# Optimization Techniques and Formal Verification for the Software Design of Boolean Algebra Based Safety-Critical Systems

Jon Perez, *Senior Member, IEEE*, Jose Luis Flores, Christian Blum, Jesús Cerquides, and Alex Abuin

***Abstract*—Artificial intelligence, and the ability to learn optimized solutions that comply with a set of safety rules, could facilitate the human-based design process of safety-critical systems. However, the reconciliation of state-of-the-art artificial intelligence technology with current safety standards and safety engineering processes is a challenge to be addressed. In this article, this publication describes a method based on optimization and on formal verification for the design of safety-critical systems that are defined by Boolean algebra. Several diverse optimization techniques and a hybrid of these approaches are used to find an optimized design that considers performance requirements, availability rules, and complies with all defined safety rules. Subsequently, this solution is translated into an alternative knowledge representation that can be formally verified and developed in compliance with currently considered safety standards. This method is evaluated with a simplified safety-critical case study.**

***Index Terms*—Ant colony optimization (ACO), artificial intelligence (AI), estimation of distribution algorithm (EDA), formal verification, functional safety, hybrid algorithm, iterated local search (ILS).**

## I. Introduction

ARTIFICIAL intelligence (AI) is at the core of recent scientific and industrial advancements and, in some applications, it is used to support safety-critical decisions where errors can lead to catastrophic and fatal consequences [1]–[4], [5]–[7]. Driven by research challenges such as autonomous driving, there is a substantial research effort to define AI solutions for the development of safety-critical systems [3], [4], [8], [9], aligned with the required evolution and definition of new safety standards. This is also of interest in other domains—for example, in industrial domains such as railway interlocking—where AI solutions could also be used to develop safety-critical systems [1], [10], [11].

Safety-critical systems, such as industrial safety protection systems, may cause a catastrophic event in case of failure (e.g., loss of human lives). They are, therefore, developed and certified with domain specific safety standards such as IEC 61508 [12] (industrial) and EN 50128 [13] (railway). The safety criticality is defined by means of a safety integrity level (SIL) value with a range from 1 to 4 [12]. For the highest criticality (SIL4), the probability of a dangerous failure is in the range of $10^{-9}$ per hour of operation, that is, approx. one dangerous failure every 114.155 years. Achieving such a low probability of a dangerous failure requires compliance with strict safety methods and techniques, in order to mitigate systematic errors (e.g., design method to reduce human, process, and tool errors) and random errors (e.g., diagnostics, fault tolerance).

Boolean algebra, or Boolean logic, is commonly used for the development of safety protection and safety control systems in industrial domains. Examples include machinery and manufacturing protection systems [14]–[17], the wind turbine safety chain (SIL3) [18], the lift safety chain and compensatory means (SIL3) [19] and railway interlocking systems (SIL4) [1], [20]. Although the computational complexity of Boolean algebra is considered to be low and design methods are mature, the effort required to find a safe and optimal design increases with growing design space dimensions to be considered by human safety engineers. A design must meet all safety rules. Moreover, it should be optimal with regard to availability and application specific performance measurements, such as maximizing the fluidity of trains in railway interlocking systems [20]. This leads to high development and certification costs, where human cognitive limitations [21] could lead not only to suboptimal designs but also to potential systematic errors.

The use of AI to assist humans in the design of safe and optimal solutions could facilitate the development of safety-critical systems, such as the examples previously described [1], as long as the generated solution is safe for its purpose and compliant

Manuscript received January 2, 2021; revised February 9, 2021 and March 23, 2021; accepted April 12, 2021. Date of publication April 20, 2021; date of current version September 29, 2021. This work was supported in part by Basque Government under Grant KK-2019-00 035, in part by the European Community's Horizon 2020 programme under Grant Crowd4SDG, in part by Humane-AI-net projects under Grant 872944 and Grant 761758, and in part by the Spanish Ministry of Science and Innovation under Grant PID2019-104156GB-I00. Paper no. TII-21-0016. *(Corresponding author: Christian Blum.)*

Jon Perez, Jose Luis Flores, and Alex Abuin are with Ikerlan Technology Research Centre, Basque Research and Technology Alliance (BRTA), 20850 Mondragón, Spain (e-mail: JMPerez@ikerlan.es; jlflores@ikerlan.es; aabuin@ikerlan.es).

Christian Blum and Jesús Cerquides are with Artificial Intelligence Research Institute (IIIA-CSIC), 08193 Bellaterra, Spain (e-mail: christian.blum@iiia.csic.es; cerquide@iiia.csic.es).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2021.3074394.

Digital Object Identifier 10.1109/TII.2021.3074394

with associated safety standards. However, current AI tools have several limitations with respect to ensuring the required systematic fault avoidance and the compliance with current safety standards. Examples concern "black box" limitations regarding the interpretability and analyzability of the solution [2], [4], [7], and compliance limitations with respect to the V-model development activities such as specification completeness and correctness, verification, validation, and testing [1], [3], [5], [6], [8], [9], [22], [23]. Because of this, AI techniques are generally considered not recommended for safety-critical systems [1]. However, as stated by Nordland, "what we need are methods for assessing and certifying processes" for the development of AI-based safety-critical systems, "and when processes can be certified, AI should be simply renamed to automated process and will be acceptable for safety related applications" [1].

This article contributes with the definition of an IEC 61508 (industrial) and EN 50128 (railway) compliant safety software design method for Boolean algebra based safety-critical systems, based on diverse optimization techniques and formal verification. For that purpose, different optimization techniques are used to find a safe design that meets all defined safety rules and is high-performing with respect to availability and performance criteria. Subsequently, this design is translated into an alternative knowledge representation that can be analyzed, formally verified and developed in compliance with selected industrial and railway safety standards. Several conceptually different optimization algorithms are used in the considered case study [estimation of distribution algorithms (EDA), iterated local search (ILS), ant colony optimization (ACO)]. A hybridization of EDA and ILS is overall the best-performing method. Finally, the formal verification activity allows the prior use of optimization tools and algorithms that are not qualified for the design of safety-critical systems.

The rest of this article is organized as follows. Section II describes basic concepts. Section III describes the proposed safety design method and Section IV outlines a simplified railway signaling case study to which the proposed method is applied and the results are explained. Note that a simplified example was chosen to show the cross-domain applicability of the approach. Finally, Section V concludes this article.

## II. Preliminaries

*AI optimization techniques*. In the context of this article, the design of Boolean algebra based safety-critical systems will be modeled as a binary optimization problem. Algorithms for solving such problems range from exact techniques that guarantee to deliver an optimal solution in bounded space and time, to heuristic techniques. However, as we will need to solve large-scale problems with various objective functions related to safety, availability, and performance, metaheuristics [24] are generally the best option. Note also that metaheuristics come with the advantage of being general ideas that can basically be adapted to any optimization problem. This means that an adaptation to a specific problem can easily be adapted to similar problems. In order to study the suitability of different metaheuristics for the case study considered in this article, we decided to implement the following three approaches, being conceptually quite different from each other.

An EDA [25], [26] is an evolutionary algorithm that samples new individuals (solutions) at each iteration from a probability distribution. In fact, an EDA algorithm can be classified depending on the complexity of the probabilistic model used to capture the interdependencies between the variables used to model the tackled problem. Univariate EDAs, for example, do not consider any dependencies, bivariate variants consider pair-wise dependencies, while multivariate approaches are the most complex ones. We decided to implement a univariate marginal distribution algorithm (UMDA) [27], which is a univariate approach that assumes that all variables are independent. For detailed information about the characteristics and different algorithms that constitute the family of EDAs, see [25] and [26].

ILS [28] is an extension of local search (hill climbing). Given a local search method, the algorithm works as follows. First, an initial solution is generated in some way. Subsequently, local search is applied to the initial solution in order to obtain the first incumbent solution. At each iteration, ILS algorithms apply three basic steps. First, a so-called perturbation mechanism is applied to the incumbent solution, resulting in a perturbed solution. Second, local search is applied to the perturbed solution resulting in an alternative local minimum. Third, the incumbent solution for the next iteration is selected between the current incumbent solution and the produced alternative local minimum. ILS is said to be able to produce high-quality solutions often very quickly. On the other side, the algorithm may sometimes fail to find the very best solutions.

