



Optimal spare management via statistical model checking: a case study in research reactors

Reza Soltani¹ · Matthias Volk² · Leonardo Diamonte³ · Milan Lopuhaä-Zwakenberg¹ · Mariëlle Stoelinga^{1,4}

Accepted: 4 April 2025 / Published online: 22 April 2025
© The Author(s) 2025

Abstract

Systematic spare management is important to optimize the twin goals of high reliability and low costs. However, existing approaches to spare management do not incorporate a detailed analysis of the effect on the absence of spares on the system's reliability. In this work, we combine fault tree analysis with statistical model checking to model spare part management as a stochastic priced timed game automaton (SPTGA). We use `UPPAAL STRATEGO` to find the number of spares that minimizes the total costs due to downtime and spare purchasing. The resulting SPTGA model can then additionally be analyzed according to a wide range of other metrics, including expected availability. We apply these techniques to the emergency shutdown system of a research nuclear reactor. In this case study, the failure probability is low, so we change the settings of `UPPAAL STRATEGO` setting to obtain reliable results about rare events. We consider both a single subsystem and the combination of two subsystems. In both situations, our methods find the optimal number of spares, minimizing cost while ensuring an expected availability of 99.96% and 99.93%, respectively.

Keywords Spare management · Fault tree · Statistical model checking · Research reactor · Uppaal

1 Introduction

Proper spare management is of crucial importance for safety critical systems: when a component breaks, it must be replaced in a timely manner.

At the same time, spare management is costly: spare parts need not only be purchased, but they must also be maintained and administered. Therefore, spare management policies must carefully trade reliability/availability versus costs.

In practice, spare management is often ad hoc, based on intuition rather than on systematic analysis. Nevertheless, spare parts optimization is a well-studied topic, especially in optimization research [1]. These approaches use multiobjective optimization to meet the goals of low costs and high reliability; spare management can also be combined with the maintenance policy which also affects these goals [2], to find an optimal joint spare/maintenance policy.

However, in many works reliability is closely linked to the availability of spare parts, rather than based on a detailed analysis of the effect of the absence of spares on the system's reliability. To overcome this shortcoming, this paper aligns spare management with a popular reliability engineering framework, namely fault tree analysis [3]: many companies already use fault trees as a part of their design process. By equipping the fault trees with a minimal amount of additional information (namely, the costs of downtime and spares), we support a more systematic method of studying tradeoffs between costs and reliability [4].

Our approach We exploit statistical model checking (SMC) to support spare parts optimization, and especially the tool `UPPAAL STRATEGO` [5], to automatically synthesize an optimal number of spare parts for a system modeled as a fault tree. Statistical model checking [6] is a state-of-the-

✉ R. Soltani
r.soltani@utwente.nl

M. Volk
m.volk@tue.nl

L. Diamonte
ladiamonte@invap.com.ar

M. Lopuhaä-Zwakenberg
m.a.lopuhaa@utwente.nl

M. Stoelinga
m.i.a.stoelinga@utwente.nl

¹ University of Twente, Enschede, the Netherlands
² Eindhoven University of Technology, Eindhoven, the Netherlands
³ INVAP SE, Bariloche, Argentina
⁴ Radboud University, Nijmegen, the Netherlands

art methodology for Monte Carlo simulation. Our key model is a stochastic priced timed game automaton (SPTGA), i.e., a transition system that models the evolution of probability and costs over time. Monte Carlo simulation can estimate (up to a confidence interval) the probability and expected values for a wide variety of random variables. In many applications, failure events have very small probabilities, so many simulation runs are needed to obtain reliable results. Therefore, the number of simulations need to be tailored to the specific setting.

When the model involves decisions, simulations can also be used to estimate the probability of desirable outcomes for a given strategy. In [4] this is used to find the reliability of a fault tree model given a fixed number of spares. More recently, however, developments in the area of machine learning allow SMC to not only analyze given strategies, but also automatically synthesize optimal strategies [7].

We start with a fault tree extended with costs for downtime and spare management, as well as the dependability metric of interest. Via statistical model checking, we synthesize a strategy for spare management that is optimal under the given dependability metric, e.g., the reliability. We further analyze this optimal strategy with respect to other metrics, such as the availability.

The case study We apply our approach to a section of the emergency shutdown system of a research reactor, developed by the company INVAP, Argentina. The section of interest of the system consists of three main subsystems: the reactor protection system, the neutron flux instrumentation, and the temperature difference at the reactor core (DTCORE). Each subsystem is implemented in triple modular redundancy. Cost is incurred by buying spares and by suffering downtime due to unreliability; the optimal spare strategy is one that balances these to obtain the lowest total cost.

Results We consider a single subsystem of the emergency shutdown system, as well as a combination of two subsystems. In both cases, we determine the optimal spare management by modeling it as a fault tree and translating it into a SPTGA. The optimal spare problem then becomes a strategy synthesis query in UPPAAL STRATEGO, which we solve using Monte Carlo simulation, coupled with rare event simulation techniques tailored to our setting. For the single subsystem, this is solved in 7 min, giving 6 spares as the optimal balance between reliability and spare part costs. Under this strategy, the probability of any downtime during the research reactor's 40-year lifespan is less than $9E-4$, with an expected availability of 99.96%. Our results for the two subsystems combined provide 6 spares for the first component and 140 spares for the second component as an optimal balance between reliability and cost of spare parts. Under

this strategy, the probability of any downtime during the research reactor's 40-year lifespan increases to 86.59%, with an expected availability of 99.93%.

Contributions Summarizing, our contributions are a systematic way to find optimal spare management strategies by combining the detailed reliability analysis of fault trees with the strategy synthesis tools of statistical model checking, together with increased simulation sizes to account for rare events. The resulting timed game automaton allows for a range of queries for further analysis of the optimal strategy. We show the validity of this method on a case study coming from nuclear research reactors.

Extension from conference paper This work is an extended version of a previous paper [8] presented at International Conference on Formal Methods for Industrial Critical Systems 2023. Compared the conference paper, we added a UPPAAL model for a system with two components and provide the subsequent analysis determining the optimal number of spares for both types of components, see Sect. 8. Further, we provide a detailed discussion on handling rare events during the Monte Carlo simulation and our choice of parameters for model checking, see Sect. 7.2.

Artefact We provide all UPPAAL models, queries and results in a publicly available artefact on Zenodo: <https://doi.org/10.5281/zenodo.7970835> [9].

1.1 Related work

Spare management Spare parts management is an active area of research, cf. [10, 11] for overviews. Hu et al. [1] surveyed the gap between the theory and practice of spare parts management. According to this work, from a product lifecycle perspective, there are three kinds of forecasting tasks, namely forecasting initial demand, ongoing demand, and demand over the final phase. There are techniques for predicting the need for spare parts based on neural networks [12–16]. The neural network based forecasting is most often used to predict continuing need and demand throughout the final phase. It can rarely be applied in predicting the initial demand since there is limited historical data on the spares' consumption when new equipment is introduced.

Zheng et al. [17] consider reordering of spare parts from dual sources, with different lead times and costs. The approach uses Markov decision processes to model the spare management and synthesizes optimal strategies through an exact algorithm. In [4], the impact of different numbers of spares is investigated. Given a dynamic fault tree with spare components, the model is translated into a probabilistic timed automata (PTA). The unavailability is then calculated by analysis of the PTA via UPPAAL. In contrast to our approach, the number of spare parts is manually fixed beforehand, and no automatic synthesis is performed.

Translation from fault trees to timed automata There are studies on optimization using timed automata. The authors of [18, 19] study the optimization of cyber-physical system behavior regarding energy consumption and its effect on system reliability by UPPAAL. Translating the fault tree into timed automata can help in better analysis and optimization. Several methods have been developed for fault tree analysis using timed automata.

In [20, 21], the authors introduced a framework for converting fault trees and attack trees to timed automata. In [20], the authors translated attack tree gates and leaves into timed automata and defined the properties in weighted CTL queries to perform model checking analysis. They used the UPPAAL CORA model checker to obtain the optimal path of the attacks. In [21], each element of an attack-fault tree is translated into a stochastic timed automaton [22]. As a result, the authors calculated the costliest system failure by equipping the attack-fault tree with stochastic model checking techniques. Ruijters et al. [23] presented a translation from fault maintenance trees – fault trees with maintenance aspects – to priced timed automata, and then applied statistical model checking for analysis. In all these papers, the authors only check the dependability metrics, such as system reliability, but do not perform any synthesis on specific parameters. In contrast, our approach allows automatically synthesizing parameters which are optimal under a given dependability metric. Furthermore, these works are less flexible than our model. In previous works, the size of a gate automaton grew linearly with the number of its children. In contrast, the size of our automata is independent of the number of children, and remains constant. The modular and configurable nature of our proposed approach aligns with Industry 4.0 principles, which emphasize flexible production environments capable of adapting to dynamic demands and shared resource utilization [24].

