1-8-2020

MILP Models for Complex System Reliability Redundancy Allocation with Mixed Components

Young-Woong Park
Iowa State University, ywpark@iastate.edu

MILP Models for Complex System Reliability Redundancy Allocation with Mixed Components

Young Woong Park
Ivy College of Business
Iowa State University, Ames, IA, USA
ywpark@iastate.edu

March 10, 2019

Abstract

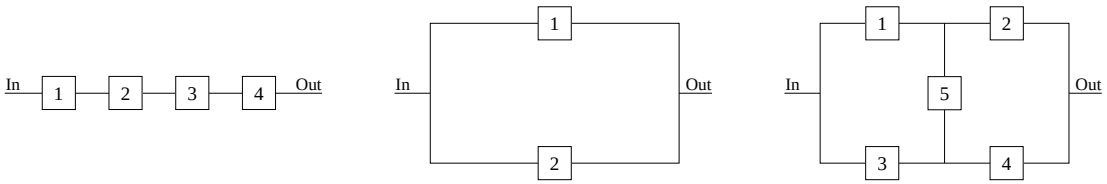
The redundancy allocation problem (RAP) aims to find an optimal allocation of redundant components subject to resource constraints. In this paper, mixed integer linear programming (MILP) models and MILP-based algorithms are proposed for complex system reliability redundancy allocation problem with mixed components, where the system have bridges or interconnecting subsystems and each subsystem can have mixed types of components. Unlike the other algorithms in the literature, the proposed MILP models view the problem from a different point of view and approximate the nonconvex nonlinear system reliability function of a complex system using random samples. The solution to the MILP converges to the optimal solution of the original problem as sample size increases. In addition, data aggregation-based algorithms are proposed to improve the solution time and quality based on the proposed MILP models. A computational experiment shows that the proposed models and algorithms converge to the optimal or best-known solution as sample size increases. The proposed algorithms outperform popular metaheuristic algorithms in the literature.

1 Introduction

In reliability analysis, the redundancy allocation problem (RAP) aims to maximize the system reliability by allocating redundant components subject to resource constraints. The system has n subsystems and each subsystem can have multiple redundant components, which allows a subsystem to function if at least one component is functioning. If a subsystem contains one type of redundant components, the subsystem is called *homogeneous*. If a subsystem is allowed to have multiple types of redundant components, the subsystem is called *heterogeneous*, which is also referred to as a subsystem with *mixed components*. The system is called heterogeneous if there exists at least one heterogeneous subsystem. Heterogeneous systems are also referred to as systems with mixed components. For RAP, one of the fundamental and popular objectives is to maximize the system reliability by placing redundant components in the subsystems while satisfying resource constraints for m resources.

For both homogeneous and heterogeneous systems, system types are defined based on the system structure or connections between the subsystems. In Figure 1, examples for three system types are presented. In a series system such as the system in Figure 1(a), subsystems are connected in a series, and failure of one subsystem makes the entire system fail. In a parallel system such as the system in Figure 1(b), subsystems are located in parallel, and the entire system fails only if all subsystems fail. When each subsystem of a series system contains a parallel system, the system is called *series–parallel*. When each subsystem of a parallel system contains series system, the system is called *parallel–series*. Finally, a system is referred to as complex when the system is more complicated than the aforementioned systems. For example, in Figure 1(c), the system has a subsystem connecting the parallel series subsystems, which is called a bridge.

RAP has been extensively studied for decades since being introduced in the 1950s by Bellman and Dreyfus [5] and Gordon [14]. It is getting increasing attention in recent years as well due to increasing level of sophistication in high-tech industrial processes [20]. Since the 1950s, many publications have studied RAP using different system types and structures, performance measures, and optimization algorithms. In terms of performance measures, the most popular measure is maximization of system reliability given a limited amount of resources, while other popular performance measures include minimization of cost given a lower bound for reliability [12, 13], maximization of percentile life of system [10, 16, 29], multi-objective problems



(a) Series system (b) Parallel system (c) Complex (bridge) system

Figure 1: System types

simultaneously optimizing reliability and cost [8, 9]. Because of their higher flexibility than homogeneous systems, recent studies focus on heterogeneous systems for various settings such as reliability maximization for series–parallel systems [22] or series–parallel multistate systems [36, 23], cost minimization for series–parallel systems [32]. Out of all the variations, we study heterogeneous systems to maximize the system reliability given a limited amount of resources. For recent surveys of the broader literature, the reader is referred to Kuo and Wan [20] and Lad *et al.* [21].

In terms of solution methodologies and algorithms for RAP, many studies focus on heuristic or metaheuristics such as genetic algorithms (GA) [3, 11, 35], tabu search [18, 19, 30], particle swarm optimization (PSO) [37], and ant colony algorithms [24] due to the complex structure of the problem. Even for the simplest system setting, a homogeneous series system, RAP is $\mathcal{NP}$-hard [7] and is difficult to solve as the objective function is nonconvex and nonlinear. Hence, only a limited number of exact algorithms have been proposed. Ha and Kuo [15] proposed a branch-and-bound algorithm based on nonconvex integer linear programming formulation for complex homogeneous systems. Caserta and Voß[6] proposed MILP for heterogeneous series systems by transforming the original problem into a multiple-choice knapsack problem. Sung and Cho [33, 34] proposed an exact algorithm for series systems with multiple choice constraints, where multiple choice constraints imply heterogeneity.

In this paper, complex systems with mixed components (heterogeneous) subject to linear resource constraints are studied. The system contains bridges, such as Figure 1(c), or interconnected subsystems, and each subsystem can have mixed types of components. For this complex system with mixed components, MILP models and algorithms are proposed based on random samples. To our knowledge, our model is the first MILP model for complex systems approximating the complex system reliability, while other studies employ metaheuristics or work directly with nonconvex nonlinear system reliability to solve the problem. Data aggregation-based algorithms are also proposed using the MILP models and adopting the Aggregate and Iterative Disaggregate (AID) algorithm framework developed by Park and Klabjan [28]. AID is an optimization algorithmic framework guaranteeing optimality by iteratively solving the problem with aggregated data. The algorithm is effective when the data size is large and has been successfully applied to select machine learning problems minimizing absolute fitting errors such as least absolute deviation regression, support vector machine (SVM) [28], and regression subset selection [27]. While the AID algorithm proposed by Park and Klabjan [28] can be used for non-machine learning problems following certain structures, tailored algorithmic design is required for a new problem. There are also iterative data aggregation-based algorithms for SVM [38, 39]. For a review of other non-iterative aggregation-based algorithms and non-sample related aggregation-based algorithms, the reader is referred to Park and Klabjan [27]. In all of the previous iterative data aggregation-based algorithms in the literature, averaging the samples (observations or records) plays a key role in data aggregation. The aggregated data is typically obtained by summing or averaging the samples in the clusters. On the other hand, our aggregation procedure is based on the minimum vector of the samples in each cluster, which is a new and distinguishing feature of our AID algorithm. Finally, the proposed algorithms are compared with two popular metaheuristic algorithms, GA and PSO, where the implementations are based on Ardakan *et al.* [3] and Wang and Li [37].

Our contributions can be summarized as follows.

1. For the first time in the literature, MILP models are proposed for complex systems to approximate nonconvex and nonlinear system reliability. While MILP for heterogeneous series–parallel systems [6] and branch-and-bound algorithm for homogeneous complex systems [15] show excellent performances,

no exact or guaranteed convergent algorithm is available for complex heterogeneous systems. Our MILP models view the problem from a completely different angle and can easily incorporate new constraints. Hence, our MILP models have the potential to be used for other complex network reliability analyses.

2. AID algorithm is proposed to improve the solution quality and speed of the MILP model. By developing problem-specific design of the AID algorithm for the MILP model, the convergence of the algorithm is shown. Unlike the other AID or iterative data aggregation algorithms in the literature, minimum vectors are used to aggregate the data instead of average vectors. Our AID algorithms converge to an optimal solution of the proposed MILP, while it is also capable of checking the solution quality with respect to the original objective function in each iteration.
3. While the proposed models and algorithms are not exact, the computational experiment shows the convergence to optimum as sample size increases and the gaps from the optimal solution are small. Among all proposed algorithms, running AID multiple times with small-size sample sets, referred to as $\mathsf{AID}_{\mathsf{rep}}$, shows the best in terms of solution speed and quality. Further, the proposed algorithms outperform the benchmark metaheuristic algorithms.

The paper is structured as follows. In Section 2, the notation and mathematical formulation of RAP for the complex system are introduced and MILP models based on random samples are proposed. In Section 3, AID algorithms based on the MILP models are proposed. In Section 4, computational experiments are presented.

2 MILP Models

In this section, MILP models are developed for heterogeneous complex systems based on sampling and graph representation. In Section 2.1, the notations and mathematical formulation are first introduced for the reliability redundancy allocation problem of the complex systems. Next, in Sections 2.2–2.4, MILP models are developed for the simpler version, homogeneous complex system, as the derivations are more straightforward. Finally, the models are extended to complex heterogeneous systems in Section 2.5.

2.1 Reliability redundancy allocation problem for complex systems

In this section, mathematical formulations are introduced for RAP of complex systems with a few illustrative examples. Through this section and the rest of the paper, the notations summarized in Table 1 are used. Among the notations introduced, the decision variables are x_{jh} and x, and functions $g_i(x)$, $f(x)$, R_j, and Q_j depend on x. All other notations are problem parameters and sets.

In Sections 2.2 - 2.4, homogeneous systems are considered to develop MILPs where x_{jh}, r_{jh}, u_{jh}, and a_{ijh} are defined only for $h = 1$. Hence, for notational convenience, let us drop h and use x_j, r_j, u_j, and a_{ij} in the corresponding sections for homogeneous systems.

For heterogeneous systems, the reliability redundancy allocation problem is written as

$$\max\{f(x)|g_i(x) \le b_i, i \in I; \sum_{h\in H_j} x_{jh} \ge 1, j \in J; 0 \le x_{jh} \le u_{jh}, h \in H_j, j \in J\}, \tag{1}$$

where x_{jh} is bounded above by u_{jh}, $h \in H_j$, $j \in J$. Function f is defined with component reliability r_{jh} for $h \in H_j$, $j \in J$, and each subsystem must have at least one component ($\sum_{h\in H} x_{jh} \ge 1$, $j \in J$).

In Problem (1), the system reliability function f depends on the configuration and structure of the system. There are also variations in resource function g_i, $i \in I$ because g_i can be a nonlinear function, a linear function, or more complex functions. In this study, resource function g_i is assumed to be linear. Hence, (1) can be written as

$$\max\{f(x)|A_i^\top x \le b_i, i \in I; \sum_{h\in H_j} x_{jh} \ge 1, j \in J; 0 \le x_{jh} \le u_{jh}, h \in H_j, j \in J\}. \tag{2}$$

Note that, due to the resource constraint, there is an implied upper bound for x_{jh}, $\min_{i\in I}\lfloor \frac{b_i}{a_{ijh}} \rfloor$. Hence, the user provided upper bound u_{jh} may be updated by $u_{jh} = \min\left\{u_{jh}, \min_{i\in I}\lfloor \frac{b_i}{a_{ijh}} \rfloor\right\}$ for all $h \in H_j$ and $j \in J$. In our implementation, u_{jh} is strengthened further. Because each system requires at least one component placed, u_{jh} is updated by $u_{jh} = \min\left\{u_{jh}, \min_{i\in I}\lfloor \frac{b_i - \sum_{j'\in J\setminus\{j\}} a_{ij'}^{\min}}{a_{ijh}} \rfloor\right\}$ for all $h \in H_j$ and $j \in J$, where

Notation	Description
n	number of subsystems in the system
J	set of subsystems, $J = \{1, 2, \cdots, n\}$
m	number of resources
I	set of resources, $I = \{1, 2, \cdots, m\}$
n_j^{H}	number of heterogeneous component types at subsystem $j \in J$
n^{H}	number of heterogeneous component types for each subsystem, when $n_j^{\mathrm{H}} = n^{\mathrm{H}}$ for all $j \in J$
H_j	set of heterogeneous component types at subsystem $j \in J$, where $H_j = \{1, 2, \cdots, n_j^{\mathrm{H}}\}$
x_{jh}	number of type $h \in H_j$ components at subsystem $j \in J$
r_{jh}	component reliability of type $h \in H_j$ at subsystem $j \in J$
x	solution matrix of x_{jh}, $h \in H_j$, $j \in J$
$g_i(x)$	amount of resource i required for x, $i \in I$
b_i	amount of resource i available, $i \in I$
$f(x)$	system reliability for solution x
a_{ijh}	amount of resource i required for a type h component at subsystem j, $i \in I$, $h \in H_j$, $j \in J$
R_j	reliability of subsystem j for solution x $(=R_j(x))$
Q_j	$= 1 - R_j$
A_i	matrix of a_{ijh}, $h \in H_j$, $j \in J$ for resource $i \in I$
b	vector of b_i, $i \in I$
u_{jh}	upper bound of x_{jh}, $h \in H_j$, $j \in J$

Table 1: List of notation

$a_{ij'}^{\min} = \min_{h \in H_{j'}} \{a_{ij'h}\}$ for $j' \in J$. This formula decreases the available amount of resource $i \in I$ by assuming the minimum possible usage of resource i to have one component for each subsystem except for $j \in J$.

In Figure 2, four example complex systems are presented to illustrate how f is defined. The example systems either have bridges or interconnecting subsystems. Subsystem $j \in J$ has n_j^{H} available types of components and can use any combination of the components. Each subsystem fails only if all of the components in the subsystem fail. Hence, reliability of subsystem j given solution x is defined as $R_j(x) = 1 - \prod_{h \in H_j} (1 - r_{jh})^{x_{jh}}$, and complement $Q_j(x) = 1 - R_j(x)$, $j \in J$, is also defined. For notational convenience, let us drop x from the notation and use R_j and Q_j instead of $R_j(x)$ and $Q_j(x)$, respectively, in the rest of the paper. The four examples in Figure 2 have the following system reliabilities given solution x.

