Phi : \mathcal{R} \to [0, 1]$ of an uncertain variable ξ is defined by $\Phi(x) = \mathcal{M}\{\xi \leq x\}$ for any real number x.

3.2 Uncertain Reliability Analysis

Uncertain reliability analysis was proposed by Liu in 2010 as a tool to deal with system reliability via uncertainty theory [16]; it is a way to model the system reliability considering the uncertain factors of the system. According to uncertainty theory, uncertain reliability analysis and uncertain risk analysis have the same root in mathematics, and they are distinguished only for the convenience of application, so reliability has similar properties with risk.

"Risk" has different definitions in literature, and the risk is defined as the "accidental loss" plus "uncertain measure of such loss" in uncertainty theory [16]. A system usually contains uncertain factors, such as lifetime, cost, and resource. The risk index is defined as the uncertain measure that some specified loss occurs, and the loss is problem-dependent [4]. Similarly, the reliability index is defined as the uncertain measure that some system is working.

Definition 4 (Reliability index [16]). Assume that a system contains uncertain variables $\xi_1, \xi_2, \ldots, \xi_n$, and there is a function R such that the system is working if and only if

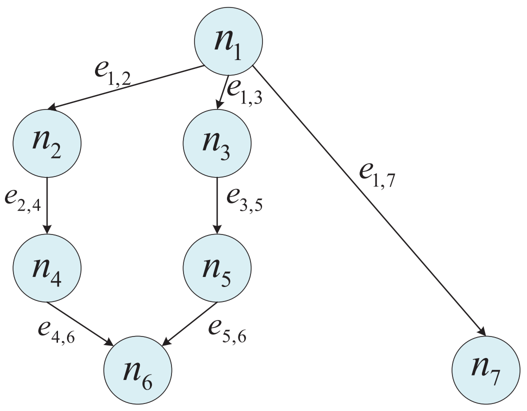

Fig. 1. An example task graph with 7 nodes.

$R(\xi_1, \xi_2, \ldots, \xi_n) \geq 0$. Then the reliability index is

$$Reliability = \mathcal{M}\left\{R(\xi_1, \xi_2, \ldots, \xi_n) \geq 0\right\}.$$

4 UNCERTAINTY MODEL

In this section, we elaborate on the uncertainty model of the CPS partitioning problem. Firstly, we give the definition of the problem. Secondly, we formulate the problem to establish the uncertainty model. Thirdly, we convert the uncertainty model to a determined one. Fourthly, we propose the reliability enhancement algorithm based on the custom genetic algorithm to solve the model. At last, we present an example of applying the uncertainty model and the algorithm.

4.1 Problem Definition

A commonly used model called task graph [15] is utilized to describe the system, as illustrated in Figure 1. The model is a directed acyclic graph where nodes represent tasks and directed edges represent dependence and communication relationships among these tasks [6]. We denote the task graph as $G = \{N, E\}$, where $N = \{n_1, n_2, \ldots, n_i\}$ is the set of nodes (i.e., the set of tasks) and $E = \{e_{1,2}, e_{2,3}, \ldots, e_{i,j}\}$ is the set of directed edges where each edge represents priority and data dependency between two tasks. We will introduce some notations defined on G in Table 1 and briefly explain what they mean.

When a task node is implemented in hardware, it is associated with a hardware implementation cost. Independent uncertain variables c_i^h is used to denote the hardware cost of node n_i. Φ_{ci}^h is uncertainty distribution of c_i^h.

Software implementation of node n_i is associated with a software cost which is usually the running time of the node. Uncertain variable t_i^s is used to denote the running time of node n_i in software implementation, and uncertain variable t_i^h is used to denote the running time of node n_i in hardware implementation. Φ_{ti}^s and Φ_{ti}^h are uncertainty distributions of t_i^s and t_i^h, respectively. If two communicating nodes n_i and n_j are implemented in different contexts, the communication time between them is considered, and $c_{i,j}$ is used to denote it.

According to the description in Section 3.2, a system can contain various uncertain factors; in this paper, we regard the lifetime as the uncertain factor of the system to perform uncertain reliability analysis. Uncertain variable l_i^s is used to denote the lifetime of node n_i in software

Table 1. Notations and Definitions

Notation	Definition
c_i^h	Hardware cost of implementing n_i in hardware
Φ_{ci}^h	Uncertainty distribution of c_i^h
t_i^s	Running time of implementing n_i in software
Φ_{ti}^s	Uncertainty distribution of t_i^s
t_i^h	Running time of implementing n_i in hardware
Φ_{ti}^h	Uncertainty distribution of t_i^h
$c_{i,j}$	Communication time between n_i and n_j
l_i^s	Lifetime of n_i in software implementation
Φ_{li}^s	Uncertainty distribution of l_i^s
l_i^h	Lifetime of n_i in hardware implementation
Φ_{li}^h	Uncertainty distribution of l_i^h

implementation, and uncertain variable l_i^h is used to denote the lifetime of node n_i in hardware implementation. Φ_{li}^s and Φ_{li}^h are uncertainty distributions of l_i^s and l_i^h, respectively.

The CPS partitioning problem defined here is to find a bipartition P of N: $P = (N_h, N_s)$, where $N_h \cup N_s = N$ and $N_h \cap N_s = \emptyset$. An auxiliary decision vector $\mathbf{x}$ $(x_1, x_2, \ldots, x_n)$ which represents the implementation way of n task nodes is used to formalize the problem. When the value of x_i is equal to 0, the task node n_i will be implemented in hardware, otherwise in software. Then, the objective of the problem is to find the optimal solution of the auxiliary decision vector. The solution needs to be found in some conditions: hardware cost, total time, and system reliability are subject to certain capacity limits C_L, T_L, and R_L, respectively. Combining with the demonstration in [2], the goal becomes finding a CPS bipartition P, satisfying $T(\mathbf{x}) \leq T_L$, $R(\mathbf{x}) \geq R_L$, and $C(\mathbf{x})$ is the minimal hardware cost.

4.2 Problem Formulation

This subsection formulates hardware cost, total time, and system reliability using uncertain variables, and finally models the partitioning problem.

4.2.1 Hardware Cost. The total cost consists of hardware cost and software cost, which represents resource consumption in hardware and software implementation of each task node. When a task node is implemented in software, it is associated with a software cost, which is usually the running time. Otherwise, it is associated with a hardware cost, which is the resource consumption cost. For the system, the total hardware cost can be calculated as the sum of hardware cost of each node implemented in hardware, because the additive calculation rule for resource consumption cost is reasonable in most cases of computation model. Finally, hardware cost $C(\mathbf{x})$ can be formalized as follows:

$$C(\mathbf{x}) = \sum_{i=1}^{n} \left[c_i^h \left(1 - x_i\right) \right].$$

4.2.2 Total Time. Here we consider the worst-case scenario where there is no overlap in the running time of each node. Then the total time is the sum of the running time of each node and the communication time among nodes. If two nodes n_i and n_j are implemented in different ways, the communication time between them is considered. Otherwise, the communication time is 0. Finally, total running time $T_{\text{run}}(\mathbf{x})$, total communication time $T_{\text{com}}(\mathbf{x})$ and total time $T(\mathbf{x})$ can be