2 Spare management for a research reactor

2.1 Research reactor

We investigate optimal spare management for research reactors. The case study stems from INVAP S.E. based in Bariloche, Argentina. This high-tech company develops – among others – research reactors, satellites, and radars.

Research reactors are nuclear reactors which are used for research purposes and not for power generation. Application areas of research reactors include research and training, analysis and testing of materials, and the production of radioisotopes used in medical diagnoses and cancer treatment. In contrast to power reactors, research reactors are smaller, operate at lower temperatures, require less fuel, and produce less fission products.

Emergency shutdown system We consider part of the emergency shutdown system of a research reactor. In case of an emergency, the system automatically stops the nuclear fission chain reaction within the reactor. This process is called “trip” or “scram.” The fission reaction is stopped by inserting neutron-absorbing control rods into the reactor core. By absorbing neutrons, the control rods stop the nuclear chain reaction and shut down the reactor. The emergency shutdown system is designed as a *fail-safe* mechanism: if it fails due to internal subsystem failures, operation of the whole research reactor is immediately stopped. In this case study, we focus on the unavailability of the research reactor operation due to the unavailability of the emergency shutdown system caused by internal failures. In particular, we do not consider the trip of the research reactor – through the emergency shutdown system – triggered by the violation of safety thresholds.

The emergency shutdown system (depicted in Fig. 1) consists of three main subsystems: (1) the reactor protection system (RPS), the (2) neutron flux instrumentation, and (3) the delta temperature reactor core (DTCore). Both the neutron flux instrumentation and the DTCore feed into the RPS via analog channels. If any of the two inputs or the RPS itself is unavailable, the emergency shutdown system becomes unavailable. The *reactor protection system* consists of three redundant subsystems, also called trains. The RPS signals a trip of the reactor if two out of the three trains are above the safety threshold or if the trains are unavailable due to internal failures. The *neutron flux instrumentation* consists of three ionization chambers. Each ionization chamber measures the neutron flux outside the core and generates an alarm if safety thresholds are violated. The neutron flux instrumentation fails if two of the ionization chambers are failed. The *Delta Temperature Reactor Core (DTCore)* consists of three redundant *thermocouples* which measure temperature differences. DTCore fails if two thermocouples are unavailable.

Figure 2 depicts a fault tree model of the emergency shutdown system. The logical OR-gate *Trip* models that the system fails if any of the three main subsystems fails. Each subsystem is modeled by a 2/3 voting-gate which fails if at least two out of the three inputs fail. The leaves represent the redundant subsystems which fail according to an associated failure distribution.

In the following, we mainly focus our analysis on the neutron flux subsystem (marked by the box in red) because it has significantly higher costs than the other components. Thus, optimal spare management is most crucial here.

2.2 Optimal spare management

Redundancies of the subcomponents ensure that the research reactor operates safely and reliably. The safety and reliability of the reactor can be analyzed with standard techniques such as fault tree analysis [3]. However, the redundancies

Fig. 1 Schematic of the emergency shutdown system

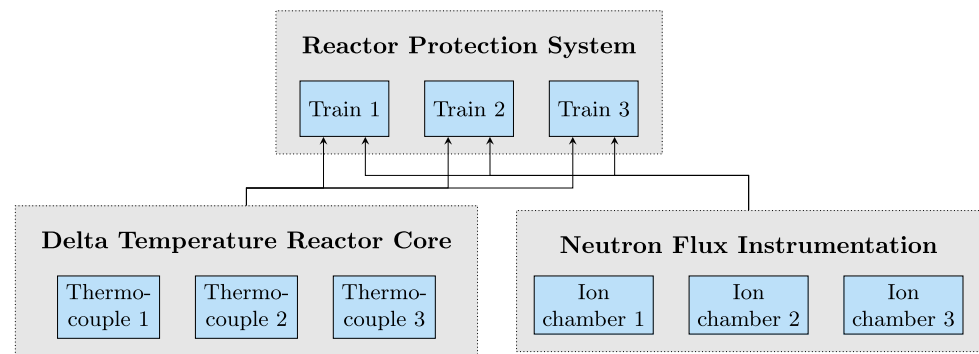
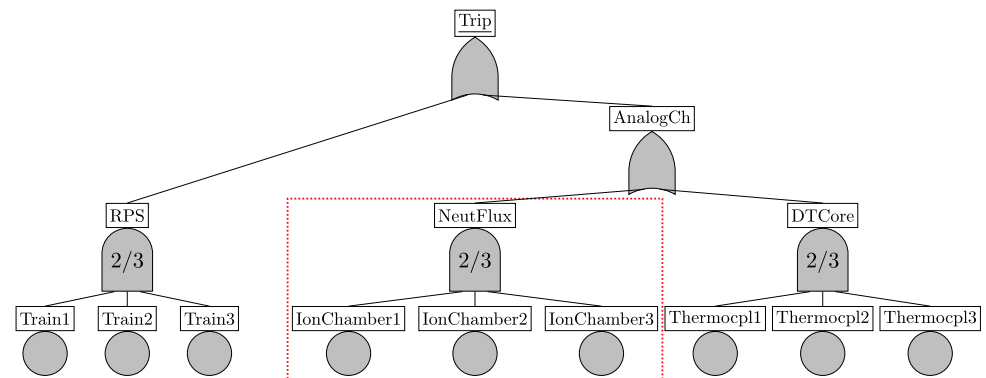


Fig. 2 Fault tree model of the emergency shutdown system



do not prevent failures, they only mitigate them. In case of a subcomponent failure, the subcomponent should be either repaired or replaced by another spare component. It is therefore crucial that enough spare components are available to allow for quick replacement of failed components. On the other hand, ordering and storing a large number of spare components costs a significant amount of money. A pressing question is therefore to find an optimal spare management policy, e.g., “how many spare components should be on stock at a given time?” The number of spares on stock should be small to minimize purchasing and storage costs but sufficiently large such that the research reactor stays operational, especially in case of component failures. The optimal strategy needs to consider various aspects: costs of spare components, warehouse capacity, delivery times for new orders, reliability requirements on the reactors, etc.

Initial stock We consider the setting where all necessary spare parts must be initially on stock. Thus, no intermediate ordering is considered. This setting is important in the context of research reactors as ordering additional spare components can take a long time or components may no longer be produced at all. The ordering process involves multiple steps such as (possibly) manufacturing the component on demand, transportation and testing of the component. Thus, the initial stock is crucial to guarantee high availability of the research reactor.

Current spare management strategies Currently, the number of spare components is often based on experience, such as keeping the square root of used components on stock. For n components which are in use, $\lceil \sqrt{n} \rceil$ components are kept on stock as spare parts. Another approach is manually considering different numbers of spare parts and calculating the system reliability with respect to each configuration. In the end, the best number of spare parts from all considered configurations is selected. However, this approach is very time-consuming as a large number of spare parts need to be considered. Furthermore, if parts of the system change, the whole process has to be performed again.

2.3 System parameters

We consider a remaining lifetime of 40 years for the research reactor. We assume a gross profit of \$1 million Argentine pesos per day of operation for the research reactor. Conversely, this means each day where the research reactor is unavailable, costs of \$1 million are accumulated.

Failure rates and costs Component failures occur according to a given failure rate. The failures can be mitigated by replacing the component with a spare one. The replacement time is given by a replacement rate. Table 1 provides the failure rates, replacement rates, and the costs for a single component of each type. The values were provided by our industrial partner INVAP S.E.

Table 1 Failure rates, replacement rates, and costs of components

Component	RPS Train	Ionization chamber	Thermocouple
Failure rate [per hour]	1×10^{-4}	1×10^{-6}	1×10^{-5}
Replacement rate [per hour]	1×10^{-2}	3×10^{-3}	3.5×10^{-3}
Costs [in \$ Argentine pesos]	8k	80k	1.2k

2.4 Performance metrics

In our analysis, we are interested in optimizing for two metrics: (1) high availability of the research reactor with (2) minimal cost. The unavailability is calculated for a lifetime of 40 years. As unavailability is also associated with costs – the loss of profit – it suffices to optimize for cost. We therefore want to find the number of initial spares such that the costs of the spares plus the costs due to system unavailability is minimal. This yields the metric Q^O we want to optimize for: Q^O . What number of spare components minimizes total cost?

