micromachines

MDPI

Article

Counterexample Generation for Probabilistic Model Checking Micro-Scale Cyber-Physical Systems

Yang Liu [1,*], Yan Ma [2], Yongsheng Yang [1] and Tingting Zheng [2]

[1] Institute of Logistics Science and Engineering, Shanghai Maritime University, Shanghai 201306, China; yangys@shmtu.edu.cn
[2] School of Computing, National University of Singapore, Singapore 117417, Singapore; yanma_nus@126.com (Y.M.); willow-2001@163.com (T.Z.)
* Correspondence: yl.nus.sg@gmail.com

Abstract: Micro-scale Cyber-Physical Systems (MCPSs) can be automatically and formally estimated by probabilistic model checking, on the level of system model MDPs (Markov Decision Processes) against desired requirements in PCTL (Probabilistic Computation Tree Logic). The counterexamples in probabilistic model checking are witnesses of requirements violation, which can provide the meaningful information for debugging, control, and synthesis of MCPSs. Solving the smallest counterexample for probabilistic model checking MDP has been proven to be an NPC (Non-deterministic Polynomial complete) problem. Although some heuristic methods are designed for this, it is usually difficult to fix the heuristic functions. In this paper, the Genetic algorithm optimized with heuristic, i.e., the heuristic Genetic algorithm, is firstly proposed to generate a counterexample for the probabilistic model checking MDP model of MCPSs. The diagnostic subgraph serves as a compact counterexample, and diagnostic paths of MDP constitute an AND/OR tree for constructing a diagnostic subgraph. Indirect path coding of the Genetic algorithm is used to extend the search range of the state space, and a heuristic crossover operator is used to generate more effective diagnostic paths. A prototype tool based on the probabilistic model checker PAT is developed, and some cases (dynamic power management and some communication protocols) are used to illustrate its feasibility and efficiency.

Keywords: probabilistic model checking; micro-scale cyber-physical systems; counterexample; genetic algorithm

Citation: Liu, Y.; Ma, Y.; Yang, Y.; Zheng, T. Counterexample Generation for Probabilistic Model Checking Micro-Scale Cyber-Physical Systems. *Micromachines* **2021**, *12*, 1059. https://doi.org/10.3390/mi12091059

Academic Editors: Hamid Reza Karimi, Birgit Vogel-Heuser and Ruwan Gopura

Received: 23 July 2021
Accepted: 29 August 2021
Published: 31 August 2021

1. Introduction

Micro-scale Cyber-Physical Systems (MCPSs) are a special kind of CPS in micromachines, which are composed of micro/nanoscale components, and the integration of computation with physical processes in micro-/nano-assembly operations. MCPSs are about the intersections, not the union, of the physical and the cyber, and the behaviors of them are defined by both cyber and physical parts of the system [1–3]. Most of constituent elements of MCPSs or MCPSs themselves are usually accompanied with stochastic behaviors. The reasons for this can be classified as three aspects: (1) MCPSs contain the randomized algorithms, e.g., leader election algorithm, consensus algorithms; (2) unreliable and unpredictable system behaviors incurred by execution environment, e.g., message loss, processor failure; (3) performance evaluation by random variables assigned artificially, e.g., reliability, availability [4,5]. As an automated and complete formal verification technique at the level of system models, probabilistic model checking can be used to estimate whether the MCPSs satisfy a desired requirement property, or avoid an undesired outcome. As shown in Figure 1, MCPSs are modeled as DTMCs (Discrete-time Markov Chains), MDPs (Markov Decision Processes), PTA (Probabilistic tic Timed Automata), etc. [5]. The achieved requirement properties, e.g., function, reliability, robust, etc., are specified by PCTL (Probabilistic Computation Tree Logic), LTL (Liner Temporal logic) with probability bounds,

PTCTL (Probabilistic Timed Computation Tree Logic) [5–7]. In this way, we can express the requirement property such as "in the MCPSs, the maximum probability to reach a set of bad states is not more than 0.001". Some probabilistic model checking tools, such as PRISM [6] and PAT [7], have been developed and applied to quantitatively estimate, control, and synthesize the MCPSs, e.g., the mobile service robot [8], unmanned undersea vehicle [9], peacemaker [10–12]. As shown in Figure 1, verifying or evaluating MCPSs is a huge challenge, which involves hardware, software, communication protocols, and so on. Probabilistic model checking has the potential to address this. At present, there are two dimensions to adapt the probabilistic model checking to estimate MCPSs: (1) horizontal, extending system models, temporal logics, and corresponding algorithms to verify more complex behaviors of MCPSs; (2) vertical, optimizing the probabilistic model checking algorithm to provide more functions and better performance for verifying a certain part of MCPSs deeply.

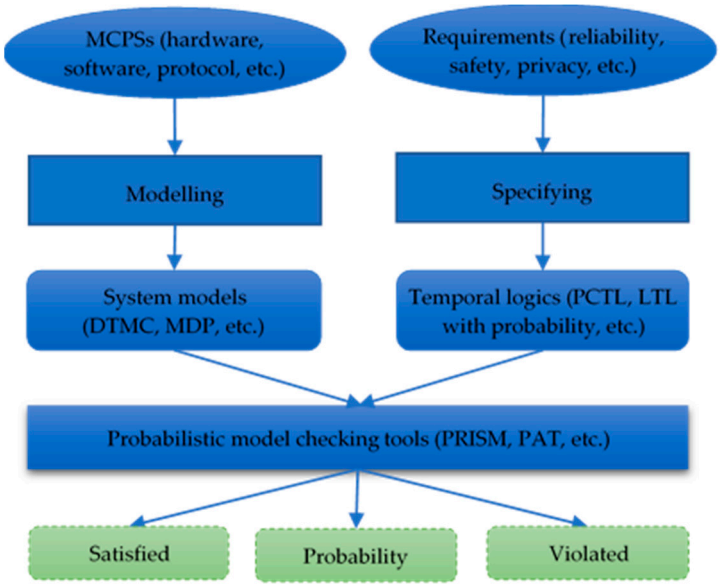

Figure 1. Schematic view of estimating MCPSs by probabilistic model checking.

This work belongs to the vertical dimension, which propose a counterexample generation method for probabilistic model checking MCPSs with nondeterministic and discrete-time stochastic behaviors. Up to now, existing probabilistic model checking tools cannot provide a counterexample directly. Counterexamples play a very important role in estimating or verifying MCPSs: (1) Counterexamples can feedback which parts of the system violate the requirements and provide diagnostic information; (2) Counterexamples are very effective in model-based testing, which can provide a reference for the design of test cases; (3) In the process of abstract refinement, counterexamples can provide guidance information for the refinement of rough abstract models [13,14]; (4) Counterexamples can be used to obtain the core of feasible plans in planning, such as task scheduling [15]; (5) Counterexamples are recently used to synthesize attacks for showing how confidentiality of systems can be broken, and the quality assurance of multi-agent systems [16].