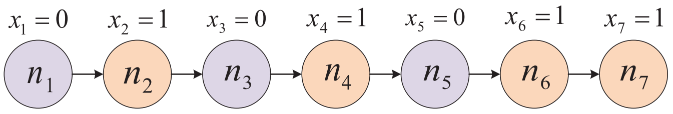

Fig. 2. System reliability calculation diagram with treating the system described by the task graph as a series system.

formalized as follows:

$$T_{\text{run}}(\mathbf{x}) = \sum_{i=1}^{n} \left[t_i^h \left(1 - x_i\right) + t_i^s x_i \right],$$

$$T_{\text{com}}(\mathbf{x}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{i,j} \left[\left(x_i - x_j \right)^2 \right],$$

$$T(\mathbf{x}) = \sum_{i=1}^{n} \left[t_i^h \left(1 - x_i\right) + t_i^s x_i \right] + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{i,j} \left[\left(x_i - x_j \right)^2 \right].$$

4.2.3 System Reliability. We deal with the system reliability via uncertainty theory; that is, we use uncertain reliability analysis mentioned in Section 3.2 to formulate the system reliability. The system reliability here is defined as the uncertain measure that the system is working in consideration of uncertain factors. We treat the system as a series system and take the lifetime of each task node as the current uncertain factor to compute the system reliability. According to uncertainty theory [16] and uncertain reliability analysis [4], for a series system in which there are n elements whose lifetimes are independent uncertain variables $l_1, l_2, \ldots, l_n$ with uncertain distributions $\Phi_{l1}, \Phi_{l2}, \ldots, \Phi_{ln}$ respectively, the lifetime of the system is an uncertain variable l with uncertainty distribution $\Psi_l(x) = \Phi_{l1}(x) \vee \Phi_{l2}(x) \vee \cdots \vee \Phi_{ln}(x)$. If we hope the system is working until time T, then the reliability index is

$$Reliability = \mathcal{M}\{l \geq T\} = 1 - \Phi_{l1}(T) \vee \Phi_{l2}(T) \vee \cdots \vee \Phi_{ln}(T),$$

where $\vee$ is maximum operator, and $\Phi_{li}(T)$ is uncertainty distribution of the lifetime of node n_i at time T.

If n_i is implemented in hardware (i.e., $x_i = 0$), $\Phi_{li}(T)$ is equal to $\Phi_{li}^h(T)$, else it is equal to $\Phi_{li}^s(T)$. Then, system reliability $R(\mathbf{x})$ is represented as follows:

$$R(\mathbf{x}) = 1 - \left[\Phi_{l1}^h(T) \left(1 - x_1\right) + \Phi_{l1}^s(T) x_1 \right] \vee \cdots \vee \left[\Phi_{ln}^h(T) \left(1 - x_n\right) + \Phi_{ln}^s(T) x_n \right].$$

For simplicity, we denote the result of the second term as $\max_{1 \leq i \leq n}\{[\Phi_{li}^h(T)(1-x_i) + \Phi_{li}^s(T)x_i]\}$, which represents the maximum of $[\Phi_{li}^h(T)(1-x_i) + \Phi_{li}^s(T)x_i]$ for $i = 1, 2, \ldots, n$. Let us take the task graph in Figure 1 as an example to illustrate the reliability of this system. For the implementation way of 7 task nodes in this graph, we assume that task nodes n_1, n_3, n_5 are implemented in hardware (i.e., x_i is equal to 0) and task nodes n_2, n_4, n_6, n_7 are implemented in software (i.e., x_i is equal to 1). According to the above description, the system described by the task graph is treated as a series system when calculating the system reliability, as shown in Figure 2. The different colors of the task nodes represent different implementations, where purple represents hardware implementation and orange represents software implementation. Then, the system reliability is $1 - \max\{\Phi_{l1}^h(T), \Phi_{l2}^s(T), \Phi_{l3}^h(T), \Phi_{l4}^s(T), \Phi_{l5}^h(T), \Phi_{l6}^s(T), \Phi_{l7}^s(T)\}$.

Finally, system reliability $R(\mathbf{x})$ can be formulated as follows:

$$R(\mathbf{x}) = 1 - \max_{1 \leq i \leq n} \left\{ \left[\Phi_{li}^h(T) \left(1 - x_i\right) + \Phi_{li}^s(T) x_i \right] \right\}. \tag{1}$$

4.2.4 Partitioning Problem. Based on the definition in Section 4.1, the partitioning problem P is modeled as follows:

$$\begin{aligned} \text{minimize} \quad & C(\mathbf{x}), \\ \text{subject to} \quad & T(\mathbf{x}) \le T_L, \\ & R(\mathbf{x}) \ge R_L, \\ & \mathbf{x} \in \{0,1\}^n, \end{aligned}$$

where $C(\mathbf{x})$, $T(\mathbf{x})$, and $R(\mathbf{x})$ follow the above formulations, T_L and R_L are capacity limits of total time and system reliability, respectively.

4.3 Problem Conversion

Considering that c_i^h, t_i^h, t_i^s, etc. are all unknown uncertain variables that may get various values depending on the context difference of systems, it is difficult to directly solve the uncertainty model. Uncertainty distribution is used to describe uncertain variables, and it is sufficient to know the uncertainty distribution rather than the uncertain variable itself in many cases. As many uncertain variables own known uncertainty distributions, it is feasible to transform the uncertainty model into a determined one to simplify the problem, by converting uncertain variables to the forms of uncertainty distributions. This subsection presents the conversion process.

Firstly, we introduce some important theorems that will be used in the conversion.

Theorem 1. *For a given set of decision variables $\boldsymbol{x}$, assume that there is a function $f(\boldsymbol{x}, \boldsymbol{\xi}) = h_1(\boldsymbol{x})\xi_1 + h_2(\boldsymbol{x})\xi_2 + \cdots + h_n(\boldsymbol{x})\xi_n + h_0(\boldsymbol{x})$, where $h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_n(\boldsymbol{x}), h_0(\boldsymbol{x})$ are real-valued functions, $\xi_1, \xi_2, \ldots, \xi_n$ are independent uncertain variables. Then we have*

$$E[f(\boldsymbol{x}, \boldsymbol{\xi})] = h_1(\boldsymbol{x})E[\xi_1] + h_2(\boldsymbol{x})E[\xi_2] + \cdots + h_n(\boldsymbol{x})E[\xi_n] + h_0(\boldsymbol{x}).$$

Proof. According to uncertainty theory [16], the linearity of expected value operator has been proven. That is, for independent uncertain variables ξ and η, $E[a\xi + b\eta] = aE[\xi] + bE[\eta]$ for any real numbers a and b. Theorem 1 follows from it immediately. □