After finding the optimal number of spares, we further want to analyze the availability of the system under this configuration. To this end, we are also interested in the following metrics:

- Q_1^A . Given a number of spares, what are the costs, i.e., costs of spares plus costs through unavailability?
- Q_2^A . Given a number of spares, what is the availability of the system within its lifetime of 40 years?
- Q_3^A . Given a number of spares, what is the probability that the system is down for less than a given threshold, e.g., a downtime of at most 30 days in total?

3 Preliminaries

3.1 Fault trees

Fault trees (FT) are a common reliability model [3] applied in diverse industries such as automotive, aerospace, and nuclear sectors [25]. A fault tree is a directed acyclic graph that serves as a model for determining the causes of system failures. It identifies how failures at lower levels propagate through the system, ultimately leading to a system-level failure. The leaves in a fault tree, called *basic events* (BE), represent atomic components that fail according to an associated failure rate. Inner nodes, called *gates*, model how failures propagate. The *voting gate* (also known as K/N gate) indicates that the associated event occurs if at least K out of its N input events have failed. The AND and OR gates are special cases of the voting gate, corresponding to N/N and $1/N$, respectively. A failure of the root node, called *top event*, represents a system failure.

3.2 Stochastic priced timed-game automata

A *Stochastic Priced Timed-Game Automaton* (SPTGA) is a Timed Game Automaton (TGA) [26] with probabilities associated to transitions between states. For each state, a price is given that represents the cost/benefit of being in that state, and there may be a cost for taking a transition. In SPTGA, there are two types of transitions, controllable and uncontrollable. A *controllable* transition refers to a transition that can be controlled or influenced by the system under consideration. The system has control over the timing and occurrence of the transitions, and their execution can impact the outcome of the game or the behavior of the system. Unlike controllable transitions, *uncontrollable* transitions cannot be influenced by the system under consideration. By distinguishing between controllable and uncontrollable transitions, SPTGA provides a framework for analyzing systems where both the player's (system's) decisions and external uncontrollable events play a role in determining the system's behavior and evolution over time. Figure 3 shows an example of SPTGA. In SPTGA, the opponent can be an antagonist, which will be a 2-player game, or stochastic, then the game will be a $1\frac{1}{2}$ -player game.

Given an SPTGA, we can split the problems of interest into two different categories. The first is the *control synthesis problem* whose input is a model of the system \mathcal{G} and a property φ . The goal is to compute a strategy σ – if such a strategy exists – such that $\mathcal{G}|\sigma$ satisfies φ . Here, $\mathcal{G}|\sigma$ represents the system composed with the strategy σ , modeled as the synchronous product of the automaton \mathcal{G} and the strategy σ . The second category is the *verification problem* (or model checking problem). Its input is a model of the system \mathcal{G} (or $\mathcal{G}|\sigma$, if subjected to a strategy) and a property φ , and its problem is determining whether \mathcal{G} satisfies φ . While the verification problem is concerned with whether a system meets a set of requirements or not, the control synthesis problem is concerned with whether the system can be *restricted* to satisfy the requirements.

Automaton symbology The automata figures use specific symbols to represent different types of states and transitions (cf. Fig. 6):

- **Committed states.** Represented with a C inside the state, committed states are temporary states that cannot delay. Transitions from these states must occur immediately, ensuring the automaton progresses without any time elapsing.

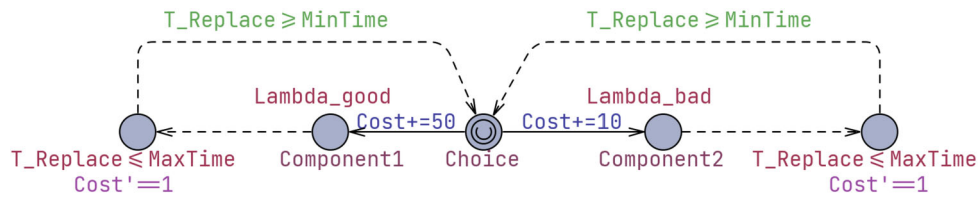


Fig. 3 The SPTGA consists of two controllable transitions (straight arrow) and four uncontrollable transitions (dashed arrow). This example shows a component that needs to be replaced when it fails. When we are in the *Choice* state, we can choose one of the two controllable transitions. One of the transitions is to use a component with a higher cost (50 units) but a lower failure rate. Another choice is a component with a lower cost (10 units) but a higher failure rate ($\lambda_{\text{bad}} > \lambda_{\text{good}}$).

- **Urgent states.** Represented with a U inside the state, urgent states cannot delay time either. However, if both urgent and committed transitions are available, committed transitions take priority and must be taken first.
- **States with inner circles.** States with a small circle drawn inside indicate initial states. These states denote the starting point of the automaton's execution.

3.3 UPPAAL STRATEGO

UPPAAL STRATEGO [5] is a powerful tool that enables users to generate, optimize, compare, and explore the effect and performance of strategies for stochastic priced timed games. It integrates the various components of UPPAAL and its two branches – UPPAAL SMC (statistical model checking) [27] and UPPAAL TIGA (synthesis for timed games) [28] – as well as the optimization method proposed in [29] (synthesis of near optimal schedulers) into one tool suite. Recently, UPPAAL STRATEGO has become a part of UPPAAL 5.0.

In UPPAAL STRATEGO, the query language contains a subset of Time Computational Tree Logic to verify the requirements of the model. We focus on the following two types of queries for a given SPTGA and a strategy. The variable N is the number of simulations to be performed and bound shows the time bound on the simulations.

1. *Probability Estimation* $\text{Pr}[\text{bound}](\psi)$ under *Strategy_Name* estimates the probability of a requirement property ψ being satisfied for a given SPTGA and strategy within a time bound.
2. *Expected Value* $\text{E}[\text{bound}; N](\psi)$ under *Strategy_Name* evaluates the minimum or maximum value of a clock or an integer value while UPPAAL STRATEGO checks the SPTGA with respect to a strategy.

In this context, simulations refer to the execution of the model under different conditions and scenarios to observe its behavior and gather information about its properties. Estimation in

When the selected component fails, it will be replaced within MinTime and MaxTime replacement time based on a uniform distribution, resulting in a cost of 1 per time-unit while the component is being replaced. With the help of SPTGA, we can synthesize a strategy to minimize the expected cost, where the choices are made in such a way that we incur the least cost

this context refers to the process of determining or approximating numerical values or probabilities associated with specific aspects of the system's behavior or performance.

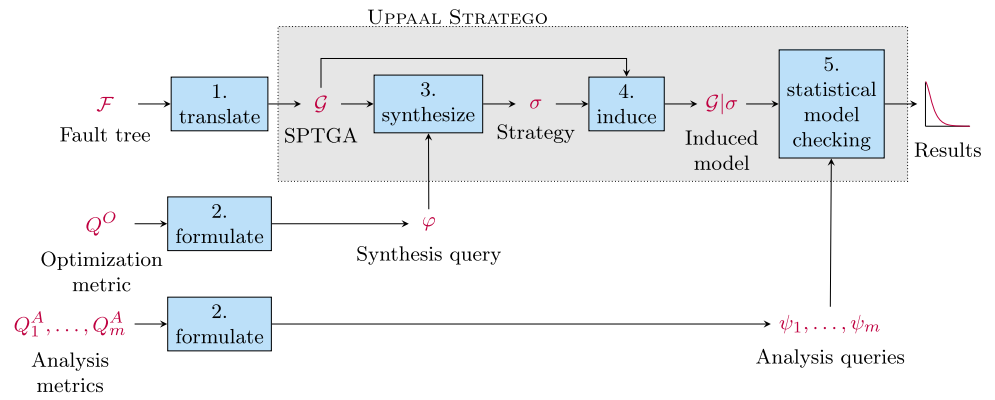
4 Methodology

Figure 4 outlines our approach for spare parts optimization via statistical model checking. The inputs to the framework are a fault tree \mathcal{F} representing the system under consideration, and the metrics Q^O and Q_1^A, \dots, Q_m^A . The approach returns the number of spare parts which optimizes Q^O , e.g., minimizes cost. Additionally, the framework also outputs the model checking results for Q_1^A, \dots, Q_m^A when using the optimal number of spares, e.g., the expected cost for having a specific amount of spares, the system's availability over its lifetime and the probability of the downtime being less than a defined threshold. The synthesis of the optimal number of spares as well as the analysis via statistical model checking are performed using UPPAAL STRATEGO [5].