1.1. Related Works

The counterexample in probabilistic model checking may be a set of diagnostic paths that satisfy a given property, and the cumulative probability mass does not satisfy a given bound. It cannot be generated during the process of probabilistic model checking and needs the dedicated algorithm [17]. We divide the existing works into two kinds of methods for generating a counterexample in probabilistic model checking: the accurate and approximate approach.

1.1.1. Accurate Approach

Han, Katoen, et al. [18] provide the theoretical and algorithmic foundations for counterexample generation in probabilistic model checking, which is the earliest research on the counterexample in probabilistic model checking. They define the counterexample for DTMC and translate the counterexample solving problem into the shortest path problem in the corresponding directed weight graph. Thus, the counterexample generation can explicitly enumerate the paths according to the probability they carry. To obtain the smallest counterexample, the enumeration stops when the cumulative probability of all generated paths exceeds the probability bound set in advance. Algorithmically, this can be done efficiently by translating this problem into a k-Shortest-Paths problem, where k is not fixed in advanced and is determined during the calculation. Eppstein and REA algorithms are applied to generate a counterexample for DTMC. On the basis of [19], the approach of calculating and representing a counterexample in a succinct way using a regular expression is presented in [20]. According to [20], the regular expressions that represent a counterexample can be calculated by the state elimination method of the automaton theory, which is guided by the k shortest paths search. The counterexample can also be compacted based on the *strongly connected components* (SCCs), and the obtained acyclic model is helpful in reducing efforts to determine the counterexample.

A hierarchical counterexample generated by performing SCC reduction is presented in [21]. Considering the unbounded-until property formula $P_{\leq p}\left(\Phi U^{\leq h}\Psi\right)$, where $h = \infty$, counterexample generation can be carried out by applying k shortest paths algorithms such as Epstein algorithm, and the time complexity of which is $O(a + blogb + c)$, where a is the number of states and b is the number of transitions of DTMC. For bound-until property $P_{\leq p}\left(\Phi U^{\leq h}\Psi\right)$, where $h \in N$, a recursive enumeration algorithm to generate counterexample is presented in [20]. Reference [22] defines the counterexample of MDP as the general DTMC. Lal and Prabhakar [14] use a counterexample generated by this method to guide the abstraction-refinement for the polyhedral probabilistic hybrid systems.

Taking PCTL path formula $\Phi_1 U \Phi_2$ as an example, the process of generating a counterexample for MDP by the Eppstein algorithm can be summarized as follows: (1) make Φ_2-states and all $\neg\Phi_1 \wedge \neg\Phi_2$-states absorbing, (2) insert a sink state and redirect all outgoing edges of Φ_2-states to it, (3) turn it into a weighted digraph, (4) implicit representation of paths, (5) represent paths by a heap, (6) find the k shortest paths as the counterexample.

1.1.2. Approximate Approach

A critical subsystem, as a subsystem of DTMC, can represent a counterexample for probabilistic model checking. It is called a minimal critical subsystem if the number of states and transitions included is minimal compared to other subsystems, and the smallest subsystem if it is minimal and carries a maximal probability to reach a target state. Chadha et al. [23] further define the counterexample as the general MDP and give the corresponding algorithm for solving the minimal counterexample. This is the first work of counterexample generation for MDP.

Generating the smallest critical subsystems to represent the counterexample is an NP-complete problem which has been proved in [23,24], and it is hard to solve with exact and complete algorithms. A heuristic search method like best first search to obtain a counterexample is presented in [25,26]. However, it may not be a smallest, or even a minimal counterexample. Symbolic methods like bounded model checking (BMC) for finding a small critical subsystem have been developed in [25,26]; it uses a symbolic representation to reduce the size of the counterexample. Another option to determine the smallest critical subsystem is to use mixed integer linear programming techniques in [27,28]. Aljzzar et al. [29] propose a directed state space search method called XBF (XZ, XUZ) to generate a counterexample, which is the first work of counterexample generation with a heuristic. There are some works in this direction, such as [13,30,31], which optimizes the heuristics to generate a counterexample for DTMC.

XBF extends the search strategy BF (Best-First) to generate a counterexample for MDP. In the framework of generating a counterexample for MDP by the Eppstein algorithm, XBF makes the following three modifications: (1) XBF records all parent states for each state; (2) XBF maintains a graph which contains, at any point in the search, the currently selected diagnostic subgraph; (3) when XBF finds the first target state, it continues the search for further target states until the whole state space has been processed, or termination is explicitly requested.

1.2. Our Contribution

This paper deals with counterexample generation for probabilistic model checking MCPSs with nondeterministic and discrete-time stochastic behaviors, in which MCPSs are modeled as MDP, and the requirements are specified as PCTL. Solving the smallest counterexample for probabilistic model checking MDP has been proven to be an NPC problem. Although some heuristic methods are designed to generate the counterexample for MDP, it is usually difficult to fix the global heuristic functions. Instead, we design the local heuristic function integrated into the Genetic algorithm (heuristic Genetic algorithm, HGA) to explore the state space, and propose a counterexample generation method for probabilistic model checking MDP against requirement property in PCTL. We use a diagnostic subgraph to represent the compact counterexample for MDP, and exploit diagnostic paths to constitute an AND/OR tree for constructing the diagnostic subgraph. We adopt the indirect path coding of the Genetic algorithm to extend the search range of state space, and the heuristic crossover operator to generate more effective diagnostic paths. A corresponding prototype tool is implemented based on PAT [16], and some MCPSs cases are used to illustrate its feasibility and efficiency.

1.3. Outline of the Paper

The rest of this paper is organized as follows. In Section 2, we introduce some preliminaries and definitions. Section 3 describes how to optimize the Genetic algorithm with a heuristic to generate a counterexample for MCPSs model MDPs. In Section 4, we show an experimental evaluation of some MCPSs cases, in order to illustrate the feasibility and efficiency. Conclusion and future work are in Section 5.

2. Preliminaries

2.1. MDP

An MDP can be viewed as an extension of DTMC, which permits both probabilistic and non-deterministic choices. In an MDP, each transition includes a non-deterministic choice of actions in state s. Formally, an MDP is defined as follows.