ACO [29] is an optimization technique inspired by the foraging behavior of natural ant colonies. At each iteration, first, a number of solutions to the tackled problem is constructed in a probabilistic way, based on greedy information and on so-called pheromone information. The best solutions from the current iteration, possibly in addition to solutions from previous iterations, are then used to update the pheromone information. Over time, the algorithm learns to produce better and better solutions.

*Formal verification—model checking*. Formal verification uses formal methods with "mathematically rigorous techniques and tools" [13] for the verification of a given algorithmic design. Model checking is an example of a formal verification technique. The model checker provides either a positive answer whenever a set of properties are proved to be satisfied, or a counter-example that shows that the design violates a given property. For that purpose, the design is modeled as a state-transition design or a state machine, and input properties are specified in temporal logic. NuSMV, for example, is an open source symbolic model checking tool, which allows us to express the properties to be verified using linear temporal logic (LTL) [30].

*Boolean algebra*. Boolean algebra uses a set of rules and laws to perform Boolean operations (e.g., $\vee$, $\wedge$) on Boolean variables that can only have two possible values ["0" (false) or "1" (true)]. Safety-critical systems that manage a set of safety digital outputs $(Y)$ based on the readings of a set of digital inputs $(X)$ are commonly developed using Boolean algebra based logic functions ($Y = F(X)$). Moreover, they are implemented with safety relays, programmable electronic, and/or software.

*Safety standards analysis (Software)*. IEC 61508 [12] is considered a reference safety standard by several domain specific standards such as, for example, in the railway domain (EN 50128 [13]), in industrial machinery (ISO 13849 [31]) and in the construction and installation of lifts (EN 81-20/21/50 [32]–[34]). For further details with respect to safety standards and certification processes see [35].

The previously referenced standards are characterized by a considerable variability of domain-specific terms and requirements. For this reason, and in order to ease the description and comprehension, the proposed design method is described using IEC 61508-3 and EN 50128 standards. However, our design method has also been defined taking into consideration additionally referenced standards, which basically refer to IEC 61508-3 techniques and requirements:

1) Wind turbines: As described in [18] and [35] and in applicable certification guidelines [36], the safety chain protection system design shall at least comply with the ISO 13849 standard. This standard references IEC 61508-3 for the development of safety-related embedded software (4.6.2), and basically recommends a subset of techniques and requirements referenced from IEC 61508-3/7.
2) Lifts: The design of safety protections, such as safety-chain and compensatory measures, shall at least consider compliance with applicable EN 81-20/21/50 safety standards that specify required safety functions and safety rules. With respect to safety software, these standards directly use or refer to a subset of techniques defined in IEC 61508-3 (e.g., EN 81-50 Table B.2).

Both IEC 61508-4 (3.2.11) and EN 50128 (3.1.42-44) provide a classification of software development tools: a class T1 tool "generates no output which can directly or indirectly contribute to the executable code," a T2 tool "supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects" and a T3 tool "generates outputs which can directly or indirectly contribute to the executable code" [12], [13]. Moreover, both standards describe tool requirements (IEC 61508-3 7.4.4; EN 50128 6.7), such as tool qualifications for T2 and T3 classes (IEC 61508-3 7.4.4) or, alternatively, perform an independent verification of the tool results as if they had been obtained manually (EN 50128 6.7.1.1).

## III. Method—Safety Software Design

This section describes the proposed safety software design method for Boolean algebra based safety-critical systems, defined in compliance with the V-model design phases, activities, and technical requirements of the considered safety standards [12], [13]. As summarized in Fig. 1, this design method proposes an optimization-based module design ["optimization module design" (B.1) and "formal verification" (B.2)] that aims to facilitate the human-based module design activity (B). The boxes shown with gray background in Fig. 1 represent V-model phases (e.g., module design) and activities (B.2) that are performed by qualified safety engineers with state-of-the-art safety techniques and tools.

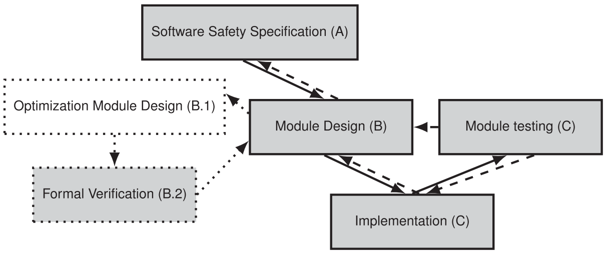

Software Safety Specification (A)
Optimization Module Design (B.1)
Module Design (B)
Module testing (C)
Formal Verification (B.2)
Implementation (C)

Fig. 1. Module design process.

TABLE I
Safety Rules [41]

| Number | Rule |
|---|---|
| Rule 1 | $(u_1) \rightarrow (\neg l_1)$ |
| Rule 2 | $(u_6) \rightarrow (\neg l_6)$ |
| Rule 3 | $(s_1) \rightarrow (\neg u_2)$ |
| Rule 4 | $(s_2) \rightarrow (\neg l_2)$ |
| Rule 5 | $(s_3) \rightarrow (\neg u_3)$ |
| Rule 6 | $(s_3) \rightarrow (\neg u_4)$ |
| Rule 7 | $(s_4) \rightarrow (\neg u_5)$ |
| Rule 8 | $(s_5) \rightarrow (\neg l_5)$ |
| Rule 9 | $(\neg u_3 \wedge \neg l_3) \vee (\neg u_5 \wedge \neg l_5) \rightarrow \neg u_1$ |
| Rule 10 | $(\neg u_3 \wedge \neg l_3) \vee (\neg u_5 \wedge \neg l_5) \rightarrow \neg l_1$ |
| Rule 11 | $(\neg u_2 \wedge \neg l_2) \vee (\neg u_4 \wedge \neg l_4) \rightarrow \neg u_6$ |
| Rule 12 | $(\neg u_2 \wedge \neg l_2) \vee (\neg u_4 \wedge \neg l_4) \rightarrow \neg l_6$ |
| Rule 13 | $(\neg u_2 \wedge \neg l_2) \rightarrow (\neg u_7)$ |
| Rule 14 | $(\neg u_5 \wedge \neg l_5) \rightarrow (\neg u_8)$ |
| Rule 15 | $(u_3) \rightarrow (\neg u_4)$ |
| Rule 16 | $(l_3) \rightarrow (\neg l_4)$ |
| Rule 17 | $(u_2) \rightarrow (\neg l_2)$ |
| Rule 18 | $(u_3) \rightarrow (\neg l_3)$ |
| Rule 19 | $(u_4) \rightarrow (\neg l_4)$ |
| Rule 20 | $(u_5) \rightarrow (\neg l_5)$ |
| Rule 21 | $(u_1 \vee l_1) \rightarrow (\neg u_7)$ |
| Rule 22 | $(u_6 \vee l_6) \rightarrow (\neg u_8)$ |

The described design method should be considered a reference method that must be adapted for a given application, domain and standard. Safety and performance measures are considered a cross-domain common minimum for any considered application. The three objective functions (relating to safety, availability, and performance) proposed in the context of the case study (see Section IV) are representative for applications such as railway interlocking [20].

### A. Software Safety Specification

As stated by IEC 61508-3 (7.2.2) and EN 50128 (7.2.4), the software safety specification defines at least the safety function(s) and their associated SILs. In addition, it must also provide the required details to allow an appropriate design, implementation, and assessment. For example, the specification of Boolean algebra based safety functions requires the specification of safety inputs ($X$) and outputs ($Y$), safety functionality ($F$), safety rules (e.g., Table I) and additional constraints—for example, related to performance—with which the design must comply (e.g., [18], [20]).

This specification is the same, regardless of the selected design process, which may, or not, be based on optimization assistance. Requirements could be managed with common class T1 tools such as text editors and specialized requirement management tools, because the specification deliverable is subject to an independent verification activity (IEC 61508-3 7.9.2.8; EN 50128 7.2.4.21) to assess the completeness and the correctness.

However, in this method, the safety specification (e.g., safety rules) common to both the optimization module design (B.1) and the formal verification (B.2), shall also be formalized with the formal method notation (IEC 61508-3 Table A.1) selected for the formal verification (model checker language).