The framework consists of five steps as indicated by the blue boxes in Fig. 4. We present each step in the following:

- Step 1.* First, we translate the given FT \mathcal{F} to an SPTGA \mathcal{G} . Following [23, 30], we employ a compositional translation methodology. That is, we translate each FT element (i.e., basic events and gates) into a separate SPTGA. Then, by combining these different components (automata), we obtain one SPTGA \mathcal{G} capturing the complete FT behavior. Details on the translation from FT to SPTGA are given in Sect. 5.
- Step 2.* The optimization metric Q^O and the analysis metrics Q_1^A, \dots, Q_m^A are formalized as temporal logic formulas φ and ψ_1, \dots, ψ_m , respectively. Details are given in Sect. 6.
- Step 3.* Given the SPTGA \mathcal{G} , we use UPPAAL STRATEGO to synthesize a strategy σ that is optimal for the given synthesis query φ , e.g., minimizes cost. The synthesized strategy σ then encodes the optimal number of spares to have on stock in the beginning.

Fig. 4 Overview of the methodology



Step 4. Applying the synthesized strategy σ on the SPTGA \mathcal{G} yields the induced model $\mathcal{G}|\sigma$ that represents the system’s behavior when using the optimal number of spare parts.

Step 5. Lastly, we use statistical model checking via UPPAAL STRATEGO to calculate the desired analysis queries ψ_1, \dots, ψ_m on the induced model $\mathcal{G}|\sigma$. The analysis yields e.g., the system’s availability or the probability of an overall downtime less than a given threshold.

5 INVAP emergency shutdown system as an SPTGA

In the following, we present the SPTGA model for the INVAP emergency shutdown system. In this section, we consider only one component in order to make the methodology easier to understand. In Sect. 8, we extend the SPTGA model to two components. In this case study, we do not fully consider the third component, the delta temperature reactor core. Spare management for this component type has less importance because the thermocouple is extremely inexpensive and manufactured by INVAP S.E. itself.

Following [30, 31], we obtain the SPTGA model from the fault tree in Fig. 2 through a compositional translation methodology. That is, we translate each FT element (i.e., basic event or gate) into a separate SPTGA. Then, by combining these different SPTGA components (automata), we obtain one SPTGA \mathcal{G} capturing the complete FT behavior. In contrast to previous works, e.g., [23], our translation is agnostic to the number of inputs in the FT. For example, we translate the behavior of a voting gate (K/N gate) to SPTGA regardless of the number of children N or threshold K . These values are dynamically defined in the system declaration without changing the model. A larger FT can be modeled by creating an instance of a translated automaton. For this, the values associated with each gate, e.g., the children’s ID, can be specified in the system declaration.

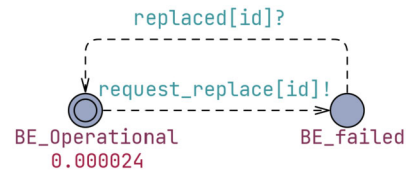


Fig. 5 SPTGA BE[id] for basic events, parameterized with their ID id. Each BE has an exponential failure rate of 0.000024

For simplicity, we start by presenting the translation of the Ionization chamber component, marked with a dashed box in Fig. 2. Later in Sect. 8, we extend the model to two components. However, our method can easily be extended for other types and is applicable to fault trees in general.

The SPTGA consists of four types of automata, which can be instantiated multiple times: BE represents a basic event, SG models the spare management, VOT represents a voting gate, and W models the warehouse storing the spares. Each element in the FT can have different states, and in SPTGA, we represent them with locations. For example, BEs are either operational or failed, so we have two different states for the associated BE. We use synchronization channels to show the link between different FT elements.

Basic events Figure 5 shows the SPTGA model for a BE. This automaton consists of two states: BE_operational denotes that the component is operational, whereas BE_failed denotes its failure. Since we have several BEs, the SPTGA is parameterized by ID [id] with the same failure rate for the same type of components. The BE’s failure rate (per day) is an exponential rate of 0.000024 (cf. Table 1 column “Ionization chamber”). In Table 1, failure rates and replacement rates are given per hour, while here we converted them into per day for convenience. When a BE fails, it communicates with the spare management automaton SG through the communication channel request_replace[id]!. The message replaced[id]? can be received from SG when the failed BE has been replaced.

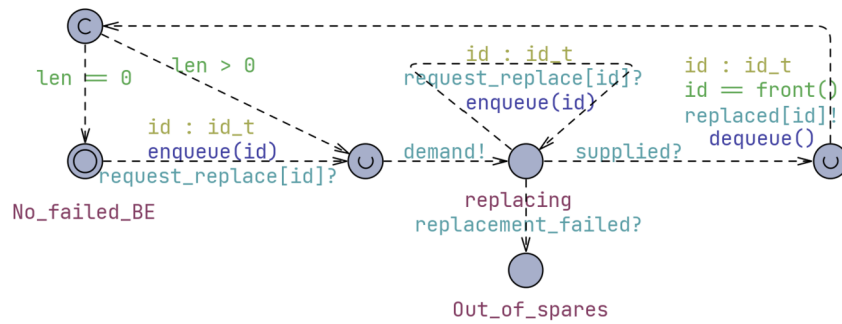
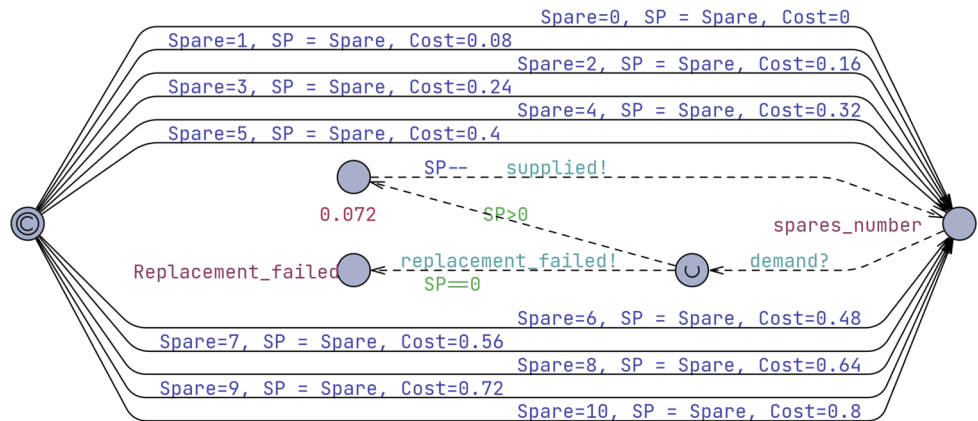


Fig. 6 The automaton for the spare management. Here request_replace[id]? and replaced[id]! are signals to communicate with the corresponding BE. The demand! and supplied? are

signals to communicate with the warehouse; len holds the length of the queue. The replacement_failed? signal will be received if no spare is available in the warehouse

Fig. 7 The automaton for the spare warehouse. The variable Spare stores the initial amount of spares in the warehouse, SP represents the number of available spares in the warehouse, and Cost represents the cost. The demand?, supplied!, and replacement_failed! are signals to communicate with the SG



Spare management For simplicity, the Spare gates are not explicitly shown in the FT in Fig. 2. However, each basic event (BE) is connected to a Spare Management gate, which ensures that failed components are replaced by spare components when available. Figure 6 shows the SPTGA representing the spare management. When SG receives the request_replace[id]? message – modeling that BE[id] has failed – it stores the ID [id] in the queue, then communicates with the warehouse through the communication channel demand!. If there is a spare part in the warehouse, then the replacement will be done, and SG will be informed by the communication channel supplied?. If there is no spare in the warehouse, then the replacement_failed? message will be received and the state Out_of_spare is entered. The enqueue() and dequeue() are functions to enqueue the ID of the failed BE or remove the successfully replaced BE from the queue, respectively. The variable len holds the length of the queue. If len equals zero, no failed BE is waiting to be replaced.

Warehouse The Warehouse automaton handles the inventory of spares and facilitates replacements by interacting with the Spare Management automaton, shown in Fig. 7. There are two different types of transitions in this automaton. Solid

lines indicate controllable transitions and dashed lines indicate uncontrollable transitions. The Spare Warehouse is the only automaton with controllable transitions. The initial state in this automaton is a committed state. A committed state cannot delay, and the outgoing transition of this state has the highest priority – even higher than the urgent state. As a result, outgoing transitions from this location are the first transitions made in the entire SPTGA, which is the initial decision for the spare quantity in the warehouse. This decision is the only one we can control. Normally, one of these transitions is chosen in a nondeterministic way. But if we synthesize a strategy, we can have control over these transitions (in the next section, we will discuss the strategy synthesis for determining the spare number). After deciding on the initial amount of spares Spare in the warehouse, this value cannot be changed later on. The variable SP represents the number of available spares in the warehouse. In the beginning, SP’s value equals Spare. The variable Cost represents the cost, which is the total cost of buying spares and the cost of system failure. Each spare costs 0.08 units (one unit is equivalent to \$1 million Argentine pesos). This cost is stored in the variable Cost at the beginning, and the cost of system failure is added later.