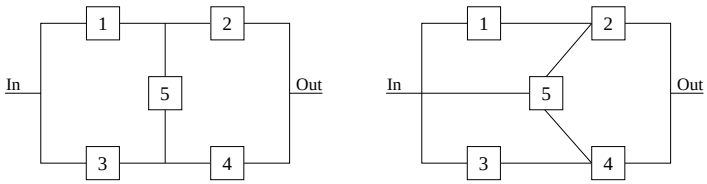
$$\begin{aligned}
&\text{System 1:} \quad f(x) = \quad R_5(1 - Q_1Q_3)(1 - Q_2Q_4) + Q_5(1 - (1 - R_1R_2)(1 - R_3R_4)) \\
&\text{System 2:} \quad f(x) = \quad R_5(1 - Q_2Q_4) + Q_5(1 - (1 - R_1R_2)(1 - R_3R_4)) \\
&\text{System 3:} \quad f(x) = \quad R_6(1 - Q_1Q_3Q_4)(1 - Q_2Q_5) + Q_6(1 - (1 - R_1R_2)(1 - (1 - Q_3Q_4)R_5)) \\
&\text{System 4:} \quad f(x) = \quad Q_7[1 - (1 - R_1R_2)(1 - R_{345}R_6)] + R_7(1 - Q_1Q_{345})(1 - Q_2Q_6)
\end{aligned}$$

For System 4, R_{345} is the reliability of Subsystems 3, 4, and 5, where $R_{345} = 1 - Q_{345}$ and $Q_{345} = Q_3 + R_3(Q_4 + Q_5 - Q_4Q_5)$.

Note that it is difficult to define a generalized $f(x)$ for a complex system. For example, $f(x)$ for Systems 1–4 in Figure 2 cannot be generalized. On the other hand, for series systems and parallel systems such as Figures 1(a) and 1(b), $f(x) = \prod_{j \in J} R_j$ and $f(x) = 1 - \prod_{j \in J} Q_j$ can be defined, respectively, for a different set J. Further, the multiplication part of $f(x)$ can be written in linear form by log transformation. For example, $\ln(\prod_{j \in J} R_j) = \sum_{j \in J} \ln(R_j)$ and $\ln(\prod_{j \in J} Q_j) = \sum_{j \in J} \ln(Q_j)$ for series systems and parallel systems. This linearization technique is one of the fundamental principles in formulating mathematical programming models for series and parallel systems [6]. Because there is no closed form $f(x)$ definition for a complex system and the $f(x)$ is complicated, it is not trivial to modify the objective function, which is nonconvex and nonlinear. To overcome this difficulty, random samples are used to formulate MILP models in Sections 2.2–2.4 to approximate the system reliability.

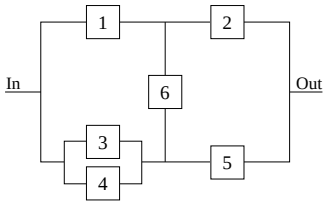
2.2 Graph representation and cuts

In this section, a procedure to represent the reliability network as a graph is presented. All reliability networks can be converted into various forms of graphs and cuts can be used to analyze the system [17, 25, 26, 31]. Early works in 1970s [17, 25, 26] used graphs to analyze complex systems based on minimal paths minimal

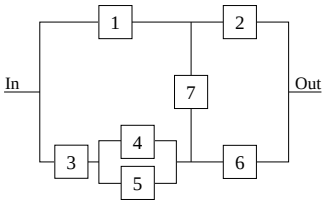


(a) System 1

(b) System 2



(c) System 3



(d) System 4

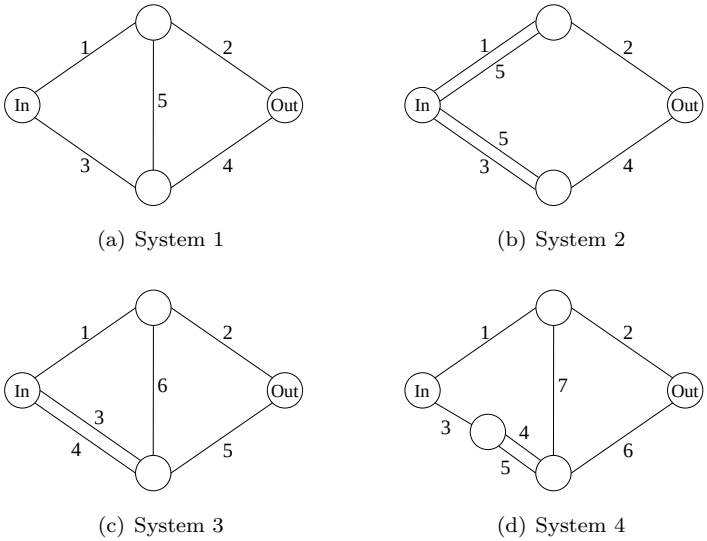
Figure 2: Complex (bridge or interconnected) network examples: (a) five subsystems including one bridge, (b) five subsystems including one interconnecting subsystem, (c) six subsystems with two series–parallel subsystems including one bridge, (d) seven subsystems with two series–parallel subsystems including one bridge

cuts enumeration and a genetic algorithm was proposed later by Sallak *et al.* [31] based on the graph representations. While many previous works use directed graphs, we focus on undirected graphs due to their simplicity while achieving the same goal. First, a reliability system network is converted into an undirected graph with a source (in) and a sink (out). In this new undirected graph, subsystem j becomes an edge and the nodes tie multiple edges (subsystems) together. When the subsystems are parallel, multiple edges connect the same two nodes in the new undirected graph.

The four reliability system networks in Figure 2 are converted into undirected graphs in Figure 3. Each subsystem in Figure 2 is an edge in Figure 3 and the numbers next to the edges in Figure 3 are subsystem numbers of the corresponding system. The nodes are created to represent the connections between subsystems. When subsystems are parallel, the corresponding edges connect the same two nodes as presented in Figure 3(c). Note also that a subsystem in the reliability network can be converted into multiple edges as illustrated in Figure 3(b). Subsystem 5 of System 2 in Figure 2(b) has connections to two subsystems and two edges are created for System 2 in Figure 3(b).

Observe that, in the example systems in Figure 2, there exists a set of subsystems such that if all subsystems in the set fail, then the entire system fails. In System 1, failure of Subsystems 1 and 3 makes the entire system fail. Similarly, failures of Subsystems 1, 4, and 5 make the entire system fail. The undirected graphs in Figure 3 can be used to formulate this property. In graph theory, a *cut* is a partition of the nodes of a graph into two disjoint subsets defined by a set of edges, and an *s–t cut* is a cut that partitions the nodes into two disjoint subsets with source and sink nodes in different subsets. For example, edges $\{2, 3, 5\}$ in Figure 3(a) forms an s–t cut because eliminating the edges in the cut partitions the nodes into two disjoint subsets of the nodes. An s–t cut is a *minimal s–t cut* if any proper subset of the cut does not form an s–t cut. For example, edges $\{1, 3, 5\}$ in Figure 3(a) is an s–t cut, but not a minimal s–t cut because a subset $\{1, 3\}$ of the cut forms an s–t cut.

Recall that the graphs in Figure 3 correspond to the networks in Figure 2 and consider an s–t cut {1,4,5} in Figure 3(a) for System 1 in Figure 2(a). If Subsystems 1, 4, and 5 fail, then the system fails. In fact, for any s–t cut, if all of the corresponding subsystems fail, then the system fails. Hence, by generating minimal s–t cuts, the minimal set of cases where the failure of the subsystems leads to a failure of the entire system can be obtained. To generate all minimal s–t cuts, the algorithms of Abel and Bicker [1] or Arunkumar and Lee [4] can be used. Both of the algorithms are designed to output all possible minimal cuts of an undirected graph. Alternatively, especially for a small graph, a simple enumeration approach also works. For each proper subset C of the edge set of the graph, test if C is a minimal s–t cut. If yes, then output C. Otherwise, discard C. The minimal s–t cuts for the four systems in Figure 3 are listed below.



(a) System 1 (b) System 2

(c) System 3 (d) System 4

Figure 3: Graph representation of complex (bridge or interconnected) system examples

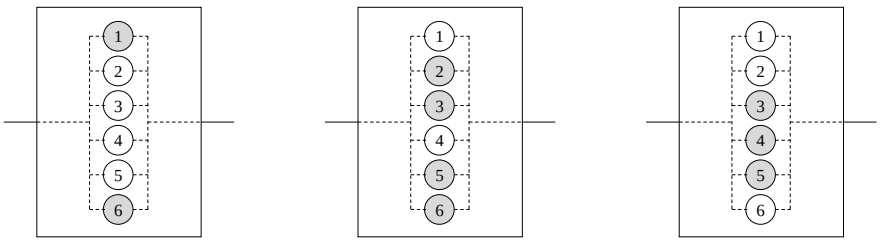
System 1: $\{1,3\}$, $\{2,4\}$, $\{1,4,5\}$, and $\{2,3,5\}$
System 2: $\{1,3,5\}$, $\{2,4\}$, $\{1,4,5\}$, and $\{2,3,5\}$
System 3: $\{2,5\}$, $\{1,5,6\}$, $\{1,3,4\}$, and $\{2,3,4,6\}$
System 4: $\{1,3\}$, $\{2,6\}$, $\{1,4,5\}$, $\{1,6,7\}$, $\{2,3,7\}$, and $\{2,4,5,7\}$

Observe that this result can also be used for series and parallel systems. For the series system with four subsystems in Figure 1(a), four s–t cuts are available: $\{1\}, \{2\}, \{3\}$, and $\{4\}$. For the parallel system with two subsystems in Figure 1(b), only one s-t cut $\{1,2\}$ is available.

2.3 Random sampling

In this section, a sampling (scenario) approach is proposed to approximate system reliability based on the graph representation presented in Section 2.2. Many samples representing possible scenarios are first generated, then the status of the system (success/fail) is determined for each sample given solution x. The overall reliability can be approximated by the percentage of the samples with working status.

First, the core concept of the sampling procedure is discussed with an illustrative example in Figure 4. Let us consider a subsystem with $u_j = 6$. When a sample is generated, the status realizations (success/fail) are generated for all six possible components, regardless of our choice of usage x_j.



(a) Sample 1 (b) Sample 2 (c) Sample 3

Figure 4: Illustration of sampling for subsystem with $u_j = 6$

In Figures 4(a)–4(c), three samples are generated and each sample has six components with a mixed status of success and failure. The large squares represent subsystem j and circles represent redundant components (of one type), where up to six redundant components can be used. Gray and white circles indicate working and

failing conditions, respectively, of the components. Because the components are redundant, they are located in parallel.

In our sampling approach, we assume that the sampled components with lower indices must be used first. In other words, if one component is used, Component 1 (circle with label 1) must be used. If two components are used, Components 1 and 2 (circles with labels 1 and 2) must be used. With this assumption, the status of the subsystem can be determined for x_j by checking if there is any working component with index $\leq x_j$. For example, if $x_j = 1$, Component 1 must be used. Hence, with $x_j = 1$, the subsystem is working for Sample 1, whereas the subsystem fails for Samples 2 and 3. Similarly, with $x_j = 2$, the subsystem is working for Samples 1 and 2, whereas the subsystem fails for Sample 3. After a simple analysis, we can conclude that $x_j = 1$ makes the subsystem work 33.3% of the time (one out of three samples), $x_j = 2$ makes the subsystem work 66.7% of the time (two out of three samples), and $x_j \geq 3$ makes the subsystem work 100% of the time. This observation will be the basis of our MILP formulation later.

In the previous example, three samples are generated for one subsystem. Let us next discuss the generation of multiple samples for the entire system. Let n^{s} be the number of samples and $S = \{1, 2, \cdots, n^{\mathrm{s}}\}$ be the set of samples. As previously assumed, all redundant components are tested across all subsystems, regardless of the number of components used in x, and a sample includes success (denoted by 1) or failure (denoted by 0) realization of each component for all possible $\sum_{j \in J} u_j$ components in the system. The sample generation algorithm is presented in Algorithm 1.

Algorithm 1 Sample Generation

Input: n^{s} (number of samples)
Output: q (samples generated)
1: **for** $s \in S = \{1, 2, \cdots, n^{\mathrm{s}}\}$
2: **for** $j \in J$
3: **for** $e \in \{1, \cdots, u_j\}$
4: Generate a uniform random number $\gamma \in [0, 1]$
5: **if** $\gamma \leq r_j$ **then** $v_e = 1$
6: **if** $\gamma > r_j$ **then** $v_e = 0$
7: **end for**
8: **if** $\sum_{e \in \{1, \cdots, u_j\}} v_e = 0$ **then** $q_{sj} = u_j + 1$
9: **else** $q_{sj} = \min \left\{ e \in \{1, \cdots, u_j\} | v_e = 1, e \in \{1, \cdots, u_j\} \right\}$
10: **end for**
11: **end for**

In Algorithm 1, sample vector $q_s = \{q_{s1}, q_{s2}, \cdots, q_{sn}\}$ is generated for all s in sample set S. Let us consider generation of q_{sj} for subsystem $j \in J$ for sample $s \in S$. In Lines 3–7, all possible u_j redundant components are tested to determine the status (success or fail). If the uniform random number is less than or equal to r_j, then the e^{th} redundant component is working. The success/fail information of all components is stored in a temporary vector v. Then, in Lines 8–9, the minimum number of redundant components is determined to have at least one working component for sample s and stored as q_{sj}. In words, q_{sj} is the index of the first redundant component with working condition ($v_i = 1$). Note that $\sum_{e \in \{1, \cdots, u_j\}} v_e = 0$ in Line 8 implies that all components failed for this sample. In this case, q_{sj} is set to $u_j + 1$, which implies that the subsystem will fail regardless of x_j decision because x_j cannot exceed u_j. Note also that as soon as the first e with $v_e = 1$ is obtained in Line 5, the for loop in Lines 3–7 can be stopped and q_{sj} Line 9 can be determined because q_{sj} is set by the first e with $v_e = 1$. Hence, the algorithm can have improved running time.

Example. Consider subsystem j with $u_j = 4$. Four redundant components are tested whether each component works by comparing random number γ and reliability r_j. Suppose we obtain $v = (0, 1, 1, 0)$. This means that the second and third redundant components work while the first and fourth components fail. By Line 8 of Algorithm 1, $q_{sj} = 2$ is obtained because 1 appears for the first time at index 2 of v. Now, if $x_j = 1$, then we must use the first redundant component and $q_{sj} = 2$ implies subsystem j fails for sample s. On the other hand, if $x_j \geq 2$, then subsystem j works for sample s. By repeating this procedure for all subsystems, a complete sample q_s is generated. Suppose $q_s = (1, 2, 3, 2, 1)$. Then, for a solution $x = (2, 2, 1, 1, 1)$, Subsystems 3 and 4 fail because $q_{s3} = 3 > 1 = x_3$ and $q_{s3} = 2 > 1 = x_3$. For a different solution $x = (1, 2, 3, 2, 2)$, all subsystems

are working for sample s.

2.4 MILP models

Based on the cuts and random samples presented in the previous sections, MILP models are formulated in this section. Multiple MILP models are proposed for easier derivations and explanations, but only the last model is recommended.

