Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 3785452, 13 pages
https://doi.org/10.1155/2021/3785452

Research Article

Task Scheduling for Multiunit Parallel Test Using Mixed-Integer Linear Programming

Zhao Yang, Han-Shan Xiao, Rui Guan, Yang Yang, and Hong-Liang Ji

China Aerodynamics Research and Development Center, Mianyang 621000, Sichuan, China

Correspondence should be addressed to Han-Shan Xiao; yangzhao@cardc.cn

Received 12 April 2020; Revised 13 November 2020; Accepted 13 January 2021; Published 29 January 2021

Academic Editor: Akemi Gálvez

Parallel test is an efficient approach for improving test efficiency in the aerospace field. To meet the challenges of implementing multiunit parallel test in practical projects, this paper presented a mixed-integer linear programming (MILP) model for solving the task scheduling problem. A novel sequence-based iterative (SBI) method is proposed to solve the model in reasonable time. The SBI method is composed of an implied sequence finding procedure (ISF) and a sequence-based iterative optimization (SBIO) procedure. The first procedure can reduce the search space by fixing free sequence variables according to the original test flowcharts, and the second procedure can solve the model iteratively in a reasonable amount of time. In addition, two indexes, namely, speed rate and average resource utilization rate, are introduced to evaluate the proposed methods comprehensively. Computational results indicate that the proposed method performs well in real-world test examples, especially for larger examples that cannot be solved by the full-space method. Furthermore, it is proved that the essence of the parallel test is trading space for time.

1. Introduction

In the aerospace field, test is an essential procedure in the manufacturing of aircraft, missiles, and satellites to guarantee the functional integrity. Every piece of equipment is a system of multiple relatively independent subsystems. When testing a system, every subsystem must pass some specific test items. In the test field, the relatively independent subsystem to be tested is called the unit under test (UUT), and the corresponding test items are called test tasks. Traditionally, the UUTs of an aerospace equipment will be tested sequentially [1, 2] using an automatic test system (ATS), which provides the necessary resources for test tasks. Sequential test methods are time-consuming [3] because of the unit-by-unit and task-by-task strategy. With the increase of test workload, sequential test severely restricts the efficiency of equipment support. Therefore, implementing parallel test is becoming highly desirable in the aerospace field [1, 4]. Parallel test means that multiple UUTs can be tested at the same time and that multiple test tasks can be performed at the same time, which can improve test efficiency without increasing the cost [4, 5].

However, implementing parallel test in projects is challenging due to several practical difficulties [1]. First, resource competition can occur because test resources in an ATS are limited under the volume and budget constraints. Commonly, there are few spare resources in an ATS, and the test tasks must be scheduled elaborately with a minimum test resource set. Second, most instrument resources can be controlled by only one software thread at a time, which leads to a task's exclusive occupation of relevant resources and threads. In the parallel test, tasks are assigned to multiple parallel software threads, with each thread performing some of the test tasks. Therefore, thread competition will occur among tasks that are assigned to the same thread. Third, the UUTs must be tested according to test flowcharts, which are strictly determined by the technical sequences and working principles. These difficulties lead to a complex NP-hard tasks scheduling problem [6, 7] that involves up to hundreds of decision variables.

Although task scheduling for multiunit parallel test is important in the aerospace test field, limited solutions have been proposed due to the difficulties that are mentioned above. Xin et al. [3] established a relation model of test tasks based on graph theory and proposed two algorithms based on graph coloring, whose correctness and feasibility were proved by both theory and simulation. Xiao and Wang [8] proposed a vertex partitioning algorithm, which was derived from graph theory to coordinate system resource conflicts and time precedence. Xia et al. [9] combined the genetic algorithm and the simulated annealing algorithm to find the multi-UUT test task array. Li [10] built a mathematical model for the parallel test system with multi-UUTs and resource constraints and solved the model using genetic algorithm. Grey and Elizalde [11] proposed a method for performing multiple tests on one or more units by dynamically searching next executable test task. Lu et al. proposed two multiobjective evolutionary algorithms based on decomposition [12, 13] for test task scheduling problem, both of which considered two minimization objectives that were the test makespan and the mean resource workload. Zhang et al. [14] improved the basic ant colony algorithm to solve the parallel test task scheduling problem and to obtain the task execution sequence with shortest test time. In reference [15], a hybrid algorithm which combined the advantages of genetic algorithm and greedy algorithm was proposed for the parallel test task scheduling problem, and the chromosome coding and crossover in genetic algorithm were improved. Dorronsoro and Pinel [16] proposed a new accurate and fast memetic parallel optimization algorithm for the independent tasks scheduling problem, which combined the virtual savant optimization framework with a parallel genetic algorithm. Zhang et al. [17] developed an optimization model that jointly considered the preventive maintenance policy and the operating unit number for the multiunit parallel production system. Kim and Kim [18] constructed an intelligent task scheduler to improve the performance and energy efficiency in a heterogeneous multicore environment.

However, the abovementioned works are not very practical when dealing with real-world test projects. Xin et al.'s [3] algorithm cannot be applied to large-scale problems because it is an enumeration method by nature. Xiao and Wang [8] do not consider dynamic test task addition and deletion; therefore, their approach is not flexible in real-world applications. In test tasks scheduling problem, test flowcharts mean predefined constraints for candidate solutions. In the works of references [9, 10, 14, 15], only simple time precedence of tasks is allowed, and it is difficult to generate legal chromosomes for the genetic algorithm, or legal candidate ant for the ant colony algorithm when considering test flowcharts. Autoscheduling method of Grey and Elizalde [11] allows the parallel execution of multiple tests by a given order without resource confliction, which means that it cannot optimize the test sequence from a global perspective. The two multiobjective evolutionary algorithms by Lu et al. [12, 13] perform well in experiment examples, but both of them supposed that the test start time and the test completion time for every test task were fixed, which means the test execution sequence cannot be optimized. The optimization algorithm for the independent tasks in reference [16] can get an accurate solution in a short computation time; however, it did not take the test flowchart into consideration. Models and methods in references [17, 18] did not take the diversity of tasks and the task-resource matching problem into consideration.

Task scheduling in multiunit parallel test is similar to scheduling problems in other domains [19–21]. For example, a farmer must optimize the use of land by selecting the best mix of crops to cultivate [22] and consider several factors simultaneously including market price variability, specific resource requests for each crop, machine availability restrictions, and timing of cultivation. Another example [23] is the creation of master theatre timetables in hospitals, which takes account of the goal of reducing the maximum number of beds that are required, surgeons' availability and preferences, variations in types of theatre, and so on. Although these problems differ in the details, the related models and approaches are inspiring to task scheduling of multiunit parallel test. To overcome the challenges mentioned above, this paper studies the task scheduling problem in multiunit parallel test and makes the following contributions:

(1) An MILP model that describes the resource competition and thread competition mathematically in multiunit parallel test is constructed. The resource competition and thread competition are described by introducing sequence variables.

(2) A novel SBI method is proposed for solving the MILP model. The SBI method includes two procedures: ISF and SBIO. The ISF procedure can reduce the search space by fixing free sequence variables according to the original test flowcharts, and the SBIO can solve the model in a reasonable amount of time.

(3) The proposed SBI method allows tasks to be added conveniently according to practical requirements. When the test flowchart changes, the project manager can simply "insert" the tasks to obtain a new optimal schedule without reoptimizing the whole problem, which minimizes the effort required to modify the test software.

The proposed test task scheduling method can be applied to develop a new ATS or improve the performance of an existing ATS without adding new test resources. In addition, the difficulty of multithread programming will be reduced because the thread competition is solved by the proposed method. These advantages make the proposed method more practical and valuable in real-world test projects.

This paper is organized as follows. In Section 2, a motivating example is presented to illustrate the concept of parallel test. In Section 3, an MILP model is built to optimize the test time of tasks for multiple UUTs. In Section 4, the SBI method for solving the proposed model is developed. Section 5 studies 7 real-world test examples to evaluate the efficiency of the proposed SBI method. Finally, conclusions and suggestions for future research are provided in Section 6.

2. Motivating Example

To illustrate the idea of multiunit parallel test, an example of 2 UUTs with 6 test tasks, which involve 4 types of resources, is demonstrated. The required resources and expected test times (test durations) of the 6 test tasks are given in Table 1. The prerequisite test sequences, which are determined by the working principle of the corresponding UUT, are shown in Figure 1.