Definition 1 (MDP). *Let AP be a set of atomic propositions. A Markov decision process M is a tuple* (S, s_{init}, A, P, L)*, where S is a finite set of states,* $s_{init} \in S$ *is the initial state, A is a set of actions,* $P: S \times A \times S \to [0, 1]$ *is a probability transition function such that for every state,* $s \in S$ *and an action* $\alpha \in A$*:* $\sum s' \in S$*,* $P(s, \alpha, s') \in \{0, 1\}$*, and* $L: S \to 2^{AP}$ *is a labeling function of atomic propositions.*

For a state $s \in S$, the probability of s to its successor state s' by action a is given by $P(s, \alpha, s')$. If and only if $\sum_{s' \in S} P(s, \alpha, s') = 1$, we call the action α enabled in the state s, otherwise, the action α is disabled. We use $A(s)$ to represent a set of actions that are enabled at state s.

A path represents the execution of the system modeled by MDP; that is, the possible event behavior of the system, which can be described as an infinite sequence of states.

Definition 2 (Path). *An (infinite) path of MDP M is an infinite sequence* $\sigma = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \ldots$ *with* $\alpha_i \in A(s_i)$ *such that* $P(s_i, \alpha_i, s_{i+1}) > 0$ *for all* $i \geq 0$.

We use $len(\sigma)$ to denote the length of σ, which is determined by the number of states. For an infinite path σ, $len(\sigma)$ is ∞. For a natural number i such that $0 \leq i < len(\sigma)$, $\sigma[i]$ refers to the ith state of σ, i.e., s_i. Using $\sigma^{(i)}$ to indicate the ith prefix of σ, formally, $\sigma^{(i)} = s_0 \xrightarrow{\alpha_0} \ldots \xrightarrow{\alpha_{i-1}} s_i$, which represents the $i+1$ state of the path σ. For $0 \leq i \leq len(\sigma)$, $A_\sigma(i)$ denotes the ith action in σ, namely α_i. For a finite path, $last(\sigma)$ denotes the last state of σ. $Paths^M$ and $Paths^M_{fin}$ represent the set of infinite paths and the set of all finite paths in M, respectively. $Paths^M(s)$ denotes the set of infinite paths which start at s and $Paths^M_{fin}(s)$ denotes the set of finite paths starting from s. Non-determinism in an MDP is resolved by schedulers (also called adversary, policy, or strategy). A scheduler for an MDP $M = (S, s_{init}, A, P, L)$ is a function mapping every finite path σ in M onto an action $d(\sigma) \in A(last(\sigma))$. According to [32], we consider the deterministic scheduler that can deterministically select actions, which induces the maximal and minimal probability measures. The scheduler converts MDP into DTMC for which the probability of paths is measurable. We refer to the set of infinite paths under this schedule as $Paths_d(s_0)$. Paths in an MDP are called valid if the paths are allowed under a given scheduler d. We give the definition of the valid path as follows.

Definition 3 (Valid path). *A finite or an infinite path σ in an MDP is valid under a scheduler d, if and only if for all i, $0 \leq i \leq l(\sigma) - 1$, it holds that $A_\sigma(i) = d\left(\sigma^{(i)}\right)$ and $A_\sigma(i)(s_{i+1}) > 0$. Otherwise, the path σ is invalid under scheduler d.*

The underlying σ-algebra is formed by the cylinder sets which are induced by finite paths under the scheduler denoted $FinitePaths_d(s_0)$. The probability of this cylinder set is computed by using the following equation:

$$Pr_d(\sigma \in FinitePaths_d(s_0) \,|\, \sigma = s_0 \xrightarrow{\alpha_0} \ldots \xrightarrow{\alpha_{i-1}} s_i) = \Pi_{0 \leq i \leq n} P(s_i, \alpha_i, s_{i+1}) \quad (1)$$

Example 1. *Figure 2 is an illustrative example for MDP, which models a simple communication protocol in MCPSs. In this MDP, state set $S = \{s_0, s_1, s_2, s_3\}$, A = {start, wait, send, restart, stop}, atomic propositions set AP = {try, succ, fail}, $L(s_0) = \varnothing$, $L(s_1)$ = {try}, $L(s_2)$ = {fail}, $L(s_3)$ = {succ}. s_0 is the initial state; it starts trying to send a message after one step. Then, there is a nondeterministic choice between: (1) waiting a step as the channel is busy, i.e., path $\sigma = s_0 \xrightarrow{start} s_1 \xrightarrow{wait} s_1$, (2) sending the message. If the latter, it is to send successfully with probability 0.99 and stop, i.e., path $\sigma = s_0 \xrightarrow{start} s_1 \xrightarrow{send} s_3 \xrightarrow{stop} s_3$; and it is failed to send with probability 0.01 and restart, i.e., path $\sigma = s_0 \xrightarrow{start} s_1 \xrightarrow{send} s_2 \xrightarrow{restart} s_0$.*

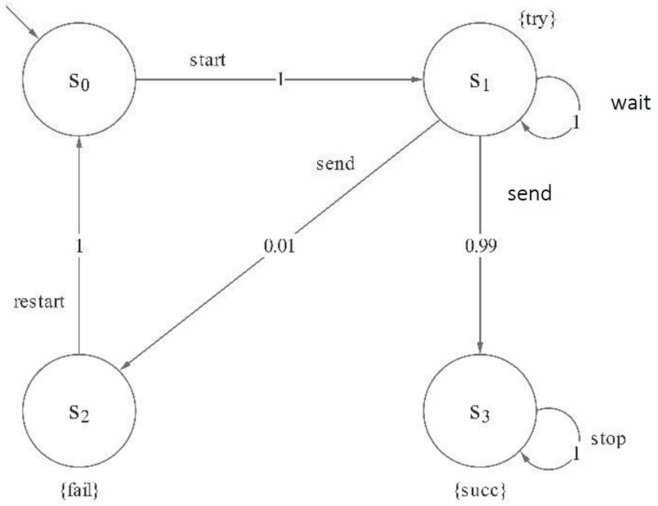

Figure 2. MDP model of a simple communication protocol.