Theorem 2. *Assume that $\xi_1, \xi_2, \ldots, \xi_n, \eta_1, \eta_2, \ldots, \eta_n$ are independent uncertain variables with uncertainty distributions $\Phi_1^{\xi}, \Phi_2^{\xi}, \ldots, \Phi_n^{\xi}, \Phi_{n+1}^{\eta}, \Phi_{n+2}^{\eta}, \ldots, \Phi_{2n}^{\eta}$ and $h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_{2n}(\boldsymbol{x}), h_0(\boldsymbol{x})$ are all nonnegative real-valued functions. Then the inequation*

$$\mathcal{M}\left\{\left(\sum_{i=1}^{n} h_i(\boldsymbol{x})\xi_i + \sum_{i=n+1}^{2n} h_i(\boldsymbol{x})\eta_i\right) \le h_0(\boldsymbol{x})\right\} \ge \alpha$$

holds if and only if

$$\sum_{i=1}^{n} h_i(\boldsymbol{x})\Phi_i^{\xi^{-1}}(\alpha) + \sum_{i=n+1}^{2n} h_i(\boldsymbol{x})\Phi_i^{\eta^{-1}}(\alpha) \le h_0(\boldsymbol{x}).$$

Proof. It has been proven in uncertainty theory [16] that if $\xi_1, \xi_2, \ldots, \xi_n$ are independent uncertain variables with uncertainty distributions $\Phi_1, \Phi_2, \ldots, \Phi_n$ and $h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_n(\mathbf{x}), h_0(\mathbf{x})$ are all nonnegative real-valued functions, then the inequation

$$\mathcal{M}\left\{\sum_{i=1}^{n} h_i(\boldsymbol{x})\xi_i \le h_0(\boldsymbol{x})\right\} \ge \alpha$$

holds if and only if

$$\sum_{i=1}^{n} h_i(\boldsymbol{x})\Phi_i^{-1}(\alpha) \le h_0(\boldsymbol{x}).$$

Let n be $2m$. Considering that all uncertain variables $\xi_1, \xi_2, \ldots, \xi_{2m}$ are independent, we denote $\xi_{m+1}, \xi_{m+2}, \ldots, \xi_{2m}$ as $\eta_1, \eta_2, \ldots, \eta_m$. Theorem 2 is then proven.

Secondly, we convert the uncertain objective and uncertain constraint to deterministic formalizations. □

Objective conversion-According to uncertainty theory [16], uncertain variable $\xi > \eta$ if and only if $E(\xi) > E(\eta)$, so the objective is converted to minimize its expected value. Similarly, for a maximized case, the objective is converted to maximize its expected value. Using Theorem 1, we formulate the expected value of $C(\mathbf{x})$ as follows:

$$E[C(\mathbf{x})] = \sum_{i=1}^{n} \left[E\left[c_i^h\right](1 - x_i) \right].$$

Finally, the uncertain cost objective function of problem P is converted to

$$\text{minimize} \quad \sum_{i=1}^{n} \left[\left(\int_0^1 \Phi_{ci}^{h^{-1}}(\alpha) d\alpha \right) (1 - x_i) \right],$$

where $\Phi_{ci}^{h^{-1}}(\alpha)$ is the inverse uncertainty distribution of c_i^h.

Constraint conversion-According to uncertain theory, considering that the uncertain constraints $g_j(\mathbf{x}, \xi) \leq 0$ do not define a deterministic feasible set, it is naturally desired that the uncertain constraints hold with a confidence level α:

$$\mathcal{M}\{g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0\} \geq \alpha.$$

Based on the basic definitions of uncertainty theory in Section 3.1, $\mathcal{M}\{g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0\}$ represents the belief degree that the event $g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0$ will occur. Therefore, this constraint means that the belief degree of $g_j(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0$ is not less than α. The value of α represents the strictness of the converted constaint. The higher the confidence level α is, the stricter the converted constraint is, which gives us less space for optimization. Then, the uncertain time constraint holds with a confidence level as follows:

$$\mathcal{M}\left\{ \sum_{i=1}^{n}(1 - x_i)t_i^h + \sum_{i=1}^{n} x_i t_i^s \leq T_L - \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} c_{ij} \left[\left(x_i - x_j\right)^2 \right] \right\} \geq \alpha.$$

Finally, using Theorem 2, the uncertain time constraint of problem P is converted to

$$\sum_{i=1}^{n} \left[\Phi_{ti}^{s^{-1}}(\alpha) - \Phi_{ti}^{h^{-1}}(\alpha) \right] x_i + \sum_{i=1}^{n} \Phi_{ti}^{h^{-1}}(\alpha) \leq T_L - \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} c_{i,j} \left[\left(x_i - x_j\right)^2 \right], \tag{2}$$

where $\Phi_{ti}^{h^{-1}}(\alpha)$ is the inverse uncertainty distribution of t_i^h, $\Phi_{ti}^{s^{-1}}(\alpha)$ is the inverse uncertainty distribution of t_i^s.

Considering that the reliability constraint has been in the form of uncertainty distribution, there is no need to convert it.

Thirdly as well as ultimately, the converted version of problem P is as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} \left[\int_0^1 \Phi_{ci}^{h^{-1}}(\alpha) d\alpha \right] (1 - x_i), \\ \text{subject to} & \sum_{i=1}^{n} \left[\Phi_{ti}^{s^{-1}}(\alpha) - \Phi_{ti}^{h^{-1}}(\alpha) \right] x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{i,j}[(x_i - x_j)^2] \leq T_L - \sum_{i=1}^{n} \Phi_{ti}^{h^{-1}}(\alpha), \\ & 1 - \max_{1 \leq i \leq n} \left\{ \left[\Phi_{li}^h(T)(1 - x_i) + \Phi_{li}^s(T) x_i \right] \right\} \geq R_L, \\ & \mathbf{x} \in \{0, 1\}^n. \end{array}$$

For safety-critical systems that care about the reliability, the objective can be set as maximizing the system reliability with uncertain cost and time constraints. In this case, the hardware cost acts as the constraint rather than the objective, and it needs to follow the rules of constraint conversion.

Using the constraint conversion method described above, the uncertain cost constraint of problem P is converted to

$$\sum_{i=1}^{n} \left[\Phi_{ci}^{h^{-1}}(\alpha)\right](1-x_i) \leq C_L, \tag{3}$$

and then we get the converted version of problem P as follows:

$$\begin{array}{ll} \text{maximize} & 1 - \max\limits_{1\leq i\leq n}\left\{\left[\Phi_{li}^{h}(T)\,(1-x_i) + \Phi_{li}^{s}(T)x_i\right]\right\}, \\ \text{subject to} & \sum_{i=1}^{n}\left[\Phi_{ti}^{s^{-1}}(\alpha) - \Phi_{ti}^{h^{-1}}(\alpha)\right]x_i + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} c_{i,j}[(x_i - x_j)^2] \leq T_L - \sum_{i=1}^{n}\Phi_{ti}^{h^{-1}}(\alpha), \\ & \sum_{i=1}^{n}\left[\Phi_{ci}^{h^{-1}}(\alpha)\right](1-x_i) \leq C_L, \\ & \mathbf{x} \in \{0,1\}^n. \end{array} \tag{4}$$

4.4 Problem Solution

In this paper, we focus on the model that maximizes the system reliability (i.e. (4)). This model aims to maximize reliability (i.e. (1)) while meeting the cost and time constraints, and finally an auxiliary decision vector $\mathbf{x}(x_1, x_2, \ldots, x_n)$ which represents the hardware and software implementation way of each node n_i will be obtained. We propose the reliability enhancement algorithm based on the custom genetic algorithm according to the characteristics of the model. The algorithm consists of two main steps.

(1) Obtain an initial solution with the maximum system reliability.
(2) Modify the initial solution under the condition of meeting total time and hardware cost constraints to get the final solution.