As shown in Table 1, some tasks use the same test resource, such as t_1 and t_5, as well as t_2 and t_4. Currently, due to the uniqueness of the resource state, two tests that use the same resource cannot be performed simultaneously. Figure 1 shows the test flowcharts of UUT_1 and UUT_2, which specify the prerequisite technological sequences. Tasks t_1 and t_2 must finish before t_3 begins, and t_4 must finish before t_5 and t_6 begins.

Figure 2(a) illustrates the concept of sequential test. In the sequential test method, all the test tasks are executed one by one using only one software thread. Resource competition does not occur and the test sequence is not violated. However, the total test time is long (140 s) and equals the sum of the test times of all tasks. Figure 2(b) illustrates the concept of multi-UUT parallel test, which can reduce the test time by 50% (70 s) with 3 threads. Using multithread technology, different tasks can be performed simultaneously. However, there are two main challenges, resource competition and technological sequence restriction. Any resource conflict in the practical test will lead to task failure and uncertainty of the test resource. If any test sequence restriction is violated, the related UUTs will suffer security risks, especially for the high-tech equipment in the aerospace field.

This example shows that the test efficiency can be improved remarkably by multi-UUT parallel test technology. The problem becomes more challenging when more tasks and resource types with complex test flowcharts are taken into consideration. Therefore, an optimization model that accounts for all of the aspects mentioned above is needed to minimize the test time of the multiunit parallel test.

Table 1: Required resources and expected test times of the 2 UUTs.

UUT	Test task	Required resources				Test time (seconds)
		r_1	r_2	r_3	r_4	
UUT_1	t_1	1	1	0	0	30
	t_2	0	0	1	0	20
	t_3	0	0	0	1	20
UUT_2	t_4	0	0	1	1	30
	t_5	1	0	0	0	20
	t_6	0	1	0	0	20

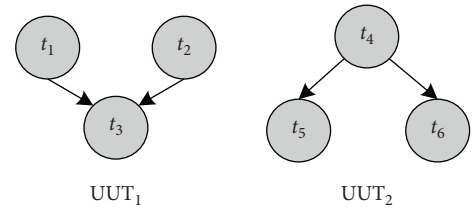

Figure 1: Test flowcharts of UUT1 and UUT2.

3. Mathematical Formulation

This paper discusses the task scheduling problem for multiunit parallel test under the following assumptions [9, 10]:

(1) Only minimum test resource sets are provided

(2) A resource can be occupied by only one test task at the same time

(3) The time for which a resource is occupied by a test task is equal to the duration of the task

(4) No test task can be interrupted and then completed later

(5) Different test tasks of the same UUT can be performed simultaneously by different threads without violating the task sequence constraints

All the assumptions are reasonable in real-world test projects. In assumption (1), the "minimum test resource sets" means sets with the minimum number of test resources (often one resource) that satisfy the test tasks. Assumption (1) is practical because the volume and budget of an ATS are often limited and because few spare resources are installed in the test system. Assumption (2) is based on the restriction that each resource can only be in one status. Therefore, only one task can be executed by a resource at a time. Assumption (3) means that a resource is occupied once the corresponding task starts and released once the task ends. Assumption (4) is because if a test task is interrupted, the UUT will work abnormally in subsequent tasks. Assumption (5) is the technological foundation of the parallel test.

The following symbols (see Table 2) are used in modelling the problem [24, 25].

The problem of task scheduling for multiunit parallel test can be stated as follows: given test tasks set T, test resources set R, test time set τ, and task-resource dependency matrix A, the objective is to find an optimal task schedule (x, s) that minimizes the overall test time without violating the prerequisite technological task sequence y^0.

This problem can be formulated as the following optimization model **M** [26, 27]:

$$\text{minimize } Z = \max_k \left(s_k + \tau_k\right), \quad k = 1, 2, \ldots, K, \tag{1}$$

under the constraints

$$\sum_{j=1}^{J} x_{kj} = 1, \quad j = 1, 2, \ldots, J, \forall k = 1, 2, \ldots, K, \tag{2}$$

$$\begin{aligned} &a_{k_1 i} + a_{k_2 i} - y_{k_1 k_2} - y_{k_2 k_1} \le 1, \\ &\quad \forall i = 1, 2, \ldots, I, \forall k_1 \ne k_2, k_1, k_2 = 1, 2, \ldots, K, \end{aligned} \tag{3}$$

$$\begin{aligned} &x_{k_1 j} + x_{k_2 j} - y_{k_1 k_2} - y_{k_2 k_1} \le 1, \\ &\quad \forall j = 1, 2, \ldots, J, \forall k_1 \ne k_2, k_1, k_2 = 1, 2, \ldots, K, \end{aligned} \tag{4}$$

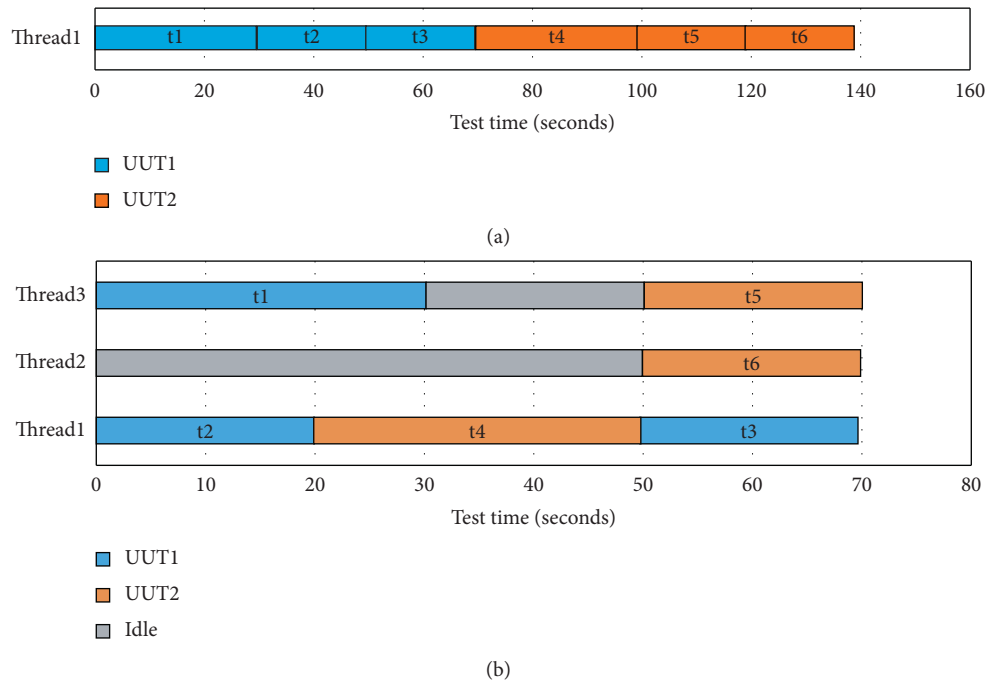

Figure 2: Alternative schedules for the motivating example. (a) Sequential test. (b) Multi-UUT parallel test.

Table 2: Parameters and variables used to formulate the optimal task scheduling problem.