## B. Module Design

This section provides a technical description of the specific optimization and formal validation activities to be carried out for the (detailed) software module design of the previously specified Boolean algebra based safety function(s), in accordance with associated software design requirements described in IEC 61508-3 (7.4.5) and EN 50128 (7.4). This is the design to be implemented (C). Prior to this design—and regardless of the selected design process; with or without assistance—it is assumed that the required activities and requirements for the software architecture and software design have been handled according to IEC 61508-3 (7.4; 7.4.3) and EN 50128 (7.3).

The proposed module design (B) starts with the optimization-based module design (B.1) and subsequently with the formal verification (B.2) to ensure that the proposed design complies with all safety rules.

### 1) Optimization Module Design (B.1):

The optimization module design starts with the definition of the optimization problem to be solved, prior to the selection of an appropriate optimization algorithm, the execution of the learning process—that is, the application of the selected algorithm to the defined problem—and finally the knowledge transformation.

*Definition of the optimization problem*. First, a solution to the problem consists of outputs for each configuration of inputs. A solution is evaluated by three different basic objective functions: safety rules evaluation (SRE), availability evaluation (AE), and performance evaluation (PE). Hereby, the objective function that is concerned with the fulfillment of the safety rules (SRE) is certainly the most important one, because a solution that does not fulfill all safety rules cannot be implemented in practise. Secondary objective functions are concerned with availability (AE) and performance criteria (PE). In principle, there are different ways to handle multiple objective functions. They may be combined as a weighted sum (or product) into a single-objective function, assigning higher weights to objective functions that are more important than others [37]. Another option is to define a lexicographic objective function that makes use of the objective functions in terms of an explicitly ordered list [38]. Finally, a third option is to solve a real multiobjective optimization problem based on Pareto optimality concepts [39].

*Selection of an appropriate optimization algorithm*. The selection of an appropriate optimization technique depends very much on the type and on the nature of the optimization problem under consideration. Important factors are, for example, the nature of the objective function and the type and the number of the constraints. Therefore, we recommend to implement and test several diverse and conceptually different algorithms in each case.

*Solving the problem*. The process of solving the previously defined problem with the chosen algorithms starts with tuning the algorithm parameters, that is, finding values for the algorithms' parameters such that the performance of each algorithm is as good as possible. Subsequently, the algorithms are applied to the problem. This process ends when one or more solutions meet all safety rules (SRE) and when the evaluations computed for other performance criteria such as availability and performance (AE and PE) are considered sufficiently high.

*Knowledge transformation*. The output of the optimization is implementation specific (e.g., binary vector), and a knowledge transformation is required to translate it into a truth table or into Boolean algebra expressions that can afterward be formally verified (B.2) and implemented (C) in compliance with applicable safety standards.

This activity is required to transform the potential "black box" design provided by the optimization algorithm into a "white box" representation (truth table) that supports interpretability, analizability, and a subsequent formal verification (B.2).

In addition to this, a truth table can easily be transformed into Boolean expressions using the Quine-McCluskey algorithm [40], and Boolean expressions can also be transformed into a truth table. The selection of the required representation is application specific (e.g., selected model checker).

### 2) Formal Verification (B.2):

The design is then subject to a formal verification activity, in compliance with applicable requirements such as IEC 61508-3 (Table A.5, C.5.12) and EN 50128 (D.28), using a model checking tool in order to "symbolically examine the entire state space" and "establish a correctness or safety property that is true for all possible inputs" [13]. The formal verification activity supports an independent verification of the design tool results that assist human designers in the generation of a design output, "which can directly or indirectly contribute to the executable code" (T3) [12], enabling the usage of currently available nonqualified optimization software, libraries, and tools in the design phase.

The compliance with safety rules has already been evaluated during the optimization module design by the SRE evaluation criteria. However, neither the optimization software, libraries, and tools are qualified (T2, T3), nor the AI researcher is required to be a qualified safety engineer, nor the required optimization design process itself needs to comply with safety standards, methods, techniques, and constraints. For this reason, as previously described, a formal verification activity is proposed. And the model software itself (to be executed by the model checker) shall also be independently verified.

Finally, the selection and usage of a qualified formal verification tool (T2) is a certification project decision beyond the scope of this publication, which could potentially simplify some of the required tool analysis, justifications, gathering of evidences and previously described verification activities (e.g., simulink design verifier, prover [30]).

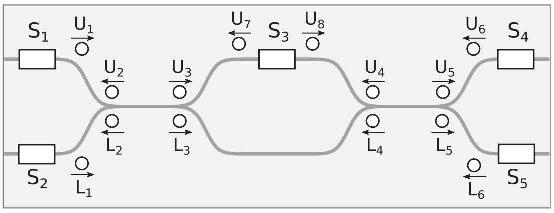


Fig. 2. Simplified railway interlocking case study. Reproduced from [41].

### C. Implementation and Module Testing

If the formal verification result is satisfactory, the optimization-based design represented as a truth table or as Boolean algebra expressions can be implemented as a software module. The implementation and testing can be performed with state-of-the-art techniques (e.g., [1], [20], [18]) as described in IEC 61508 (7.4.6, 7.4.7) and EN 50128 (7.5). This is because the implementation and testing activities do not depend on the selected design process, which may work with or without optimization assistance.

## IV. Railway Signaling Case-Study

In order to demonstrate the potential usefulness of the method described in the previous section, we consider the simplified railway interlocking case study described in [41] as a guiding simplified example. In this way, we intend to show the cross-domain applicability of our method for the development and certification of safety-critical systems up to the highest integrity level (SIL4). This public case study has been chosen to facilitate the comprehension, analysis, and reproduction of the results as opposed to using private case studies with application specific complexities (e.g., [1], [18], [19]) such as detailed characteristics and constraints of the railway interlocking domain [20].

### A. Optimization Module Design

*1) Definition of the Optimization Problem:* Fig. 2 shows a simplified railway interlocking system defined in [41], supporting the safe bidirectional movement of trains in five railway sections ($\{S_1,\ldots,S_5\}$). As explained in [1], a railway interlocking system is a computer-based SIL4 system that controls railway objects (e.g., signals) in a delimited geographical area based on static design time information (e.g., track layout) and dynamic input information (e.g., sections state).

At any time, a section $S_j$ has exactly one of two possible values, that is, $[0,1] = [\text{free}, \text{busy}]$. Moreover, for each section $S_j$, there is a binary variable $D_j \in \{0,1\}$. In case $S_j = 1$ (that is, section $S_j$ is busy), the value of $D_j$ indicates the direction of the train at $S_j$. More specifically, if $D_j = 0$, the intention of the train at $S_j$ is to move to the left, to the right otherwise. Note that each possible setting of the variables in $\{S_1,\ldots,S_5,D_1,\ldots,D_5\}$ is called a *train configuration* $X^i$, which is represented by a binary vector

$$X^i = (s_1^i,\ldots,s_5^i,d_1^i,\ldots,d_5^i) \tag{1}$$

where $s_j^i \in \{0,1\}$ is the value of $S_j$ in configuration $X^i$, and $d_j^i \in \{0,1\}$ is the direction of the train at $S_j$ in configuration $X^i$. It is easy to verify that there are exactly 243 different train configurations. They form the set of input configurations. For each input configuration $X^i$, the setting of 14 traffic light object based signals ($\{U_1,\ldots,U_8,L_1,\ldots,L_6\}$) is required; see Fig. 2. In this simplified example, traffic light objects can only have two possible values, $[0,1] = [\text{red}, \text{green}]$, where "0" is considered the default safe state (traffic light state "red"). A vector of output values for a train configuration $X^i$ is henceforth denoted by

$$Y^i = (u_1^i,\ldots,u_8^i,l_1^i,\ldots,l_6^i). \tag{2}$$

In other words, a candidate setting $Y^i$ of the traffic lights for a specific train configuration $X^i$ is represented by a binary vector of 14 elements. The first eight elements, $(u_1^i, ..., u_8^i)$, represent the states of the eight upper track traffic lights, while the following six elements, $(l_1^i, ..., l_6^i)$, represent the states of the six lower track traffic lights.