First, consider the status of subsystem $j \in J$ for sample $s \in S$. Let z_{sj} be a binary variable to indicate the status of subsystem $j \in J$ for sample $s \in S$, where $z_{sj} = 1$ indicates working or success and $z_{sj} = 0$ indicates failure. Constraint

$$x_j \geq q_{sj} z_{sj}$$

correctly defines z_{sj} for $j \in J$ and $s \in S$ because $z_{sj} = 1$ is possible only if $x_j \geq q_{sj}$ and z_{sj} will be implicitly maximized by the objective function presented later.

Next, consider the status of the overall system. Let n^{C} be the number of cuts in the system and $C = \{1, 2, \cdots, n^{\mathrm{C}}\}$ be the set of cuts. For cut c, let J_c be the set of subsystems in cut $c \in C$. For example, System 1 in Figures 2 and 3 has $C = \{1, 2, 3, 4\}$, $J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{1, 4, 5\}$, and $J_5 = \{2, 3, 5\}$. Let us define binary variable y_s for $s \in S$ as follows.

$$y_s = \begin{cases} 1 & \text{if system works for sample } s, \\ 0 & \text{otherwise,} \end{cases} \qquad s \in S$$

For sample $s \in S$ and cut $c \in C$, constraint

$$\sum_{j \in J_c} z_{sj} \geq y_s$$

checks if the system fails due to cut c for sample s. Note that the system works for sample s only if $\sum_{j \in J_c} z_{js} \geq 1$ for all cuts in C.

Finally, the system reliability R_S is defined as $R_S = \frac{\sum_{s \in S} y_s}{n^{\mathrm{S}}}$. Because maximizing R_S and $\sum_{s \in S} y_s$ are equivalent given a sample set S, the following MILP for (2) is formulated.

$$\begin{array}{lll} \max & \displaystyle\sum_{s \in S} y_s & \\ \text{s.t.} & \displaystyle\sum_{j \in J} a_{ij} x_j \leq b_i, & i \in I, \\ & \displaystyle\sum_{j \in J_c} z_{sj} \geq y_s, & c \in C, s \in S, \\ & x_j \geq q_{sj} z_{sj}, & j \in J, s \in S, \\ & 1 \leq x_j \leq u_j,\ x_j \in \mathbb{Z}, & j \in J, \\ & z_{sj} \in \{0, 1\}, & j \in J, s \in S, \\ & y_s \in \{0, 1\}, & s \in S \end{array} \tag{3}$$

Proposition 1. Let $(\bar{x}, \bar{z}, \bar{y})$ be an optimal solution for (3). As sample size n^{S} increases, $\frac{\sum_{s \in S} \bar{y}_s}{n^{\mathrm{S}}}$ converges to $f(\bar{x})$.

Observe that each sample $s \in S$ is independent and identically distributed and $\frac{\sum_{s \in S} \bar{y}_s}{n^{\mathrm{S}}}$ is the sample average estimating the probability of working given $\bar{x}$. Because $f(\bar{x})$ is also the expected system reliability given $\bar{x}$, by the law of large numbers, $\frac{\sum_{s \in S} y_s}{|S|}$ converges to $f(\bar{x})$ as sample size n^{S} increases. Note that the size of the binary variable matrix z is $n \times n^{\mathrm{S}}$. Hence, the size of MILP (3) can be very large even for a system with several subsystems and samples. For example, with 10 subsystems and 10,000 samples, the number of binary variables in z is 100,000.

Improved model

Note that identical samples can be generated multiple times. The number of binary variables can be decreased by eliminating the duplicate samples and giving weights based on the number of duplicate samples. Let us define a new sample set $\bar{S}$ containing nonduplicate qs. Mathematically, $\bar{S} = \{s \subseteq S | q_s \neq q_{s'}, s' \in S \setminus \{s\}\}$. Let w_s, $s \in \bar{S}$, be the number of samples in S such that $q_s = q_t$, $t \in S$, and let $\bar{n}_s = |\bar{S}|$. Then, a weighted version of (3) can be formulated.

$$
\begin{aligned}
\max \quad & \sum_{s \in \bar{S}} w_s y_s & & \\
\text{s.t.} \quad & \sum_{j \in J} a_{ij} x_j \leq b_i, & & i \in I, \\
& \sum_{j \in J_c} z_{sj} \geq y_s, & & c \in C, s \in \bar{S}, \\
& x_j \geq q_{sj} z_{sj}, & & j \in J, s \in \bar{S}, \\
& 1 \leq x_j \leq u_j,\ x_j \in \mathbb{Z}, & & j \in J, \\
& 0 \leq y_s \leq 1, & & s \in \bar{S}, \\
& z_{sj} \in \{0, 1\}, & & j \in J, s \in \bar{S}
\end{aligned}
\tag{4}
$$

Note that, in (4), binary variables y_s are relaxed to be continuous. Because the left hand side of $\sum_{j \in J_c} z_{sj} \geq y_s$ is an integer and the objective function is maximizing y_s, an optimal solution must have an integer value for y_s. The relative cardinality of $|\bar{S}|$ compared with $|S|$ depends on the number of component types. If there is a small number of subsystems or heterogeneous components, $\frac{|\bar{S}|}{|S|}$ can be very small. On the other hand, if the system has many subsystems and heterogeneous components, then $\frac{|\bar{S}|}{|S|}$ can be almost equal to 1 as it is less likely to have duplicate samples. In this case, (4) becomes less efficient.

Final model

Note that the number of binary variables in (4) is $(n+1) \cdot \bar{n}_s$, which depends on the number of (nonduplicate) samples. When $n^{\mathrm{S}} \approx \bar{n}^{\mathrm{S}} = 16,000$, for System 1 with $n^{\mathrm{H}} = 4$, the number of binary variables z_{sj} will be approximately 78,000. Hence, solving (4) becomes intractable even for this small system with five subsystems. To resolve this issue, an improved MILP is proposed by eliminating binary variables defined for samples.

Let $K_j = \{0, 1, 2, \cdots, u_j\}$ be the set of possible number of components at subsystem j. In addition to integer variables x_j, let us define binary variables to represent the number of components in the subsystems.

$$
x^b_{jk} = \begin{cases} 1 & \text{if } x_j = k, \\ 0 & \text{otherwise,} \end{cases} \quad k \in K_j, j \in J
$$

When these binary variables are used, exactly one of x^b_{jk}, $k \in K_j$, must be 1 for subsystem $j \in J$. Hence, assignment constraints $\sum_{k \in K_j} x^b_{jk} = 1, j \in J$, must be included.

Next, consider a sample $s \in \bar{S}$ for subsystem $j \in J$. If the number of components at subsystem j is greater than q_{sj}, then subsystem j is working for sample s. Let p_{sjk} indicate whether subsystem j is working when there are k redundant components for sample s. If $p_{sjk} = 1$, then subsystem j is working when there are k redundant components for sample s. If $p_{sjk} = 0$, then subsystem j is not working. Hence, p_{sjk} is defined based on q_{sj}.

$$
p_{sjk} = \begin{cases} 1 & \text{if } k \geq q_{sj}, \\ 0 & \text{otherwise,} \end{cases} \quad s \in \bar{S}, k \in K_j, j \in J
$$

With the new variables and parameters x^b_{jk} and p_{sjk}, binary variables z_{sj} in (4) can be removed. In detail, constraint

$$
\textstyle\sum_{k \in K_j} p_{sjk} \cdot x^b_{jk}
$$

replaces z_{sj} for $j \in J$ and $s \in \bar{S}$. The value of this equation will be either 0 or 1. The equation becomes 1 only if $x_j = k$ and $p_{sjk} = 1$. Otherwise, when $x_j = k$ and $p_{sjk} = 0$, the equation becomes 0. Finally, the following

MILP is proposed.

$$\max \quad \sum_{s\in\bar{S}} w_s y_s \tag{5a}$$

$$s.t. \quad \sum_{j\in J} a_{ij}x_j \le b_i, \qquad i \in I, \tag{5b}$$

$$\sum_{k\in K_j} x^b_{jk} = 1, \qquad j \in J, \tag{5c}$$

$$\sum_{j\in J_c}\sum_{k\in K_j} p_{sjk}\cdot x^b_{jk} \ge y_s, \qquad c \in C, s \in \bar{S}, \tag{5d}$$

$$x_j = \sum_{k\in K_j} k\cdot x^b_{jk}, \qquad j \in J, \tag{5e}$$

$$x^b_{jk} \in \{0,1\}, \qquad k \in K_j, j \in J, \tag{5f}$$

$$0 \le y_s \le 1, \qquad s \in \bar{S}, \tag{5g}$$

$$1 \le x_j \le u_j, \qquad j \in J \tag{5h}$$

Constraint (5b) is a resource constraint, and Constraint (5c) forces exactly one of $x^b_{jk}, k \in K_j$ to be one to define x_j correctly in (5e). Constraint (5d) defines y_s to check if the system is working for sample $s \in S$. Because of the definition of x_j in (5e), an integer requirement for $x_j \in \mathbb{Z}$ in (5h) is not needed.

Note that the number of binary variables, $n \cdot \left(\sum_{j\in J}(u_j+1)\right)$, does not depend on the number of samples. Although (5) requires more binary variables for the number of components used compared with (4), it can be more efficient than the previous MILP models (3) and (4) when n^{S} (or $\bar{n}^{\mathrm{S}}$) is large. This is because the number of constraints for $\sum_{j\in J_c} z_{sj} \ge y_s$ and $x_j \ge q_{sj}z_{sj}$ in (3) is $(n+n^{\mathrm{C}})\cdot n^{\mathrm{S}}$, whereas the number of corresponding constraints (5d) of (5) is $n^{\mathrm{C}}\cdot n^{\mathrm{S}}$. As n^{S} increases, the difference in the number of constraints becomes large.

2.5 MILP for Heterogeneous Systems

Through Sections 2.2–2.4, MILP model (5) is developed to solve RAP for complex homogeneous systems. In this section, MILP (5) is modified for complex heterogeneous systems. In Table 1, notations x_{jh}, r_{jh}, a_{ijh}, and u_{jh} are introduced for heterogeneous systems. Similar to the parameters and variables defined for homogeneous system (K_j, x^b_{jk}, q_{sj}, and p_{sjk}) in Sections 2.2–2.4, the following additional parameters and decision variables can be defined for heterogeneous systems.

$K_{jh} = \{0, 1, 2, \cdots, u_{jh}\}$: set of possible number of type h components at subsystem j, $h \in H_j, j \in J$

$$x^b_{jhk} = \begin{cases} 1 & \text{if } x_{jh} = k, \\ 0 & \text{otherwise,} \end{cases} \quad k \in K_{jh}, h \in H_j, j \in J$$

q_{sjh}: outcome of type h component at subsystem j for sample s, $h \in H_j, j \in J, s \in \bar{S}$

$$p_{sjhk} = \begin{cases} 1 & \text{if } k \ge q_{sjh}, \\ 0 & \text{otherwise,} \end{cases} \quad s \in \bar{S}, k \in K_{jh}, h \in H_j, j \in J$$

Note that the graph representations presented in Section 2.2 can be directly used because the representations do not distinguish homogeneous and heterogeneous subsystems. For the sampling procedure in Section 2.3, the procedure in Algorithm 1 can be used with a few changes to generate q_{sjh} for $h \in H_j, j \in J, s \in \bar{S}$. Then, nonduplicate set S is obtained from $\bar{S}$. Based on these changes and some trivial modifications, MILP (5) can be modified for heterogeneous systems as follows.

$$\max \quad \sum_{s\in\bar{S}} w_s y_s \tag{6a}$$

$$s.t. \quad \sum_{j\in J}\sum_{h\in H_j} a_{ijh}x_{jh} \le b_i, \qquad i \in I, \tag{6b}$$

$$\sum_{j\in J_c}\sum_{h\in H}\sum_{k\in K_{jh}} p_{sjhk}\cdot x^b_{jhk} \ge y_s, \qquad c \in C, s \in \bar{S}, \tag{6c}$$

$$\sum_{k \in K_{jh}} x^b_{jhk} = 1, \qquad h \in H_j, j \in J, \tag{6d}$$

$$x_{jh} = \sum_{k \in K_{jh}} k \cdot x^b_{jhk}, \qquad h \in H_j, j \in J, \tag{6e}$$

$$\sum_{h \in H_j} x_{jh} \geq 1, \qquad j \in J, \tag{6f}$$

$$0 \leq x_{jh} \leq u_{jh}, x_{jh} \in \mathbb{Z} \qquad h \in H_j, j \in J, \tag{6g}$$

$$x^b_{jhk} \in \{0, 1\}, \qquad k \in K_{jh}, h \in H_j, j \in J, \tag{6h}$$

$$0 \leq y_s \leq 1, \qquad s \in \bar{S} \tag{6i}$$

Observe that all constraints are similar to those of (5), except for (6f). In (5), Constraint (5h) ensures at least one component should be used for subsystem j. In (6), Constraint (6f) does the same role by summing all mixed components in subsystem j.

The final remark is that the sample-based MILP formulations have the potential to be used for other network reliability analyses. Note that any constraint on x can be easily added to (6). Hence, for other versions of complex system RAP with different objective functions, (6) can be used with a few trivial modifications. Further, by designing tailored constraints defining y_s and sample data (p_{sjk}) structure, (6) can be used for other network reliability analyses.

3 AID Algorithms

In this section, an AID algorithm is proposed for RAP based on the MILP model (6). The AID algorithm, presented in Algorithm 2, follows the basic framework proposed by Park and Klabjan [28]. The algorithm starts with initial clusters and aggregated data (Line 1), and iteratively disaggregate the clusters while the optimality condition is not met (Lines 2–7). The only yet significant differences from the basic framework of Park and Klabjan [28] are the aggregation technique in Line 3 and additional checkup in Line 6. First, instead of the standard average function for data aggregation, the minimum function is used in Line 3. Second, because the solution quality with respect to the original problem (2) can be checked given a feasible solution, in each iteration of AID, the best solution for the original objective function $f(x)$ is updated and kept separately independent of the best sample-based objective value.

Algorithm 2 AID (*tol*)

Input: *tol* (tolerance for optimality gap)

1: Define initial clusters
2: **Do**
3: Create aggregated data and solve aggregated problem
4: Check optimality condition
5: **if** optimality condition is violated **then** decluster the current clusters
6: **if** $f(x^{aid}) > f(x^{best})$ **then** update x^{best}
7: **While** optimality condition is not satisfied or optimality gap is greater than *tol*

The AID algorithm requires several mandatory components tailored to a particular optimization problem: (i) definition of aggregated data, (ii) declustering procedure and criteria, (iii) definition of aggregated problem, and (iv) optimality condition [28]. Let us first define sets related to clusters.