Symbols	Description
R	$R = \{r_1, r_2, \ldots, r_i, \ldots r_I\}$ is a set of various test resources, where I is the number of test resources
N	N is the total number of UUTs to be tested
T	$T = T_1 \cup T_2 \cdots \cup T_n \cdots \cup T_N = \{t_1, t_2, \ldots, t_k, \ldots, t_K\}$; K is the number of test tasks for all UUTs, and t_k represents the kth test task in all tasks
T_n	$T_n = \{t_{1n}, t_{2n}, \ldots, t_{kn}, \ldots, t_{K_n}\}$ is the set of test tasks for UUT$_n$, K_n is the number of test tasks for UUT$_n$, and t_{k_n} represents the kth test task for UUT$_n$
K	$K = \sum_{n=1}^{N} K_n$ is the total number of test tasks for all UUTs
τ	Test time vector $\tau = \{\tau_1, \tau_2, \ldots, \tau_k, \ldots, \tau_K\}$, and τ_k represents the time needed for test k
A	$K \times I$ task-resource dependency matrix; the $(k \times i)$th element $a_{ki} = 1$ if test k needs resource I, otherwise $a_{ki} = 0$
s	Task start time vector $s = \{s_1, s_2, \ldots, s_k, \ldots, s_K\}$, and s_k is a continuous variable that represents the start time of test k
y	$K \times K$ task sequence matrix; the $(k_1 \times k_2)$th element $y_{k_1k_2} = 1$ if there is a technological sequence between t_{k1} and t_{k2}, which means t_{k1} finishes before test t_{k1} begins, otherwise $y_{k_1k_2} = 0$. y^0 is the prerequisite technological task sequence, which is defined by the test flowchart
x	$K \times J$ matrix; the $(k \times j)$th element$x_{kj} = 1$ if the kth test task is assigned to thread j, otherwise $x_{kj} = 0$. J is the total number of test threads

$$s_{k_1} + \tau_{k_1} \le s_{k_2} + U \times \left(1 - y_{k_1k_2}\right), \quad \forall k_1, k_2 = 1, 2, \ldots, K, \tag{5}$$

$$s_k \ge 0, \quad \forall k = 1, 2, \ldots, K, \tag{6}$$

$$y_{k_1k_2} = 0, \quad \forall k_1 = k_2, k_1, k_2 = 1, 2, ..., K, \tag{7}$$

$$y_{k_1k_2} + y_{k_2k_1} \le 1, \quad \forall k_1, k_2 = 1, 2, \ldots, K, \tag{8}$$

$$y_{k_1k_2} \ge y^0_{k_1k_2}, \quad \forall k_1, k_2 = 1, 2, \ldots, K, \tag{9}$$

$$y_{k_1k_2} \in \{0, 1\}, \quad \forall k_1 \ne k_2, k_1, k_2 = 1, 2, \ldots, K, \tag{10}$$

$$x_{kj} \in \{0, 1\}, \quad \forall k = 1, 2, \ldots, K, \forall j = 1, 2, \ldots, J. \tag{11}$$

The objective function (1) minimizes the time of the latest completed test task. Constraint (2) ensures that every task is performed by exactly one software thread. Constraint (3) enforces an execution sequence between different tasks that need the same resource [28]. By determining an execution sequence, the resource competition problem will be solved. Constraint (4) enforces an execution sequence of different tasks that are assigned to the same thread. Constraint (5) enforces the time sequence when a sequence (either a prerequisite technological sequence or execution sequence) exists between two tasks, and U is a big-M parameter. Constraint (6) ensures that the starting time of every test task is bigger than zero. Constraint (7) ensures that no sequence is required for the task itself. Constraint (8) ensures that the sequences of tasks k_1 and k_2 will not conflict with each other. Constraint (9) requires the sequence matrix y to

conform with prerequisite technological task sequence y^0. Constraints (10) and (11) define x and y as binary variables.

The model **M** is an MILP model, in which s is a continuous variable and x and y are binary variables. The difficulty in solving this problem is due to the combination of sequencing and task assignment binaries (y and x). In small-scale problems, this model can be solved in a reasonable amount of time. However, for real-world test projects, the computation time increases exponentially with the number of tasks [26]. Therefore, it is necessary to explore reasonable methods for obtaining the optimal (or suboptimal) solution.

4. SBI Method

In real-world applications, the number of test tasks is large and the task-resource dependency is complex. Therefore, the problem is intractable with a full-space method [29]. In this section, the SBI method is proposed for solving the model **M**. The proposed SBI method is accomplished by the ISF SBIO procedure, and its scheme is shown in Figure 3.

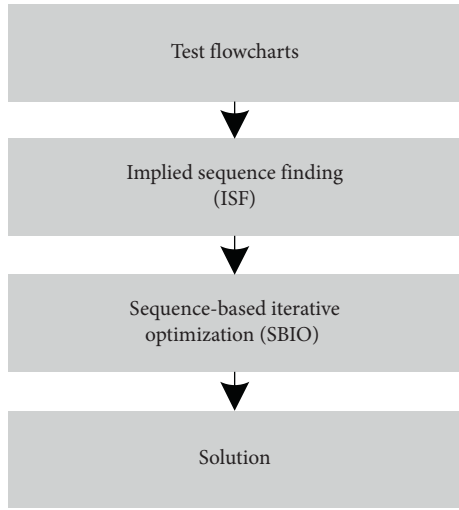

Figure 3: Scheme of the SBI method.

4.1. ISF Procedure. By test flowcharts, the prerequisite technological sequences can be fixed intuitively. For instance, $y_{12} = 1$, $y_{23} = 1$, $y_{52} = 1$, $y_{14} = 1$, $y_{43} = 1$, $y_{36} = 1$ can be set directly according to the test flowcharts shown in Figure 4(a). Although there is no prerequisite technological sequence for t_1 and t_3, t_1 must finish before t_3 begins because t_1 finishes before t_2 begins and t_2 finishes before t_3 begins. The sequences of t_1 and t_3 are called implied sequences [30]. It is difficult to find them one by one intuitively in large problems with many test tasks.

To derive all implied sequences, the ISF following procedure is proposed:

For all test tasks of UUT_n to be scheduled $k_1 = 1$: K_n

For all test tasks of UUT_n to be scheduled $k_2 = 1$: K_n

For all test tasks of UUT_n to be scheduled $k_3 = 1$: K_n

If $y_{k_1k_2} = 1$ AND $y_{k_2k_3} = 1$, then $y_{k_1k_3} = 1$

For the test flowchart shown in Figure 4(a), the implied sequences identified by the ISF procedure are shown by dashed lines in Figure 4(b). This ISF procedure is inspired in [28], in which the test tasks must be labelled elaborately according to their technological sequences. If the technological sequences are changed or new tasks are added to the flowcharts, the label numbers must be upgraded with the technological sequences, which brings great inconvenience to practical test project management [31]. In the ISF procedure proposed by this paper, the test tasks can be labelled freely. The ISF procedure can substantially reduce the computation time, as shown in Section 5.

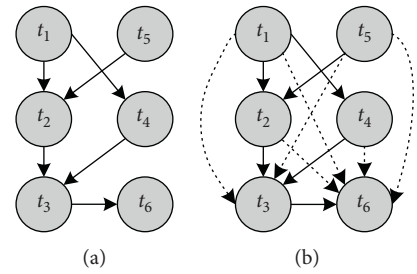

Figure 4: Application of the ISF procedure: (a) Original test flowchart. (b) Technological sequences after the ISF procedure.

4.2. SBIO Procedure. In model **M**, the sequence matrix y is a key variable, which includes a $K \times K$ sequence binary elements. These binary elements can be classified into three types depending on their sources:

(1) Prerequisite technological sequence (PTS): PTS can be fixed intuitively according to the original test flowcharts. The function of PTS is to ensure all the prerequisite technological sequences are followed.

(2) Implied sequence (IS): IS is the sequence identified by the ISF procedure. The function of IS is to reduce the number of binary variables to be optimized, thereby shortening the computation time.

(3) Free sequence (FS): FS is the indeterminate element to be optimized in matrix y after the ISF procedure. FS is used to obtain a minimum objective function Z when the tasks use the same resource or are assigned to the same thread. The combination of FS is a leading contributor to the high computational burden.

By the ISF procedure, the PTS and IS of every UUT can be determined. Thus, the sequence matrix for every UUT is dense (y (1), y (2), . . ., y (N) in Figure 5). However, for the integrated matrix of multiple UUTs, there are many FSs to be optimized (gray region in Figure 5). It is time-consuming to solve the model **M**. Of the three types of sequence elements, PTS and IS are determined by test flowcharts. This makes fixing FS before optimization the only way to reduce the computation time. Based on the above analysis, model **M** is reformulated and the SBIO procedure is proposed.

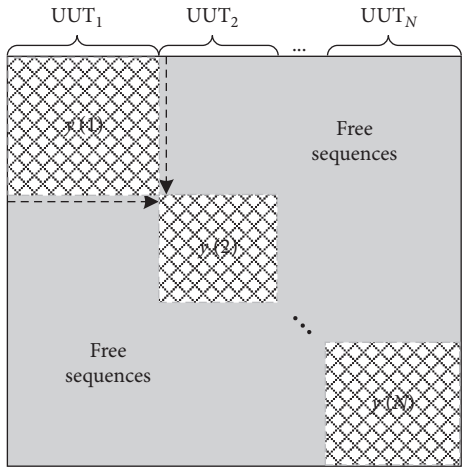

Figure 5: Structure of sequence matrix y.