Summarizing, a complete candidate solution $(Y)$ provides traffic light settings (outputs) for all $N = 243$ train configurations, where $Y^i$ is defined in (2)

$$Y = (Y^1,\ldots,Y^N). \tag{3}$$

A possible candidate solution $Y$ is evaluated according to three different objective functions: SRE, AE, and PE. Each of them will be described in the following in detail. The safety function requirement can be informally described as "avoid the collision of train(s) by proper traffic light settings $(Y)$ dependent on the section occupation and direction values $(X)$." In other words, two trains entering the same section simultaneously is a situation that must be avoided. This safety requirement is encoded by means of a set of 22 Boolean rules defined in [41], each one expressed by a premise and a conclusion; see also Table I. Given a candidate solution $Y$, the safety-related objective function value $\text{SRE}(Y)$ is defined as follows:

$$\text{SRE}(Y) = \sum_{i=1}^{N}\sum_{k=1}^{22} r_k^i \tag{4}$$

where $r_k^i \in \{0,1\}$ is a binary value with the following meaning. If $r_k^i = 0$, the $k$th rule from Table I is fulfilled, while with $r_k^i = 1$, this is not the case. In other words, function SRE() counts the total number of safety rule violations of a candidate solution $Y$. This function must be minimized.

The second function, AE(), is concerned with availability, which can informally be described as "avoid trains getting blocked by constant red traffic lights." Availability is often evaluated by simulation in sophisticated railway signaling systems. However, for the purpose of our simplified case study, availability is measured as follows. First, we identify the following six situations in which availability is important.

1) Sit. 1: train at $S_1$, direction "right" (traffic light: $u_1$).
2) Sit. 2: train at $S_2$, direction "right" (traffic light: $l_1$).
3) Sit. 3: train at $S_3$, direction "right" (traffic light: $u_8$).
4) Sit. 4: train at $S_3$, direction "left" (traffic light: $u_7$).
5) Sit. 5: train at $S_4$, direction "left" (traffic light: $u_6$).
6) Sit. 6: train at $S_5$, direction "left" (traffic light: $l_6$).

Let $\delta_k \subset \{1, \ldots, N\}$ for each situation $k = 1, \ldots, 6$ be the set of all indices of those train configurations in which the respective situation is present. $\delta_1$, for example, contains the indices of all train configurations with a train at section $S_1$ that wants to leave to the right. Given a solution $Y$, function $\text{AE}_k()$ measures for each situation $k = 1, \ldots, 6$, the fraction of train configurations in which the respective train has a green light. In the case of $k = 1$, for example, $\text{AE}_k(\text{Y})$ is defined as follows:

$$\text{AE}_1(Y) = \frac{\sum_{i \in \delta_1} u_1^i}{|\delta_1|}. \tag{5}$$

In the case of the remaining five situations, function $\text{AE}_k()$ is defined correspondingly. Finally, the complete availability measure $\text{AE}(Y)$ of a solution $Y$ is defined as follows:

$$\text{AE}(Y) = \min_{k=1,\ldots,6} \text{AE}_k(Y). \tag{6}$$

This function must be maximized, that is, we intend to maximize—as much as possible—the availability for trains at the railway section that is worst off. Nevertheless, we noticed that maximizing the average availability by means of a subordinate objective function helps the algorithms to maximize AE(). Therefore, $\text{AEA}(Y) = (\sum_{k=1}^{6} \text{AE}_k(Y))/6$ is also considered (see as follows).

Finally, the last function, PE(), measures the performance of a solution in terms of the "railway network traffic flow capability." This performance measure balances the presence of trains and the possibilities to allow the trains to proceed. If there are no trains, it does not matter how many traffic lights are set to green, and if there are many available trains but no green lights it is not possible to transport passengers. For the purpose of our simplified case study, the performance-related objective function value $\text{PE}(Y)$ is defined as follows:

$$\text{PE}(Y) = \sum_{i=1}^{N} \left( \sum_{j=1}^{5} s_j^i \right) \times \left( \sum_{j=1}^{8} u_j^i + \sum_{j=1}^{6} l_j^i \right). \tag{7}$$

As the three objective functions can be clearly ordered according to decreasing importance [first SRE(), then AE(), and finally PE()], we decided to tackle this problem by means of a lexicographic objective function, $F()$, which is indirectly defined as follows. Given two solutions $Y$ and $Y'$, it holds that $F(Y) < F(Y')$ if and only if:

1) $\text{SRE}(Y) < \text{SRE}(\text{Y'})$ or
2) $\text{SRE}(Y) = \text{SRE}(\text{Y'})$ and $\text{AE}(Y) > \text{AE}(Y')$ or
3) $\text{SRE}(Y) = \text{SRE}(\text{Y'})$ and $\text{AE}(Y) = \text{AE}(Y')$ and $\text{AEA}(Y) > \text{AEA}(Y')$ or
4) $\text{SRE}(Y) = \text{SRE}(\text{Y'})$ and $\text{AE}(Y) = \text{AE}(Y')$ and $\text{AEA}(Y) = \text{AEA}(Y')$ and $\text{PE}(Y) > \text{PE}(Y')$.

*2) Implementation of the Optimization Algorithms:* As already mentioned in Section II, we decided to implement an EDA variant, which is known as UMDA. The pseudocode of this EDA is given in Algorithm 1. As input, the algorithm requires values for the following three parameters.

1) $p_{\text{size}}$: the population size.
2) $n_{\text{sel}}$: the number of solutions chosen from the population for the estimation of the new probability distribution $\mathbf{D}$.
3) $c_{\text{lim}}$: the maximum number of consecutive iterations without improvement of the restart-best solution $Y^{\text{rb}}$.

**Algorithm 1:** EDA.

```
 1: input: values for parameters $p_{\text{size}}$, $n_{\text{sel}}$, $c_{\text{lim}}$
 2: $P = \texttt{GenerateInitialPopulation}(n_{\text{sel}})$
 3: $Y^{\text{pbest}} = \texttt{BestSolutionFrom}(P)$
 4: $Y^{\text{bsf}} = Y^{\text{pbest}}$, $Y^{\text{rb}} = Y^{\text{pbest}}$, $c_{\text{noimpr}} = 0$
 5: while CPU time limit not reached do
 6:   if $c_{\text{noimpr}} \geq c_{\text{lim}}$ then
 7:     $P = \texttt{GenerateInitialPopulation}(n_{\text{sel}})$
 8:     $Y^{\text{pbest}} = \texttt{BestSolutionFrom}(P)$
 9:     $Y^{\text{rb}} = Y^{\text{pbest}}$
10:     $c_{\text{noimpr}} = 0$
11:   end if
12:   $P_{sel} = \texttt{Select}(P, n_{\text{sel}})$      $\triangleright P_{sel} \subseteq P$
13:   $\mathbf{D} \leftarrow \texttt{EstimateDistribution}(P_{sel} \cup Y^{\text{pbest}})$
14:   $P \leftarrow \texttt{SampleDistribution}(\mathbf{D}, p_{\text{size}})$
15:   $Y^{\text{pbest}} = \texttt{BestSolutionFrom}(P)$
16:   if $F(Y^{\text{pbest}}) < F(Y^{\text{rb}})$ then
17:     $Y^{\text{rb}} = Y^{\text{pbest}}$
18:     $c_{\text{noimpr}} = 0$
19:   else
20:     $c_{\text{noimpr}} = c_{\text{noimpr}} + 1$
21:   end if
22:   if $F(Y^{\text{rb}}) < F(Y^{\text{bsf}})$ then $Y^{\text{bsf}} = Y^{\text{rb}}$
23: end while
24: output: $Y^{\text{bsf}}$, the best solution found
```

At the start of the algorithm, an initial population of solutions $(P)$ is generated uniformly at random (line 2). Moreover, the best-so-far solution $Y^{\text{bsf}}$ and the restart-best solution $Y^{\text{rb}}$ are initialized with the population-best solution $Y^{\text{pbest}}$, and the counter for consecutive nonimproving iterations $(c_{\text{noimpr}})$ is initialized to zero (line 4). Then, the following cycle is repeated until the CPU time limit is reached. First, in lines 6–11, the algorithm performs a restart if necessary, that is, if $c_{\text{noimpr}} \geq c_{\text{lim}}$. Second, the best $n_{\text{sel}}$ solutions from $P$ are selected and stored in $P_{\text{sel}}$ (line 12). Third, a probability distribution $\mathbf{D}$ is estimated in which the probability $\mathbf{p}(y_j = 1)$ for generating value "1" for position $j$ of a solution $Y$ is defined as follows:

$$\mathbf{p}(y_j = 1) = \frac{|\{Y' \in P_{\text{sel}} \cup Y^{\text{bsf}} \text{ s.t. } {y'}_j = 1\}|}{|P_{\text{sel}} \cup Y^{\text{bsf}}|}. \tag{8}$$

Obviously it holds that $\mathbf{p}(y_j = 0) = 1 - \mathbf{p}(y_j = 1)$. Next, $p_{\text{size}}$ solutions are sampled from $\mathbf{D}$ and stored in the new population $P$ (line 14). Finally, the restart-best solution is updated, the counter for consecutive nonimproving solutions is updated accordingly (lines 16–21), and the best-so-far solution $Y^{\text{bsf}}$ is updated.