$\mathbb{K}^t = \{1, 2, \cdots, |\mathbb{K}^t|\}$: index set of the clusters in iteration t
$\mathbb{C}^t = \{\mathbb{C}^t_1, \mathbb{C}^t_2, \cdots, \mathbb{C}^t_{|\mathbb{K}^t|}\}$: set of clusters in iteration t, where $\mathbb{C}^t_\xi$ is a subset of $\bar{S}$ for any ξ in $\mathbb{K}^t$

Next, given clusters $\mathbb{C}^t$, the aggregated data are defined as follows.

$$q_{\xi jh} = \min_{s \in \mathbb{C}^t_\xi} \{q_{sjh}\}, \ \xi \in \mathbb{K}^t, h \in H_j, j \in J$$

$$p_{\xi jhk} = \begin{cases} 1 & \text{if } k \geq q_{\xi jh}, \\ 0 & \text{otherwise,} \end{cases} \quad \xi \in \mathbb{K}^t, k \in K_{jh}, h \in H_j, j \in J$$

$v_\xi = \sum_{s \in \mathbb{C}^t_\xi} w_s$, $\xi \in \mathbb{K}^t$

We emphasize here that the definition of $q_{\xi jh}$ is distinguished from all existing AID implementations [27, 28] because others use averages instead of minimum function. Recall that each sample in $\bar{S}$ represents a scenario. Given a solution $x \in \mathbb{Z}^n_+$, a status function $V_s(x)$ for sample s needs to be defined. Let $V_s(x) = 1$ if the system is working with solution x for sample $s \in \bar{S}$ and $V_s(x) = 0$ otherwise. Let us also define $V_\xi(x)$ for aggregated sample ξ. For each cluster $\xi \in \mathbb{K}^t$, the cluster is declustered or partitioned based on the following declustering procedure.

1. If $V_s(x) = V_\xi(x)$ for all $s \in \mathbb{C}^t_\xi$, do nothing.
2. If $V_s(x) = 0$ for all $s \in \mathbb{C}^t_\xi$ and $V_\xi(x) = 1$, randomly divide $\mathbb{C}^t_\xi$ into two clusters.
3. Otherwise ($0 < \sum_{s \in \mathbb{C}^t_\xi} V_s(x) < |\mathbb{C}^t_\xi|$), divide $\mathbb{C}^t_\xi$ into two clusters: $\mathbb{C}^{t+1}_{\xi 1} = \{s \in \mathbb{C}^t_\xi | V_s(x) = 1\}$ and $\mathbb{C}^{t+1}_{\xi 2} = \{s \in \mathbb{C}^t_\xi | V_s(x) = 0\}$

In words, the current cluster remains the same if the statuses of the original samples in the cluster are equal to the status of the aggregated sample. Otherwise, the current cluster is partitioned based on the statuses of the original samples.

The decision variables associated with $\bar{S}$ are also redefined: y_ξ, $t_{\xi jh}$, and $z_{\xi k}$ are defined similar to y_s, t_{sjh}, and z_{sk}, respectively. However, x_j and a_{ijh} remain the same because they are not associated with the samples. We start the analysis with the following property of aggregated samples.

Lemma 1. Let $\mathbb{K}$ and $\mathbb{K}'$ be the index sets of clusters, and $\mathbb{C}$ and $\mathbb{C}'$ be the corresponding sets of clusters. Let us consider two aggregated samples $\xi \in \mathbb{K}$ and $\xi' \in \mathbb{K}'$ and the corresponding clusters $\mathbb{C}_\xi$ and $\mathbb{C}'_{\xi'}$ with $\mathbb{C}_\xi \subseteq \mathbb{C}'_{\xi'}$. Then, for any feasible solution $\bar{x}$ for (2), $V_\xi(\bar{x}) \leq V_{\xi'}(\bar{x})$ holds.

Proof. Let q_ξ and $q_{\xi'}$ be the sample vectors for $\xi \in \mathbb{K}$ and $\xi' \in \mathbb{K}'$, respectively. Then, for any $h \in H_j, j \in J_k, k \in K$,

$$q_{\xi jh} = \min_{s \in \mathbb{C}_\xi} \{q_{sjh}\} \geq \min_{s \in \mathbb{C}'_{\xi'}} \{q_{sjh}\} = q_{\xi' jh}$$

holds, where the inequality is true because $\mathbb{C}_\xi \subseteq \mathbb{C}'_{\xi'}$ and the equations are due to the definition. Recall that $q_{\xi jh}$ is the minimum number of components needed to prevent failure of type h component at subsystem j for sample ξ. Hence, given $\bar{x}$, if the system is working for aggregated sample ξ, then it is also working for aggregated sample ξ' because $q_{\xi jh} \geq q_{\xi' jh}$. This ends the proof. □

The aggregated problem is a weighted version of (6) and defined as follows.

$$\begin{aligned}
F_t = \quad \max \quad & \sum_{\xi \in \mathbb{K}^t} v_\xi y_\xi & & \\
\text{s.t.} \quad & \sum_{j \in J} \sum_{h \in H_j} a_{ijh} x_{jh} \leq b_i, & & i \in I, \\
& \sum_{j \in J_c} \sum_{h \in H_j} \sum_{k \in K_{jh}} p_{\xi jhk} \cdot x^b_{jhk} \geq y_\xi, & & c \in C, \xi \in \mathbb{K}^t, \\
& \sum_{h \in H_j} x_{jh} \geq 1, & & j \in J, \\
& x_{jh} = \sum_{k \in K_{jh}} k \cdot x^b_{jhk}, & & h \in H_j, j \in J \\
& 0 \leq x_{jh} \leq u_{jh}, x_{jh} \in \mathbb{Z} & & h \in H_j, j \in J, \\
& x^b_{jhk} \in \{0,1\}, & & k \in K_{jh}, h \in H_j, j \in J, \\
& 0 \leq y_\xi \leq 1, & & \xi \in \mathbb{K}^t
\end{aligned} \tag{7}$$

Let $\bar{X}^t = (\bar{x}^t, (\bar{x}^b)^t, \bar{y}^t)$ be an optimal solution for (7). Then, $\bar{x}^t$ can generate a feasible solution to (6) because all binary variables can be determined given $\bar{x}^t$ and samples in $\bar{S}$. Let $X^* = (x^*, (x^b)^*, y^*)$ be optimal solutions for (6). Let $\bar{x}^*$ be a feasible solution for (7) obtained by transforming x^* and let $\hat{x}^t$ be a feasible solution for (7) obtained by transforming $\bar{x}^t$. In Proposition 2 and Lemma 2, the validity of optimality condition and nonincreasing optimal objective function values of the aggregated problems are shown.

Proposition 2. If $V_s(\bar{x}^t) = V_\xi(\bar{x}^t)$ for all $s \in \mathbb{C}^t_\xi$ and $\xi \in \mathbb{K}^t$, then $\hat{x}^t$ is an optimal solution.

Proof. First, let us assume that $\mathbb{K}$ is the index set of clusters where each cluster is a sample in $\bar{S}$ and $\mathbb{C}$ be the corresponding cluster set. Then, all samples s in $\mathbb{C}^t_\xi$, $\xi \in \mathbb{K}^t$, have corresponding clusters in $\mathbb{C}$. Further, there exists exactly one cluster in $\mathbb{K}^t$ such that $\mathbb{C}_{\xi'} \subseteq \mathbb{C}^t_\xi$ for $\xi' \in \mathbb{K}$ and $\xi \in \mathbb{K}^t$. Note that, given x^*, $V_s(x^*) = y^*_s$ and $V_{\xi'}(x^*) = \bar{y}^*_{\xi'}$ hold. Then, by Lemma 1, we must have $y^*_s \le \bar{y}^*_\xi$ for any $s \in \mathbb{C}^t_\xi$ and $\xi \in \mathbb{K}^t$.

Next, we derive

$$\sum_{s\in\bar{S}} v_s y^*_s = \sum_{\xi\in\mathbb{K}^t}\sum_{s\in\mathbb{C}^t_\xi} v_s y^*_s \le \sum_{\xi\in\mathbb{K}^t}\sum_{s\in\mathbb{C}^t_\xi} v_s \bar{y}^*_\xi = \sum_{\xi\in\mathbb{K}^t} v_\xi \bar{y}^*_\xi \le \sum_{\xi\in\mathbb{K}^t} v_\xi \bar{y}^t_\xi = \sum_{\xi\in\mathbb{K}^t}\sum_{s\in\mathbb{C}^t_\xi} v_s \bar{y}^t_\xi = \sum_{s\in\bar{S}} v_s \bar{y}^t_s,$$

where the first inequality holds because $y^*_s \le \bar{y}^*_\xi$ for all $\xi \in \mathbb{K}^t$ and $s \in \mathbb{C}^t_\xi$, the second equality holds by the definition, the second inequality holds because $\bar{y}^t$ is an optimal solution for (7), and the last equality is true because $V_s(\bar{x}^t) = V_\xi(\bar{x}^t)$.

Finally, the inequality $\sum_{s\in\bar{S}} v_s y^*_s \le \sum_{s\in\bar{S}} v_s \bar{y}^t_s$ implies that $\bar{x}^t$ is an optimal solution for (6). □

Lemma 2. For each iteration $t \ge 2$, $F_{t-1} \ge F_t$ holds.

Proof. For simplicity, let us assume that $\{\mathbb{C}^{t-1}_1\} = \mathbb{C}^{t-1} \setminus \mathbb{C}^t$, $\{\mathbb{C}^t_1, \mathbb{C}^t_2\} = \mathbb{C}^t \setminus \mathbb{C}^{t-1}$, and $\mathbb{C}^{t-1}_1 = \mathbb{C}^t_1 \cup \mathbb{C}^t_2$. That is, $\mathbb{C}^{t-1}_1$ is the only cluster in $\mathbb{C}^{t-1}$ such that the entries in $\mathbb{C}^{t-1}_1$ have both 0 and 1 for $V_s(\bar{x}^{t-1})$, and $\mathbb{C}^{t-1}_1$ is partitioned into $\mathbb{C}^t_1$ and $\mathbb{C}^t_2$ for iteration t.

We can derive

$$\begin{aligned}
F_{t-1} &= v^{t-1}_1 \bar{y}^{t-1}_1 + \sum_{\xi\in\mathbb{K}^{t-1}\setminus\{1\}} v^{t-1}_\xi \bar{y}^{t-1}_\xi \;\ge\; v^{t-1}_1 \max\{\bar{y}^t_1, \bar{y}^t_2\} + \sum_{\xi\in\mathbb{K}^{t-1}\setminus\{1\}} v^{t-1}_\xi \bar{y}^t_\xi \\
&= (v^t_1 + v^t_2)\max\{\bar{y}^t_1, \bar{y}^t_2\} + \sum_{\xi\in\mathbb{K}^{t-1}\setminus\{1\}} v^{t-1}_\xi \bar{y}^t_\xi \;\ge\; v^t_1 \bar{y}^t_1 + v^t_2 \bar{y}^t_2 + \sum_{\xi\in\mathbb{K}^{t-1}\setminus\{1\}} v^{t-1}_\xi \bar{y}^t_\xi \\
&= v^t_1 \bar{y}^t_1 + v^t_2 \bar{y}^t_2 + \sum_{\xi\in\mathbb{K}^t\setminus\{1,2\}} v^t_\xi \bar{y}^t_\xi \;=\; F_t,
\end{aligned}$$

where the first inequality holds because $\mathbb{C}^t_1 \subseteq \mathbb{C}^{t-1}_1$ and $\mathbb{C}^t_2 \subseteq \mathbb{C}^{t-1}_1$ imply $\bar{y}^t_1 \le \bar{y}^{t-1}_1$ and $\bar{y}^t_2 \le \bar{y}^{t-1}_1$ by Lemma 1. Therefore, we have $F_{t-1} \ge F_t$. □

Let $E_t = f(\bar{x}^t)$ for iteration t. Note that, during the iterations of AID, the best objective function value of the original problem E^{best}_t can be updated by $E^{best}_t = \max\{E_1, E_2, \cdots, E_t\}$. Then, by Lemma 2, $E^{best}_t \le F_t \le F_{t-1} \le F_{t-2} \le \cdots \le F_1$ holds and the following proposition also holds.

Proposition 3. AID for (6) monotonically converges to the optimal solution.

We emphasize here that optimality of AID is with respect to a sample-based MILP model (6) and AID does not guarantee optimality for the original problem (2). The relative optimality gap of AID can be calculated with gap (%) $= \frac{F_t - E^{best}_t}{E^{best}_t} \times 100$ and the algorithm terminates if this gap is less than the tolerance (Line 7 of Algorithm 2).

4 Computational Experiment

In the computational experiment, six algorithms were tested.

1. MILP: MILP model (6) from Section 2
2. AID: AID algorithm from Section 3 with $tol = 10^{-5}$ terminating with guaranteed optimality
3. AID_{heur}: AID algorithm with $tol = 0.002$ terminating without guaranteed optimality
4. AID_{rep}: multiple runs of AID with $n^{\text{S}} = 1000$ and $tol = 0.002$
5. GA: multiple runs of genetic algorithm (GA)
6. PSO: multiple runs of particle swarm optimizaiton (PSO).

Although the AID algorithm is designed to solve the problem optimally, there are several reasons why the exact optimality for RAP is not as important as other problems. First, the base MILP model (6) is only an approximation to the actual objective function. Second, the parameters of the original problem (such as component reliability) are obtained from statistical estimation and are not exactly deterministic. Finally, by relaxing the tolerance, execution time can be reduced [27]. Therefore, in addition to the original AID algorithm, we test two heuristic versions of the AID algorithm $\mathsf{AID}_{\mathsf{heur}}$ and $\mathsf{AID}_{\mathsf{rep}}$.

We will present three experiment sets in this section. We will first demonstrate the characteristics and performances of MILP and AID. Second, we will compare all of the proposed algorithms, both exact approximations and heuristics, to determine the most competitive algorithm. Third, because no exact algorithm exists for (2), we will compare the best of the proposed algorithms with two metaheuristic algorithms in the literature. In detail, the first set of experiment compares MILP and AID for benchmark instances for homogeneous systems in Section 4.2. Next, the second set of experiment compares MILP, AID, and $\mathsf{AID}_{\mathsf{heur}}$ for random instances for heterogeneous systems in Section 4.3. The results in Sections 4.2 and 4.3 show that combining multiple runs of $\mathsf{AID}_{\mathsf{heur}}$ with small-size sample sets can be beneficial. Hence, the performances of all proposed algorithms including $\mathsf{AID}_{\mathsf{rep}}$ are compared in Section 4.4. Finally the third set of experiment in Section 4.5 compares $\mathsf{AID}_{\mathsf{rep}}$, the best algorithm among the four proposed algorithms, with two state-of-the-art metaheuristic algorithms in the literature. The implementation details of the metaheuristic algorithms are available in Section 4.5.