A reformulated model $\mathbf{M}$ (k) is proposed:

$$\text{minimize } Z = \max_{k_1}\left(s_{k_1} + \tau_{k_1}\right), \quad k_1 = 1, 2, \ldots, k, \tag{12}$$

under the constraints

$$\sum_{j=1}^{J} x_{k_1 j} = 1, \quad j = 1, 2, \ldots, J, \forall k_1 = 1, 2, \ldots, k, \tag{13}$$

$$\begin{aligned} &a_{k_1 i} + a_{k_2 i} - y_{k_1 k_2} - y_{k_2 k_1} \le 1, \\ &\quad \forall i = 1, 2, \ldots, I, \forall k_1 \ne k_2, k_1, k_2 = 1, 2, \ldots, k, \end{aligned} \tag{14}$$

$$\begin{aligned} &x_{k_1 j} + x_{k_2 j} - y_{k_1 k_2} - y_{k_2 k_1} \le 1, \\ &\quad \forall j = 1, 2, \ldots, J, \forall k_1 \ne k_2, k_1, k_2 = 1, 2, \ldots, k, \end{aligned} \tag{15}$$

$$s_{k_1} + \tau_{k_1} \le s_{k_2} + U \times \left(1 - y_{k_1 k_2}\right), \quad \forall k_1, k_2 = 1, 2, \ldots, k, \tag{16}$$

$$s_{k_1} \ge 0, \quad \forall k_1 = 1, 2, \ldots, k, \tag{17}$$

$$y_{k_1 k_2} = 0, \quad \forall k_1 = k_2, k_1, k_2 = 1, 2, \ldots, k, \tag{18}$$

$$y_{k_1 k_2} + y_{k_2 k_1} \le 1, \quad \forall k_1, k_2 = 1, 2, \ldots, k, \tag{19}$$

$$y_{k_1 k_2} \ge y^0_{k_1 k_2} + \sum_{l=1}^{k-1} y^l_{k_1 k_2}, \quad \forall k_1, k_2 = 1, 2, \ldots, k, \tag{20}$$

$$y_{k_1 k_2} \in \{0, 1\}, \quad \forall k_1 \ne k_2, k_1, k_2 = 1, 2, \ldots, k, \tag{21}$$

$$x_{k_1 j} \in \{0, 1\}, \quad \forall j = 1, 2, \ldots, J, \forall k_1 = 1, 2, \ldots, k. \tag{22}$$

Compared to model $\mathbf{M}$, model $\mathbf{M}$ (k) restricts the tasks to be scheduled to a partial test set $\{t_1, t_2, \ldots, t_k\}$. Constraint (9) in $\mathbf{M}$ is reformulated as constraint (20) in $\mathbf{M}$ (k). $y^l_{k_1 k_2}$ is the optimization result of model $\mathbf{M}$ (l).

The SBIO procedure for solving model $\mathbf{M}$ (k) is shown in Figure 6. The initial sequence matrix y^1 is set to 0 because of its 1×1 dimension and lack of prerequisite test sequence for itself. For every iteration k, the SBIO procedure will solve the model $\mathbf{M}$ (k) to obtain y^k. After adding PTS of t_{k+1} to y^k, the ISF procedure will be performed to reduce the number of FSs. Then, the sequence matrix y^k obtained by solving $\mathbf{M}$ (k) will be used to rebuild $\mathbf{M}$ $(k+1)$. The reservation mechanism of the last step's optimization result will reduce the computation time of the current step. Finally, model $\mathbf{M}$ (K) (i.e., model $\mathbf{M}$) will be solved and the corresponding optimization result will be output as the solution.

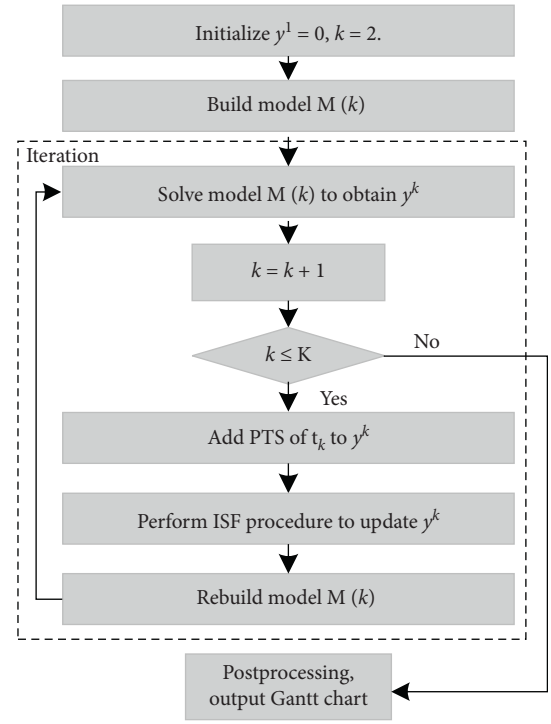

Figure 6: SBIO procedure.

5. Experiments and Results

5.1. Experiment Design. In this section, 7 real-world examples which involve including 3 UUTs, 31 test tasks, and 10 resources were studied to evaluate the SBI method. All the examples were solved on a PC with an Intel Core i5-4590 3.3 GHz CPU using CPLEX 12.6. The relative optimality tolerance was set to 0.1%. The other CPLEX options have been set as the default for all models. In all models, the big-M parameter was set as the sum of all test tasks times, multiplied by a safety factor 2, so no feasible solutions would be cut off [28].

$$U = 2 \times \sum_{k_1=1}^{k} \tau_{k_1}, \quad \forall k_1 = 1, 2, \ldots, k. \tag{23}$$

The task-resource dependency matrix and corresponding test time are shown in Table 3. The configurations of the 7 examples are given in Table 4. The test flowcharts of 3 UUTs are presented in Figure 7. The small examples (1, 2, and 3) each involve only one UUT. The medium examples (4, 5, and

Table 3: Problem data: task-resource dependency matrix and test times of 3 UUTs.

UUT	Test task	Required resources set R										Test time τ (s)
		r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	
UUT_1	T_1											
	t_1	1	0	1	0	0	0	0	0	0	0	18
	t_2	1	1	0	0	0	0	0	0	0	0	20
	t_3	1	0	0	0	1	0	0	0	0	0	42
	t_4	0	0	0	0	0	1	0	0	1	0	120
	t_5	1	0	0	0	0	1	0	0	0	0	26
	t_6	0	1	0	0	0	0	1	0	0	0	25
	t_7	0	1	1	0	0	0	0	0	0	0	40
	t_8	1	0	1	0	0	0	0	0	0	0	26
	t_9	0	1	0	1	0	0	0	0	0	0	60
	t_{10}	0	0	0	1	0	0	0	0	0	0	15
	t_{11}	1	0	0	1	0	0	0	0	0	0	16
	t_{12}	0	0	1	0	1	0	0	0	0	0	20
UUT_2	T_2											
	t_{13}	0	0	0	0	0	1	0	0	0	0	30
	t_{14}	0	0	1	0	0	0	0	0	1	0	75
	t_{15}	0	0	0	0	1	1	0	0	0	0	60
	t_{16}	0	0	0	0	1	0	0	1	0	0	145
	t_{17}	1	0	0	0	0	1	0	0	0	0	54
	t_{18}	0	0	0	1	0	0	0	0	0	0	120
	t_{19}	0	0	0	0	1	0	0	0	0	0	135
	t_{20}	0	0	0	0	0	1	0	0	0	0	10
	t_{21}	0	0	0	0	0	0	1	0	0	0	32
	t_{22}	1	1	0	0	1	0	0	0	0	0	25
UUT_3	T_3											
	t_{23}	1	0	0	0	0	0	0	1	0	1	15
	t_{24}	0	0	1	1	0	1	0	0	0	0	32
	t_{25}	0	0	0	0	0	1	0	1	0	0	40
	t_{26}	0	0	0	0	0	0	0	1	0	1	55
	t_{27}	1	0	0	1	0	1	0	0	0	0	55
	t_{28}	1	0	0	1	0	0	0	0	0	0	24
	t_{29}	0	1	0	0	0	0	1	0	0	0	75
	t_{30}	0	0	1	0	0	0	0	0	1	0	81
	t_{31}	0	0	0	0	0	0	0	1	1	0	30

Table 4: Configuration of 7 examples.