2.2. Probabilistic Computation Tree Logic

We consider the calculation of the (constrained) reachability probability in MDP. Let a finite MDP $M = (S, s_{init}, A, P, L)$ and $B \subseteq S$ a set of a target. The maximal probability of reaching a state in B from the initial state of MDP is equivalent to determining $Pr^{max}(s \vDash_M \Diamond B) = sup_D Pr(s \vDash \Diamond B)$. Note that the supremum ranges over all, potentially infinitely many, schedulers for M. Probabilistic computation tree logic (PCTL) is a probabilistic branching-time temporal logic. PCTL state formulae over the atomic propositions set AP are formed according to the following grammar:

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid P_{\sim p}(\phi)$$

where $a \in AP$, ϕ is a path formula, $\sim \in \{<, \leq, >, \geq\}$, and $p \in [0, 1]$. PCTL path formula is formed according to the following grammar:

$$\phi ::= \Phi_1 U \Phi_2 \Big| \Phi_1 U^{\leq n} \Phi_2$$

where Φ_1 and Φ_2 are state formulae, and $n \in \mathrm{IN}_{\geq 0} U \{\infty\}$. The formula $\Phi_1 U \Phi_2$ means that Φ_2 is satisfied and all preceding states satisfy Φ_1. The path formula $\Phi_1 U^{\leq n} \Phi_2$ is the step-bounded variant of $\Phi_1 U \Phi_2$. $n = \infty$ in a path formula means that it is unbounded until formula.

Let $s \in S$ be a state and σ be an infinite path under D, where D denotes the set of all schedulers in MDP; the semantics of PCTL formulas over MDP are defined by satisfaction relation $\vDash_D$:

$$s \vDash_D P_{\sim p}(\phi) \text{ iff } Pr_s^D(\left\{\sigma \in Path^D(s) \middle| \sigma \vDash_D \phi\right\}) \sim p$$

$\sigma \vDash_D \Phi U^{\leq h} \Psi$ iff $\exists i \leq h$, such that $\sigma[i] \vDash_D \Psi$ and $\exists \forall j < i : \sigma[j] \vDash_D \Phi$.

We abbreviate $Pr_s^D(\{\sigma \in Path^D(s) | \sigma \vDash_D \phi\})$ as $Pr_s^D(\phi)$ for convenience. It needs to calculate the probability of each path under D starting from s and satisfying ϕ, and determine which state satisfies the formula $P_{\sim p}(\phi)$. We use the set $Sat(P_{\sim p}(\phi))$ to represent all states that satisfy $P_{\sim p}(\phi)$. For MDP $M = (S, s_{init}, A, P, L)$, a formula $P_{\sim p}(\phi)$ is satisfied if and only if for every $d \in D : Pr_d(\phi) \sim p$, where $Pr_d(\phi)$ denotes the probability of the set of all finite paths satisfying ϕ under scheduler d. The probability of paths in MDP is only defined in a given scheduler d. Probabilistic model checking MDP should consider the maximizing or minimizing probability values incurred by the different schedulers. Let $Pr^{max}(\phi)$ denote the maximal probability mass at which a MDP M satisfies ϕ, $Pr^{max}(\phi) = max(Pr_d(\phi) | d \in D)$, and dually, the minimal probability $Pr^{min}(\phi) = min(Pr_d(\phi) \Big| d \in D)$. For properties in the upper bound, it is obvious that $M \vDash P_{\leq p}(\phi) \Leftrightarrow Pr^{max}(\phi) > p$.

2.3. Genetic Algorithm

The Genetic algorithm is a heuristic search, inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. The Genetic algorithm has been applied to a board range of learning and optimization problems [33]. It begins with a set of a random population of coded candidate solutions which are called the chromosome. Then, the fitness is evaluated, measured by the fitness function of each candidate solution in the current population, and the fittest candidate solution is selected as parent of the next generation. The next step is to generate a second-generation population of solutions from those selected through crossover and mutation. The fittest parents and the new offspring form a new population. We give the standard Genetic algorithm in pseudocode, as shown in Algorithm 1.

Algorithm 1. Standard Genetic algorithm.

START
Generate the initial population
Compute fitness
REPEAT
    Selection
    Crossover
    Mutation
    Compute fitness
UNTIL population has converged
STOP

3. Counterexample Generation with Heuristic Genetic Algorithm

In this section, we adopt a diagnostic subgraph to represent the counterexample, and optimize the Genetic algorithm with heuristic (heuristic Genetic algorithm, HGA) to generate a counterexample for probabilistic model checking MCPSs. The system model of MCPSs is MDP, and the requirement property is expressed by PCTL, i.e., $P_{\sim p}(\phi)$. We will focus on the formula $P_{\leq p}(\phi)$ in PCTL; the lower bounded formulas $P_{>p}(\phi)$ and $P_{\geq p}(\phi)$ can be converted to upper bound formulas.

3.1. Counterexample Represented by Diagnostic Subgraph

In classical model checking, the checked property will determine the shape of a counterexample. For linear temporal logic such as LTL, a failure path serves as a counterexample; and for CTL, the path as a counterexample is only for a subclass of global quantifier formulas. In probabilistic model checking, the situation is very complex. Let $\phi = \Phi_1 U^{\leq n} \Phi_2$, $s \not\models_D P_{\leq p}(\phi)$, if and only if $Pr_s^D(\phi) > p$ holds. Consequently, if $P_{\leq p}(\phi)$ does not satisfy at state s, then the probability sum of all paths which satisfy formula ϕ with initial state s exceeds the probability bound p. It is obvious that the counterexample can be represented by finite paths. For a low-bounded property formula $P_{\geq p}(\phi)$, the definition of the counterexample does not make sense, since the empty set satisfies this condition; diagnostic path $Pr_s^D(\phi) < P$ does not carry useful information. Therefore, in the following, we only consider the upper bound.

The PCTL property $P_{\leq p}(\phi)$ is refused in an MDP, if there exists at least one scheduler d such that the probability mass of the paths $FinitePaths_d(s_0)$ satisfying ϕ under d exceeds p, where $FinitePaths_d(s_0)$ means a set of paths starting from state s_0 and satisfying ϕ under a scheduler d. We denote this set by $FinitePaths_d(s_0 \vDash \varphi)$. These finite paths are also called diagnostic path.