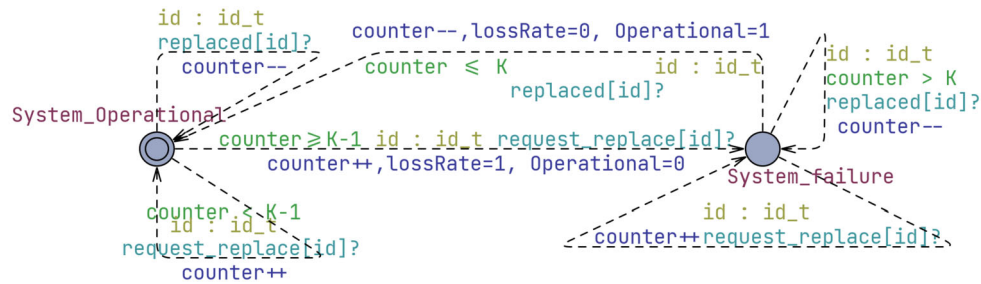


Fig. 8 The K out of N voting automaton. The counter keeps track of failed children. Variables *Operational* and *lossRate* are used only for monitoring and cost calculation purposes. The

request_replace[id]? is the received signal from the corresponding BE. The *replaced[id]?* is the received signal from SG

During system operations, if a replacement is needed for a failed BE, the warehouse receives the *request demand?* from SG. If a spare is available in the warehouse, the failed BE will be replaced according to an exponential rate of 0.072 per day, and the warehouse sends the *supplied!* signal to SG. If no spare is available anymore, the *replacement_failed!* signal will be transmitted.

Voting gate Figure 8 shows the SPTGA model for a K/N voting gate. This automaton consists of two states, *System_operational* and *System_failure*. The counter keeps track of failed children.

When a BE fails (*request_replace[id]?*), the counter value is incremented. If the counter value is at least K , then the voting gate is failed and the automaton goes into the *System_failure* state. When a failed BE is replaced (*replaced[id]?*), the counter value is decremented. If the total number of failed BEs (the counter value) is less than K , then the automaton goes into the *System_working* state. Variables *Operational* and *lossRate* are used only for monitoring and cost calculation purposes. The monitoring automaton is not shown for a single-component analysis since it only has one state to keep track of costs.

For the OR gate, which is a $1/N$ voting gate, the same automaton can be used with the parameter K set to 1. However, the specific OR gate automaton discussed later in Sect. 8 has been separately modeled to address additional system-level interactions unique to that context.

6 Formal queries

This section will explain the queries we have used in our model verification. After modeling the fault tree using SPTGA, we use these queries to analyze the model further. We use two types of queries for statistical model checking to calculate the desired dependability metrics, namely *strategy queries* and *statistical queries*.

6.1 Strategy queries

By giving them names, strategy queries make it possible to save, load, reuse, and modify the strategies. We use this query in two ways, with the *controller synthesis queries* or *learning queries*.

Strategy as controller synthesis query Controller synthesis queries synthesize a strategy that makes a given control objective goal true, i.e., regardless of the environment choice, the goal predicate is always true. If no such strategy exists, then false is returned. The format of this query for our case is

```
strategy SPCount# = control: A[] Spares == i
```

This query computes a strategy where, regardless of the environment choice, goal predicate – the number of the primary spares must be equal to i – is always true. Such a query allows manually fixing the number of spares. For example, in Fig. 7, which depicts the Warehouse automaton, the initial state contains controllable transitions that determine the number of spare parts in the warehouse. The controller synthesis query selects one of these transitions based on the specified optimization criteria, such as minimizing the overall cost.

Strategy as a learning query A learning query specifies the objective for which an optimal strategy should be synthesized. We use the following query:

```
strategy MinCost = minE(Cost) [<=lifetime]:
    <>GlobalTime==lifetime
```

Strategy *MinCost* minimizes the expected *Cost* value within the given *lifetime* (e.g., 40 years which is equivalent to 14,600 days). In order to determine the strategy for exactly the given lifetime, we use the predicate *GlobalTime==lifetime* that stops together with the simulation time bound. *GlobalTime* is a clock that is never reset.

Table 2 Changed settings in UPPAAL

	Number of successful runs	Maximum number of runs	Number of good runs	Number of runs to evaluate	Probability uncertainty
Default value	200	500	100	100	0.05
Our value	20,000	50,000	10,000	1,000,000	0.01

6.2 Statistical queries

We use statistical queries to formally specify the analysis metrics Q_1^A, \dots, Q_m^A from Sect. 2.4:

$E [\leq \text{lifetime}; 1000000]$ (max: Cost) under Strategy_Name

This query estimates the maximal value of Cost within the given time span by running 100,000 simulations. The under Strategy_Name indicates that the query is subjected to the strategy Strategy_Name,

$E [\leq \text{lifetime}; 1000000]$ (max: ODay/lifetime) under Strategy_Name

This query is similar to the previous one, except that the estimated value is the system's availability. Variable ODay stores the operational time of the system. It is important to note that using min instead of max in these two queries would return the overall minimal value, which initially would be 0. This is because, at the start of the simulation, no uptime has accumulated, yet. Since our goal is to estimate the expected availability under the given strategy, we use max to capture the best-case operational performance rather than the trivial lower bound:

$\text{Pr}[\leq \text{lifetime}] (\langle \text{DTime} \rangle 0)$ under Strategy_Name

This query estimates the probability that the system is down at all within its lifetime. Variable DTime stores the downtime of the system.

7 Analysis of a single-component system

This section presents the analysis of the system with a single component. The statistical model checking results were obtained using UPPAAL STRATEGO and its requirement specification language [32].

7.1 Analysis results

In this section, we limit our following analysis to the neutron flux subsystem. This component type has significantly higher costs than the other subsystems and thus, optimal spare management is most crucial here.

Settings We run our experiments on a MacBook Air with M1 chip and 16 GB RAM. We use UPPAAL 5.0, which includes UPPAAL STRATEGO. We run the statistical queries

with a confidence interval of 95%. We changed some of UPPAAL's default settings to capture rare events and increase accuracy, see Table 2 for the details. The parameters listed in this table are specific to the learning algorithm and statistical model checking in UPPAAL STRATEGO. Below is a brief explanation of each parameter:

- **Number of successful runs** specifies the minimum number of successful simulation runs required to consider the learning process valid.
- **Maximum number of runs** defines the maximum number of simulation attempts allowed during the learning process.
- **Number of good runs** specifies the number of simulation runs in which the selected strategy performs well according to the optimization criteria (e.g., minimizing cost or maximizing availability). These runs contribute to refining the strategy by reinforcing successful decision patterns. This term does not specifically originate from rare event simulation but rather from the reinforcement learning process used in UPPAAL STRATEGO.
- **Number of runs to evaluate** determines the number of simulations used to evaluate the synthesized strategy during validation.
- **Probability uncertainty** represents the acceptable margin of error for probability estimates. This parameter applies only to the last statistical query, which calculates the probability of downtime, as the other statistical queries explicitly declare the number of simulations (100,000).

These adjustments ensure accurate results, particularly for rare-event scenarios, by increasing the number of simulations and refining the uncertainty bounds. Except for the probability uncertainty, which is related to statistical model checking, the other four parameters relate to UPPAAL STRATEGO's learning algorithm.