Firstly, we find an initial solution. By observing the objective of the model, we find that the goal of the final solution is to minimize the maximum value of the set $\{[\Phi_{li}^{h}(T)(1-x_i) + \Phi_{li}^{s}(T)x_i]\}$ for $i = 1, 2, \ldots, n$. As described in Section 4.2, for each x_i, $\Phi_{li}(T)$ is either $\Phi_{li}^{s}(T)$ or $\Phi_{li}^{h}(T)$. That is, for each element in the set, its value is $\Phi_{li}^{h}(T)$ when x_i is equal to 0, and $\Phi_{li}^{s}(T)$ when x_i is equal to 1. Therefore, if each x_i in the auxiliary decision vector $\mathbf{x}(x_1, x_2, \ldots, x_n)$ can make its corresponding $\Phi_{li}(T)$ take the smaller one of $\Phi_{li}^{h}(T)$ and $\Phi_{li}^{s}(T)$, it can be sure that the maximum value of the set calculated by $\mathbf{x}$ is the smallest. Meanwhile, there is no need to determine the value of each x_i in $\mathbf{x}(x_1, x_2, \ldots, x_n)$ and we can get the smallest value. We make each x_i be determined in descending order of the value of its corresponding $\Phi_{li}(T)$. If the $\Phi_{li}(T)$ value of the currently determined x_i (i.e., the smaller one of $\Phi_{li}^{h}(T)$ and $\Phi_{li}^{s}(T)$) is the largest among rest all $\Phi_{li}(T)$ (for the determined x_i, it is the smaller one of corresponding $\Phi_{li}^{h}(T)$ and $\Phi_{li}^{s}(T)$; for the undetermined x_i, it is corresponding $\Phi_{li}^{h}(T)$ and $\Phi_{li}^{s}(T)$), the remaining undetermined x_i can be arbitrarily set to obtain an initial solution. In this way, only several x_i in the initial solution $\mathbf{x}(x_1, x_2, \ldots, x_n)$ are determined, and the current maximum value of the set is still the smallest.

Secondly, we modify the initial solution. According to the determined x_i in the initial solution $\mathbf{x}$, we calculate corresponding hardware cost and time, and judge whether any of them exceeds its capacity limit (C_L or T_L), and finally modify the initial solution. Because the original objective of maximizing system reliability has been achieved in the first step, the remaining problem can be converted according to the actual problem. For example, for hard real-time applications, it can be converted to minimize hardware cost under the time constraint; for cost-constrained applications, it can be converted to minimize time under the hardware cost constraint. Many algorithms can be used for the remaining problem and we choose to use the genetic algorithm. Details of the method are shown in Algorithm 1, and we explain the algorithm by combining it with an example.

ALGORITHM 1: The Reliability Enhancement Algorithm

```
Input: belief degree $\alpha$, time limit $T_L$, cost limit $C_L$, working time $T$, task nodes number $n$, task
    graph $G$ (including all relevant parameters)
Output: solution $\mathbf{x}(x_1, x_2, \ldots, x_n)$
 1: Compute all values of $\Phi^h_{li}(T)$ and $\Phi^s_{li}(T)$ respectively for $i = 1, 2, \ldots, n$, and put them together
    into Max;
 2: Initialize an empty list MaxIndex = [ ];
 3: Initialize a random solution $X = (x_1, x_2, \ldots, x_n)$
 4: for ($i \leftarrow 1; i \leq n; i$++) do
 5:     max ← the maximum value of Max
 6:     $k \leftarrow$ the subscript of $x_k$ corresponding to max ;
 7:     if ($k \notin$ MaxIndex) then
 8:         Assign 0 or 1 to $x_k$ according to max;
 9:         Put $k$ in MaxIndex;
10:         Update Max with removing max;
11:     else
12:         break;
13:     end if
14: end for
15: Compute CurrentCost for ($x_j, j \in$ MaxIndex) according to Inequation (3);
16: Compute CurrentTime for ($x_j, j \in$ MaxIndex) according to Inequation (2);
17: while (CurrentCost > $C_L$ or CurrentTime > $T_L$) do
18:     Remove the last element of MaxIndex;
19:     $m \leftarrow$ the last element of MaxIndex;
20:     Update $x_m$ with fliping its value;
21:     Compute CurrentCost for ($x_j, j \in$ MaxIndex) according to Inequation (3);
22:     Compute CurrentTime for ($x_j, j \in$ MaxIndex) according to Inequation (2);
23: end while
24: Record ($x_j, j \in$ MaxIndex) to the solution $X'$;
25: $C_N \leftarrow C_L -$ CurrentCost;
26: $T_N \leftarrow T_L -$ CurrentTime;
27: Apply the custom genetic algorithm to ($x_j, j \notin$ MaxIndex) for getting the solution $X''$;
28: Combine $X'$ and $X''$ to generate the solution $\mathbf{x}$;
29: Compute Cost for ($x_i \in \mathbf{x}$) according to Inequation (3);
30: Compute Time for ($x_i \in \mathbf{x}$) according to Inequation (2);
31: while (Cost > $C_L$ or Time > $T_L$) do
32:     Repeat the steps on lines 18–22 to modify the initial solution again;
33:     Repeat the steps on lines 24–30 to get the new solution;
34: end while
35: return $\mathbf{x}$;
```

Lines 1–3 finish the initialization process. A random solution X is initialized to record the determined x_i (i.e., the assigned x_i). An empty list *MaxIndex* is initialized to record the subscript of the determined x_i. All $\Phi^h_{li}(T)$ and $\Phi^s_{li}(T)$ for $i = 1, 2, \ldots, n$ are computed and put into *Max*.

Lines 4–14 obtain the initial solution with the maximum system reliability. Line 5 finds the maximum value of *Max*. Suppose that $\Phi^s_{l4}(T)$ is the current maximum. Line 6 finds the subscript of x_4 (i.e., $k = 4$). Lines 8–10 perform corresponding operations according to the subscript k obtained in line 6. x_4 is assigned value 0 to make sure that $\Phi^s_{l4}(T)$ will not be taken (i.e., let $\Phi_{l4}(T)$ be $\Phi^h_{l4}(T)$ for the reason that $\Phi^h_{l4}(T)$ is smaller than $\Phi^s_{l4}(T)$). Then the subscript 4 is recorded in *MaxIndex*, and $\Phi^s_{l4}(T)$ is removed from *Max*. Above steps are repeated until the current subscript k has been recorded in *MaxIndex*. The initial solution X is obtained in this way, in which only the x_i whose subscript has been in *MaxIndex* is determined.

Lines 15–24 preliminarily modify the initial solution. Lines 15–16 compute corresponding total time *CurrentTime* and hardware cost *CurrentCost* of the determined x_i (i.e., the subscript of x_i is in *MaxIndex*). If any of them exceeds its capacity limit (C_L or T_L), a series of operations are performed on the initial solution to avoid it. The last element of *MaxIndex* which represents the last determined x_i is removed to reduce the number of the determined x_i. Because as the number of the determined x_i decreases, both C_L and T_L decrease. Lines 19–20 flip the value of x_m corresponding to the last element m in the current *MaxIndex* to ensure that the optimal solution that meets constraints can be obtained. Then, *CurrentTime* and *CurrentCost* are recomputed. Above steps are repeated until both *CurrentTime* and *CurrentCost* do not exceed their capacity limits. Line 24 records all the determined x_i in X'.