Definition 4 (Diagnostic paths). *Let M be an MDP model of a MCPS, and upper bounded formula $P_{\leq p}(\phi)$ be a requirement property under consideration. A counterexample of the property $P_{\leq p}(\phi)$ for MDP M is a set X of diagnostic paths in M such that $Pr^{max}(\phi) > p$ holds.*

Example 2. *Let us consider the example of MDP shown in Figure 3 and the property $P_{\leq 0.4}(aUb)$. This property is violated in this model since there exists a scheduler that induces a set of diagnostic paths that satisfy formula aUb and their probability mass greater than 0.4. We have the following diagnostic paths: $\sigma_1 = s_0 \xrightarrow{\alpha_0} s_2 \xrightarrow{\alpha_2} s_3$, $\sigma_2 = s_0 \xrightarrow{\alpha_0} s_4 \xrightarrow{\alpha_4} s_3$, $\sigma_3 = s_0 \xrightarrow{\alpha_0} s_2 \xrightarrow{\alpha_2} s_4 \xrightarrow{\alpha_4} s_3$ and their respective probabilities are 0.2, 0.15, 0.09. The set of three diagnostic paths is $C = \{\sigma_1\sigma_2\sigma_3\}$, and the probability is 0.44, which is greater than the given probability bound of 0.4. Thus, set C is a counterexample of the property of model violation.*

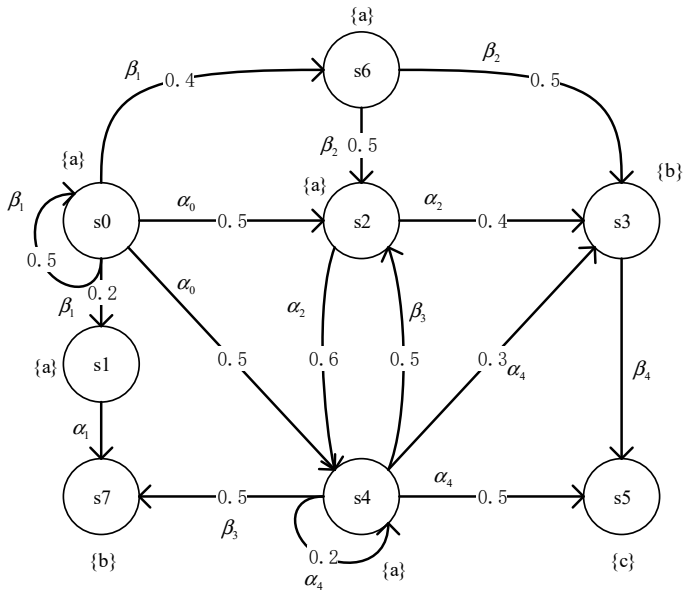

Figure 3. A Markov decision process.

We argue that a counterexample in probabilistic model checking should satisfy the following three conditions: (1) explain why the probabilistic system does not satisfy the property; (2) represent violation of a class of requirement properties; (3) identify errors in the system simply and specifically. A set X of diagnostic paths can show the violation of a property, but it contains too much redundancy information. Thus, we define the diagnostic subgraph to represent a counterexample for MDP.

An MDP can be regarded as a weighted directed graph where the vertices of graph represent states of MDP and the edges of graph represent transitions of MDP. Thus, the transition probability between states in the MDP can be converted into the weight between the vertices in the graph.

Definition 5 (Subgraph of MDP). *Let $G = (V, E, W)$ be a weighted digraph generated by an MDP $M = (S, s_{init}, A, P, L)$; a subgraph of M is $G' = \left(V', E', W'\right)$, where G′ is a part of G with $V' \subseteq V$, $E' \subseteq E$, $W' \subseteq W$ such that $\left\{(s, s') \subseteq E' \middle| \forall s, s' \subseteq V', \left(s, s'\right) \subseteq E\right\}$.*

The subgraph of MDP not only contains states and transitions of paths, but also all transitions connecting them in the original MDP.

Definition 6 (Diagnostic subgraph). *Let $M = (S, s_{init}, A, P, L)$ be an MDP; the counterexample for $s \not\models_D \mathcal{P}_{\leq p}(\phi)$ ($s \not\models_D \mathcal{P}_{<p}(\phi)$) is a diagnostic subgraph such that $Pr_s^D(\phi) > p$ ($s \not\models_D \mathcal{P}_{\geq p}(\phi)$) does not hold.*

Theorem 1. *For MDPs, a Counterexample represented by the diagnostic sub-graph, has less than or equal to the state space of the counterexample represented by diagnostic paths.*

Proof of Theorem 1. If a counterexample for MDP is one diagnostic path, it also can be represented with a diagnostic subgraph with this diagnostic path. If a counterexample for MDP is a measurable subset $X = \{X_1, X_2 \ldots X_n,\} \subseteq \{\pi \in FinitePaths_d(s) | \pi \vDash_D \phi\}$ such that $Pr_s^D(\phi) > p$ ($s \not\models_D \mathcal{P}_{\geq p}(\phi)$), each path in the measurable subset can be derived from a diagnostic loop path. We can get that the probability of a diagnostic loop path is greater than or equal to the diagnostic path. Thus, the state space of the counterexample represented by a diagnostic subgraph is less than or equal to the counterexample represented by the diagnostic paths. □

3.2. Genetic Algorithm with Heuristic (HGA)

The most critical issue of counterexample generation with HGA is how to code paths in the weighted directed graph for MDP as a chromosome. In order to represent all possible paths in the graph, we adopt a priority-based coding method. Then, we use heuristic crossover operators to generate better individuals than the parents, which can improve the search speed. We calculate the shortest path by the fitness function.

3.2.1. Coding the Path

According to [34,35], there are two main coding schemas of the Genetic algorithm for solving the shortest path, namely, direct and indirect coding. Direct coding is based on the identification number of the node. The disadvantage of this coding is that an invalid path is generated because the random sequence of node identification numbers may not correspond to a valid path. Therefore, direct coding is not a good choice. We choose priority-based indirect coding, which is used in [35,36].

Indirect coding has significant advantages in generating finite paths compared to direct coding schemes. The indirect coding scheme used in the Genetic algorithm is shown in Figure 4. The initial node identification number appears directly on the path; it is different from the path that uses the instruction information about forming the path node. The guidance information is the priority of each node in a weighted directed graph. In the initialization phase, the priorities are randomly distributed. The path starting from the initial node and terminating at the target node is generated by a sequential node attach procedure. In each step of the path construction, if there are several successor nodes to consider, then the node with the highest path priority is selected. Repeat the above process until reaching the target node.

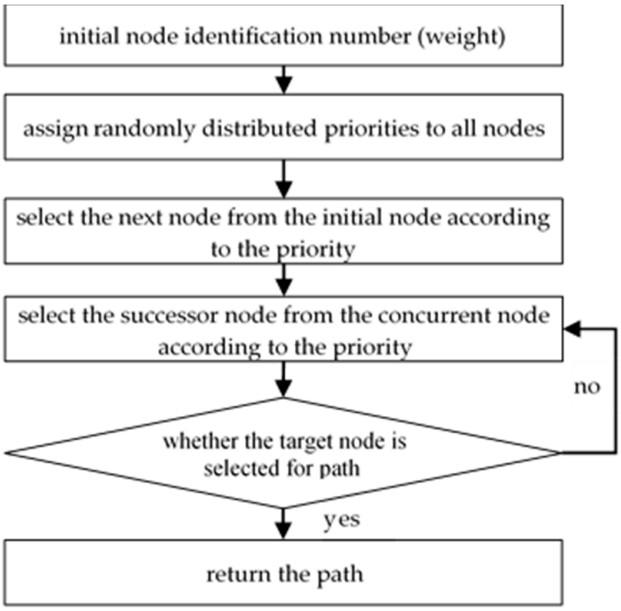

Figure 4. Indirect coding flow.