Our second algorithm is known as ILS. The pseudocode is provided in Algorithm 2. It requires values for parameters $0 < pp_{\text{LB}} < pp_{\text{UB}} < 1$, which are the lower bound, respectively, the upper bound, of the perturbation strength. Moreover, $c_{\text{lim}}$ is the maximum number of times that the algorithms' current perturbation strength $pp_{\text{cur}}$ is allowed us to surpass the predefined upper

**Algorithm 2:** ILS.

```
1:  input: values for parameters $pp_{\mathrm{LB}}$, $pp_{\mathrm{UB}}$, $c_{\mathrm{lim}}$
2:  $Y^{\mathrm{cur}}$ = GenerateInitialSolution()
3:  $Y^{\mathrm{cur}}$ = LocalSearch($Y^{\mathrm{cur}}$)
4:  $Y^{\mathrm{bsf}} = Y^{\mathrm{cur}}$, $c_{\mathrm{noimpr}} = 0$, $pp_{\mathrm{cur}} = pp_{\mathrm{LB}}$
5:  while CPU time limit not reached do
6:    if $c_{\mathrm{noimpr}} < c_{\mathrm{lim}}$ then
7:      $Y^{\mathrm{iter}}$ = Perturbation($Y^{\mathrm{cur}}$, $pp_{\mathrm{cur}}$)
8:    else
9:      $Y^{\mathrm{iter}}$ = StrongPerturbation($Y^{\mathrm{cur}}$)
10:   end if
11:   $Y^{\mathrm{iter}}$ = LocalSearch($Y^{\mathrm{iter}}$)
12:   if $F(Y^{\mathrm{iter}}) < F(Y^{\mathrm{bsf}})$ then $Y^{\mathrm{bsf}} = Y^{\mathrm{iter}}$
13:   if $F(Y^{\mathrm{iter}}) < F(Y^{\mathrm{cur}})$ or $c_{\mathrm{noimpr}} \geq c_{\mathrm{lim}}$ then
14:     $Y^{\mathrm{cur}} = Y^{\mathrm{iter}}$, $pp_{\mathrm{cur}} = pp_{\mathrm{LB}}$
15:     if $c_{\mathrm{noimpr}} \geq c_{\mathrm{lim}}$ then $c_{\mathrm{noimpr}} = 0$
16:   else
17:     $pp_{\mathrm{cur}} = pp_{\mathrm{cur}} + 0.01$
18:     if $pp_{\mathrm{cur}} > pp_{\mathrm{UB}}$ then
19:       $pp_{\mathrm{cur}} = pp_{\mathrm{LB}}$, $c_{\mathrm{noimpr}} = c_{\mathrm{noimpr}} + 1$
20:     end if
21:   end if
22: end while
23: output: $Y^{\mathrm{bsf}}$, the best solution found
```

bound ($pp_{\mathrm{UB}}$). Once this happens, a strong perturbation of the current solution ($Y^{\mathrm{cur}}$) is executed (see line 9 of the algorithm).

The algorithm starts by generating an initial solution uniformly at random (line 2) and by subsequently applying local search to this solution (line 3). Local search tries to swap each bit of the input solution exactly once. In case such a bit-swap improves the solution, it is immediately executed. The order in which the bits are considered is as follows. The train configurations are considered in a random order, and—in this order—the 14 b for each train configuration are also treated in a random order. After the application of local search, the best-so-far solution $Y^{\mathrm{bsf}}$, the no-improvement counter $c_{\mathrm{noimpr}}$, and the current perturbation strength $pp_{\mathrm{cur}}$ are initialized (line 4). Then, at each iteration, a random perturbation of the current solution $Y^{\mathrm{cur}}$ is performed. The standard perturbation (line 7) works as follows. Each train configuration is considered one after the other, and with a probability of $pp_{\mathrm{cur}}$ all 14 corresponding bits in $Y^{\mathrm{cur}}$ are set to value “0” On the other side, when $c_{\mathrm{noimpr}} \geq c_{\mathrm{lim}}$, the standard perturbation is replaced by a stronger, and conceptually different, perturbation (line 9). In particular, the stronger perturbation considers the six availability-related situations in random order and—with a fixed probability of 0.3—it sets all related bits of the corresponding solution to “0.” When dealing with situation 1, for example, the strong perturbation mechanism would set all bits $u_1^i$ (for all $i = 1, \ldots, N$) to “0.” After the application of local search to the perturbed solution $Y^{\mathrm{iter}}$, the remainder of the algorithm iteration deals with updates of solutions and counters. In particular, in case of no improvement of solution $Y^{\mathrm{cur}}$, the strength of the standard perturbation mechanism is increased by 0.01 (line 17). Otherwise, it is set back to the lower bound (line 19). The idea behind this mechanism is as follows. In case of a successful iteration, the perturbation strength should be small in order to search in the vicinity of the new solution in subsequent iterations. Otherwise, the perturbation strength is increased in order to enlarge the search radius.

**Algorithm 3:** ACO.

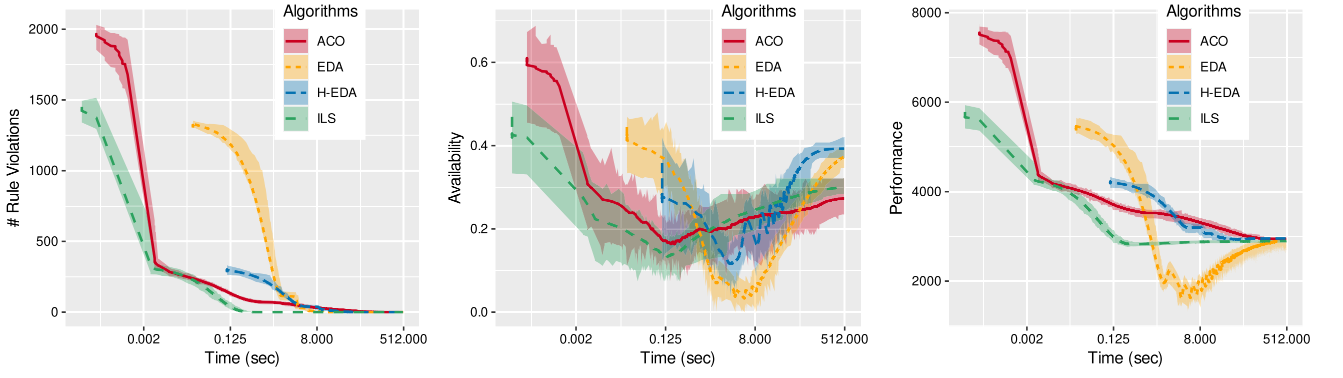
```
1:  input: values for parameters $n_{\mathrm{ants}}$, $l_{\mathrm{rate}}$, $d_{\mathrm{rate}}$
2:  InitializePheromoneValues($\mathcal{T}$)
3:  $Y^{\mathrm{bsf}} = Y^{\mathrm{rb}} = \mathbb{1}$
4:  while CPU time limit not reached do
5:    $\mathcal{S} = \emptyset$
6:    for $i = 1, \ldots, n_{\mathrm{ants}}$ do
7:      $Y$ = ConstructSolution($\mathcal{T}$, $d_{\mathrm{rate}}$)
8:      $\mathcal{S} = \mathcal{S} \cup \{Y\}$
9:    end for
10:   $Y^{\mathrm{ib}} = \mathrm{argmin}\{f(Y) \mid Y \in \mathcal{S}\}$
11:   $Y^{\mathrm{ib}}$ = LocalSearch($Y^{\mathrm{ib}}$)
12:   Update($Y^{\mathrm{bsf}}$, $Y^{\mathrm{rb}}$, $Y^{\mathrm{ib}}$)
13:   $c$ = ComputeConvergenceFactor($\mathcal{T}$)
14:   UpdatePheromoneValues($Y^{\mathrm{bsf}}$, $Y^{\mathrm{rb}}$, $Y^{\mathrm{ib}}$, $\mathcal{T}$,
      $c$, $l_{\mathrm{rate}}$)
15: end while
16: output: $Y^{\mathrm{bsf}}$, the best solution found
```