Lines 25–26 use original limits C_L and T_L to subtract hardware cost and total time calculated from the determined x_i respectively, thereby becoming new limits C_N and T_N which are then used to solve the undetermined x_i (i.e., the unassigned x_i).

Lines 27–28 further modify the current solution with the objective of minimizing hardware cost. The custom genetic algorithm illustrated in Algorithm 2 is applied to find the approximate optimal solution for the rest x_i, where strategies of fitness function, selection, crossover, and mutation are chosen from [2]. Then, the final solution $\mathbf{x}$ is obtained.

Lines 29–35 make sure that the optimal solution obtained this way meets the constraints of cost and time.

ALGORITHM 2: The Custom Genetic Algorithm

```
1:  Initialize the first population $P$;
2:  Compute the fitness of each individual in $P$;
3:  Record the individual with the lowest fitness to the solution $X''$;
4:  while (termination conditions) do
5:      Clear new population $P'$;
6:      while (size of $P' \leq$ size of $P$) do
7:          Perform selection in $P$ to generate two parents $p_1$ and $p_2$;
8:          Perform crossover on $p_1$ and $p_2$ to generate two offsprings $o_1$ and $o_2$;
9:          Perform mutation on $o_1$ and $o_2$ to generate two individuals $o_1'$ and $o_2'$;
10:         Put individuals $o_1'$ and $o_2'$ into new population $P'$;
11:     end while
12:     Find the individual $o_f'$ with the lowest fitness in $P'$ ;
13:     if (fitness($o_f'$) $\leq$ fitness($X''$)) then
14:         $X'' \leftarrow o_f'$;
15:     end if
16:     $P \leftarrow P'$;
17: end while
18: return $X''$
```

4.5 Problem Example

In this subsection, we apply the uncertainty model described in Section 4.3 (i.e. (4)) and the reliability enhancement algorithm described in Section 4.4 to the task graph of Figure 1. We assume that each parameter of each node is a normal uncertain variable, denoted by $\mathcal{N}(\mu,\sigma)$. The normal uncertain variable's definition comes from uncertainty theory and is completely different from the normal variable we are familiar with, and the same is true for the normal uncertainty distribution. According to uncertainty theory [16], the uncertainty distribution of normal uncertain variable $\mathcal{N}(\mu,\sigma)$ is

$$\Phi(x) = \left(1 + \exp\left(\frac{\pi(\mu - x)}{\sqrt{3}\sigma}\right)\right)^{-1},$$

and the inverse uncertainty distribution of normal uncertain variable $\mathcal{N}(\mu,\sigma)$ is

$$\Phi^{-1}(\alpha) = \mu + \frac{\sigma\sqrt{3}}{\pi}\ln\frac{\alpha}{1-\alpha}.$$

We make the hardware cost of each node meet the following assumptions: $c_1^h = \mathcal{N}(5,1)$, $c_2^h = \mathcal{N}(2,0.5)$, $c_3^h = \mathcal{N}(10,3)$, $c_4^h = \mathcal{N}(8,2)$, $c_5^h = \mathcal{N}(5,1)$, $c_6^h = \mathcal{N}(7,2)$, $c_7^h = \mathcal{N}(8,3)$. We make the software running time and hardware running time of each node meet the following assumptions: $t_1^h = \mathcal{N}(2,0.1)$, $t_2^h = \mathcal{N}(5,0.1)$, $t_3^h = \mathcal{N}(4,0.1)$, $t_4^h = \mathcal{N}(6,0.1)$, $t_5^h = \mathcal{N}(8,0.1)$, $t_6^h = \mathcal{N}(3,0.1)$, $t_7^h = \mathcal{N}(1,0.1)$, $t_1^s = \mathcal{N}(20,0.2)$, $t_2^s = \mathcal{N}(50,0.2)$, $t_3^s = \mathcal{N}(30,0.2)$, $t_4^s = \mathcal{N}(50,0.2)$, $t_5^s = \mathcal{N}(20,0.2)$, $t_6^s = \mathcal{N}(30,0.2)$, $t_7^s = \mathcal{N}(20,0.2)$. We make the communication time between every two nodes meet the following assumptions: $c_{1,2} = c_{1,3} = c_{1,7} = c_{2,4} = c_{3,5} = c_{4,6} = c_{5,6} = 20$. We make the software lifetime and hardware lifetime of each node meet the following assumptions: $l_1^h = \mathcal{N}(132,10)$, $l_2^h = \mathcal{N}(140,10)$, $l_3^h = \mathcal{N}(143,10)$, $l_4^h = \mathcal{N}(155,10)$, $l_5^h = \mathcal{N}(150,10)$, $l_6^h = \mathcal{N}(142,10)$, $l_7^h = \mathcal{N}(125,10)$, $l_1^s = \mathcal{N}(156,12)$, $l_2^s = \mathcal{N}(135,12)$, $l_3^s = \mathcal{N}(152,12)$, $l_4^s = \mathcal{N}(140,12)$, $l_5^s = \mathcal{N}(128,12)$, $l_6^s = \mathcal{N}(146,12)$, $l_7^s = \mathcal{N}(154,12)$.

Suppose that we want the system to work until time 145, the total time not to exceed 200, and the hardware cost not to exceed 30. If we hope that the uncertain constraints hold with the confidence level $\alpha = 0.9$, the objective becomes:

$$\text{maximize} \quad 1 - \max\left\{\begin{array}{l} [0.9136\,(1-x_1) + 0.1594x_1], \\ [0.7124\,(1-x_2) + 0.8193x_2], \\ [0.5897\,(1-x_3) + 0.2577x_3], \\ [0.1402\,(1-x_4) + 0.6804x_4], \\ [0.2876\,(1-x_5) + 0.9289x_5], \\ [0.6328\,(1-x_6) + 0.4623x_6], \\ [0.9741\,(1-x_7) + 0.2042x_7] \end{array}\right\},$$

and the constraints become:

$$\begin{aligned}&(20.24 - 2.12)x_1 + (50.24 - 5.12)x_2 + (30.24 - 4.12)x_3 + (50.24 - 6.12)x_4 + \\ &(20.24 - 8.12)x_5 + (30.24 - 3.12)x_6 + (20.24 - 1.12)x_7 + 20[(x_1 - x_2)]^2 + \\ &20[(x_1 - x_3)]^2 + 20[(x_1 - x_7)]^2 + 20[(x_2 - x_4)]^2 + 20[(x_3 - x_5)]^2 + 20[(x_4 - x_6)]^2 + \\ &20[(x_5 - x_6)]^2 + 2.12 + 5.12 + 4.12 + 6.12 + + 8.12 + 3.12 + 1.12 \leq 200,\end{aligned}$$

$$\begin{aligned}&6.21(1 - x_1) + 2.61(1 - x_2) + 13.63(1 - x_3) + 10.42(1 - x_4) + \\ &6.21(1 - x_5) + 9.42(1 - x_6) + 11.63(1 - x_7) \leq 30.\end{aligned}$$

Using the reliability enhancement algorithm, we get the following results about $\mathbf{x}\,(x_1, x_2, x_3, x_4, x_5,\ \ x_6, x_7)$: $\mathbf{x} = \{1, 0, 1, 0, 0, 0, 1\}$. That is, task nodes n_1, n_3, n_7 are implemented in software, and task nodes n_2, n_4, n_5, n_6 are implemented in hardware.