3.2.2. Fitness Function

The path quality is measured by the fitness function. In order to make the counterexample as small as possible, the goal of the fitness function is to minimize the search for the candidate path. The fitness function is described as follows:

$$f = \left(\sum_{e \in C} \omega(e)\right)^{-1} \quad (2)$$

where C is the path set in the graph, that is, each chromosome in the Genetic algorithm. e is an edge of path C, and $\omega(e)$ is the weight of the edge.

The construction process of diagnostic paths is selecting individuals with higher fitness from the current population and eliminating individuals with lower fitness. Assuming that the population size is N and fitness of the ith chromosome is p_i, then the probability of which the ith chromosome is selected is

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} \tag{3}$$

3.2.3. Heuristic Crossover Operator

The crossover operation replaces the reorganization of the partial structure of two parent individuals to generate a new individual. This paper uses a heuristic crossover approach. For the cross of individuals c_1 and c_2, we firstly generate a cross break-point K randomly, then copy the Kth to Nth gene string c_{11} in c_1 to the back of c_2 and delete the same gene in c_2 as in c_{11}; the same process is performed to c_2, the Kth to Nth gene string c_{22} of c_2 is copied to the back of c_1, and the same gene in c_{11} as in gene string c_{22} is deleted. This produces two legal intermediate individuals c_3, c_4 then selects two individuals with high fitness as crossover offspring from c_1, c_2, c_3, c_4 and puts the selected descendants into the mating pool for the next crossover.

3.2.4. Mutation Operator

The mutation operation is to change the gene values of certain gene spots on chromosome individuals in the population. Its purpose is to enable Genetic algorithms to have local random search capabilities and maintain group diversity. Suppose we perform mutation operations on individual A: [s a_1 a_2 a_3 a_4 a_5 t]; select a gene block Y: [a_2 a_3 a_4 a_5] on A first, and then randomly generate a path X: [a_2 R a_5] from a_2 to a_5 in graph, where R represents all genes on the path X, i.e., diagnostic paths. Then, individual A after the mutation operation is [s a_1 a_2 R a_5 t].

3.3. Generating Counterexample with HGA

The diagnostic paths are added in the set R, and we need to filter out the invalid paths in R. We call the set of all diagnostic paths from R which are valid under d the maximum of R. The goal is to provide a counterexample that contains only valid paths.

Definition 7 (Maximum of R). *Let d be a maximizing scheduler of R, and the set of all diagnostic paths which are valid under d is called a maximum of R.*

If R is a counterexample, then each maximum of R only contains the valid paths. Now, we need to compute a maximum X of R and to compute its probability $Pr^{max}(X)$. To this end, we define the compatible paths as follows.

Definition 8 (Compatible paths set). *A set of paths C is called compatible, if and only if there exists a scheduler d such that all paths in C are valid under d.*

We proposed this concept because each scheduler compatible subset X of R with maximal probability is a maximum of R. For diagnostic paths, the first prefix after the common prefix is decisive for scheduler compatibility. When a branch exists in a state, the diagnostic path is not compatible. We can always define a scheduler that allows a set of diagnostic paths to branch in an action. Therefore, the diagnostic path of the branch in the action is compatible. Checking the scheduler compatibility can be done as follows. R can be implemented as an AND/OR tree in order to check scheduler compatibility. Each diagnostic path is stored in R by mapping its states and actions to nodes. The OR nodes map to the state nodes, in which the scheduler makes a decision. The AND nodes map to the probability decisions after an action has been chosen by the scheduler. For a searched

diagnostic path $\sigma = s_0 \xrightarrow{\alpha_0} \ldots \xrightarrow{\alpha_{k-1}} s_k$, we interpret σ as an alternating sequence of OR and AND nodes, i.e., $S_\sigma = (s_0, \alpha_0 \ldots \alpha_{k-1}, s_k)$. If R is empty, we add all nodes and edges in diagnostic path σ directly to R. If R is not empty, determine whether the longest prefix of S_σ is already included in R, which starts from root s_0, i.e., whether the longest prefix is shared by any path in S_σ and R. When the longest prefix is included in the tree, insert diagnostic path σ into the tree. The remainder of diagnostic path σ becomes a new subtree, with the node ending with the longest prefix as root. Then, assign a probability to each node of this diagnostic path σ in a bottom-up manner. For AND root node in diagnostic path σ, its probability value is the sum of the values of its child nodes. The value marked on the leaf is the probability of the path from the root to it. The value marked in the internal AND node is the sum of values of its child nodes, and the value of the OR node is the maximum value of all child node values. Algorithm 2 illustrates the flows to a counterexample generation for MDP with HGA.

Algorithm 2. Counterexample generation for MDP with HGA.

Step1: Initialize R and X as an empty set, respectively, that is, without edges and nodes.
Step2: Generate a diagnostic path σ by HGA in Section 3.2
Step3: Replace diagnostic path σ with the sequence $S_\sigma = < s_0, \alpha_0 \ldots s_i \alpha_i >$ in the AND/OR tree
Step4: If R is empty, then directly add the nodes and edges in the σ and go to step 8.
Step5: Determine whether the longest prefix in S_σ already exists in R, starting from the initial node s_0.
Step6: Add the remainder of S_σ to the last node in the longest prefix.
Step7: Assign mark M_p to each node in R from the bottom up, where M_p is the probability of each node
Step8: If a node n in S_σ is an AND node, then its probability value M_p is the sum of the child-nodes' value.
Step9: If a node n in S_σ is an OR node, then its probability value M_p is the maximum of the child-nodes' value.
Step10: If $Pr_{max}(X) > p$ holds, return the counterexample.
Step11: Go to Step2

Theorem 2. *The Algorithm 2 will terminate after generating a counterexample.*

Proof of Theorem 2. Assume that Algorithm 2 does not terminate, only an infinite counterexample for $s \not\models_D P_{\leq p}(\phi)$ is generated, i.e., the counterexample consists of an infinite path. Let $C = \{\sigma_1, \sigma_2 \ldots\}$ be an infinite counterexample; we have $\sum_{i=1}^{\infty} Pr(\sigma_i) = \lim_{j \to \infty} \sum_{i=1}^{j} Pr(\sigma_i) > p$, where $\sum_{i=1}^{\infty} Pr(\sigma_i) = L$ and $\sum_{i=1}^{j} Pr(\sigma_i) = a_j$. According to the characteristics of the limit, this means that:

$$\forall \varepsilon > 0.\ \exists N \in \mathrm{IN}.\ \forall n \geq N.\ |a_n - \mathrm{L}| < \varepsilon \tag{4}$$