Results Table 3 shows the results of the strategy and statistical queries performed on the SPTGA from Sect. 5. First, learning query MinCost is used to synthesize the optimal number of spares needed to reduce the cost. In this case, the strategy identifies six as the optimal number of spares for the neutron flux subsystem. However, MinCost does not directly modify model parameters such as the initial number of spare parts. The optimal spare quantity is a result of strategy synthesis, but the model itself still allows flexibility in its initial configuration. To explicitly enforce a fixed number

Table 3 Verification results for the neutron flux subsystem

Query	Result	Comp. time
strategy MinCost = minE(Cost) [≤ 14600]: $\langle \rangle$ GlobalTime==14600	✓	411.00 s
E [$\leq 14600; 100000$] (min: Spare) under MinCost	6	0.01 s
strategy SPCount6 = control: A[] Spare==6	✓	0.01 s
E [$\leq 14600; 100000$] (max: Cost) under SPCount6	0.488±0.0030	2.46 s
E [$\leq 14600; 100000$] (max: ODay/14600) under SPCount6	0.99965±0.00009	2.45 s
Pr[≤ 14600] ($\langle \rangle$ DTime>0) under SPCount6	≤ 0.0009	0.02 s

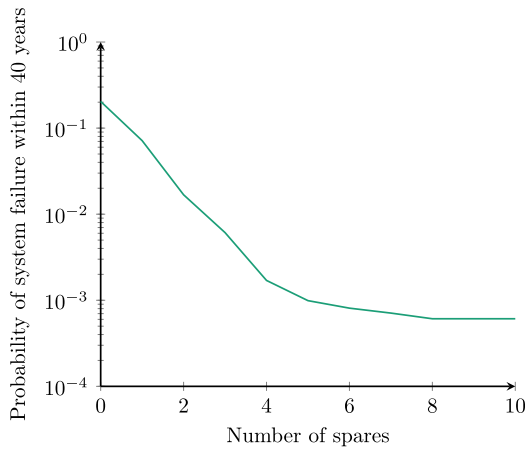


Fig. 9 Probability of any system failure within 40 years

of spares, a separate controller synthesis query (SPCount6) is required. This approach ensures that we can analyze both unconstrained and constrained scenarios under the learned strategy. The controller synthesis query (SPCount6) explicitly fixes the number of spares to 6 for further analysis. After finding the optimal number of spares, we use a controller synthesis query to obtain a new strategy SPCount6 that fixes the number of spares to 6. Afterwards, we analyze this configuration with respect to the given statistical queries. The expected cost – the sum of the spares’ cost and the cost due to unavailability – with six spares is \$488k. Most of the cost stems from the cost of the spares (\$480k). The system’s availability is 99.96% which corresponds to an expected downtime of 6 days. The probability of encountering a system failure within 40 years is less than $9E-4$. This can also be seen in Fig. 9 which shows the probability of having any downtime for different spare numbers.

Figure 10 shows the overall cost (on logarithmic scale) for different number of spares. We see that six spares indeed minimize the cost. For fewer than six spares, the cost due to system failure is too large, whereas for more than six spares, the cost of the spares becomes the major factor. In particular, buying more than six spares incurs more expenses but only marginally decreases downtime.

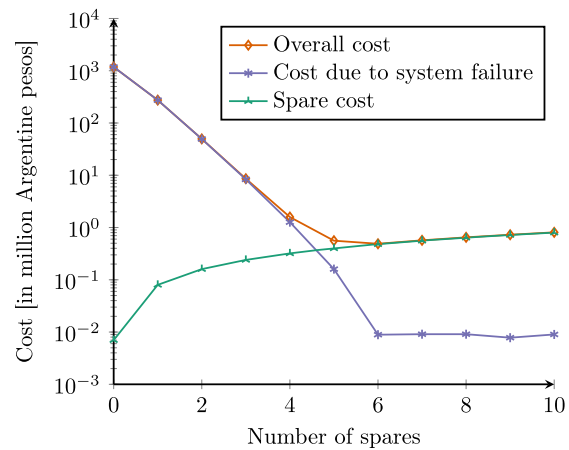


Fig. 10 Costs for different number of spares

7.2 Discussion

Rare events For six spares, the probability of any downtime within 40 years is less than $9E-4$. Thus, a system failure is a rare event and most of the time, the system will not be down at all. However, even one system failure has drastic consequences (\$1 million loss per day). It is, therefore, crucial to take these rare events into account. However, statistical model checking has issues with rare events and we had to significantly increase the number of runs in UPPAAL to obtain correct results. With the default settings of UPPAAL STRATEGO, the strategy synthesis via the learning query returned an incorrect number of spares – due to not encountering the rare events during the limited number of simulation runs. For instance, the model provides two spares as the optimal solution in the default settings. We discovered this issue by performing standard statistical model checking for a fixed number of spares, similar to Fig. 10. By increasing the number of simulation runs (cf. Table 2), we obtained the correct number of spares, again validated by statistical model checking.

Model flexibility In our SPTGA model, we use controllable transitions to choose the initial number of spares. This allows using controller synthesis queries to manually fix the

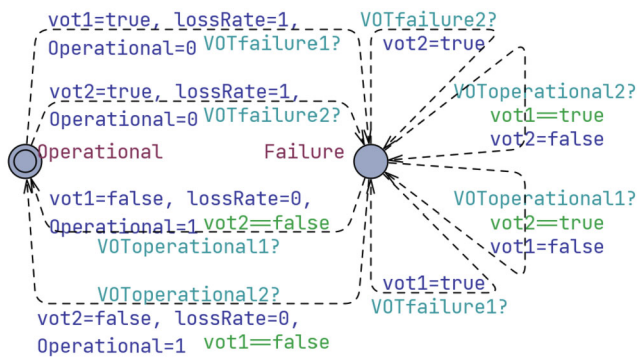


Fig. 11 SPTGA model for the OR gate

number of spares instead of creating multiple models for different numbers of spares. This modeling choice also provides flexibility for future extensions where a larger number of configurations need to be considered, such as synthesizing spare numbers for multiple components. Instead of exhaustively checking all possible configurations, this approach allows the use of UPPAAL STRATEGO's optimization algorithms to find the best configuration.

Performance We started with UPPAAL STRATEGO version 4.1.20 but switched to the recently released UPPAAL 5.0 as the latter performed significantly faster. The MinCost strategy synthesis took several days in UPPAAL STRATEGO 4.1.20, but only minutes in UPPAAL 5.0.

8 Analysis of a two-component system

In Sect. 5, for the convenience of presenting the methodology, we model the emergency shutdown system by only considering the neutron flux instrumentation component. The analysis and implementation of the model are presented in Sect. 7. In this section, we will examine the emergency system with two components. As mentioned earlier, this study does not consider the third component of the emergency shutdown system, the thermocouple. Spare management for this component type has less importance because the thermocouple is extremely inexpensive and manufactured by INVAP S.E. itself.

The neutron flux instrumentation feeds into the RPS Train via analog channels. The emergency shutdown system becomes unavailable if two out of the three components of either the Ionization chambers (components for the neutron flux instrumentation) or the RPS Train itself are unavailable. This section considers the neutron flux instrumentation (Ionization chamber) and the RPS Train together.

8.1 SPTGA models

OR gate In two-component modeling, we have an additional automaton, the OR gate. Figure 11 shows the SPTGA

model for an OR gate. This automaton has two states, either Operational or Failure. The OR gate SPTGA is connected with the voting gates through four communication channels: VOTOperational1, VOTfailure1, VOTOperational2, and VOTfailure2. Variables Operational and lossRate are used only for monitoring and cost calculation purposes. The Boolean variables vot1 and vot2 represent the failure of the voting gates associated with each of the components.

Alternatively, the need for a separate OR gate automaton can be eliminated by enhancing the VOT automaton with an additional communication channel. This channel would inform its parent gate when the voting gate recovers from a failure, allowing the parent gate to track subsystem statuses dynamically. Such an approach could streamline the modeling process by reducing the number of distinct automata required, albeit at the cost of added complexity within the voting gate automaton itself.

Extended warehouse In modeling the reactors' emergency shutdown system with two components, the SPTGA for the warehouse differs. The warehouse is currently configured in such a way that only one repair team is operational, and only one part can be replaced at any given moment. Figure 12 shows the warehouse SPTGA. At first, we have three states with committed locations. The three initial committed states in the extended warehouse automaton are used to explicitly manage the initialization of spare parts for multiple components, ensuring flexibility in future extensions where initialization logic may differ between components. While these states could be compressed into a single committed state with assignments of variables (Spare1, Spare2) as effects of outgoing transitions, this approach was chosen to maintain clarity and modularity in the current implementation.

There are both controllable and uncontrollable transitions in this automaton. By synthesizing a strategy, we can control the controllable transitions. After deciding on the initial amount of spares for both components in the warehouse, this value cannot be changed later on. Note that the RPS train is considered in groups of ten. The reason for this is that many of them will be needed in the given time frame, and due to their lower cost compared to the Ionization chamber, it is possible to buy (slightly) more of them.

In the current model, replacements are performed sequentially by a single repair team, regardless of whether multiple components fail simultaneously. This decision reflects a practical constraint in many real-world systems, where repair resources are limited. However, if the system allows for parallel replacements by different teams, the warehouse automaton could be modified to include separate repair processes for each component type. This would require extending the automaton to track multiple parallel replacement activities.