5 EXPERIMENTAL RESULTS

In this paper, we are concerned about the uncertainty in CPS partitioning, especially the uncertainty of reliability, so the paper mainly contains two tasks: (1) describing the conversion method that can be applied to various forms of uncertain variables to deal with uncertainty; (2) applying uncertain reliability analysis to the CPS partitioning problem to consider the uncertainty of reliability. Considering that we have described the conversion method in detail and given an example in Section 4, we focus on reliability in the experiment. Because our system reliability is obtained through uncertain reliability analysis, which includes the uncertainty of reliability and is different from the reliability studied in other literature, we mainly study the performance of our system reliability in this section, not the performance of the model or algorithm itself. Firstly, because our system reliability is determined by multiple parameters, we study the influence of these parameters on it. Secondly, because the reliability studied in other literature is greatly affected by the number of task nodes, we study the relationship between the number of task nodes and our system reliability. Thirdly, because our system reliability is obtained by applying uncertain reliability analysis to the CPS partitioning problem, we study the influence of our system reliability on partitioning.

5.1 Reliability Parameter Test

In this subsection, we investigate the influence of different parameters on system reliability. This research is based on the reliability (i.e., (1)) formulated in Section 4.2, which is defined based on its expected working time T and is determined by $\Phi^h_{li}(T)$ and $\Phi^s_{li}(T)$ (i.e., uncertainty distributions of l^h_i and l^s_i). l^h_i and l^s_i can be set as different types of uncertain variables, such as linear, normal, zigzag, and lognormal. We set lifetimes l^s_i, l^h_i of node n_i as normal uncertain variables and denote them as $\mathcal{N}(\mu^s_{li}, \sigma^s_{li})$, $\mathcal{N}(\mu^h_{li}, \sigma^h_{li})$, respectively. According to the formula for the uncertainty distribution of the normal uncertainty variable $\mathcal{N}(\mu, \sigma)$ given in Section 4.5, μ, σ, and T are the parameters that affect the system reliability.

For a task node n_i implemented in a specific way (software or hardware), its lifetime is an uncertain variable with a known uncertainty distribution (i.e., μ_i and σ_i are determinate), so the value of $\Phi_i(T)$ is mainly determined by T, as is the system reliability. That is, both μ and σ are determinate, and only T is variable, so we study the impact of T on the system reliability under the condition of μ, σ being determinate.

We take the number of task nodes as 30. Parameters μ^s_{li}, μ^h_{li} are generated as uniform random numbers in [2500, 2600], and σ^s_{li}, σ^h_{li} are generated as uniform random numbers in [100, 200], which imitates the parameter generation method in [12]. We set the value of T according to the values of μ and σ; T is set to 2300, 2500, a random number generated from [2500, 2600], 2600, and 2800, respectively. We randomly generate 50 solutions to calculate their system reliability corresponding to different T and show them in Figure 3.

It can be seen that the system reliability decreases as T increases. Because as the working time of the system increases, the risk of the system increases, and the reliability decreases accordingly. Meanwhile, we can see that (1) when the value of T is a random number generated from [2500, 2600], the system reliability varies greatly each time because it changes with the change of T, as shown by the yellow line in Figure 3; (2) when the value of T is a constant such as 2300, 2500, 2600, and 2800, the system reliability is relatively stable because T is fixed.

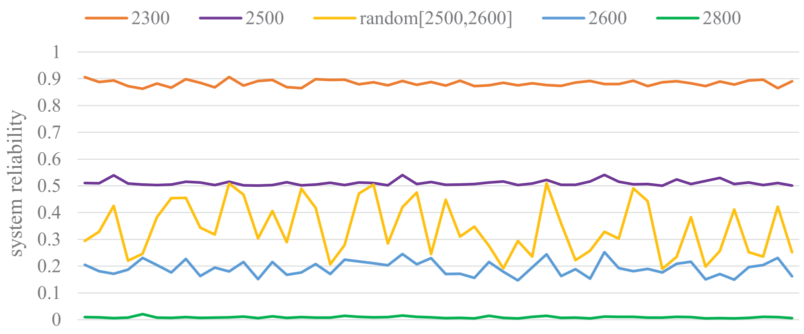

Fig. 3. System reliability corresponding to different T.

Table 2. System Reliability Corresponding to the Different Number of Task Nodes and Different T

n	T = 2300	2500	random [2500, 2600]	2600	2800
10	0.89717	0.52915	0.24534	0.23433	0.01445
20	0.89114	0.51640	0.31316	0.20231	0.00914
30	0.88379	0.50940	0.22693	0.19185	0.00852
40	0.87873	0.50624	0.29380	0.18620	0.00802
50	0.87891	0.50621	0.21514	0.17504	0.00765

It can be noted that the system reliability is slightly greater than 0.5 when T is 2500. This is because μ_{li}^s and μ_{li}^h are randomly generated from [2500, 2600]. The closer T is to μ_i, the closer each $\Phi_i(T)$ is to 0.5, so the closer the system reliability is to 0.5. Obviously, the system reliability is greater than 0.5 when T is less than 2500.

5.2 Reliability Relationship Test

The reliability in other literature is generally calculated as the product of different task nodes' reliability; as the number of task nodes increases, the reliability decreases sharply even if the reliability of each task node is very high [12, 25]. Considering that our system reliability is obtained through uncertain reliability analysis, it is different from the reliability studied in other literature, and we want to know whether the number of task nodes affects our reliability. In this subsection, we investigate the relationship between the number of task nodes and system reliability.

We mainly focus on the influence of n on reliability. The experimental result in Section 5.1 shows that the parameter T affects the reliability, so we change the value of n while T is fixed, and perform experiments on several sets of different T values. Parameters μ_{li}^s, μ_{li}^h, and T follow the settings in Section 5.1. We randomly generate a set of solutions for the cases, where the number of task nodes is 10, 20, 30, 40, and 50, respectively, to calculate their system reliability corresponding to different T. We test each instance 50 times and calculate the average values. The result is listed in Table 2.

Values in the table correspond to the system reliability with different T values when the number of task nodes is 10, 20, 30, 40, and 50, respectively. It can be observed from columns 2, 3, 5, and 6 that under the condition that T has been determinate, the system reliability does not change significantly with the growth of task node numbers, which shows that the number of task nodes hardly affects the reliability obtained through uncertain reliability analysis.