Let $0 < \varepsilon < L - p$. According to (3), for some $n \geq N$, $|a_n - L| < \varepsilon \Rightarrow |a_n - L| < L - p$; but the finite set $C' = \{\sigma_1, \sigma_2 \ldots \sigma_n\}$ is also a counterexample as $Pr(C') > p$. This is a contradiction. Therefore, the algorithm will terminate after generating a counterexample. □

3.4. An Example

We consider the MDP as shown in Figure 3 and the property $P_{\leq 0.5}(aUb)$. The AND/OR tree of it is shown in Figure 5. Searching the state space by HGA and generating the most probable diagnostic paths, then add it to the AND/OR tree, which is the implementation of R. s_2 is the AND node, the probability of which is the sum of its child-nodes' value, i.e., 0.35. s_4 is the OR node, the probability of which is the maximum of child-nodes' value, i.e., 0.25. The diagnostic paths we are interested in are marked with thick lines. The number identified next to each node is based on the probability calcu-

lated in Algorithm 2, the probability of the root node is 0.6, which means the maximum probability of identification is 0.6. Since 0.6 > 0.5, the maximum that was obtained is a counterexample, as shown in Figure 5.

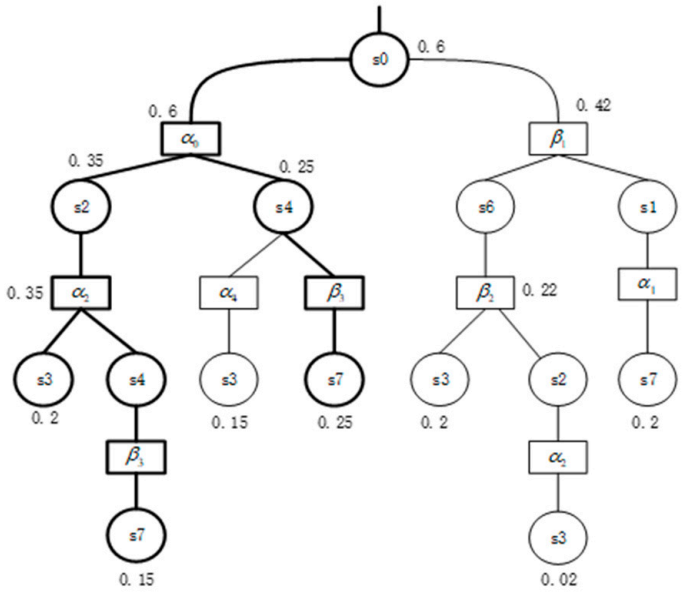

Figure 5. The AND/OR tree.

4. Experimentation

We develop a prototype tool CX-HGA for counterexample generation with HGA, based on PAT [7,16]. It is developed in the Java language with explicit-state data structures (sparse matrices, bit-sets, etc.). MDPs are described with PAT language, and the probability of the diagnostic subgraph is calculated by the PAT engine. All experiments are performed on a machine with Intel Pentium CPU 3.2 GHz speed and 1 GB memory. In our algorithms, population size is set to 100 and the evolution generation is 3000. Crossover probability is 0.90 and mutation probability is 0.10. By some PRISM Benchmark cases, we compare HGA with the XBF-based (XZ and XUZ) algorithm and Eppstein algorithm (Eppstein) for counterexample generation, which are implemented in DIPRO [37]. We also compare HGA and Genetic algorithm (GA) with directed coding. Note that DTMCs are a proper subset of MDPs. Any Markov chain is an MDP in which, for any state s, A (s) is just a singleton set. Thus, the experiments also include two DTMC cases for comparing with the existing works. We select 4 cases for demonstrating the effectiveness of the proposed method, which are either a typical MCPS or some important constituent protocols of MCPSs.

4.1. Dynamic Power Management

This case is the power dynamics management of the IBM disk drive [38,39]. It is a typical MCPS, and can be used for various areas, e.g., multi-robot collaboration. Its purpose is to minimize the power consumption while minimizing the impact on performance. It is modeled as a DTMC in PAT with approximately 140,000 states and transitions. We are interested in the event that is 100 or more requests getting lost within 400 milliseconds. The time consumption of transition in DTMC is 0.5 milliseconds, and the step number of property is 800. Therefore, the state formula corresponding to this event is $(true\ U^{\leq 800} lost = 100)$. The probability calculated by PAT is 0.014725297, if the time-bound property of $P_{<0.014725297}(true\ U^{\leq 800}\ lost = 100)$ is violated, which is a time-bound property of DTMC. We generate the counterexample for it by HGA, GA, XZ, XUZ, and Eppstein algorithms.

As shown in Figure 6, it is obvious that the Eppstein algorithm is almost parallel to the y-axis for a counterexample probability close to zero. HGA, XZ, and XUZ can successfully

provide counterexamples for all possible upper bounds with a reasonable computational effort. HGA finds the first increment of the counterexample after exploring a smaller fraction of the state space than XZ and XUZ. The Probability value of the first increment which is generated by HGA is greater than 0.011, and the reason for this is that we use a diagnostic subgraph to represent counterexample. After exploring about 2000 states and transitions, all algorithms can implement the second increment of the counterexample, and generate a counterexample after finding the second increment, because the total probability value exceeds the given probability. In the process of finding a counterexample, HGA explores fewer states and transitions than XZ and XUZ algorithms, except for the range between 0.006 and 0.011. HGA reaches the probability bound within about 3 s and consumes 250 KB memory. Meanwhile, GA, XZ, and XUZ algorithms reach the probability bound within about 3 s and 300 KB. GA generates a counterexample exploring more states and transitions, because it uses directed coding methods to generate many invalid paths and crossover operators used by GA cannot generate better offspring.

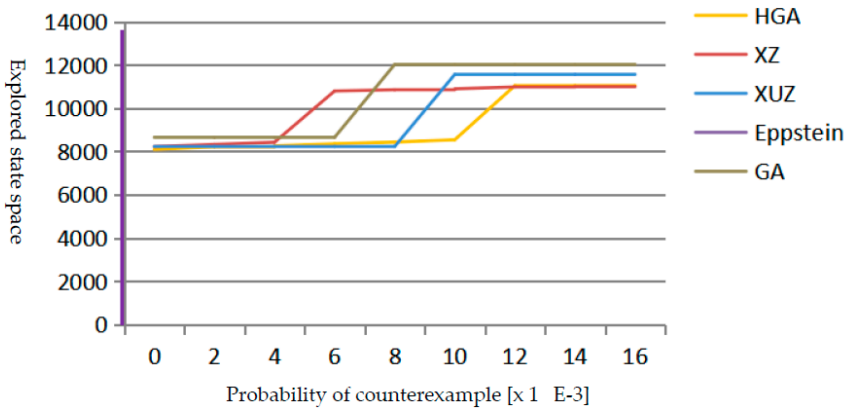

Figure 6. Counterexample probability of power dynamics management.