Due to the fact that combinations of different algorithms often lead to improved techniques, we also studied ways of hybridizing EDA and ILS. The best form of hybridization is obtained by choosing EDA as the main algorithm and by applying LocalSearch() exclusively to the best solution of $P$ after lines 2, 7, and 14. The resulting hybrid algorithm is henceforth labeled H-EDA.

Finally, we also implemented an ACO algorithm known as $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System in the hypercube framework (see [42]). This approach (see Algorithm 3) requires values for the following three important input parameters.

1) $n_{\mathrm{ants}}$: the number of solution constructions per iteration.
2) $l_{\mathrm{rate}}$: the learning rate (between 0 and 1).
3) $d_{\mathrm{rate}}$: the determinism rate (between 0 and 1). Lower values result in less deterministic solution constructions.

Our algorithm works with a pheromone value $\tau_j \in \mathcal{T}$ for each position $j$ of a binary solution $Y$. All pheromone values are initially set to 0.5 in function InitializePheromoneValues($\mathcal{T}$); line 2. Then, the *best-so-far* solution $Y^{\mathrm{bsf}}$ and the *restart-best* solution $Y^{\mathrm{rb}}$ are initialized to a low-quality solution: the one with all-ones (all traffic lights set to green in all train configurations). Then, at each iteration, $n_{\mathrm{ants}}$ solutions are constructed in function ConstructSolution($\mathcal{T}$, $d_{\mathrm{rate}}$); line 7. After applying local search to the *iteration-best* solution $Y^{\mathrm{ib}}$, $Y^{\mathrm{bsf}}$, respectively $Y^{\mathrm{rb}}$, are updated with $Y^{\mathrm{ib}}$, if necessary; line 12. Finally, the convergence factor is computed and the pheromone values are updated (lines 13 and 14). This is done in exactly the same way as described in [42]. The only aspect that remains to be described is the construction of a solution $Y$. In particular, for each position $j$ of solution $Y$, the following is done. First, a random nunber



(a) (b) (c)

Fig. 3. Evolution of the three quality measures over time. (a) Number of safety rule violations. (b) Evolution of the availability measure. (c) Evolution of the performance measure.

$r_1 \in [0, 1]$ is chosen. In case $r_1 \leq d_{\text{rate}}$, position $j$ is set to "1" if $\tau_j \geq 0.5$, and to "0" otherwise. If, however, $r_1 < d_{\text{rate}}$, a second random number $r_2 \in [0, 1]$ is drawn and position $j$ is set to "1" if $r_2 \leq \tau_j$, and to "0" otherwise. Finally, note that we include, by default, the use of local search in ACO, because this is nowadays a standard procedure.

*3) Experimental Evaluation:* After implementing the four algorithms (EDA, ILS, H-EDA, and ACO) in C++, the algorithm parameters were tuned using the scientific parameter tuning tool `irace` [43], with a CPU time limit of 500 s for each run. The resulting parameter values are as follows.

1) EDA: $p_{\text{size}} = 200$, $n_{\text{sel}} = 50$, $c_{\text{lim}} = 10$.
2) ILS: $pp_{\text{LB}} = 0.0318$, $pp_{\text{UB}} = 0.0515$, $c_{\text{lim}} = 500$.
3) H-EDA: $p_{\text{size}} = 1000$, $n_{\text{sel}} = 20$, $c_{\text{lim}} = 5$.
4) ACO: $n_{\text{ants}} = 5$, $l_{\text{rate}} = 0.05$, $d_{\text{rate}} = 0.33$.

Afterward, the four algorithms were applied 100 times each—that is, using 100 different random seeds—with a CPU time limit of 500 s per run to the optimization problem defined in the previous section.[1] The outcome is shown in graphical form in Fig. 3 . Note that—in all three graphics—the lines indicate the average performance, while the confidence ribbons indicate the performance over 100 runs. Moreover, the $x$-axis of the three graphics are shown in log-scale in order to focus on the early stages of the search process.

Fig. 3(a) shows the evolution of the number of safety rule violations over time. All algorithms clearly reach solutions that do not violate any safety rules very quickly. ILS achieves that after about 0.125 s, while EDA requires close to eight seconds. H-EDA already starts off with much better solutions than EDA, due to the application of local search. While most of the ACO runs quickly produce solutions without safety rule violations, there are a few runs in which this is only achieved after 100–200 s. In general, the variability in algorithm behavior over 100 runs is rather low. Fig. 3(b) presents the algorithms' evolution for what concerns availability. All algorithms start with rather high availability values (caused by solutions with many safety rule violations). While the algorithms move toward solutions with no safety rule violations, the availability value of the produced solutions becomes worse. However, as soon as the algorithms reach areas of the search space with safe solutions, they start to optimize availability. Again, ILS and ACO are faster in doing so. However, EDA clearly outperforms both ILS and ACO at about 150 s into the search process. The best algorithm concerning the availability measure is H-EDA, which nicely inherits the strong aspects of both EDA and ILS. H-EDA basically behaves like EDA, but it inherits the speed of ILS and is, therefore, able to outperform EDA. Finally, Fig. 3(c) shows the algorithms' performance for what concerns performance. The graphic indicates that the optimization of the performance measure happens alongside the optimization of availability. The zig-zag behavior in the case of EDA, starting at around five seconds, indicates that, every time the algorithm is able to find improved solutions concerning availability, the performance measure slightly decreases for a moment before improving again. This behavior is neither seen for ILS nor for ACO, and to a less extent in the case of H-EDA, which is again the best-performing algorithm. Note that the corresponding numerical values—apart from function SRE for which all algorithms always obtain zero—are provided in Table II.

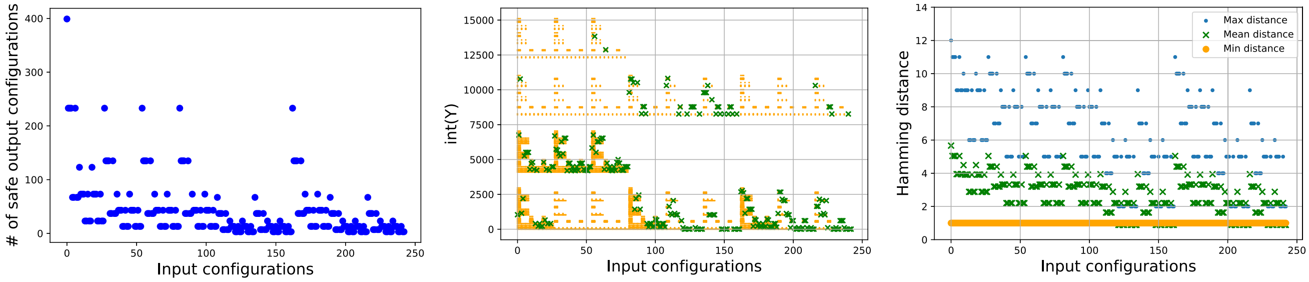
TABLE II
OBJECTIVE FUNCTION VALUES (AE AND PE) OF THE BEST SOLUTIONS FOUND BY EDA, ILS, H-EDA, AND ACO

| | AE | | PE | |
|---|---|---|---|---|
| | best | average (std) | best | average (std) |
| EDA | 0.395062 | 0.371975 (0.01) | 2946 | 2921.15 (17.04) |
| ILS | 0.320988 | 0.298642 (0.01) | 2909 | 2896.13 (13.13) |
| H-EDA | 0.419753 | 0.392716 (0.01) | 2954 | 2954.00 (0.0) |
| ACO | 0.320988 | 0.272963 (0.02) | 2950 | 2946.36 (5.78) |

Moreover, average solution qualities over 100 runs are also provided, together with the corresponding standard deviations.