When T takes an uncertain random value within a certain range, the system reliability fluctuates as the number of task nodes changes, which can be seen from the 4th column. This is because the

Table 3. Partitioning Results with Considering Reliability or Not

Number	Limits		With reliability concern			Without reliability concern			Improvement %		
of nodes	T_L	C_L	Time	Cost	Reliability	Time	Cost	Reliability	Time	Cost	Reliability
6	250	160	190	121	0.9632	202	56	0.9364	−6.3	53.7	2.8
	320	150	245	107	0.9420	279	33	0.9304	−13.9	69.2	1.2
	185	269	159	230	0.9198	142	190	0.9033	10.7	17.4	1.8
7	271	398	205	315	0.9583	267	47.6	0.9196	−30.2	84.9	4.0
	274	261	167	171	0.9682	201	29	0.8761	−20.4	83.0	9.5
	357	234	308	139	0.9377	340	74	0.9085	−10.4	46.8	3.1
11	549	378	488	109	0.9100	525	80	0.9054	−7.6	26.6	0.5
	452	570	372	160	0.9486	440	39	0.9296	−18.3	75.6	2.0
	558	318	369	298	0.9234	531	23	0.8662	−43.9	92.3	6.2
14	505	367	474	127	0.9196	497	12	0.9012	−4.9	90.6	2.0
	1174	1056	981	221	0.9139	1164	54	0.8896	−18.7	75.6	2.7
	441	657	361	495	0.8889	427	286	0.8694	−18.3	42.2	2.2
22	965	860	898	386	0.8933	925	244	0.8691	−3.0	36.8	2.7
	542	979	532	735	0.9036	509	660	0.8867	4.3	10.2	1.9
	985	800	843	207	0.9121	973	56	0.8969	−15.4	72.9	1.7
25	951	862	882	336	0.8929	938	261	0.8775	−6.3	22.3	1.7
	989	1297	900	518	0.9214	986	243	0.8652	−9.6	53.1	6.1
	1454	1027	1395	208	0.9063	1438	36	0.8811	−3.1	82.7	2.8
58	2458	1862	2367	592	0.8938	2457	407	0.8752	−3.8	31.3	2.1
	3062	2980	2900	449	0.9036	3060	210	0.8649	−5.5	53.2	4.3
	1497	2892	1298	2052	0.9190	1488	1256	0.8855	−14.6	38.8	3.6

system reliability is related to the T value which is taken randomly, and always follows the law of decreasing with the growth of T.

5.3 Reliability Contrast Test

For the purpose of investigating the influence of the system reliability on partitioning, and comparing with prior works, we make a series of settings for the maximizing system reliability model to achieve parameters similar to those in [12].

Cost c_i^h and time t_i^s, t_i^h are set as linear uncertain variables, denoted by $\mathcal{L}(a_{ci}^h, b_{ci}^h)$, $\mathcal{L}(a_{ti}^s, b_{ti}^s)$, and $\mathcal{L}(a_{ti}^h, b_{ti}^h)$ respectively, where a_{ci}^h, b_{ci}^h, a_{ti}^s, b_{ti}^s, a_{ti}^h, and b_{ti}^h are nonnegative real numbers. The inverse uncertainty distribution of linear uncertain variable $\mathcal{L}(a, b)$ is

$$\Phi^{-1}(\alpha) = (1 - \alpha)a + \alpha b.$$

Here α is equal to 1. a_{ci}^h, b_{ci}^h, a_{ti}^s, b_{ti}^s, a_{ti}^h, and b_{ti}^h are generated in the same way as described in [12]. a_{ci}^h is randomly generated from [0, 100], and b_{ci}^h is set as $a_{ci}^h + \beta_{ci}^h$, where β_{ci}^h is a nonnegative constant. a_{ti}^h is randomly generated from [0, 10], and b_{ti}^h is set as $a_{ti}^h + \beta_{ti}^h$, where β_{ti}^h is a nonnegative constant. a_{ti}^s is randomly generated from $[b_{ti}^h, 100]$ and b_{ti}^s is set as $a_{ti}^s + \beta_{ti}^s$, where β_{ti}^s is a nonnegative constant. $c_{i,j}$ is generated from $[0, \frac{1}{5} \cdot max(b_{ti}^s)]$. T_L is generated from $[\sum_{i=1}^n a_{ti}^h, \sum_{i=1}^n b_{ti}^s]$. C_L is generated from $[\sum_{i=1}^n a_{ci}^h, \sum_{i=1}^n b_{ci}^h]$.

We set parameters about the system reliability based on the experimental results in Section 5.1. μ_{li}^s, μ_{li}^h are generated from [2500, 2600], and σ_{li}^s, σ_{li}^h are generated from [100, 200], and T is set to 2300. Considering that we focus on the uncertainty, we cannot determine all the parameters to do experiments like [12], and we can only imitate its experimental method. We randomly generate 7 directed acyclic task graphs with different numbers of nodes, and the number of nodes is the same as in [12]. For each task graph, we first use Algorithm 2 to find the minimum hardware cost without reliability concern. Then, we use Algorithm 1 to the task graph to

Table 4. Comparison Results with Ref. [12]

Number of nodes	Ref. [12]		Our method	
	Reliability	Average	Reliability	Average
6	0.8250	0.8139	0.9632	0.9417
	0.8000		0.9420	
	0.8166		0.9198	
7	0.7525	0.7474	0.9583	0.9547
	0.7448		0.9682	
	0.7449		0.9377	
11	0.4704	0.4671	0.9100	0.9273
	0.4655		0.9486	
	0.4655		0.9234	
14	0.3916	0.3890	0.9196	0.9075
	0.3837		0.9139	
	0.3916		0.8889	
22	0.1441	0.1326	0.8933	0.9030
	0.1309		0.9036	
	0.1228		0.9121	
25	0.1325	0.1284	0.8929	0.9069
	0.1271		0.9214	
	0.1257		0.9063	
58	0.0061	0.0062	0.8938	0.9055
	0.0064		0.9036	
	0.0062		0.9190	

enhance system reliability. For each task graph, we separately record the results of three sets of random data, including the time, cost, and reliability calculated using the two algorithms, as listed in Table 3.

Considering that all the parameters that affect the results are randomly generated, there may be a situation where the upper limit and the obtained result have a large difference. It can be seen from Table 3 that compared with Algorithm 2 that does not consider reliability, the results obtained by Algorithm 1 have improved reliability, but the reliability improvement rate is not high. On the one hand, from the reliability values listed in Table 3, it can be seen that the system reliability obtained is relatively high even if the partitioning is performed using Algorithm 2, so the optimization space is small, resulting in a low reliability improvement rate when using Algorithm 1. On the other hand, although the reliability improvement rate is not high, the reliability is indeed improved when using Algorithm 1, which shows that Algorithm 1 can enhance the reliability. As can be seen from the data in the last vertical group in Table 3, this improvement comes at the expense of increasing the cost, within the allowable time limit. Similarly, we can also make this improvement at the expense of increasing the time, within the allowable cost limit. Therefore, if we need higher reliability and do not have stricter requirements on time or cost, Algorithm 1 can spend time or cost overhead to pursue reliability enhancement while meeting the time and cost constraints.

Furthermore, we compare our reliability with the reliability of Ref. [12]. Due to our different calculation methods and parameter requirements, we only compare the two in terms of change trend and the order of magnitude, as shown in Table 4. It can be seen that the reliability of [12] decreases sharply with the growth in the number of nodes, while our reliability has not changed significantly. By observing the reliability corresponding to different numbers of nodes, we can see that our reliability is much greater than the reliability of [12] in the order of magnitude when there are more nodes. For example, when the number of task nodes is 58, the reliability of [12] has dropped to less than 0.01, while our reliability still remains above 0.1.