All algorithms are implemented in C# and CPLEX and a personal computer with 8 GB RAM and Intel Core i7 (2.40 GHz dual core) was used. The source code of all proposed and benchmark algorithms is available in the online supplement. Let $f(x)$ and $\hat{f}(x)$ be the objective function values of x for the original problem (2) and sample-based MILP (6), respectively. Let x^* and $\hat{x}^*$ be the optimal solutions for (2) and (6), respectively. When the optimal solution is unknown, x^* and $\hat{x}^*$ are the best known solutions. To compare the performances of different algorithms, multiple performance measures are used.

$\delta = |f(\hat{x}^*) - \hat{f}(\hat{x}^*)|$
$\Delta = f(x^*) - f(\hat{x}^*)$
%opt: percentage of obtaining an optimal solution (benchmark instances)
%best: similar to *%opt*, but used when exact optimality is not verified (random instances)
time: execution time in seconds

Note that our models optimize sample-based objective function (6), which is an approximation to the actual system reliability. To check how close $\hat{f}(x)$ is to $f(x)$, sample approximation gap δ is defined, which can check the effectiveness of sample-based model (6). Next, Δ is calculated to check how our solution $\hat{x}^*$ is different from the optimal or best solution of the original problem. Finally, *%opt* is defined to check how often our algorithms can find an optimal solution for benchmark instances and *%best* is used for random instances, where optimality is not verified. Although the exact optimality is not verified for random instances, all algorithms converge to a specific value for each instance, which implies the solutions are very likely to be the optimal solution.

4.1 Benchmark and random instances

Although there are many benchmark instances, including the well-known instances of Coit and Konak [9], many of them are either for homogeneous systems, heterogeneous series–parallel systems, or heterogeneous complex systems with additional decisions for the reliability of the components. While our algorithms can also be used for heterogeneous series–parallel systems, we believe the existing algorithms such as Caserta and Voß[6] are much more efficient for this simpler version of RAP. Hence, complex homogeneous system examples in Table 2 from the literature are used in the experiment.

Instance	System	n	m	Type	Reference
Benchmark 1	System 1 in Figure 2	5	1	Homogeneous	Aggarwal [2]
Benchmark 2	System 2 in Figure 2	5	1	Homogeneous	Aggarwal [2]
Benchmark 3	System 4 in Figure 2	7	2	Homogeneous	Ryoo [30]

Table 2: Benchmark instances

Two instances (Benchmarks 1 and 3) are directly from the literature, while Benchmark 2 adopts the resource and reliability data of Aggarwal [2] for the network in Figure 2(b). The data can be found in each of the papers. The optimal solutions and system reliabilities are known.

Benchmark 1: $x^* = [3\ 1\ 2\ 2\ 1]$ with $R^* = 0.9932$
Benchmark 2: $x^* = [1\ 3\ 1\ 1\ 4]$ with $R^* = 0.9993$
Benchmark 3: $x^* = [1\ 2\ 2\ 2\ 1\ 3\ 1]$ with $R^* = 0.9949$

Note that the benchmark instances in Table 2 can be quickly solved by the branch-and-bound algorithm of Ha and Kuo [15], where their algorithm takes less than 0.1 seconds to solve the similar size problems. We use these benchmark instances only for demonstrating the behavior of the proposed models and algorithms.

Next, random instances are generated for heterogeneous complex systems based on Systems 1–4 in Figure 2. Note that Systems 1 and 2 can share instances while different s–t cuts are needed. Hence, instances are generated for $n \in \{5, 6, 7\}$ and Systems 1 and 2 share the instances with $n = 5$. For each $n \in \{5, 6, 7\}$, instances are generated for $n^{\mathrm{H}} = 2, 3, 4$. Hence, there are nine pairs of (n, n^{H}). For each (n, n^{H}) pair, four random instances are generated. In summary, a total of 36 random instances are generated and each system in Figure 2 gets 12 instances.

Given a uniform random number $\gamma_{jh} \in [0, 1]$ for type h component at subsystem j, values are created by

$$
\begin{aligned}
r_{jh} &= \mathsf{round}(0.5 + 0.05\sqrt{n^{\mathrm{H}}} + 0.2\gamma_{jh}, 2), \\
a_{ijh} &= \mathsf{round}(1 + \gamma + 3\gamma_{jh}, 2), \\
b_i &= \mathsf{round}\Big(\Big\{\textstyle\sum_{j\in J}\big[\min_{h\in H} a_{ijh}\big]\Big\}(1.5 + 0.5\gamma), 0\Big),
\end{aligned}
$$

where $\gamma \in [0, 1]$ is randomly generated for each time needed. To avoid unnecessary numerical errors, b_i is rounded to an integer and two digits are kept after the decimal point for r_{jh} and a_{ijh}. Note that r_{jh} and a_{ijh} are correlated through γ_{jh}. If γ_{jh} is large, then both r_{jh} and a_{ijh} are large, while there are some random variations due to γ. The formula for b_i ensures that the available resource should be between 150% and 200% compared with the case when the minimum amount of resource is used to make the system work (at least one component should be placed for each subsystem). In Table 3, the types of the randomly generated instances are summarized. The instances and the best known solutions are available in the online supplement.

System	Network	n	n^{H}	m	Type
System 1	System 1 in Figure 2	5	2,3,4	2	Heterogeneous
System 2	System 2 in Figure 2	5	2,3,4	2	Heterogeneous
System 3	System 3 in Figure 2	6	2,3,4	2	Heterogeneous
System 4	System 4 in Figure 2	7	2,3,4	2	Heterogeneous

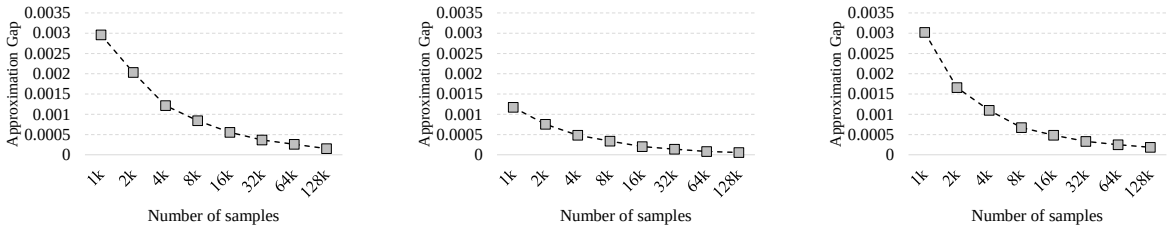
Table 3: Types of random instances

4.2 Result of MILP and AID for benchmark instances

For each benchmark instance, 100 sample sets were used for each of the sample sizes $|S|$ in $\{1, 2, 4, 8, 16, 32, 64, 128\} \times 1,000$. Then, the averages of the 100 results are reported in Figures 5 and 6. In all plots, gray squares with a dashed black line represent MILP and black circles with a gray line represent AID. For all series in all plots, each marker represents the average of the results obtained from the 100 different sample sets.

In Figure 5, the approximation gap (δ) of MILP model (5) is plotted to check the convergence of the sample-based objective function of MILP to the actual system reliability. As sample size n^{S} increases, δ decreases and converges to near zero value. Among the three benchmark instances, the δ values of Benchmark 2 are the smallest for all sample sizes. We think this is due to the structure of the system, as Subsystem 5 of Benchmark 2 provides more alternatives to achieve higher system reliability than Subsystem 5 of Benchmark 1.

In Figure 6, MILP and AID are compared based on the three by three plot matrix that summarizes the performance measures for the three benchmark instances. Each row of the plot matrix is for a performance measure, and each column of the plot matrix is for a benchmark instance. For example, the plot in the third column of the first row shows %*opt* values for Benchmark 3 over increasing sample sizes.



(a) Benchmark 1 (b) Benchmark 2 (c) Benchmark 3

Figure 5: Approximation gap (δ) of MILP for benchmark instances

Let us consider *%opt* in the first row of the plot matrix. Even when sample size n^{S} is 1000, both MILP and AID are able to find an optimal solution with approximately 20% and 30%, respectively, on average across the three benchmark instances. For both algorithms, *%opt* converges to 100% as the sample size increases. However, the *%opt* values of AID are always higher than that of MILP, as AID examines multiple solutions with respect to the original objective function in each iteration. Among the three instances, *%opt* values of Benchmark 2 show slower convergence due to the structure of the system. Having an interconnected subsystem in Benchmark 2 generates multiple optimal solutions for the sample-based objectives and this prevents finding the optimal solution to the original problem when the sample size is small.

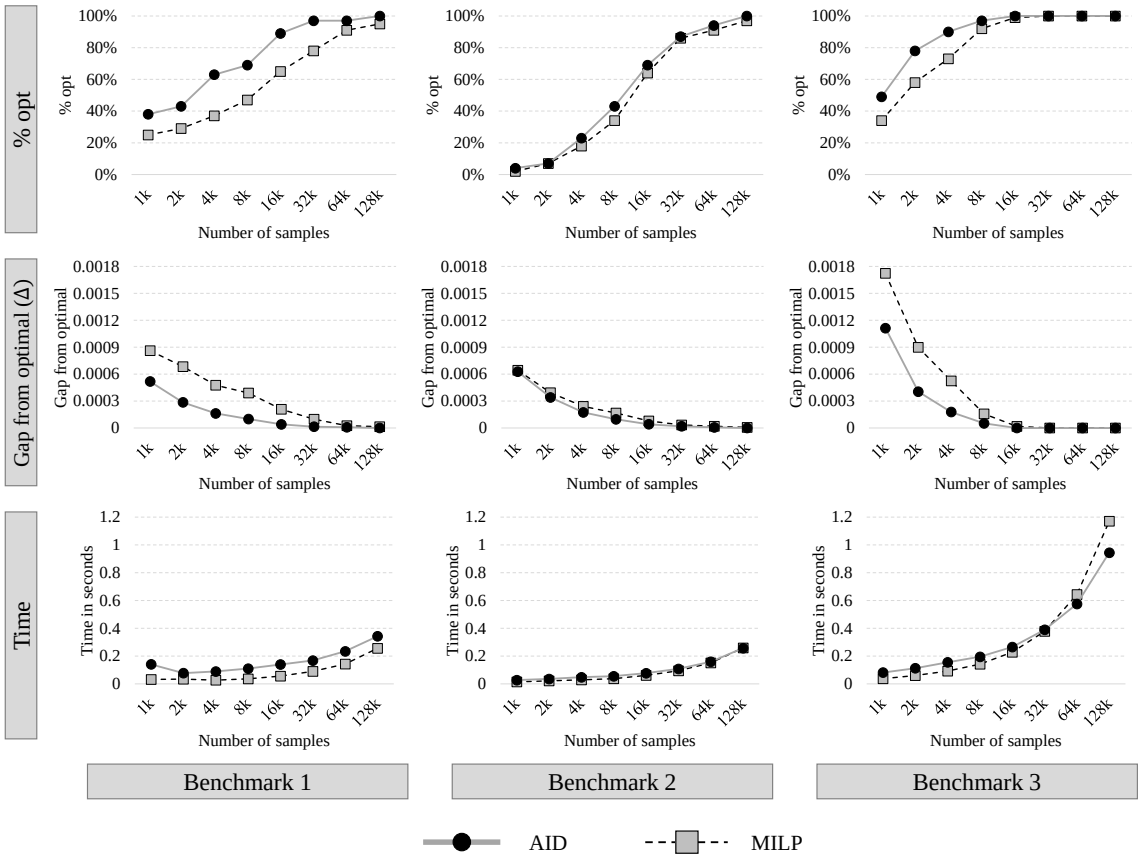


Figure 6: Result of MILP and AID for benchmark instances

The values of Δ also show the effectiveness of both algorithms. The average gap Δ decreases and converges to zero as the sample size increases. When the sample size is small, Δ values vary over different systems. However, for the largest sample size, all systems have near-zero Δ values. Hence, with a sufficiently large n^{S},

both algorithms find an optimal solution. The execution times for all instances and sample sizes are within a second in most cases. The average execution times of MILP for the largest sample size $n^{\mathrm{S}} = 128,000$ are 0.26, 0.26, and 1.17 seconds for Benchmarks 1,2, and 3, respectively. The average times of AID for the largest sample size $n^{\mathrm{S}} = 128,000$ are 0.34, 0.26, and 0.94 seconds for Benchmarks 1,2, and 3, respectively. The plots in the third row show that the execution times increase with sample size. Between the two algorithms, MILP is faster for Benchmarks 1 and 2. However, for a larger system (Benchmark 3), MILP is faster when the sample size is small but slower when $n^{\mathrm{S}} \geq 64,000$.

In summary, the result shows (i) MILP and AID converge to optimal with higher probabilities as n^{S} increases, (ii) AID converges faster than MILP, and (iii) AID becomes attractive as n^{S} increases as it is faster and has higher *%opt* values.

4.3 Result of MILP and AID for random instances

In Table 4, the results for the 48 random instances are presented. For each instance, 20 sample sets are tested for each of sample sizes $n^{\mathrm{S}} = 1000, 2000, 4000$, and 8000. After calculating the performance measures, the results are aggregated by n and n^{H} to present the averages. Hence, each row in Table 4 is the average of 80 results (20 sample sets $\times$ 4 instances).