*4) Search Space Analysis:* Finally, we would like to point out that—even in the context of this simplified railway signaling case study—the size of the search space is rather huge. In particular, the search space includes $2^{3402}$ candidate solutions (translating into a number with 1025 digits). Approximately

[1] Note that the number of required repetitions of a stochastic algorithm strongly depends on its variability. However, general recommendations range nowadays from 30 to 100 repetitions [44]



(a) (b) (c)

Fig. 4. (a) Each input configuration ($x$-axis), the set of feasible output configurations (binary vectors) converted to integer. Moreover, the green crosses show those output configurations that are found in the best solution derived by H-EDA. (b) Number of feasible output configurations for each of the 243 input configurations. Finally, (c) Maximum, minimum, and mean Hamming distance between the feasible output configurations of each input configuration.

TABLE III
PART OF THE LEARNED TRUTH TABLE DERIVED FROM THE BEST SOLUTION PRODUCED BY H-EDA WITHIN 100 RUNS

| $X =$ $\{s_1 \dots s_5, d_1 \dots d_5\}$ | $Y =$ $\{u_1 \dots u_8, l_1 \dots l_6\}$ |
|---|---|
| 00000 , 00000 | 00010000 , 010000 |
| 00001 , 00000 | 01101001 , 100100 |
| 00010 , 00000 | 00010001 , 111010 |
| 00011 , 00000 | 00100010 , 010101 |
| 00100 , 00000 | 01000011 , 000110 |
| . . . . . , . . . . . | . . . . . . . . , . . . . . . |
| 11110 , 00000 | 00000001 , 101010 |
| 11111 , 00000 | 00000000 , 000100 |

$2^{1092}$ of these candidate solutions (a number with 329 digits) comply with all safety rules. Given that the feasible search space is only a tiny fraction of the complete search space, our algorithms do a very good job in finding feasible (safety-compliant) solutions in a fraction of a second (ILS), respectively, in a few seconds (EDA, H-EDA, and ACO).

Fig. 4 is concerned with a search space analysis. The number of feasible output configurations depends very much on the number of trains present in the input configuration [see Fig. 4(a)]. As no trains are present in input configuration 1, for example, there are 398 feasible output configurations. The orange-colored dots in Fig. 4(b) show, for each of the 243 input configurations, the feasible output configurations (binary vectors) converted to integer values. Moreover, the green crosses show those output configurations present in the best solution found by H-EDA. Finally, the Hamming distances (minimum, maximum, and mean) between the feasible output configurations of each input configuration are shown in Fig. 4(c). Interestingly, the mean Hamming distances are rather high, which—at least partially—explains why EDA outperforms ILS for this problem. This is because sometimes, in order to move from the current solution to a better solution, the output configuration for a certain input configuration must be changed considerably. This cannot be achieved with a local search procedure based on one-flip moves. This aspect should be considered in future work.

*5) Knowledge Transformation:* Implementation specific knowledge transformation is applied to convert the optimized design to the truth table shown in Table III. The output configuration of each nonvalid input configuration—that is, an input configuration, which is not among the 243 valid ones—is defined as the default safe output in which all traffic lights are set to "red" ($Y^i = (0, \dots .0)$). In this way, the truth table is fully represented with the learned combinations ($N = 243$ rows) and default safe state combinations.

## B. Formal Verification

The learned Boolean expressions listed as truth table (see Table III), or equivalent Boolean algebra expressions, must be formally verified with respect to the safety rules listed in Table I. In this specific use case, the truth table is used to model a transition system where the states of the traffic lights ($Y^i$) are specified based on the input values ($X^i$) that indicate section occupation and train direction. Moreover, the properties that the system must satisfy correspond to the safety rules listed in Table I, which can be translated from premise and conclusion expressions to LTL properties. The formalization of the model in NuSMV has been done in the following way.

1) Input and output values are defined ($X, Y$).
2) The start condition (INIT clause in NuSMV) is defined with the default safe state output with all signals in "red."
3) State transitions are represented as formulas in which the combination of input values (truth table entry $X^i$) defines the state of the output signals ($Y^i$). As the truth table defines output values for all input value combinations, the model is fully represented.
4) The properties the system model must satisfy, described as safety rules in Table I, are modeled using LTL properties. With this logic, propositional formulas can be expressed using "Always" ("G": in all states) and "Next" ("X": in the next state) time operators.

The formal verification tool NuSMV is executed with the described system design model. The result, as expected, is that the proposed optimized design complies with all the safety rules expressed in terms of LTL properties.

# V. CONCLUSION

In this article, an IEC 61508 (industrial) and EN 50128 (railway) compliant safety software design method for Boolean algebra based safety-critical systems was proposed to facilitate the human-based design process, which combines optimization techniques and formal verification in compliance with currently considered safety standards. The multiobjective optimization was performed by a hybrid algorithm (H-EDA), which combines the speed of ILS with the optimization performance of EDA, in order to propose an optimized safe design, chosen from a large and scattered Boolean design search space. Note that especially the proposed EDA algorithm is easily applicable to similar problems because it does not use problem-specific information.

In order to support further analysis and the potential adaptation to other applications, domains and standards, the developed case-study software and results for both the optimized design and formal verification activities are available at [45].

In future work, we plan to test our methodology in the context of real case studies, such as, for example, the automated design of safety-critical protection systems for industrial applications and lifts, which is nowadays still commonly done manually by experienced safety personnel. Besides, we plan to extend the presented optimization techniques to guide fault injection experiments to maximize the detection of output errors catalogued as safety errors.

## Acknowledgment

The authors would like to thank A. Urkidi (Ikerlan) for performing a safety compliance review.

## References

[1] O. Nordland, "Can artificial intelligence be safe?," in *Probabilistic Safety Assessment and Management*, C. Spitzer, U. Schmocker, and V. N. Dang, Eds. London, U.K.:Springer, 2004, pp. 400–405.

[2] D. Castelvecchi, "Can we open the black box of AI?," *Nature*, vol. 538, no. 7623, pp. 20–23, 2016.

[3] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE Int. J. Trans. Saf.*, vol. 4, no. 1, pp. 15–24, 2016.

[4] R. Salay and K. Czarnecki, "Using machine learning safely in automotive software: An assessment and adaption of software process requirements in ISO 26262," 2018, *arXiv:1808.01614*.

[5] K. Mainzer, *How Safe is Artificial Intelligence*?, Berlin, Germany: Springer, 2020, pp. 243–266.

[6] A. Pereira and C. Thomas, "Challenges of machine learning applied to safety-critical cyber-physical systems," *Mach. Learn. Knowl. Extraction*, vol. 2, no. 4, pp. 579–602, 2020.

[7] F. R. Ward and I. Habli, "An assurance case pattern for the interpretability of machine learning in safety-critical systems," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.*, 2020, pp. 395–407.

[8] N. Rajabli *et al.*, "Software verification and validation of safe autonomous cars: A systematic literature review," *IEEE Access*, vol. 9, pp. 4797–4819, Dec. 2021.

[9] D. J. Hand and S. Khan, "Validating and verifying AI systems,*" Patterns*, vol. 1, no. 3, pp. 1–3, 2020.

[10] J. Athavale, A. Baldovin, and M. Paulitsch, "Trends and functional safety certification strategies for advanced railway automation systems," in *Proc. IEEE Int. Rel. Phys. Symp.*, 2020, pp. 1–7.

[11] J. Athavale *et al.*, "AI and reliability trends in safety-critical autonomous systems on ground and air," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops*, 2020, pp. 74–77.

[12] *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*, IEC 61508(-1/7), 2010.

[13] *Railway Applications: Communication, Signalling and Processing Systems—Software for Railway Control and Protection Systems*, EN 50128:2011/A1:2020, 2020.