Example	1	2	3	4	5	6	7
UUTs	1	2	3	1&2	1&3	2&3	1&2&3
Total tasks (K)	12	10	9	22	21	19	31

6) involve two UUTs, respectively. The large example involves all 3 UUTs. For every example, three scenarios (using 2, 3, and 4 threads) and three methods (full-space method without ISF, full-space method with ISF, and SBI method) for parallel test are studied to find the optimal solution.

Two indexes, speed rate (SR) and average resource utilization rate (ARUR), are used to evaluate the efficiency of the proposed method comprehensively.

The SR is defined as the ratio of the sequential test time and the parallel test time for the same task set.

$$\mathrm{SR} = \frac{\sum_{k=1}^{K} \tau_k}{\max_k \left(s_k + \tau_k\right)}, \quad k = 1, 2, \ldots, K. \tag{24}$$

The ARUR is defined as the integral average of the instantaneous resource utilization rate over the whole test.

$$\mathrm{ARUR} = \frac{\int_0^t \left(n_\tau / N\right) \mathrm{d}\tau}{t} \times 100\%. \tag{25}$$

In (25), t is the completion time of the whole test, n_τ is the number of utilized resources at time τ, and N is the total number of utilized resources during the whole test. The SR can evaluate the extent of the speedup of the parallel test compared to the sequential test. The ARUR can indicate the overall level of resource utilization.

5.2. Model Statistics and Experiment Results. The model statistics and solution details for all examples are given in Table 5. Examples are labelled in the format "*E*-*T*," where "*E*" stands for the example number and "*T*" represents the number of threads that are used. For example, label "7-3" means that example 7 is solved using 3 threads. The recorded model statistics are the numbers of constraints and binary and continuous variables in model **M**. The solution details of both the full-space method and the SBI method are recorded for comparison. The search time is limited to 2000 seconds for all examples. The computation times of the full-space method are recorded in the two cases. The FIS procedure is adopted in the first case but not the second. The computation time of the SBI method includes the total time needed for the ISF and SBIO procedures.

The advantage of the proposed MILP model lies in its unlimited descriptive ability, which means that it can describe test task scheduling problem with any number of tasks, resources, and threads. What is more, every practical constraint for test resources, threads, and sequences can be described clearly by the corresponding equation or inequality in model **M**(k). However, the number of binary variables and total constraints will increase sharply with the number of tasks, resources, and threads getting larger (see "Model details" in Table 5), which is caused by the combination of the three. This will lead to more memory to be consumed when solving the model. Considering that the memory of the modern computer is large enough to deal with it and the computation time will be reduced by the proposed SBI method, this disadvantage can be overcome.

As shown in Table 5, the optimal solution cannot be found in 2000 s by the full-space method without the ISF procedure in examples 7-2, 7-3, and 7-4. Nor can it be found by the full-space method with the ISF procedure in 2000 s in examples 7-3 and 7-4. However, the optimal (or suboptimal) solutions can be found by the proposed SBI method in all examples.

5.3. Discussion

5.3.1. Computation Time. The computation time of all examples is shown in Figure 8 on the natural logarithmic scale. In all examples, the full-space method without the ISF procedure takes the most computation time. The full-space method with the ISF procedure takes the least computation time in small problems that involve one UUT (examples 1, 2, and 3). However, for medium and large problems with 2 or 3 UUTs (examples 4-7), the SBI method takes the least computation time. Therefore, the time efficiency of the SBI

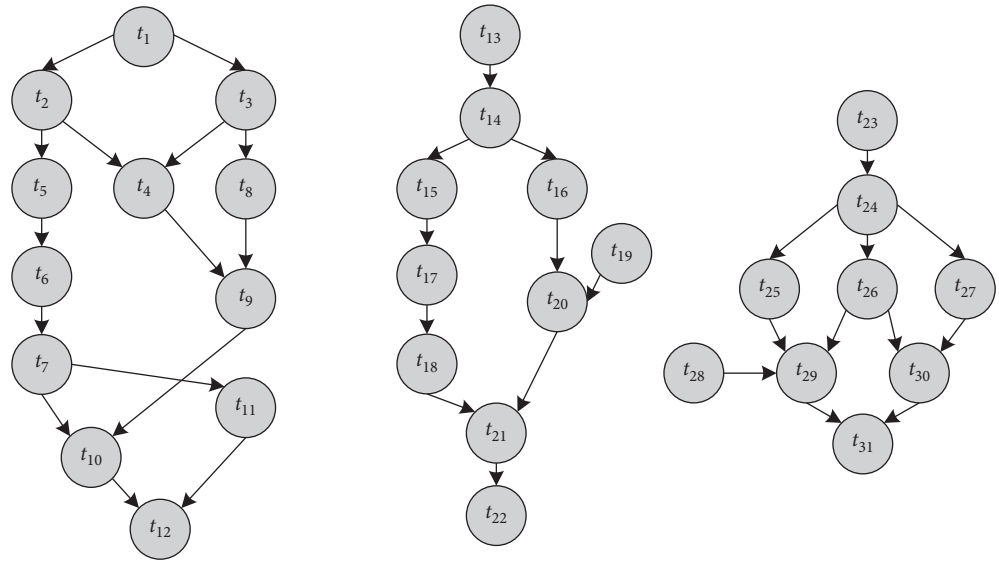

Figure 7: Test flowcharts of UUT1, UUT2, and UUT3.

Table 5: Model statistics and solution details of 7 examples.

Example	Model details			Full-space method			SBI method			
	Binary variables	Continuous variables	Constraints	Optimal solution (s)	Computation time without ISF (s)	Computation time with ISF (s)	Best solution (s)	Computation time (s)	SR	ARUR
1-2	156	12	605	321	16.2	1.7	321	9.4	1.33	26.1
2-2	110	10	407	461	11.1	1.4	461	6.6	1.49	26.7
3-2	90	9	334	253	8.1	1.3	253	5.6	1.61	36.0
4-2	506	22	2065	728	485.7	66.4	728	35.3	1.53	28.5
5-2	462	21	1898	577	366.6	53.2	577	32.1	1.45	30.4
6-2	380	19	1532	732	206.2	32.8	732	24.8	1.49	29.3
7-2	992	31	4150	999[a]	2000	1066.3	999[a]	86.5	1.52	29.9
1-3	168	12	869	321	25.1	2.1	321	10.5	1.33	26.1
2-3	120	10	587	456	16.5	1.7	456	7.1	1.5	27.0
3-3	99	9	478	253	12.1	1.5	253	5.9	1.61	36.1
4-3	528	22	2989	658	758.5	89.6	658	44.4	1.69	31.5
5-3	483	21	2738	533	593.6	68.9	533	40	1.57	32.9
6-3	399	19	2216	619	321.6	43.1	619	30.7	1.77	34.7
7-3	1023	31	6010	None	2000	2000	815	113.5	1.87	36.7
1-4	180	12	1133	321	31.9	2.5	321	11.5	1.33	26.1
2-4	130	10	767	456	22.1	1.9	456	7.6	1.5	27.0
3-4	108	9	622	253	16.1	1.7	253	6.3	1.61	36.0
4-4	550	22	3913	577	1077.5	108	577	53.4	1.93	37.1
5-4	504	21	3578	553	844.6	83.2	533	47.9	1.57	32.9
6-4	418	19	2900	619	461.8	54.4	619	36.3	1.77	34.7
7-4	1054	31	7870	None	2000	2000	815	141.2	1.87	36.7

[a]Solution of examples 7-2 is found by the full-space method with ISF.

method is better than that of the full-space method for large problems, such as in practical multiple UUTs parallel test applications.