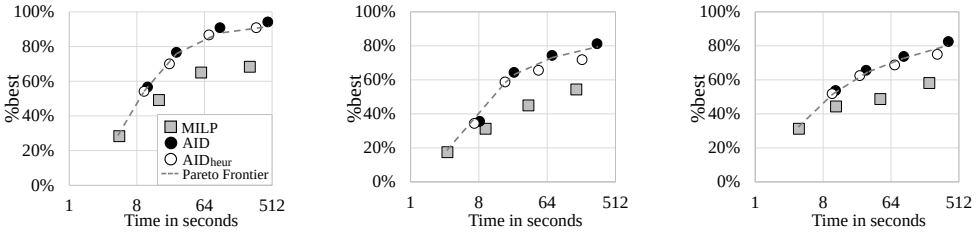
	Data	Samples		MILP				AID			$\mathsf{AID}_{\mathsf{heur}}$ (tol = 0.002)		
	(n, n^{H})	$\lvert S\rvert$	$\lvert\bar{S}\rvert$	Time	%best	Δ	δ	Time	%best	Δ	Time	%best	Δ
S1	(5,2)	1000	762	0.7	18%	0.0044	0.0093	1.9	55%	0.0020	1.7	53%	0.0021
	(5,2)	2000	1363	1.8	33%	0.0027	0.0054	4.5	78%	0.0004	3.8	75%	0.0006
	(5,2)	4000	2375	4.1	55%	0.0018	0.0035	9.2	93%	0.0000	7.3	88%	0.0001
	(5,2)	8000	4038	11.9	75%	0.0007	0.0021	20.0	95%	0.0000	15.9	93%	0.0000
	(5,3)	1000	974	1.8	23%	0.0053	0.0122	7.1	50%	0.0013	6.4	50%	0.0013
	(5,3)	2000	1913	5.0	38%	0.0025	0.0065	17.4	88%	0.0004	15.2	88%	0.0004
	(5,3)	4000	3722	18.4	53%	0.0014	0.0042	46.1	93%	0.0002	36.9	90%	0.0002
	(5,3)	8000	7164	75.4	68%	0.0008	0.0025	152.0	95%	0.0001	109.1	88%	0.0003
	(5,4)	1000	997	2.3	30%	0.0036	0.0102	9.9	58%	0.0008	8.6	58%	0.0009
	(5,4)	2000	1990	7.1	53%	0.0023	0.0065	25.1	80%	0.0001	21.2	78%	0.0002
	(5,4)	4000	3963	28.7	45%	0.0013	0.0046	67.5	78%	0.0002	52.9	73%	0.0002
	(5,4)	8000	7874	122.4	55%	0.0010	0.0031	259.4	85%	0.0000	186.6	80%	0.0001
S2	(5,2)	1000	762	0.8	18%	0.0043	0.0087	1.5	35%	0.0017	1.2	35%	0.0018
	(5,2)	2000	1363	1.6	30%	0.0024	0.0047	4.2	43%	0.0011	3.0	38%	0.0013
	(5,2)	4000	2375	3.8	45%	0.0014	0.0028	9.2	68%	0.0005	6.6	58%	0.0007
	(5,2)	8000	4038	11.2	70%	0.0005	0.0015	19.6	88%	0.0001	12.8	78%	0.0003
	(5,3)	1000	974	2.1	30%	0.0034	0.0083	5.2	38%	0.0016	4.5	38%	0.0017
	(5,3)	2000	1913	5.3	40%	0.0016	0.0046	14.4	73%	0.0004	10.7	65%	0.0007
	(5,3)	4000	3722	18.6	50%	0.0012	0.0034	35.6	75%	0.0004	24.6	68%	0.0006
	(5,3)	8000	7164	76.4	65%	0.0004	0.0020	107.3	88%	0.0001	68.6	75%	0.0002
	(5,4)	1000	997	2.9	33%	0.0038	0.0096	10.9	45%	0.0011	9.5	40%	0.0012
	(5,4)	2000	1990	8.7	33%	0.0024	0.0063	30.6	48%	0.0006	24.2	45%	0.0008
	(5,4)	4000	3963	33.4	43%	0.0014	0.0039	82.5	60%	0.0004	60.5	58%	0.0004
	(5,4)	8000	7874	152.6	58%	0.0007	0.0027	287.7	73%	0.0002	197.9	58%	0.0004
S3	(6,2)	1000	925	1.0	5%	0.0032	0.0083	2.1	53%	0.0005	1.7	53%	0.0007
	(6,2)	2000	1767	2.3	5%	0.0023	0.0050	5.4	50%	0.0004	4.1	45%	0.0005
	(6,2)	4000	3317	6.6	13%	0.0009	0.0028	11.2	63%	0.0002	8.0	50%	0.0003
	(6,2)	8000	6141	20.8	25%	0.0006	0.0020	26.1	75%	0.0001	17.5	63%	0.0002
	(6,3)	1000	995	2.9	3%	0.0037	0.0098	7.8	15%	0.0015	6.2	13%	0.0018
	(6,3)	2000	1982	9.2	0%	0.0031	0.0067	26.0	25%	0.0009	18.1	18%	0.0013
	(6,3)	4000	3936	35.1	8%	0.0016	0.0041	75.9	35%	0.0005	45.8	13%	0.0008
	(6,3)	8000	7795	151.0	15%	0.0012	0.0028	267.7	50%	0.0003	147.8	40%	0.0005
	(6,4)	1000	1000	3.2	15%	0.0039	0.0088	9.6	45%	0.0010	8.6	43%	0.0011
	(6,4)	2000	1999	8.8	30%	0.0020	0.0051	26.6	58%	0.0005	21.5	53%	0.0006
	(6,4)	4000	3998	35.1	33%	0.0018	0.0036	77.0	65%	0.0005	55.9	58%	0.0005
	(6,4)	8000	7992	149.4	45%	0.0012	0.0025	282.8	78%	0.0002	182.3	68%	0.0002
S4	(7,2)	1000	972	1.7	23%	0.0032	0.0094	4.0	63%	0.0003	3.5	58%	0.0003
	(7,2)	2000	1907	4.6	38%	0.0017	0.0052	8.2	80%	0.0002	6.9	70%	0.0002
	(7,2)	4000	3708	13.2	50%	0.0009	0.0035	18.7	85%	0.0001	13.9	80%	0.0001
	(7,2)	8000	7127	52.4	60%	0.0003	0.0020	53.4	95%	0.0000	37.1	93%	0.0000
	(7,3)	1000	999	5.5	15%	0.0049	0.0089	12.8	40%	0.0024	10.8	38%	0.0025
	(7,3)	2000	1997	19.7	48%	0.0020	0.0046	35.7	73%	0.0007	27.0	65%	0.0010
	(7,3)	4000	3991	72.7	70%	0.0008	0.0028	139.2	95%	0.0001	89.2	93%	0.0002
	(7,3)	8000	7970	318.7	70%	0.0007	0.0019	640.8	93%	0.0001	415.4	85%	0.0002
	(7,4)	1000	1000	6.8	48%	0.0031	0.0078	16.6	68%	0.0011	15.6	68%	0.0011
	(7,4)	2000	2000	23.0	63%	0.0013	0.0045	37.3	78%	0.0004	31.6	75%	0.0005
	(7,4)	4000	4000	87.5	75%	0.0004	0.0025	151.8	93%	0.0000	117.2	88%	0.0001
	(7,4)	8000	7999	403.8	75%	0.0003	0.0016	653.3	95%	0.0000	491.4	95%	0.0000

Table 4: Results of MILP, AID, and $\mathsf{AID}_{\mathsf{heur}}$ for random instances

For each (n, n^{H}) pair, as n^{S} increases, all algorithms have increasing execution times, *%best*, and Δ values.

This implies that, if n^{S} is larger, then the algorithms are more likely to provide an optimal solution. However, the execution times increase quickly. While all three algorithms MILP, AID, and $\mathsf{AID}_{\mathsf{heur}}$ find the optimal solution with higher chance as n^{S} increases, the convergence rates of *%best* are different for the three algorithms. The *%best* value of MILP is up to 33% lower than the other two. In terms of the solution speed, MILP is the fastest, while the relative speed up by MILP is decreasing in n^{S}. Hence, if the sample size is very large, it is beneficial to run AID or$\mathsf{AID}_{\mathsf{heur}}$.

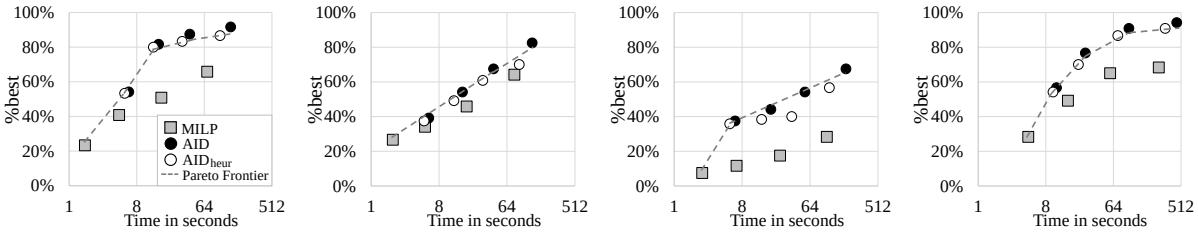
To summarize the result in Table 4, scatter plots for two core performance measures *%best* and *time* are presented in Figures 7 and 8 by aggregating the result by n^{S}, n^{H}, and system types. Each plot in Figure 7 is for the aggregated result by n^{S} and n^{H}, where the four markers of each algorithm are for $n^{\mathrm{S}} = 1000$, 2000, 4000, and 8000. Similarly, Each plot in Figure 8 is for the aggregated result by n^{S} and system type. In all plots, the horizontal and vertical axes represent *time* and *%best*, respectively, and the horizontal axes are in log scale to display the trend better. In each plot, Pareto frontiers (dotted lines) are also plotted to indicate the most efficient algorithm achieving the highest *%best* value per unit time.



(a) $n^{\mathrm{H}} = 2$ (b) $n^{\mathrm{H}} = 3$ (c) $n^{\mathrm{H}} = 4$

Figure 7: Scatter plots of time and *%best* by n^{H} for random instances

In all plots of Figures 7 and 8, AID and $\mathsf{AID}_{\mathsf{heur}}$ outperform MILP. Given a fixed time, either AID or $\mathsf{AID}_{\mathsf{heur}}$ provides the highest *%best* values. The Pareto frontier lines are mostly formed by AID and $\mathsf{AID}_{\mathsf{heur}}$, which implies the two algorithms are more efficient than MILP. In the three plots of Figure 7, the execution times are increasing and *%best* values are decreasing as n^{H} increases.



(a) System 1 (b) System 2 (c) System 3 (d) System 4

Figure 8: Scatter plots of time and *%best* by system types for random instances

In the four plots of Figure 8, the execution times are increasing as n increases (Systems 1–4 have $n = 5, 5, 6,$ and 7, respectively). While the convergences of *%best* do not have a clear trend over n or system types, we think this is due to the random instances and different system structures.

4.4 Comparison of all proposed algorithms

In Table 4, $\mathsf{AID}_{\mathsf{heur}}$ with 1000 samples frequently finds optimal solutions (45% on average) in a few seconds (6.5 seconds on average). This observation naturally suggests a heuristic algorithm by running $\mathsf{AID}_{\mathsf{rep}}$ multiple times with different sample sets. Let n^{R} be the number of sample sets (or number of replications). In this section, the performance of $\mathsf{AID}_{\mathsf{rep}}$ is tested by running the algorithm with $n^{\mathrm{R}} = 5$, 10, 15, 20, and 25 sample sets over five different random number sets, where each sample set contains $n^{\mathrm{S}} = 1000$ samples.

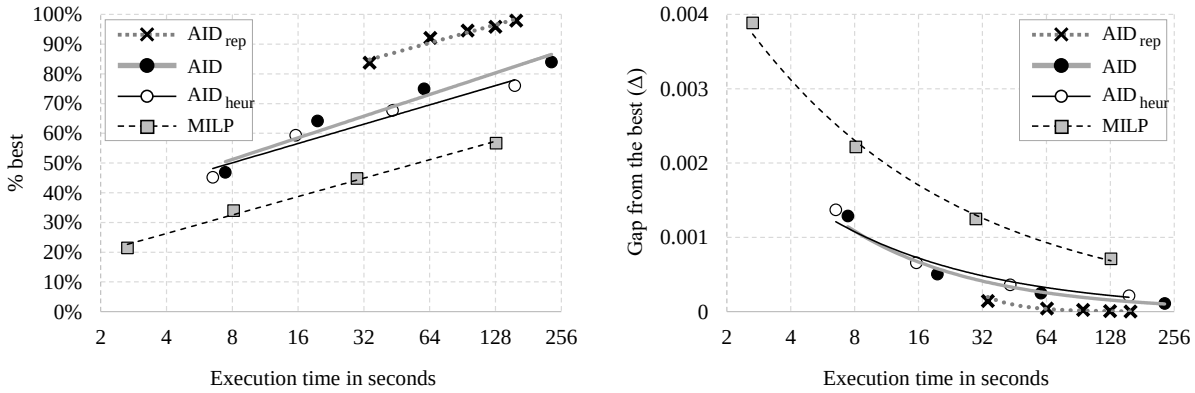
In Table 5, the result for $\mathsf{AID}_{\mathsf{rep}}$ is presented, where each row shows the average of 20 results (4 instances $\times$ 5 random number sets). For most of the cases, $\mathsf{AID}_{\mathsf{heur}}$ is able to find the best known solution with 100%

probability, while *%best* values are lower for Systems 2 and 4 with $n^{\mathrm{H}} = 3$. Because System 2 with $n^{\mathrm{H}} = 4$ has near-perfect *%best* values, we think that the bad performances for Systems 2 and 4 with $n^{\mathrm{H}} = 3$ are due to the random instances and sample sets.

		$n^{\mathrm{R}} = 5$			$n^{\mathrm{R}} = 10$			$n^{\mathrm{R}} = 15$			$n^{\mathrm{R}} = 20$			$n^{\mathrm{R}} = 25$		
	n^{H}	Time	%best	Δ	Time	%best	Δ	Time	%best	Δ	Time	%best	Δ	Time	%best	Δ
S1	2	9	100%	0.0000	16	100%	0.0000	25	100%	0.0000	33	100%	0.0000	42	100%	0.0000
	3	33	100%	0.0000	62	100%	0.0000	95	100%	0.0000	125	100%	0.0000	156	100%	0.0000
	4	48	85%	0.0001	90	90%	0.0000	133	95%	0.0000	176	95%	0.0000	224	100%	0.0000
S2	2	7	70%	0.0004	13	100%	0.0000	19	100%	0.0000	26	100%	0.0000	32	100%	0.0000
	3	22	80%	0.0002	45	95%	0.0000	66	95%	0.0000	89	95%	0.0000	110	100%	0.0000
	4	51	90%	0.0001	94	95%	0.0000	137	95%	0.0000	183	95%	0.0000	238	100%	0.0000
S3	2	9	90%	0.0000	18	95%	0.0000	26	100%	0.0000	34	100%	0.0000	43	100%	0.0000
	3	31	30%	0.0005	57	45%	0.0003	85	65%	0.0002	115	80%	0.0000	142	85%	0.0000
	4	44	85%	0.0001	82	85%	0.0000	122	85%	0.0000	166	85%	0.0000	203	90%	0.0000
S4	2	19	95%	0.0000	38	100%	0.0000	57	100%	0.0000	78	100%	0.0000	94	100%	0.0000
	3	53	90%	0.0004	103	100%	0.0000	151	100%	0.0000	203	100%	0.0000	249	100%	0.0000
	4	81	90%	0.0001	155	100%	0.0000	229	100%	0.0000	306	100%	0.0000	380	100%	0.0000

Table 5: Result of $\mathsf{AID}_{\mathsf{rep}}$ for random instances

In Figure 9, all four proposed algorithms are compared based on the scatter plots of execution time, *%best*, and Δ. The horizontal axis is for *time* in both plots and the vertical axis is for *%best* and Δ in Figures 9(a) and 9(b), respectively. The horizontal axes are in log scale to display the trend better. For MILP, AID, $\mathsf{AID}_{\mathsf{heur}}$, the four markers represent the aggregated result for $n^{\mathrm{S}} = 1000$, 2000, 4000, and 8000. Hence, each marker is an average of 320 results over different system types and instances. For $\mathsf{AID}_{\mathsf{rep}}$, the five markers represent $n^{\mathrm{R}} = 5$,10,15,20, and 25. In the scatter plot, an algorithm with smaller execution time is better for a fixed *%best* value (or Δ). Similarly, an algorithm with higher *%best* value (or lower Δ) is better for a fixed execution time. For each algorithm, a trend line is added to represent the trend of *%best* (or Δ) over changing execution times, where the equations are available in the caption.