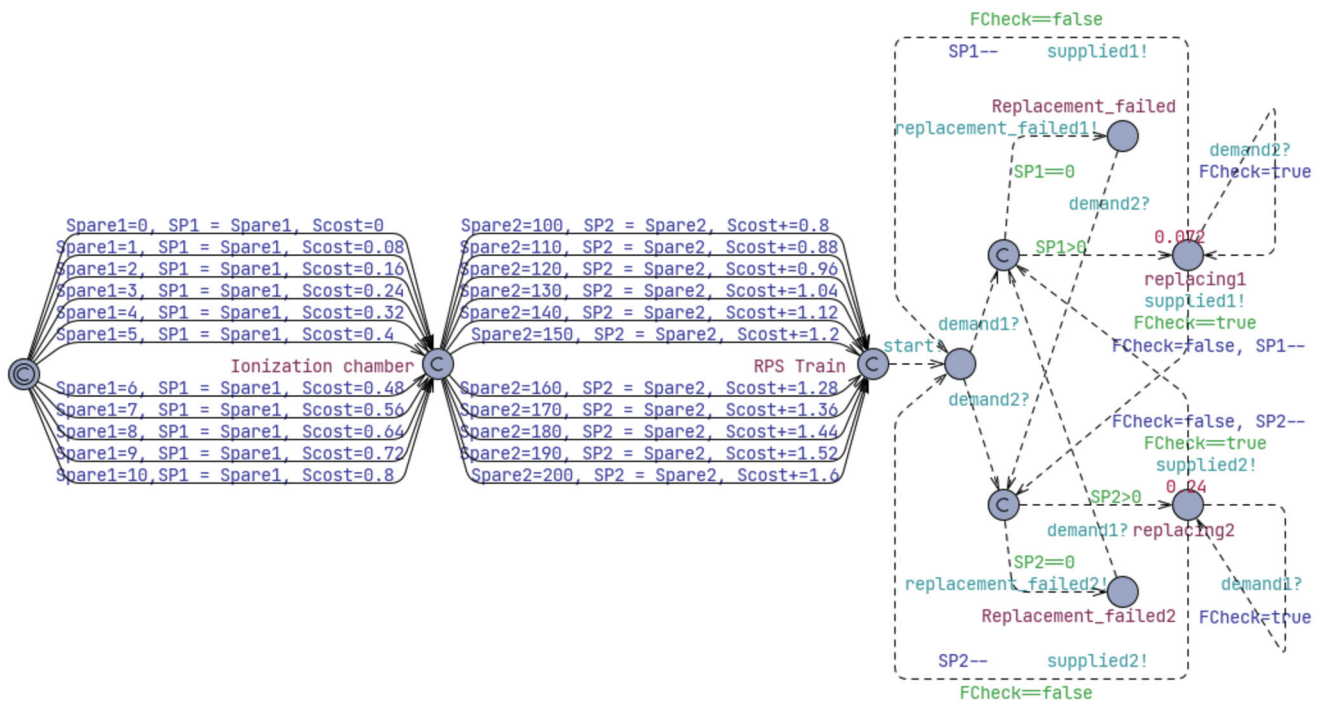


Fig. 12 SPTGA model for the warehouse gate (considering two components)

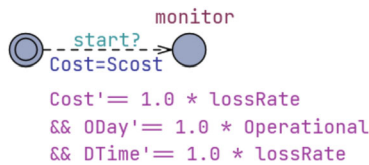


Fig. 13 SPTGA model for the monitoring automaton

Figure 12’s warehouse SPTGA shares many similarities with Fig. 7’s warehouse SPTGA. In the new warehouse model, the two states Replacing1 and Replacing2, with rates of 0.072 and 0.24, respectively, indicate the replacement rate of two distinct components, the Ionization chamber and RPS train. In this model, the Boolean variable FCheck indicates the failure of another component (of a different type) while the present component is being replaced. The main goal of using FCheck is to follow a “fairness policy” which alternates between the components of both types when performing the replacements.

Monitoring automaton Figure 13 shows the SPTGA model for a monitoring automaton. This automaton is solely for the purpose of monitoring the system’s cost and availability. The monitoring process begins with the Start? communication channel once the warehouse automaton has fixed the quantity of spare components. The Cost variable takes the value of Scost, which is equivalent to the cost of purchasing spare components. Depending on whether the system failed or not, the variables LossRate and Operational receive

values of 0 or 1. The variables Cost, ODay, and Dtime modify their values on a continuous basis.

8.2 Results

Similar to Sect. 6, we use two types of queries for model checking and calculate the desired dependability metrics, namely strategy queries and statistical queries. In Table 3, we used a different approach to calculate the maximum cost and availability of the system by using the strategy as a controller synthesis query, as discussed in Sect. 6, to fix the number of spare parts on the optimal number. In Table 4, we did not use a controller synthesis query because the learning query stores the model with the optimal number of spares. We use statistical queries to formally specify the analysis metrics Q_1^A, \dots, Q_m^A from Sect. 2.4 for two components.

Table 4 shows the results of the strategy and statistical queries performed on the SPTGA based on two components. The first query in Table 4 is a learning query that is used to synthesize the optimal number of spares needed for both components to reduce the cost. This query is the same as before. In the two-component system, the MinCost strategy synthesizes the optimal number of spares, determining 6 spares for the ionization chamber and 140 spares for the RPS train. As with the single-component case, MinCost identifies the optimal configuration but does not directly enforce it in the model. Subsequent statistical queries are performed under the strategy to evaluate the associated cost and availability metrics. We use the second and third queries to obtain

Table 4 Verification results for the two-component system

Query	Result	Comp. time
strategy MinCost = minE(Cost) [≤ 14600]: $\langle \rangle$ GlobalTime==14600	✓	45,995.00 s
E [$\leq 14600; 100000$] (min: Spare1) under MinCost	6	0.01 s
E [$\leq 14600; 100000$] (min: Spare2) under MinCost	140	0.01 s
E [$\leq 14600; 100000$] (max: Cost) under MinCost	10.400 \pm 0.140	2 s
E [$\leq 14600; 100000$] (max: ODay/14600) under MinCost	0.99932 \pm 0.00013	2 s
Pr[≤ 14600] ($\langle \rangle$ DTime >0) under MinCost	0.8659 \pm 0.0009	4.12 s

the optimal number of spares per component type from the synthesized strategy. The analysis yields that 6 is the optimal number of spares for the ionization chamber and 140 spares should be on stock for the RPS train.

We further analyze the optimal configuration using statistical queries. The system's availability is 99.93%, which corresponds to an expected downtime of 11 days. The probability of encountering a system failure within 40 years is 86.59%. The results reveal a substantial increase in the probability of downtime when comparing single and combined subsystems. This is primarily due to shared constraints, such as a single repair team handling multiple subsystem failures, and higher failure rates in certain subsystems like the RPS train.

8.3 Discussion

Comparison to single component For the ionization chamber, we obtain the same number of spares (6) through both the single component and the two-component system, see Tables 3 and 4. This indicates that the spare management for both component types could be considered independently in this case. However, such an independence does not hold in general: if one component is significantly more failure-prone, the other component becomes less important and thus, might require less spares.

System downtime The two-component system exhibits a dramatic increase in the probability of failure (0.866) compared to the one-component system ($9E-4$). One reason is that only one replacement per time unit is considered in the warehouse model. Another reason is that the failure rate of RPS train is two orders of magnitude larger than the ionization chamber. RPS train components therefore fail more often and it is more likely that two of the components are failed simultaneously—even with the higher replacement rates. The higher failure rates of RPS train is also reflected by the fact that 140 spares need to be on stock.

9 Conclusion and future work

We presented an approach for optimal spare management by combining fault tree analysis and statistical model checking. Our approach finds the optimal number of spares as a trade-off between costs and reliability. Given fault trees equipped with costs and spares, we translate them into stochastic priced timed game automata (SPTGA) and use UPPAAL STRATEGO to synthesize an optimal strategy, i.e., the optimal number of spares. We applied our approach to two subsystems of the emergency shutdown system of a research reactor. Using UPPAAL STRATEGO, we found the optimal number of spares for both a system with one component and a system consisting of two different types of components.

Future work There are multiple directions for future extensions. First, we can take advantage of UPPAAL's modeling flexibility to extend the SPTGA modeling techniques and include, e.g., a dynamic warehouse model allowing on-demand ordering, variable delivery times, or use dynamic fault trees [23] as the initial model. These extensions allow for more realistic modeling of more complex spare management scenarios.

A second follow-up to the current work would be to automate the translation from fault trees to SPTGAs. This translation would provide a convenient way to analyze larger systems consisting of various types of components. UPPAAL offers extensive language for defining templates in the form of extended timed automata. It supports template instantiation. The grammar for Instantiation can define new templates based on preexisting ones. This allows for such an automated process. In order to extend the translation to spare management, however, one needs to develop a standardized format to express spare management scenarios.