5.3.2. Resource Competition and Thread Competition. The best solutions (i.e., those with the minimum tasks completion time) with different number of threads are shown in Figure 9. For small problems (examples 1, 2, and 3), using different numbers of threads has no remarkable effect on the value of the solution. The best solution of example 2 using 2 threads is associated with longer computation time (461 s) than the solutions that are obtained using 3 and 4 threads (456 s) because some tasks must "wait" to be performed when the number of threads is limited. However, for medium and large problems (examples 4-7),

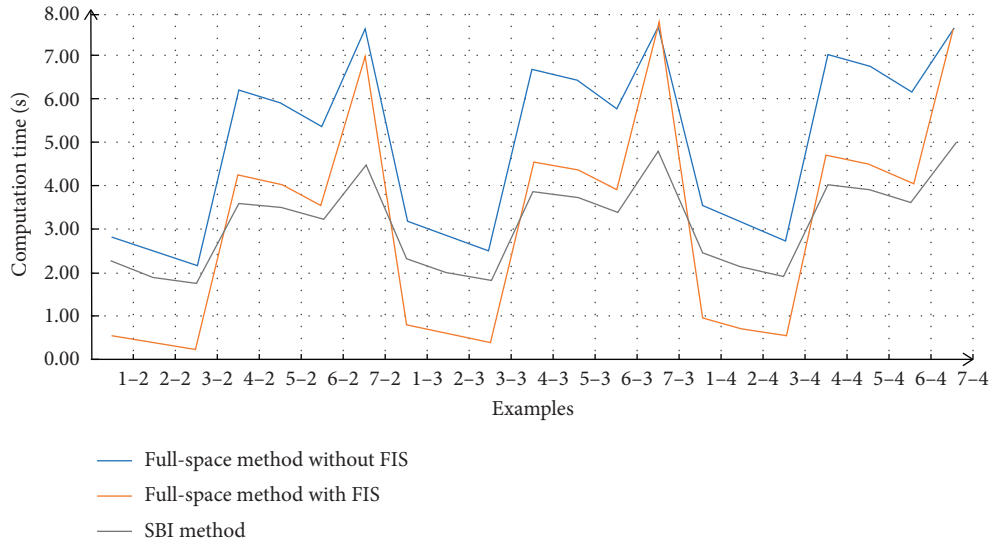

Figure 8: Computation time of all the examples.

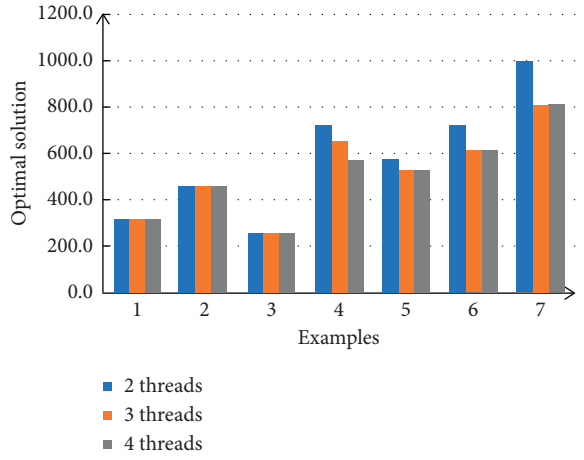

Figure 9: Best solutions with different threads.

using more threads to schedule test tasks can reduce the completion time of the whole test. Nevertheless, it does not promise an affirmatory optimal solution reduction (see examples 4-6 in Figure 9) because the completion time is restricted by test resources other than the number of threads. In practical test projects, using too many threads will lead to higher complexity of resource control. For this reason, applying fewer threads is preferable when the solutions are equal. For the biggest problem (example 7), the same solution can be found using either 3 or 4 threads.

The Gantt chart of the best solution for all the examples are shown in Figure 10, in which the resource competition and thread competition are efficiently solved, and this will facilitate the software programming and resource control in practice.

5.3.3. Effect of Adding Tasks. To evaluate the effect of adding tasks in the proposed model, three tasks (t_{32}, t_{33}, and t_{34}) were randomly generated and inserted into the test flowchart in example 7. The three tasks were generated by following rules: (1) the required resources, with a maximum number of 3, were randomly selected from the given resource set; (2) test time for each task was a randomly generated real number within the bounds $[\min(\tau_k), \max(\tau_k)]$, $k = 1, 2, \ldots, 31$; and (3) each task was inserted into one of the test flowchart, which means $t_{32} \in T_1, t_{33} \in T_2, t_{34} \in T_3$. The test sequences were also randomly generated. Table 6 and Figure 11 report the details of three randomly generated tasks. Example 7 with tasks t_{32}, t_{33}, and t_{34} was denoted as example 7^*. Example 7^* was solved based on model **M** (31) of example 7 using 3 threads, and the model statics and solution details are recorded in Table 7.

The additional computation time for adding a task to the proposed model is less than 10 seconds (see Table 7). When adding a new task to test projects, the developer simply needs to describe the resource requirements and redefines the test flowchart. In comparison, the procedure proposed by Maravelias and Grossmann [28] has to label

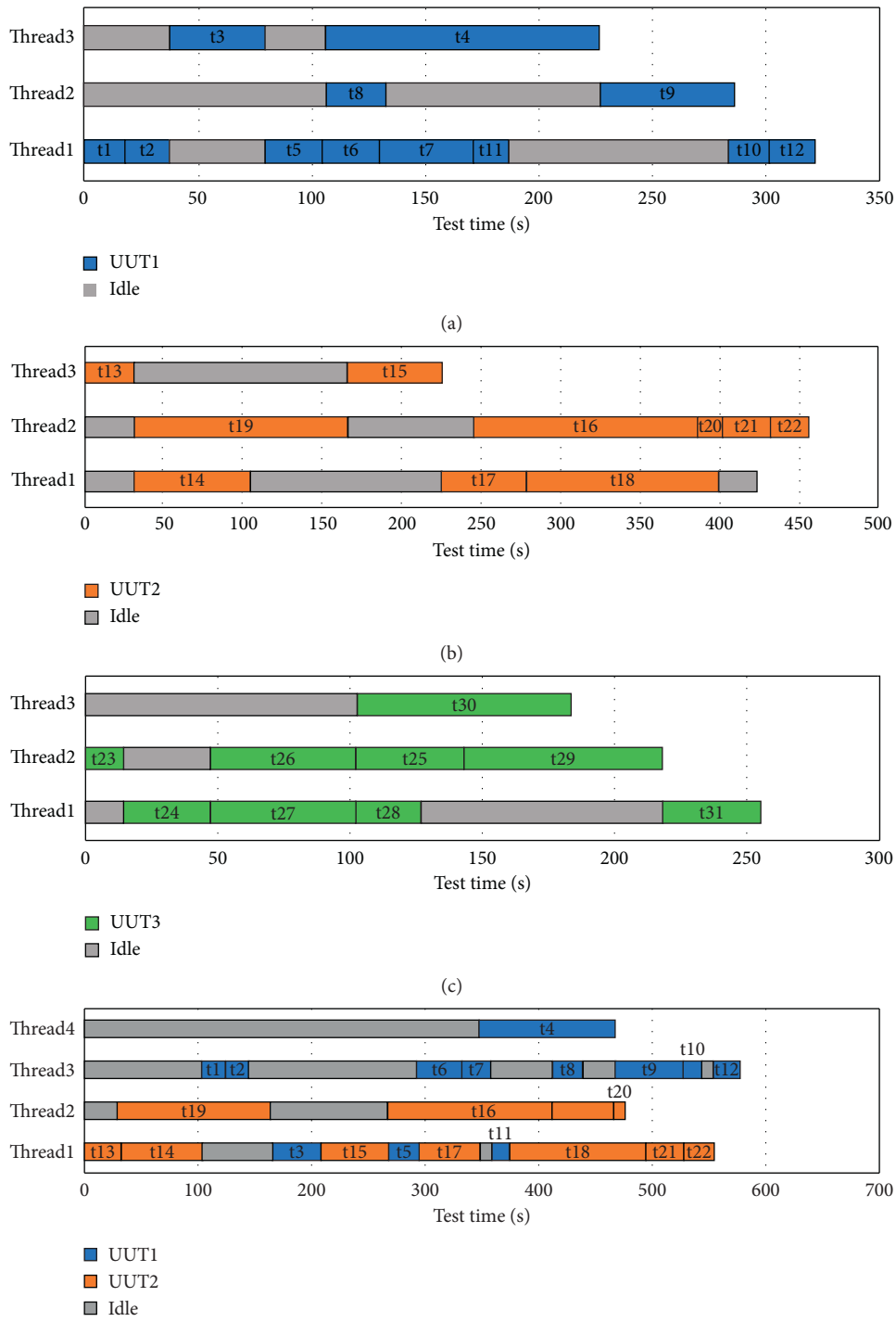

Figure 10: Continued.