[14] D. Darvas *et al.*, "Formal verification of safety PLC based control software," in *Proc. Int. Conf. Integr. Formal Methods*, 2016, pp. 508–522.

[15] R. Pichard *et al.*, "Safety of manufacturing systems controllers by logical constraints with safety filter," *IEEE Trans. Control Syst. Technol.*, vol. 27, no. 4, pp. 1659–1667, Jul. 2019.

[16] R. Schroeder *et al.*, "Safety system of W7-X neutral beam injection heating system," *Fusion Eng. Des.*, vol. 161, 2020, Art. no. 111922.

[17] R. Pichard *et al.*, "Consistency checking of safety constraints for manufacturing systems with graph analysis," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 1193–1198, 2017.

[18] J. Perez *et al.*, "A safety concept for an IEC 61508 compliant fail-safe wind power mixed-criticality embedded system based on multi-core partitioning," in *Proc. 20th Ada-Eur. Int. Conf. Reliable Softw. Technol.*, 2015, pp. 3–17.

[19] A. Soury, D. Genon-Catalot, and J. Thiriet, "New lift safety architecture to meet PESSRAL requirements," in *Proc. 2nd World Symp. Web Appl. Netw.*, Inst. Inf. Commun. Technol., Electron. Appl. Math., Université catholique de Louvain, Louvain-la-Neuve, Belgium, 2015, pp. 1–5.

[20] Q. Cappart, "Verification of railway interlocking systems and optimisation of railway traffic," Ph.D. dissertation, Inst. Inf. Commun. Technol., Electron. Appl. Math. (ICTEAM), Louvain School of Eng. (EPL), Universit Catholique de Louvain 814 (UCL), Louvain-la-Neuve, Belgium, 2017.

[21] G. S. Halford, R. Baker, J. E. McCredden, and J. D. Bain, "How many variables can humans process?," *Psychol. Sci.*, vol. 16, no. 1, pp. 70–76, 2005.

[22] E. Vassev, "Safe Artificial Intelligence and Formal Methods," in *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*. Berlin, Germany: Springer, 2016, pp. 704–713.

[23] T. Menzies and C. Pecheur, *Verification and Validation and Artificial Intelligence*. New York, NY, USA: Elsevier, 2005, pp. 153–201.

[24] M. Gendreau and J. Y. Potvin, Eds., *Handbook of Metaheuristics (Operations Research and Management Science*, 146), 3rd ed. Berlin, Germany: Springer, 2019.

[25] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Norwell, MA, USA: Kluwer, 2001.

[26] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A survey of optimization by building and using probabilistic models," *Comput. Optim. Appl.*, vol. 21, no. 1, pp. 5–20, 2002.

[27] C. González, J. Lozano, and P. Larrañaga, "Mathematical modelling of UMDAc algorithm with tournament selection. Behaviour on linear and quadratic functions," *Int. J. Approx. Reasoning*, vol. 31, no. 3, pp. 313–340, 2002.

[28] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search: Framework and applications," in *Handbook of Metaheuristics*. Berlin, Germany: Springer, 2019, pp. 129–168.

[29] M. Dorigo and T. Stützle, "Ant colony optimization: Overview and recent advances," in *Handbook of Metaheuristics*. Berlin, Germany: Springer, 2019, pp. 311–351.

[30] A. Ferrari *et al.*, "Survey on formal methods and tools in railways: The astrail approach," in *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*. Berlin, Germany: Springer, 2019, pp. 226–241.

[31] *Safety of Machinery—Safety-Related Parts of Control Systems—Part 1: General Principles for Design*, ISO 13849-1:2015, 2015.

[32] *Safety Rules for the Construction and Installation of Lifts. Lifts for the Transport of Persons and Goods. Passenger and Goods Passenger Lifts*, EN 81-20:2014, 2014.

[33] *Safety Rules for the Construction and Installation of Lifts. Lifts for the Transport of Persons and Goods. New Passenger and Goods Passenger Lifts in Existing Building*, EN 81-21:2018, 2018.

[34] *Safety Rules for the Construction and Installation of Lifts. Examinations and Tests. Design Rules, Calculations, Examinations and Tests of Lift Components*, EN 81-50:2014, 2014.

[35] I. Martinez *et al.*, *Safety Certification of Mixed-Criticality Systems*. Boca Raton, FL, USA: CRC Press, 2018.

[36] "Rules and guidelines, industrial services—Guideline for the certification of wind turbines," Germanischer Lloyd, Hamburg, Germany, Rep. GL 2010, 2010.

[37] S. Kaddani *et al.*, "Weighted sum model with partial preference information: Application to multi-objective optimization," *Eur. J. Oper. Res.*, vol. 260, no. 2, pp. 665–679, 2017.

[38] Z. Al Chami, H. Manier, and M.-A. Manier, "A lexicographic approach for the bi-objective selective pickup and delivery problem with time windows and paired demands," *Ann. Oper. Res.*, vol. 273, no. 1/2, pp. 237–255, 2019.

[39] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89509, Jan. 2020.

[40] E. J. McCluskey, "Minimization of boolean functions," *Bell Syst. Tech. J.*, vol. 35, no. 6, pp. 1417–1444, 1956.

[41] E. Castillo, J. M. Gutiérrez, and A. S. Hadi, *Expert Systems and Probabilistic Network Models (Monographs in Computer Science)*. Berlin, Germany: Springer-Verlag, 2011.

[42] C. Blum and M. Dorigo, “The hyper-cube framework for ant colony optimization,” *IEEE Trans. Syst., Man, Cybern., Part B (Cybern.)*, vol. 34, no. 2, pp. 1161–1172, Apr. 2004.

[43] M. López-Ibánez *et al.*, “The IRACE package: Iterated racing for automatic algorithm configuration,” *Oper. Res. Perspectives*, vol. 3, pp. 43–58, 2016.

[44] J. Brownlee, “Statistical methods for machine learning: Discover how to transform data into knowledge with Python,” *in Machine Learning Mastery*, 2018. [Online]. Available: https://books.google.com/books/about/Statistical_Methods_for_Machine_Learning.html?id=386nDwAAQBAJ

[45] “Repository Containing Software, Results, and Verification Tools,” 2021. [Online]. Available: https://github.com/ccblum/optimization_safety_critical_systems

**Jon Perez** (Senior Member, IEEE) received the Ph.D. degree in computer science from TU Wien, Vienna, Austria, in 2011.

He is currently a Principal Researcher with Ikerlan, Gipuzkoa, Spain, in the field of dependable autonomous systems. He has worked for more than 15 years in the development and certification of industrial safety-critical systems such as on-board SIL4 railway signalling systems (ERTMS/ETCS). His research interests include dependability, machine learning, and cybersecurity technologies.

**Jose Luis Flores** received the M.Sc. degree in robotics and advanced control from the University of the Basque Country, San Sebastián, Spain, in 2003.

He is currently a Researcher with Ikerlan Technology Research Center, Mondragón, Spain, within the Cybersecurity in Embedded Systems team. His main interest is related to artificial intelligence and cybersecurity. As such, the main lines he works on in each organization are embedded system security with Ikerlan, and machine learning and optimization with the university.

**Christian Blum** received the Ph.D. degree in applied sciences from the Free University of Brussels, Brussels, Belgium, in 2004.

He is currently a Senior Research Scientist with the Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain. His research interests include solving difficult optimization problems using swarm intelligence techniques as well as combinations of metaheuristics with exact techniques.

**Jesús Cerquides** received the Ph.D. degree in artificial intelligence from the UPC BarcelonaTech, Barcelona, Spain, in 2001.

He is currently a Senior Research Scientist with the Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain. His research interests include machine learning, probabilistic modeling, optimization, citizen science, and causality.

**Alex Abuin** is currently working toward the Ph.D. degree in computer science, formal verification (model checking) with Ikerlan Technology Research Center (Dependable Embedded Systems), Mondragón, Spain and with the University of the Basque Country, San Sebastián, Spain.

He has done all his university studies with the Faculty of Computer Science, University of the Basque Country and has been a part of Ikerlan, Gipuzkoa, Spain, since 2014. There are two main research lines/topics in which he works: on the one hand, the realization of Certified Model Checking through Context-Based Temporal Tableaux Method and on the other hand, the application of formal methods in the industry.