(a) *%best* (b) Δ

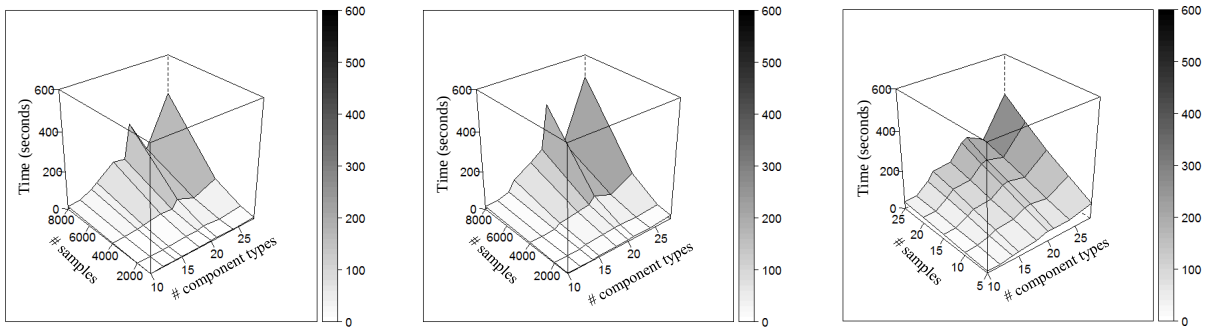
Figure 9: Scatter plots of *%best* and Δ against execution time. In Figure (a), equations for trend lines are $\%best = 1.39 + 8.95\ln(\mathrm{time})$ for MILP, $\%best = 29.35 + 10.51\ln(\mathrm{time})$ for AID, $\%best = 30.59 + 9.37\ln(\mathrm{time})$ for $\mathsf{AID}_{\mathsf{heur}}$, and $\%best = 53.75 + 8.81\ln(\mathrm{time})$ for $\mathsf{AID}_{\mathsf{rep}}$. In Figure (b), equations for trend lines are $\Delta = 0.0057(\mathrm{time})^{-0.437}$ for MILP, $\Delta = 0.0046(\mathrm{time})^{-0.699}$ for AID, $\Delta = 0.0036(\mathrm{time})^{-0.577}$ $\mathsf{AID}_{\mathsf{heur}}$, and $\Delta = 2.2605(\mathrm{time})^{-2.652}$ for $\mathsf{AID}_{\mathsf{rep}}$

For a fixed execution time 128 seconds, $\mathsf{AID}_{\mathsf{rep}}$ achieves the highest *%best* value (98%), whereas AID, $\mathsf{AID}_{\mathsf{heur}}$, and MILP achieves approximately 80%, 75%, and 57%, respectively. For a fixed *%best* value 83%, the execution time of $\mathsf{AID}_{\mathsf{rep}}$ (33 seconds) is approximately seven times faster than AID (230 seconds), while the other two algorithms are expected spend more than 256 seconds to achieve $\%best \geq 83\%$.

The equations of the trend lines also confirm the superiority of $\mathsf{AID}_{\mathsf{rep}}$. Given a fixed target $\%best = 99\%$, the expected execution times are 170, 755, 1481, and 54510 seconds for $\mathsf{AID}_{\mathsf{rep}}$, AID, $\mathsf{AID}_{\mathsf{heur}}$, and MILP, respectively. Given a fixed target $\Delta = 0.0001$, the expected execution times are 44, 239, 498, and 10424

seconds for $\mathsf{AID}_{\mathsf{rep}}$, AID, $\mathsf{AID}_{\mathsf{heur}}$, and MILP, respectively. The largest intercept 53.75 of $\mathsf{AID}_{\mathsf{rep}}$ indicates that the algorithm would give the highest $\%best$ value when a very small amount of time is allowed. The largest slope 10.51 of AID indicates that $\%best$ increases faster per unit time than the other algorithms do.

In all of the results presented, the execution times are closely related to the number of component types ($n \cdot n^{\mathrm{H}}$), number of samples (n^{S}), and the number of replications (n^{R}). The execution times tend to increase rapidly in increasing values of the three parameters. To show the weakness of the proposed algorithms, three-dimensional plots are presented for the three algorithms MILP, $\mathsf{AID}_{\mathsf{heur}}$, and $\mathsf{AID}_{\mathsf{rep}}$ in Figure 10. The plot for AID is omitted as it looks similar to $\mathsf{AID}_{\mathsf{heur}}$. In each of the 3-D plots, the X and Y axes (horizontal) are for number of component types ($n \cdot n^{\mathrm{H}}$) and number of samples (or number of replications), and the Z axis (vertical) is for execution time.



(a) MILP (b) $\mathsf{AID}_{\mathsf{heur}}$ (c) $\mathsf{AID}_{\mathsf{rep}}$

Figure 10: 3-D plot of number of component types, number of samples (or replications), and execution times

Given a smaller value of $n \cdot n^{\mathrm{H}}$ or $n^{\mathrm{S}}(n^{\mathrm{R}})$, the execution times of the three algorithms slowly increase. However, when $n \cdot n^{\mathrm{H}}$ and $n^{\mathrm{S}}(n^{\mathrm{R}})$ increase, the execution time increases rapidly. Hence, the algorithms are likely to spend a very long time for instances with large values of n, n^{H}, or $n^{\mathrm{S}}(n^{\mathrm{R}})$. Among the three algorithms, $\mathsf{AID}_{\mathsf{rep}}$ is the most scalable as it has the smallest slope for execution time over increasing $n \cdot n^{\mathrm{H}}$ and n^{R}.

4.5 Comparison with metaheuristic algorithms

In the previous section, we conclude that $\mathsf{AID}_{\mathsf{rep}}$ is the most efficient algorithm among the proposed algorithms. However, because $\mathsf{AID}_{\mathsf{rep}}$ uses a limited number of samples ($|S| = 1,000$) and does not guarantee an approximated optimal, it can be regarded as a heuristic. Hence, in this section, the performance of $\mathsf{AID}_{\mathsf{rep}}$ is compared with two popular metaheuristic algorithms in the literature. The GA and PSO algorithms are implemented in C# based on Ardakan *et al.* [3] and Wang and Li [37], respectively. While the original GA and PSO algorithms do not solve exactly the same problems with ours, the algorithms are designed to handle heterogeneous components and can solve our problem (2) with simple modifications. In our implementations, all of the algorithmic components and parameter settings are adopted directly from the corresponding papers [3, 37]. For GA, $MaxGen$=300 (number of generations), $\gamma_1 = 0.5$ (penalty for constraint violations), N_{pop}=1000 (number of population), and $r_C = 0.7, r_M = 0.4, p_M = 0.25$ (parameters for mutation and crossover operators) are used. Among all crossover and mutation operators proposed by Ardakan *et al.* [3], two crossover operators (Double Point, Max-Min) and two mutation operators (Simple, Max-Min) are used as suggested by the authors for complex systems. For PSO, $N_{pa} = 300$ (number of particles), $\gamma_1 = 0.4$ (initial penalty for constraint violations), $N_{it} = 100$ (number of iterations), and $w = 0.6, l_1 = l_2 = 1.7$ (parameters for velocity updates) are used. For both algorithms, to incorporate the constraints, the following penalty-based objective function is used.

$$\max f(x) + \gamma_1 \Big[\textstyle\sum_{i \in I} \max\{A_i^\top x - b_i, 0\} + \sum_{j \in J} \max\{1 - \sum_{h \in H_j} x_{jh}, 0\} + \sum_{j \in J} \sum_{h \in H_j} \max\{x_{jh} - u_{jh}, 0\} \Big]$$

For both algorithms, this penalized objective function (fitness) is used throughout the iterations and the best feasible solution is kept separately.

Note that the performance of the algorithms depends on the random number sets, because both algorithms use random numbers to generate or modify solutions. Therefore, running the algorithms multiple times

and reporting the best solution will improve the performance of the algorithms and reduce the performance variations due to random numbers. Similar to $\mathsf{AID}_{\mathsf{rep}}$, n^{R} is used to indicate the number of replications, and various values of n^{R} are tested for both benchmark algorithms. The n^{R} values are selected to have ranges of execution times similar to $\mathsf{AID}_{\mathsf{rep}}$. The performance of GA is tested for $n^{\mathrm{R}} = 5, 10, 20, 40$, and 80 with five different random number sets for each n^{R}. The performance of PSO is tested for $n^{\mathrm{R}} = 80, 160, 320, 640$ and 1280 with five different random number sets for each n^{R}.

In Tables 6 and 7, the results of GA and PSO for the random instances, respectively, are presented, where each row shows the average of 20 results (4 instances × 5 random number sets). The results can be compared to the performance of $\mathsf{AID}_{\mathsf{rep}}$ in Table 5.

		$n^{\mathrm{R}} = 5$			$n^{\mathrm{R}} = 10$			$n^{\mathrm{R}} = 20$			$n^{\mathrm{R}} = 40$			$n^{\mathrm{R}} = 80$		
	n^{H}	Time	%best	Δ	Time	%best	Δ	Time	%best	Δ	Time	%best	Δ	Time	%best	Δ
S1	2	6	30%	0.0020	11	60%	0.0009	23	75%	0.0004	46	80%	0.0002	91	95%	0.0002
	3	7	55%	0.0037	13	70%	0.0020	27	90%	0.0003	54	95%	0.0001	108	95%	0.0001
	4	8	40%	0.0042	16	60%	0.0026	32	80%	0.0009	64	80%	0.0009	127	80%	0.0009
S2	2	6	80%	0.0005	11	80%	0.0002	23	95%	0.0001	45	100%	0.0000	91	100%	0.0000
	3	7	35%	0.0045	14	45%	0.0039	27	60%	0.0029	54	85%	0.0012	108	90%	0.0007
	4	8	55%	0.0018	16	55%	0.0014	32	60%	0.0011	64	65%	0.0006	127	70%	0.0003
S3	2	6	40%	0.0005	13	40%	0.0004	26	65%	0.0001	52	75%	0.0001	104	75%	0.0000
	3	8	30%	0.0015	16	40%	0.0007	32	40%	0.0005	63	55%	0.0003	127	60%	0.0002
	4	9	5%	0.0024	19	30%	0.0016	37	35%	0.0012	75	40%	0.0012	149	50%	0.0009
S4	2	7	25%	0.0031	15	35%	0.0020	30	50%	0.0017	59	60%	0.0014	118	70%	0.0009
	3	9	0%	0.0057	18	0%	0.0047	36	0%	0.0040	73	5%	0.0034	146	5%	0.0034
	4	11	10%	0.0108	22	15%	0.0092	43	20%	0.0074	87	25%	0.0057	173	25%	0.0051

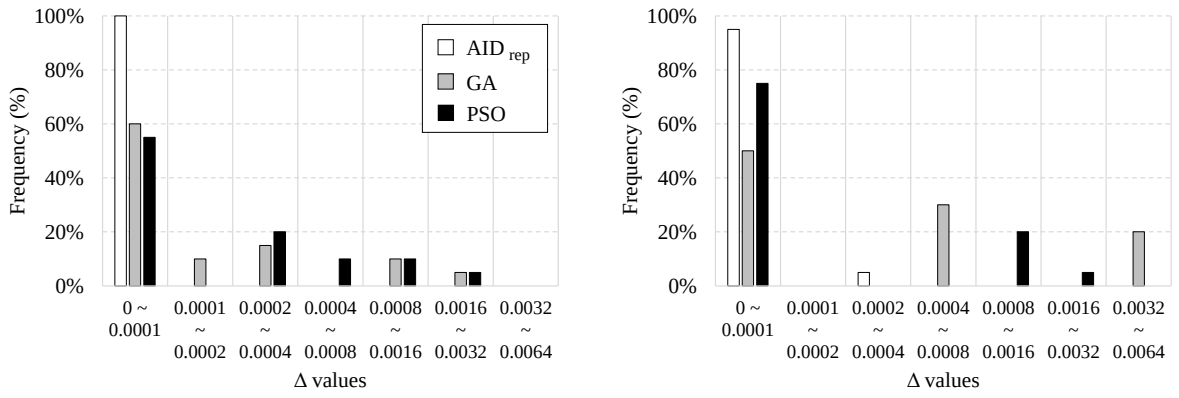
Table 6: Result of GA for random instances

Both algorithms show improving performances in increasing n^{R}. When n^{R} is larger, %*best* values are larger and Δ values are smaller. For a fixed system type, both algorithms tend to perform worse as n^{H} increases. The %*best* values decrease and Δ values increase rapidly as n^{H} increases. The average performances in Tables 6 and 7 imply that the solution quality of both metaheuristic algorithms can be very low for difficult problems when n^{H} is large. For example, consider System 4 with $n^{\mathrm{H}} = 4$. For 80 replications of GA, it takes 173 seconds to report a solution, the chance of getting the best known solution is 25% and the average gap from the best known objective value is 0.0051. For 1280 replications of PSO, it takes 155 seconds to report a solution, the chance of getting the best known solution is 15% and the average gap from the best known objective value is 0.0066. Between the two metaheuristic algorithms with the largest n^{R}, PSO performs slightly better than GA. The performance of PSO is perfect when $n^{\mathrm{H}} = 2$. However, the performance of PSO decreases rapidly as n^{H} increases.