Another follow-up to the current work could extend this framework to more complex tree-based risk analyses, such as Attack-Fault-Defense Trees (AFDT), which integrate safety, security, and countermeasure modeling into a unified structure [33]. Such extensions could provide deeper insights into the interdependencies between failures and adversarial actions in cyber-physical systems.

Lastly, in order to analyze larger systems, a closer look into performance improvements in UPPAAL 5.0 compared to UPPAAL STRATEGO is needed. In particular, one needs to study what causes the increase in computation time seen for the two-component system. Regarding the analysis, dedicated methods to support rare-event simulation with UPPAAL are beneficial to keep the number of simulation runs at a feasible level.

Funding information This work has been partially funded by the NWO grant NWA.1160.18.238 (PrimaVera), by the ERC Consolidator Grant 864075 (CAESAR) and by EU Horizon 2020 project MISSION, number 101008233.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Hu, Q., Boylan, J.E., Chen, H., Labib, A.: OR in spare parts management: a review. *Eur. J. Oper. Res.* **266**(2), 395–414 (2018). <https://doi.org/10.1016/j.ejor.2017.07.058>
- Zhang, X., Zeng, J.: Joint optimization of condition-based opportunistic maintenance and spare parts provisioning policy in multiunit systems. *Eur. J. Oper. Res.* **262**(2), 479–498 (2017). <https://doi.org/10.1016/j.ejor.2017.03.019>
- Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. *Comput. Sci. Rev.* **15**, 29–62 (2015). <https://doi.org/10.1016/j.cosrev.2015.03.001>
- Heijblom, R., Postma, W., Natarajan, V., Stoelinga, M.: In: 2018 Annual Reliability and Maintainability Symposium (RAMS), pp. 1–7 (2018). <https://doi.org/10.1109/RAM.2018.8463074>
- David, A., Jensen, P.G., Larsen, K.G., Mikucionis, M., Taankvist, J.H.: In: TACAS. Lecture Notes in Computer Science, vol. 9035, pp. 206–211. Springer, Berlin (2015). https://doi.org/10.1007/978-3-662-46681-0_16
- Legay, A., Delahaye, B., Bensalem, S.: In: RV, Lecture Notes in Computer Science, vol. 6418, pp. 122–135. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-16612-9_11
- Chatain, T., David, A., Larsen, K.G.: In: ADHS, IFAC Proceedings Volumes, vol. 42, pp. 238–243. Elsevier, Amsterdam (2009). <https://doi.org/10.3182/20090916-3-ES-3003.00042>
- Soltani, R., Volk, M., Diamonte, L., Lopuhaä-Zwakenberg, M., Stoelinga, M.: In: International Conference on Formal Methods for Industrial Critical Systems, pp. 205–223. Springer, Berlin (2023). https://doi.org/10.1007/978-3-031-43681-9_12
- Soltani, R., Volk, M., Diamonte, L., Lopuhaä-Zwakenberg, M., Stoelinga, M.: Artefact for “optimal spare management via statistical model checking: a case study in research reactors” (2023). <https://doi.org/10.5281/zenodo.8199172>
- Zhang, S., Huang, K., Yuan, Y.: Spare parts inventory management: a literature review. *Sustainability* **13**(5) (2021). <https://doi.org/10.3390/su13052460>
- Tusar, M.I.H., Sarker, B.R.: Spare parts control strategies for offshore wind farms: a critical review and comparative study. *Wind Eng.* **46**(5), 1629–1656 (2022). <https://doi.org/10.1177/0309524X221095258>
- Chen, F., Chen, Y., Kuo, J.: Applying moving back-propagation neural network and moving fuzzy-neuron network to predict the requirement of critical spare parts. *Expert Syst. Appl.* **37**(9), 6695–6704 (2010). <https://doi.org/10.1016/j.eswa.2010.04.037>
- Gutierrez, R.S., Solis, A.O., Mukhopadhyay, S.: Lumpy demand forecasting using neural networks. *Int. J. Prod. Econ.* **111**(2), 409–420 (2008). <https://doi.org/10.1016/j.ijpe.2007.01.007>. Special Section on Sustainable Supply Chain
- Kourentzes, N.: Intermittent demand forecasts with neural networks. *Int. J. Prod. Econ.* **143**(1), 198–206 (2013). <https://doi.org/10.1016/j.ijpe.2013.01.009>
- Li, S.G., Kuo, X.: The inventory management system for automobile spare parts in a central warehouse. *Expert Syst. Appl.* **34**(2), 1144–1153 (2008). <https://doi.org/10.1016/j.eswa.2006.12.003>
- Wu, P., Hung, Y., Lin, Z.: Intelligent forecasting system based on integration of electromagnetism-like mechanism and fuzzy neural network. *Expert Syst. Appl.* **41**(6), 2660–2677 (2014). <https://doi.org/10.1016/j.eswa.2013.11.007>
- Zheng, M., Ye, H., Wang, D., Pan, E.: Joint optimization of condition-based maintenance and spare parts orders for multi-unit systems with dual sourcing. *Reliab. Eng. Syst. Saf.* **210**, 107512 (2021). <https://doi.org/10.1016/j.res.2021.107512>
- Soltani, R., Kang, E.Y., Mena, J.E.H.: Verification and optimization of cyber-physical systems: Preprint for FedCSIS (2021). [arXiv:2109.01574](https://arxiv.org/abs/2109.01574). arXiv preprint
- Soltani, R., Kang, E.Y., Mena, J.E.H.: In: FedCSIS (Position Papers), pp. 205–210 (2021). <https://doi.org/10.15439/2021F125>
- Kumar, R., Ruijters, E., Stoelinga, M.: In: FORMATS. Lecture Notes in Computer Science, vol. 9268, pp. 156–171. Springer, Berlin (2015). https://doi.org/10.1007/978-3-319-22975-1_11
- Kumar, R., Stoelinga, M.: In: HASE, pp. 25–32. IEEE Comput. Soc., Los Alamitos (2017). <https://doi.org/10.1109/HASE.2017.12>
- David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: In: FORMATS. Lecture Notes in Computer Science, vol. 6919, pp. 80–96. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-24310-3_7
- Ruijters, E., Stoelinga, M.: In: ISoLA (1). Lecture Notes in Computer Science, vol. 9952, pp. 151–165 (2016). https://doi.org/10.1007/978-3-319-47166-2_10
- Jepsen, S.C., Worm, T., Johansen, A., Lazarova-Molnar, S., Kjærgaard, M.B., Kang, E.Y., Friederich, J., Heredia Mena, J.E., Soltani, R., Sørensen, S.L., Hjort Schwee, J.: In: 2021 International Symposium ELMAR, pp. 143–150 (2021). <https://doi.org/10.1109/ELMAR52657.2021.9550961>
- Kabir, S.: An overview of fault tree analysis and its application in model based dependability analysis. *Expert Syst. Appl.* **77**, 114–135 (2017). <https://doi.org/10.1016/j.eswa.2017.01.058>
- Maler, O., Pnueli, A., Sifakis, J.: In: STACS. Lecture Notes in Computer Science, vol. 900, pp. 229–242. Springer, Berlin (1995). https://doi.org/10.1007/3-540-59042-0_76
- Bulychev, P.E., David, A., Larsen, K.G., Mikucionis, M., Poulsen, D.B., Legay, A., Wang, Z.: In: QAPL, EPTCS, vol. 85, pp. 1–16 (2012). <https://doi.org/10.4204/EPTCS.85.1>
- Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: In: CAV. Lecture Notes in Computer Science, vol. 4590, pp. 121–125. Springer, Berlin (2007). https://doi.org/10.1007/978-3-540-73368-3_14
- David, A., Jensen, P.G., Larsen, K.G., Legay, A., Lime, D., Sørensen, M.G., Taankvist, J.H.: In: ATVA. Lecture Notes in Computer Science, vol. 8837, pp. 129–145. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-11936-6_10

30. Boudali, H., Crouzen, P., Stoelinga, M.: In: DSN, pp. 708–717. IEEE Comput. Soc., Los Alamitos (2007). <https://doi.org/10.1109/DSN.2007.37>
31. Ruijters, E., Guck, D., van Noort, M., Stoelinga, M.: In: DSN, pp. 662–669. IEEE Comput. Soc., Los Alamitos (2016). <https://doi.org/10.1109/DSN.2016.67>
32. UPPAAL requirements specification language. <https://docs.uppaal.org/language-reference/requirements-specification/>. Accessed: 2023-05-26
33. Soltani, R., Lopuhaä-Zwakenberg, M., Stoelinga, M.: In: Ceccarelli, A., Trapp, M., Bondavalli, A., Bitsch, F. (eds.) Computer Safety, Reliability, and Security, pp. 218–232. Springer, Cham (2024)

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.