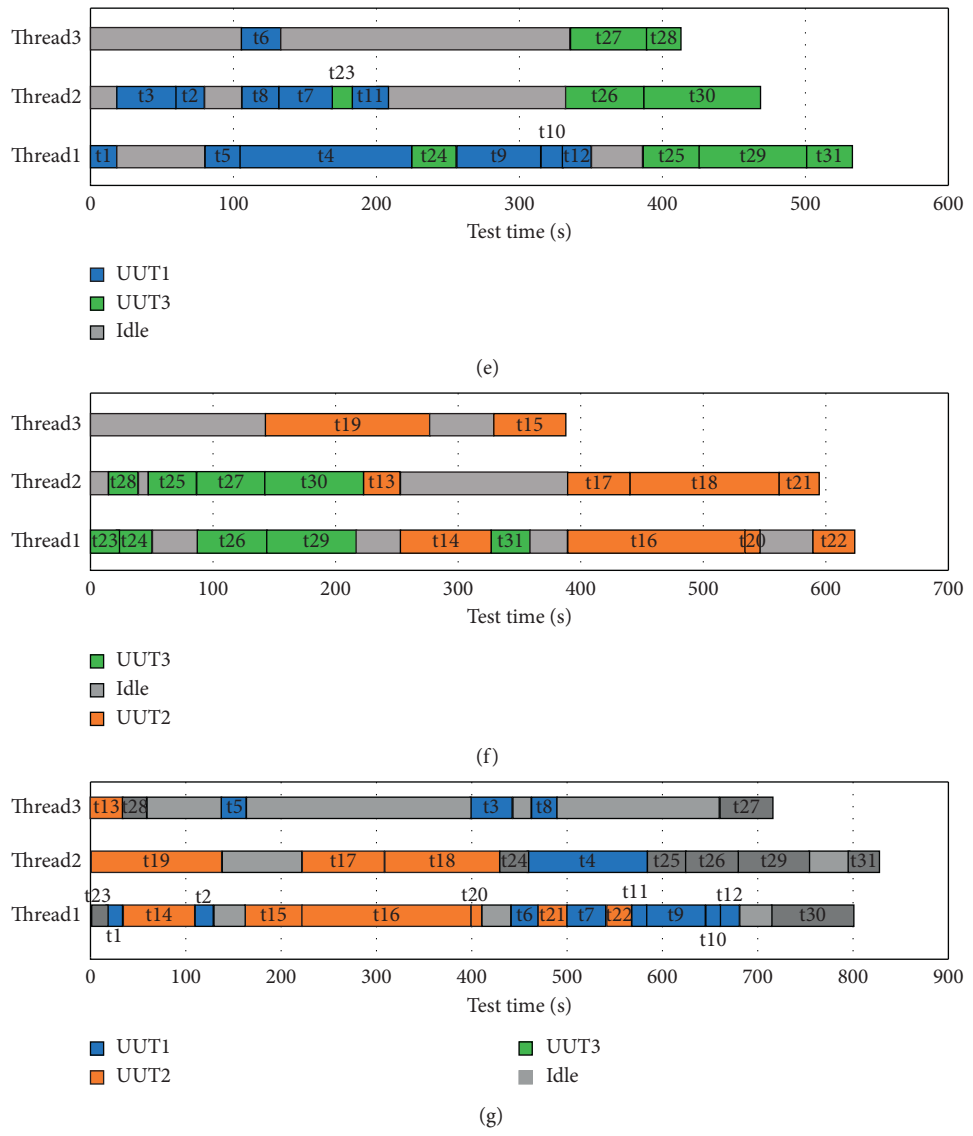

Figure 10: Gantt chart of the best solution. (a) Example 1, (b) example 2, (c) example 3, (d) example 4, (e) example 5, (f) example 6, and (g) example 7.

Table 6: Problem data: task-resource dependency matrix and test time of t_{32}, t_{33}, and t_{34}.

UUT	Test task		Required resources set R										Test time τ (s)
			r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	
UUT_1	T_1	t_{32}	0	0	0	0	1	0	0	0	0	0	23.1
UUT_2	T_2	t_{33}	0	1	0	0	0	1	0	0	0	0	95.4
UUT_3	T_3	t_{34}	0	1	0	1	0	0	0	1	0	0	133.3

the test tasks elaborately according to their technological sequences, which means the whole model must be reoptimized. Therefore, the SBI method will bring great convenience to practical application when adding new test tasks.

5.3.4. SR and ARUR. The SR and ARUR for examples 4-7 are shown in Figure 12. The SR of parallel test for all examples ranges from 1.33 (example 1) to 1.97 (example 4 using 4 threads), which means that the parallel test consumes 50.7% (1/1.97) to 75% (1/1.33) of the time

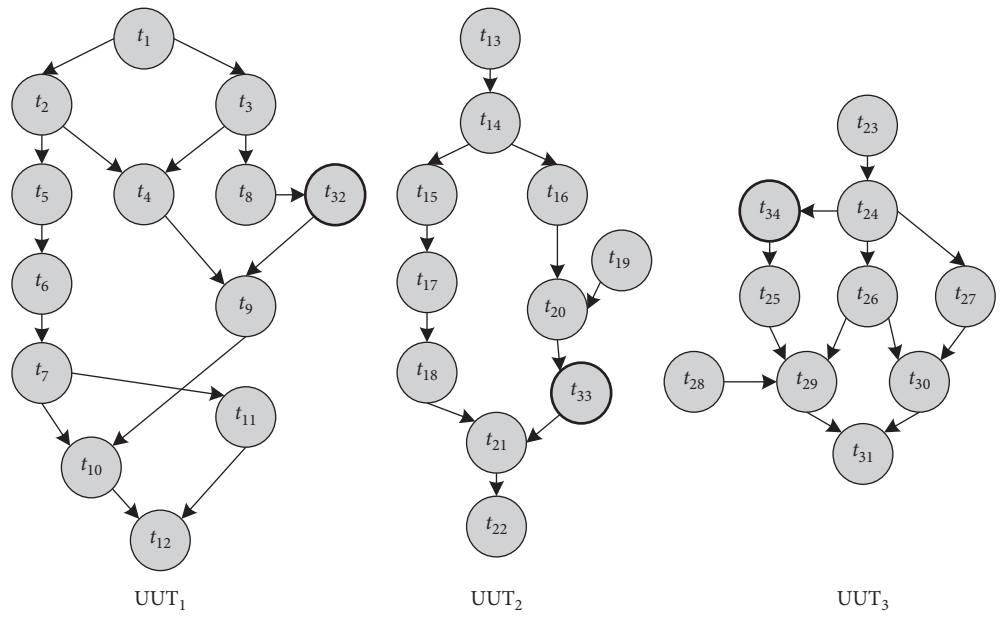

Figure 11: Test flowcharts of UUT1, UUT2, and UUT3 with t_{32}, t_{33}, and t_{34}.

Table 7: Model statistics and solution details of example 7* using 3 threads.

Model	Binary variables	Continuous variables	Constraints	Best solution (s)	Additional computation time (s)	SR	ARUR (%)
M (32)	1088	32	6395	815	5.2	1.93	38.0
M (33)	1155	33	6415	833.4	6.4	1.97	38.1
M (34)	1224	34	7247	853.4	9.1	2.07	47.1

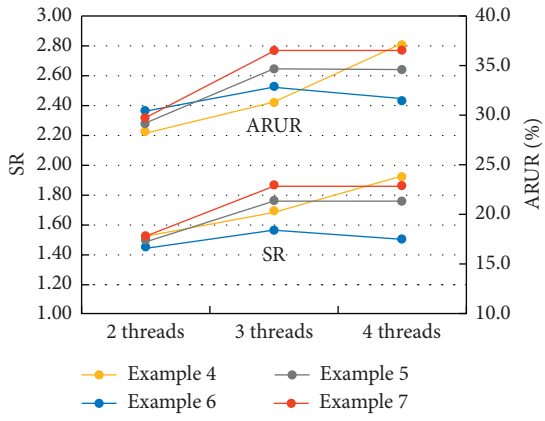

Figure 12: SR and ARUR of examples 4-7.

compared to the sequential test. From the computational results shown in Figure 12, it is concluded that the SR will increase with the ARUR for the same tasks set, which proves that the multi-UUT parallel test trades space for time.