		$n^{\mathrm{R}} = 80$			$n^{\mathrm{R}} = 160$			$n^{\mathrm{R}} = 320$			$n^{\mathrm{R}} = 640$			$n^{\mathrm{R}} = 1280$		
	n^{H}	Time	%best	Δ	Time	%best	Δ	Time	%best	Δ	Time	%best	Δ	Time	%best	Δ
S1	2	5	75%	0.0007	10	85%	0.0005	19	100%	0.0000	39	100%	0.0000	76	100%	0.0000
	3	6	45%	0.0012	12	50%	0.0010	24	70%	0.0006	48	90%	0.0001	96	100%	0.0000
	4	7	15%	0.0034	15	25%	0.0025	29	40%	0.0021	58	50%	0.0015	116	70%	0.0001
S2	2	5	95%	0.0000	10	100%	0.0000	20	100%	0.0000	39	100%	0.0000	77	100%	0.0000
	3	6	45%	0.0009	12	55%	0.0003	24	65%	0.0003	48	90%	0.0001	96	100%	0.0000
	4	7	45%	0.0016	15	60%	0.0005	29	65%	0.0005	58	85%	0.0001	116	85%	0.0001
S3	2	6	70%	0.0001	12	90%	0.0000	24	95%	0.0000	49	100%	0.0000	96	100%	0.0000
	3	7	10%	0.0014	14	15%	0.0010	29	35%	0.0006	58	45%	0.0004	115	55%	0.0003
	4	8	25%	0.0032	17	30%	0.0019	34	40%	0.0009	68	55%	0.0006	135	60%	0.0005
S4	2	7	60%	0.0002	13	75%	0.0001	26	85%	0.0001	52	90%	0.0001	103	100%	0.0000
	3	8	5%	0.0037	17	15%	0.0026	34	20%	0.0020	67	35%	0.0014	134	50%	0.0008
	4	10	0%	0.0173	19	0%	0.0138	39	0%	0.0117	78	0%	0.0102	155	15%	0.0066

Table 7: Result of PSO for random instances

Next, the gap from the best (Δ) values for the three algorithms ($\mathsf{AID}_{\mathsf{rep}}$ with $n^{\mathrm{R}} = 25$, GA with $n^{\mathrm{R}} = 80$, and PSO with $n^{\mathrm{R}} = 1280$) are compared to show how robust each algorithm is. In Figure 11, the distributions of Δ values are presented for System 3 with $n^{\mathrm{H}} = 3$ and 4. The two system types are selected because $\mathsf{AID}_{\mathsf{rep}}$ does not achieve 100% for %best. In each column chart of Figure 11, the horizontal and vertical axes are for the bins for Δ values and the frequencies, respectively. The column charts show that $\mathsf{AID}_{\mathsf{rep}}$ returns near-best solution (within 0.0001 from the best objective function value) with 100% and 95% chances. On the other hand, while GA and PSO have the average Δ values less than 0.001, bad solutions are occasionally returned.

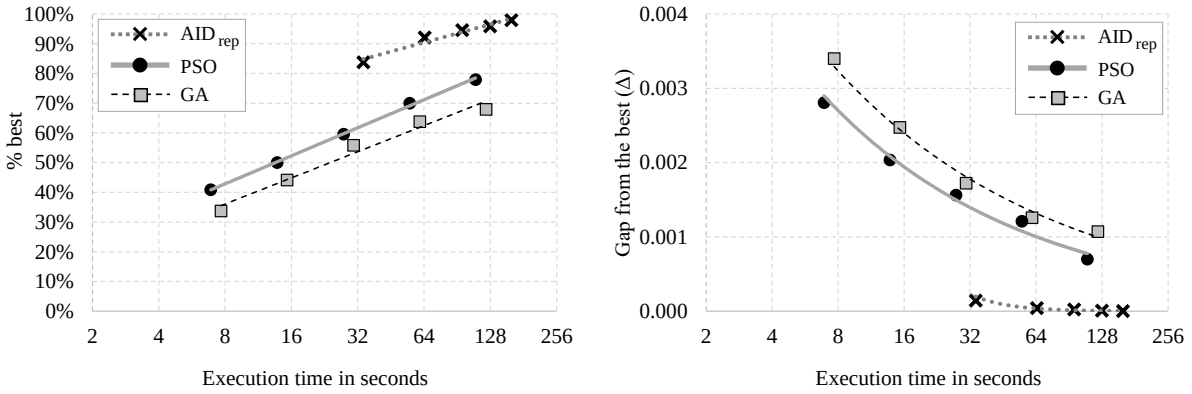


(a) System 3 with $n^{\mathrm{H}} = 3$

(b) System 3 with $n^{\mathrm{H}} = 4$

Figure 11: Distributions of Δ values of $\mathsf{AID}_{\mathsf{rep}}$, GA, and PSO

In Figure 12, the three algorithms are compared based on the scatter plots of execution time, *%best*, and Δ. The horizontal axes are for execution times in both plots and the vertical axes are for *%best* and Δ in Figures 12(a) and 12(b), respectively. The horizontal axes are in log scale to display the trend better. For all algorithms, the five markers of the corresponding series are for the result of the five n^{R} values. For each algorithm, a trend line is added to represent the trend of *%best* (or Δ) over changing execution times, where the equations are available in the caption. For *%best* and Δ, logarithmic and power functions are used, respectively.



(a) *%best*

(b) Δ

Figure 12: Scatter plots of *%best* and Δ against execution time. In Figure (a), equations for trend lines are $\%best = 13.61 + 14.53 \ln(\text{time})$ for PSO, $\%best = 9.65 + 12.69 \ln(\text{time})$ for GA, and $\%best = 53.75 + 8.81 \ln(\text{time})$ for $\mathsf{AID}_{\mathsf{rep}}$. In Figure (b), equations for trend lines are $\Delta = 0.0073(\text{time})^{-0.476}$ for PSO, $\Delta = 0.0079(\text{time})^{-0.431}$ for GA, and $\Delta = 2.2605(\text{time})^{-2.652}$ for $\mathsf{AID}_{\mathsf{rep}}$

The scatter plots clearly show that $\mathsf{AID}_{\mathsf{rep}}$ outperforms the two metaheuristic algorithms. Given a fixed time, it find the best solution with the highest chance and the average gap is the smallest among the algorithms compared. The equations of the trend lines also confirm the superiority of $\mathsf{AID}_{\mathsf{rep}}$. Given a fixed target $\%best = 99\%$, the expected execution times are 170, 496, and 1142 seconds for $\mathsf{AID}_{\mathsf{rep}}$, PSO, and GA, respectively. Given a fixed target $\Delta = 0.0001$, the expected execution times are 44, 8214, and 25284 seconds for $\mathsf{AID}_{\mathsf{rep}}$, PSO, and GA, respectively. For both criteria, $\mathsf{AID}_{\mathsf{rep}}$ achieves the target value with the fastest time.

5 Conclusion

In this paper, MILP models and AID algorithms are proposed for heterogeneous complex systems based on random samples. While there exist exact models and algorithms for homogeneous complex systems and heterogeneous series systems, there is no exact algorithm for the heterogeneous complex systems. Our MILP model is the first mathematical model with linear objective function approximating the nonconvex nonlinear system reliability function of complex system. Our models are formulated based on random samples, which is different from the existing models and algorithms, and the approximation errors decrease as sample size increases. The AID algorithm is also proposed using a new aggregation approach to improve solution quality and time.

Although the proposed models and algorithms are not exact, the computational experiment confirms the convergence to an optimal or near-optimal solution as sample size increases. It is also shown that an optimal or near-optimal solution can be obtained with a small number of samples. Among all versions of the algorithms proposed, $\mathsf{AID}_{\mathsf{rep}}$ performs the best in practice. The comparison with the two metaheuristic algorithms also confirms that $\mathsf{AID}_{\mathsf{rep}}$ is very competitive.

One of the straightforward limitations of the proposed algorithms is the high computational complexity. The algorithms have rapidly increasing execution times in problem size, as shown in Figure 10. Because of this limitation, complex heterogeneous systems with five, six, and seven subsystems are tested, whereas the well-known benchmark instances of Coit and Konak [9] consider serial heterogeneous systems with 20 subsystems. Hence, improving the efficiency of the current models and algorithms is necessary to solve the problem for larger complex systems.

Several future research directions are available. It should be worthwhile to improve the MILP model by reducing the number of constraints or variables, as the current models and algorithms cannot solve larger problems to optimality within a reasonable amount of time. In addition, because the model includes popular constraints such as Knapsack and assignment constraints, developing relaxation-based algorithms is promising. Finally, extending the models to other versions of RAP, such as cost minimization or maximization of percentile life of the system, can be studied.

References

[1] U. Abel and R. Bicker. Determination of all minimal cut-sets between a vertex pair in an undirected graph. *IEEE Transactions on Reliability*, R-31(2):167–171, 1982.

[2] K. K. Aggarwal. Redundancy optimization in general systems. *IEEE Transactions on Reliability*, 25(5):330–332, 1976.

[3] M. A. Ardakan, M. Sima, A. Z. Hamadani, and D. W. Coit. A novel strategy for redundant components in reliability–redundancy allocation problems. *IIE Transactions*, 48(11):1043–1057, 2016.

[4] S. Arunkumar and S. H. Lee. Enumeration of all minimal cut-sets for a node pair in a graph. *IEEE Transactions on Reliability*, R-28(1):51–55, 1979.

[5] R. Bellman and S. Dreyfus. Dynamic programming and the reliability of multicomponent devices. *Operations Research*, 6(2):200–206, 1958.

[6] M. Caserta and S. Voß. An exact algorithm for the reliability redundancy allocation problem. *European Journal of Operational Research*, 244(1):110 – 116, 2015.

[7] M.-S. Chern. On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters*, 11(5):309–315, 1992.

[8] D. W. Coit, T. Jin, and N. Wattanapongsakorn. System optimization with component reliability estimation uncertainty: a multi-criteria approach. *IEEE Transactions on Reliability*, 53(3):369–380, Sept 2004.

[9] D. W. Coit and A. Konak. Multiple weighted objectives heuristic for the redundancy allocation problem. *IEEE Transactions on Reliability*, 55(3):551–558, Sept 2006.

[10] D. W. Coit and A. E. Smith. Genetic algorithm to maximize a lower-bound for system time-to-failure with uncertain component weibull parameters. *Computers & Industrial Engineering*, 41(4):423 – 440, 2002.

[11] D. W. Coit, A. E. Smith, and D. M. Tate. Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS Journal on Computing*, 8(2):173–182, 1996.

[12] M. Djerdjour and K. Rekab. A branch and bound algorithm for designing reliable systems at a minimum cost. *Applied Mathematics and Computation*, 118(2):247 – 259, 2001.

[13] A. O. C. Elegbede, C. Chu, K. H. Adjallah, and F. Yalaoui. Reliability allocation through cost minimization. *IEEE Transactions on Reliability*, 52(1):106–111, 2003.

[14] R. Gordon. Optimum component redundancy for maximum system reliability. *Operations Research*, 5(2):229–243, 1957.

[15] C. Ha and W. Kuo. Reliability redundancy allocation: an improved realization for nonconvex nonlinear programming problems. *European Journal of Operational Research*, 171(1):24–38, 2006.

[16] K. O. Kim and W. Kuo. Percentile life and reliability as performance measures in optimal system design. *IIE Transactions*, 35(12):1133–1142, 2003.

[17] Y. H. Kim, K. E. Case, and P. M. Ghare. A method for computing complex system reliability. *IEEE Transactions on Reliability*, R-21(4):215–219, Nov 1972.

[18] S. Kulturel-Konak, B. A. Norman, D. W. Coit, and A. E. Smith. Exploiting tabu search memory in constrained problems. *INFORMS Journal on Computing*, 16(3):241–254, 2004.

[19] S. Kulturel-Konak, A. E. Smith, and D. W. Coit. Efficiently solving the redundancy allocation problem using tabu search. *IIE Transactions*, 35(6):515–526, 2003.

[20] W. Kuo and R. Wan. Recent advances in optimal reliability allocation. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 37(2):143–156, 2007.

[21] B. K. Lad, M. S. Kulkarni, and K. B. Misra. Optimal reliability design of a system. *Handbook of performability engineering*, pages 499–519, 2008.

[22] G. Levitin, A. Lisnianski, and D. Elmakis. Structure optimization of power system with different redundant elements. *Electric Power Systems Research*, 43(1):19 – 27, 1997.

[23] C.-Y. Li, X. Chen, X.-S. Yi, and J.-Y. Tao. Heterogeneous redundancy optimization for multi-state series–parallel systems subject to common cause failures. *Reliability Engineering & System Safety*, 95(3):202 – 207, 2010.

[24] Y.-C. Liang and A. E. Smith. An ant colony optimization algorithm for the redundancy allocation problem. *IEEE Transactions on Reliability*, 53(3):417–423, Sept 2004.

[25] P. Lin, B. Leon, and T. Huang. A new algorithm for symbolic system reliability analysis. *IEEE Transactions on Reliability*, 25(1):2–15, 1976.

[26] K. B. Misra. An algorithm for the reliability evaluation of redundant networks. *IEEE Transactions on Reliability*, R-19(4):146–151, Nov 1970.

[27] Y. W. Park. Optimization for l1-norm error fitting via data aggregation. *arXiv preprint arXiv:1703.04864*, 2017.

[28] Y. W. Park and D. Klabjan. An aggregate and iterative disaggregate algorithm with proven optimality in machine learning. *Machine Learning*, 105(2):199–232, 2016.

[29] V. R. Prasad, W. Kuo, and K. O. Kim. Maximization of a percentile life of a series system through component redundancy allocation. *IIE Transactions*, 33(12):1071–1079, Dec 2001.

[30] H. S. Ryoo. Robust metaheuristic algorithm for redundancy optimization in large-scale complex systems. *Annals of Operations Research*, 133(1-4):209–228, 2005.

[31] M. Sallak, C. Simon, and J.-F. Aubry. A reliability graph approach for availability and redundancy allocation: application to safety instrumented systems. *Technical report*, 2009.

[32] V. K. Sharma, M. Agarwal, and K. Sen. Reliability evaluation and optimal design in heterogeneous multi-state series-parallel systems. *Information Sciences*, 181(2):362 – 378, 2011.

[33] C. Sung and Y. Cho. Reliability optimization of a series system with multiple-choice and budget constraints. *European Journal of Operational Research*, 127(1):159 – 171, 2000.

[34] C. S. Sung and Y. K. Cho. Branch-and-bound redundancy optimization for a series system with multiple-choice constraints. *IEEE Transactions on Reliability*, 48(2):108–117, 1999.

[35] R. Tavakkoli-Moghaddam, J. Safari, and F. Sassani. Reliability optimization of series-parallel systems with a choice of redundancy strategies using a genetic algorithm. *Reliability Engineering & System Safety*, 93(4):550 – 556, 2008.

[36] Y. Wang and L. Li. Heterogeneous redundancy allocation for series-parallel multi-state systems using hybrid particle swarm optimization and local search. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 42(2):464–474, March 2012.

[37] Y. Wang and L. Li. A pso algorithm for constrained redundancy allocation in multi-state systems with bridge topology. *Computers & Industrial Engineering*, 68:13 – 22, 2014.

[38] H. Yu, J. Yang, and J. Han. Classifying large data sets using svm with hierarchical clusters. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 306–315, 2003.

[39] H. Yu, J. Yang, J. Han, and X. Li. Making SVMs scalable to large data sets using hierarchical cluster indexing. *Data Mining and Knowledge Discovery*, 11(3):295–321, 2005.