6 CONCLUSION

In this paper, we propose an uncertainty theory based CPS partitioning method with uncertain reliability analysis. We use uncertain variables to formulate cost, time, and reliability for establishing our uncertainty model; this is the first work to include the uncertainty of reliability into the CPS partitioning. Our conversion method can be applied to various forms of uncertain variables. Our reliability enhancement algorithm can enhance the system reliability while satisfying time and cost constraints. Our system reliability obtained through uncertain reliability analysis does not change significantly with the growth of task module numbers.

REFERENCES

[1] J. Albuquerque, C. Coelho, C. F. Cavalcanti, D. Cecilio da Silva, and A. O. Fernandes. 1999. System-level partitioning with uncertainty. In *Proceedings of the Seventh International Workshop on Hardware/Software Codesign (CODES'99).* 198–202.

[2] P. Arato, S. Juhasz, Z. A. Mann, A. Orban, and D. Papp. 2003. Hardware-software partitioning in embedded system design. In *Proceedings of IEEE International Symposium on Intelligent Signal Processing, 2003.* 197–202.

[3] Péter Arató, Zoltán Ádám Mann, and András Orbán. 2005. Algorithmic aspects of Hardware/Software partitioning. *ACM Transactions on Design Automation of Electronic Systems* 10, 1 (Jan. 2005), 136–156.

[4] B. Liu. 2010. Uncertain risk analysis and uncertain reliability analysis. *Journal of Uncertain Systems* 4 (Jan. 2010), 163–170.

[5] G. Dartmann, H. Song, and A. Schmeink. 2019. *Big Data Analytics for Cyber-Physical Systems: Machine Learning for the Internet of Things.* Elsevier, Amsterdam.

[6] E. Ilavarasan and Perumal Thambidurai. 2007. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Science* 3 (Feb. 2007).

[7] N. Govil, R. Shrestha, and S. R. Chowdhury. 2017. PGMA: An algorithmic approach for multi-objective hardware software partitioning. *Microprocessors and Microsystems* 54 (Oct. 2017), 83–96.

[8] Z. A. Halim, B. S. Babu, and M. Mustaffa. 2020. Hardware software partitioning using four levels hybrid algorithm technique. In *Proceedings of 2020 IEEE 10th Symposium on Computer Applications Industrial Electronics (ISCAIE).* 42–47.

[9] T. Ham, Y. Lee, S. Seo, S. Kim, H. Choi, S. Jung, and J. Lee. 2021. ELSA: Hardware-Software co-design for efficient, lightweight self-attention mechanism in neural networks. In *Proceedings of 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA).* 692–705.

[10] N. Hou, F. He, Y. Zhou, and Y. Chen. 2020. An efficient GPU-based parallel tabu search algorithm for hardware/software co-design. *Frontiers of Computer Science* 14, 5 (March 2020), 1–18.

[11] Y. Jiang, H. Song, Y. Yang, H. Liu, M. Gu, Y. Guan, J. Sun, and L. Sha. 2018. Dependable model-driven development of CPS: From stateflow simulation to verified implementation. *ACM Transactions on Cyber-Physical Systems* 3, 1 (Aug. 2018), 1–31.

[12] Y. Jiang, M. Wang, X. Jiao, H. Song, H. Kong, R. Wang, Y. Liu, J. Wang, and J. Sun. 2019. Uncertainty theory based reliability-centric cyber-physical system design. In *Proceedings of 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData).* 208–215.

[13] Y. Jiang, H. Zhang, X. Jiao, X. Song, W. N. N. Hung, M. Gu, and J. Sun. 2012. Uncertain model and algorithm for hardware/software partitioning. In *Proceedings of the 2012 IEEE Computer Society Annual Symposium on VLSI.* IEEE Computer Society, 243–248.

[14] Jay Lee, Behrad Bagheri, and Hung-An Kao. 2015. A Cyber-physical systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters* 3 (Jan. 2015), 18–23.

[15] Clifford Liem and Pierre Paulin. 1997. Hardware/Software Co-Design: Principles and Practice. (Jan. 1997).

[16] B. Liu. 2010. *Uncertainty Theory.* Springer-Verlag, Berlin.

[17] A. Ouyang, X. Peng, J. Liu, and A. Sallam. 2017. Hardware/Software partitioning for heterogenous MPSoC considering communication overhead. *International Journal of Parallel Programming* 45, 4 (Aug. 2017), 899–922.

[18] N. Rashid, G. Quiros, and M. A. A. Faruque. 2019. A survivability-aware cyber-physical systems design methodology. In *Proceedings of 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. 848–853.
[19] E. Sha, L. Wang, Q. F. Zhuge, J. Zhang, and J. Liu. 2015. Power efficiency for hardware/software partitioning with time and area constraints on MPSoC. *International Journal of Parallel Programming* 43, 3 (June 2015), 381–402.
[20] W. Shi, J. Wu, G. Jiang, and S. Lam. 2019. Multiple-Choice hardware/software partitioning for tree task-graph on MPSoC. *Comput. J.* 63, 5 (Feb. 2019), 688–700.
[21] H. Song, G. Fink, and S. Jeschke. 2018. *Security and Privacy in Cyber-Physical Systems: Foundations, Principles and Applications*. Wiley, Hoboken.
[22] H. Song, D. Rawat, S. Jeschke, and C. Brecher. 2016. *Cyber-Physical Systems: Foundations, Principles and Applications*. Academic Press, Boston.
[23] H. Song, R. Srinivasan, T. Sookoor, and S. Jeschke. 2017. *Smart Cities: Foundations, Principles and Applications*. Wiley, Hoboken.
[24] N. Talati, K. May, A. Behroozi, Y. Yang, K. Kaszyk, and C. Vasiladiotis. 2021. Prodigy: Improving the memory latency of data-indirect irregular workloads using hardware-software co-design. In *Proceedings of 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 654–667.
[25] S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, Y. Xie, and W. Hung. 2005. Reliability-centric hardware/software co-design. In *Proceedings of 6th International Symposium on Quality Electronic Design (ISQED'05)*. 375–380.
[26] R. Wang, W. N. N. Hung, G. Yang, and X. Song. 2016. Uncertainty model for configurable hardware/software and resource partitioning. *IEEE Trans. Comput.* 65, 10 (Jan. 2016), 3217–3223.
[27] J. Wu, B. Chang, and T. Srikanthan. 2009. A hybrid branch-and-bound strategy for hardware/software partitioning. In *Proceedings of 2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*. 641–644.
[28] J. Wu, T. Srikanthan, and G. Chen. 2010. Algorithmic aspects of hardware/software partitioning: 1D search algorithms. *IEEE Trans. Comput.* 59, 4 (April 2010), 532–544.
[29] Q. Xiao, S. Zheng, B. Wu, P. Xu, X. Qian, and Y. Liang. 2021. HASCO: Towards agile hardware and software co-design for tensor computation. In *Proceedings of 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1055–1068.
[30] X. H. Yan, F. Z. He, and Y. L. Chen. 2017. A novel hardware/software partitioning method based on position disturbed particle swarm optimization with invasive weed optimization. *Journal of Computer Science and Technology* 32, 2 (March 2017), 340–355.

Received May 2021; revised September 2021; accepted October 2021