6. Conclusions

In this paper, an MILP optimization model for the scheduling of multiunit parallel test is presented. This model uses the sequence variables to describe the resource competition and the thread competition, which takes the original test flowcharts into consideration. To solve the model in a reasonable amount of time, the proposed model is reformulated and a novel SBI method is proposed. The SBI method is accomplished by the ISF and SBIO procedures. The ISF procedure reduces the search space by fixing FS variables according to the original test flowcharts, and the SBIO procedure can solve the reformulated model in a reasonable amount of time. Moreover, the SBI method is evaluated comprehensively in terms of test duration, SR, and ARUR using different numbers of threads. The computational results prove that the SBI method performs well in real-world test examples, especially for larger examples that cannot be solved by the full-space method. The method presented in this paper can be applied to many task scheduling problems that involve resource competition, such as software test, and agrochemical and pharmaceutical products development. Future research will concentrate on developing a method for task scheduling problems that have high resource requirements for every resource type.

Data Availability

Some or all data and models used during the study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] W. H. Li, Y. P. Wang, and X. H. Wang, "Implementing parallel test in traditional serial framework ATS," in *Proceedings of the International Conference on Electronic Measurement & Instruments*, pp. 143–146, Chengdu, China, August 2011.

[2] C. Heide and R. Hoover, "Optimizing test systems for operational test benefits using parallel test capable instruments," in *Proceedings of the Autotestcon*, pp. 499–503, Salt Lake City, UT, USA, September 2008.

[3] L. Xin, S. T. S, and H. Lu, "Algorithms of tasks scheduling in parallel test based on graph coloring theory," *Journal of Beijing University of Aeronautics & Astronautics*, vol. 33, no. 9, pp. 1068–1071, 2007.

[4] T. Adachi, A. Pramanick, and M. Elston, "Parallel, multi-DUT testing in an open architecture test system," in *Proceedings of the IEEE International Test Conference*, pp. 881–890, Austin, TX, USA, November 2005.

[5] M. Q. Xiao, X. P. Zhu, and R. Xia, "Summary of parallel test technology," *Journal of Air Force Engineering University*, vol. 6, no. 3, pp. 22–25, 2005.

[6] S. K. Sahni, "Algorithms for scheduling independent tasks," *Journal of the ACM*, vol. 23, no. 1, pp. 116–127, 1976.

[7] M. Mastrolilli and O. Svensson, "Hardness of Approximating flow and job shop scheduling problems," *Journal of the ACM*, vol. 58, no. 5, pp. 1–32, 2011.

[8] X. Xiao and J. Q. Wang, "An application of vertex partition for parallel test tasks scheduling in automatic test system," in *Proceedings of the International Conference on Computer Science and Software Engineering*, pp. 723–726, Hubei, China, December 2008.

[9] R. Xia, M. Q. Xiao, J. J. Cheng et al., "Optimizing the multi-UUT parallel test task scheduling based on multi-objective GASA," in *Proceedings of the 8th International Conference on Electronic Measurement and Instruments*, pp. 839–844, Xi'an, China, August 2007.

[10] Z. Q. Li, "Optimization for the parallel test task scheduling based on GA," in *Proceedings of the International Conference on Information Science and Engineering*, pp. 5223–5226, Hangzhou, China, December 2010.

[11] J. A. Grey and D. Elizalde, "Auto-scheduling of tests," US Patent No. 8234089, 2012.

[12] H. Lu, Z. Zhu, X. Wang, and L. Yin, "A variable neighborhood MOEA/D for multiobjective test task scheduling problem," *Mathematical Problems in Engineering*, vol. 2014, Article ID 423621, 14 pages, 2014.

[13] H. Lu, L. Yin, X. Wang et al., "Chaotic multiobjective evolutionary algorithm based on decomposition for test task scheduling problem," *Mathematical Problems in Engineering*, vol. 2014, Article ID 640764, 25 pages, 2014.

[14] G. Q. Zhang, G. Q. Shi, H. G. Zhao, and Y. H. Chen, "A parallel test task scheduling of integrated avionics system based on the ant colony algorithm," *Applied Mechanics and Materials*, vol. 713–715, pp. 2069–2072, 2015.

[15] Y. Qin and X. Liang, "Research on hybrid genetic algorithm for parallel test task scheduling," *Foreign Electronic Measurement Technology*, vol. 9, no. 35, pp. 72–75, 2016.

[16] B. Dorronsoro and F. Pinel, "Combining machine learning and genetic algorithms to solve the independent tasks scheduling problem," in *Proceedings of the 3rd IEEE International Conference on Cybernetics*, pp. 1–8, Exeter, UK, June 2017.

[17] W. Zhang, W. Gao, Z. Wang et al., "Joint optimization on preventive maintenance and operating unit number for multi-unit parallel production system with one repairman and spare unit," in *Proceedings of the 2018 Prognostics and System Health Management Conference (PHM-Chongqing)*, pp. 833–838, Chongqing, China, October 2018.

[18] S. I. Kim and J.-K. Kim, "A method to construct task scheduling algorithms for heterogeneous multi-core systems," *IEEE Access*, vol. 7, pp. 142640–142651, 2019.

[19] O. Koné, C. Artigues, P. Lopez, and M. Mongeau, "Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources," *Flexible Services and Manufacturing Journal*, vol. 25, no. 1-2, pp. 25–47, 2013.

[20] H. Kim and Y. K. Chang, "Mission scheduling optimization of SAR satellite constellation for minimizing system response time," *Aerospace Science and Technology*, vol. 40, pp. 17–32, 2015.

[21] Y. Ye, S. Liang, and Y. Zhu, "A mixed-integer linear programming-based scheduling model for refined-oil shipping," *Computers & Chemical Engineering*, vol. 99, pp. 106–116, 2017.

[22] C. Filippi, R. Mansini, and E. Stevanato, "Mixed integer linear programming models for optimal crop selection," *Computers & Operations Research*, vol. 81, pp. 26–39, 2017.

[23] M. L. Penn, C. N. Potts, and P. R. Harper, "Multiple criteria mixed-integer programming for incorporating multiple factors into the development of master operating theatre timetables," *European Journal of Operational Research*, vol. 262, no. 1, pp. 194–206, 2017.

[24] S. Baruah, V. Bonifaci, G. D'angelo et al., "Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems," *Journal of the ACM*, vol. 62, no. 2, pp. 1–33, 2015.

[25] R. Vujanic, P. Goulart, and M. Morari, "Robust optimization of schedules affected by uncertain events," *Journal of Optimization Theory and Applications*, vol. 171, no. 3, pp. 1033–1054, 2016.

[26] V. Jain and I. E. Grossmann, "Resource-constrained scheduling of tests in new product development," *Industrial & Engineering Chemistry Research*, vol. 38, no. 8, pp. 3013–3026, 1999.

[27] M. Radmanesh and M. Kumar, "Flight formation of UAVs in presence of moving obstacles using fast-dynamic mixed integer linear programming," *Aerospace Science & Technology*, vol. 50, pp. 149–160, 2016.

[28] C. T. Maravelias and I. E. Grossmann, "Optimal resource investment and scheduling of tests for new product development," *Computers & Chemical Engineering*, vol. 28, no. 6-7, pp. 1021–1038, 2004.

[29] C. T. Maravelias and I. E. Grossmann, "Simultaneous planning for new product development and batch manufacturing facilities," *Industrial & Engineering Chemistry Research*, vol. 40, no. 26, pp. 6147–6164, 2001.

[30] C. T. Maravelias and I. E. Grossmann, "Logic Inference and a decomposition algorithm for the resource-constrained scheduling of testing tasks in the development of new pharmaceutical and agrochemical products," *Handbook on Modelling for Discrete Optimization*, Springer, Boston, MA, USA, pp. 265–289, 2006.

[31] C. Artigues, P. Michelon, and S. Reusser, "Insertion techniques for static and dynamic resource-constrained project scheduling," *European Journal of Operational Research*, vol. 149, no. 2, pp. 249–267, 2003.