4.2. Synchronous Leader Election Protocol

This case is the synchronous leader election protocol [40,41]. It selects a unique leader node from N identical network nodes of the MCPSs. At each round, every node chooses a random number from $\{0, \ldots, K\}$ as its ID. A node with the maximum unique ID will be elected as leader; otherwise, a new round begins between these nodes. The protocol system is modeled in PAT as a DTMC with 12,400 states and 16,495 transitions. Here, we set N and K as 4 and 8, respectively. This process will not stop until the leader is elected. The probability of the formula $(F <= (L * (N + 1)$ "*elected*"$))$ is 0.95703125 by PAT, where "elected" indicates that a node has been selected as the leader, and L is set to be 1. We generate a counterexample of property $P_{\leq 0.95703125}(F <= (L * (N + 1)$ "*elected*"$))$ by HGA, GA, XZ, XUZ, and Eppstein algorithms.

As shown in Figure 7, we can see that the Eppstein and XUZ algorithms grow almost parallel to the y-axis, so the probability of the counterexample is zero. HGA and XZ can generate a counterexample for probability upper bound by reasonable computational effort. The HGA finds the first increment after exploring about half of the entire state space. However, the probability of the first increment of counterexample is less than 0.5. After exploring all the states and transitions, the HGA and XZ algorithms can find the second increment. HGA explores fewer states and transitions than XZ; and HGA finds the first increment with consumption of about 150 s and 360 KB memory. The GA algorithm consumes too much memory to find a counterexample, the reason being that direct coding produces more invalid paths.

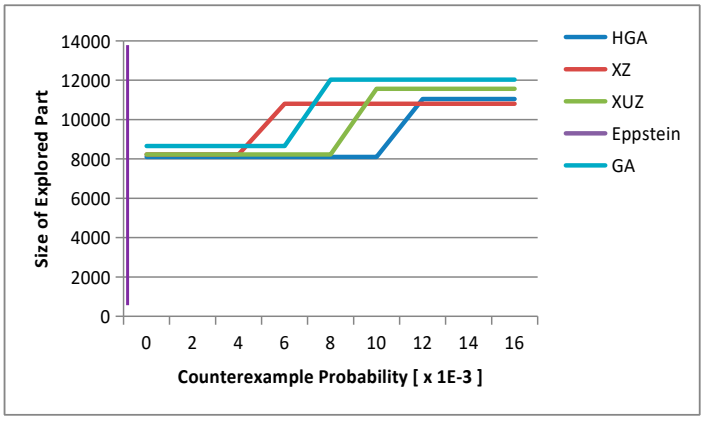

Figure 7. Counterexample probability of leader election.

4.3. Zeroconf Protocol

This case is a Zeroconf protocol [42]. It offers a distributed "plug-and-play" solution, in which the address configuration is managed by an individual MCPSs device when it is connected to the network. Each station has a single-message buffer and is cyclically attended by the server. The buffer could store the messages that it wants to send; in such cases, messages are not relevant after reconfiguring, and thus keeping these messages can slow down the network and make hosts reconfigure when they do not need to. We only consider version networks where the host does not do anything (No Reset) on these messages.

It is modeled as an MDP in PAT. The number of abstract hosts is denoted by N, the number of probes to send is denoted by K, and the probability of message loss is denoted by a loss. In this experiment, N and loss are set to 1000 and 0.1, respectively. K is set to 2 or 4. We are interested in the probability that a host picks an IP address already in use. In Table 1, "Time" and "Memory" represent the run time (in seconds) and memory (in KB). We compare Eppstein, XUZ, GA, and HGA on the two k-value variants of the model, as shown in Tables 2 and 3. When the searched state space is very big, Eppstein and XUZ algorithms cannot get a counterexample. They are recorded as "Failed", which denotes out of memory. The time consumption of HGA is less than Eppstein, XUZ, and GA, at the expense of very little memory, especially when the searched state and transitions are very large.

Table 1. Model information of MDP in PAT.

K	2	4
States	77,279	276,781
Transitions	180,442	640,721
Time	24.868	77.430
Memory	3584.0	11,878.4

Table 2. Run time of counterexample generation.

K	Probability	Eppstein	XUZ	GA	HGA
2	0.01	19.076	17.161	16.105	15.176
	0.04	169.06	165.090	156.223	105.103
	0.08	909.082	801.214	601.31	520.282
	0.1	2198.009	1996.071	1077.56	809.072
4	0.01	93.040	105.943	37.09	29.235
	0.04	786.047	518.177	509.43	435.034
	0.08	5963.085	4163.076	2607.38	1998.089
	0.1	Failed	Failed	5901.92	4076.92

Table 3. Memory consumption of counterexample generation.

K	Probability	Eppstein	XUZ	GA	HGA
2	0.01	797.9	702.8	699.4	690.2
	0.04	3798.4	3301.7	2863.5	2831.3
	0.08	7802.2	7032.9	5051.2	4960.3
	0.1	18,915.6	16,970.3	10,967.9	9818.9
4	0.01	7849.2	7649.3	7734.6	5782.34
	0.04	37,851.7	37,091.8	15,736.6	9825.4
	0.08	87,869.7	85,981.8	36,751.5	17,838.3
	0.1	Failed	Failed	99,302.2	48,804.2

4.4. Bounded Retransmission Protocol

BRP (bounded retransmission protocol) is a variant of the alternating bit protocol [43] for the compositional MCPSs. It sends a file in several chunks, but allows only a bounded number of retransmissions of each chunk. It is modeled as an MDP in PAT, where N represents the number of blocks, and MAX represents the maximum number of retransmits allowed for each block. In the experiment, MAX is set to 5 and the size of state space is scaled by N. It is modeled as an MDP in PAT; we are interested in the probability that the sender does not report a successful transmission. Model information is shown in Table 4, and experimental results are shown in Tables 5 and 6, the fields of which have the same meaning as Tables 1–3, respectively.

The non-deterministic analysis of MDP results in 137,313 transitions at N = 640 and 685,153 transitions at N = 3200. As can be seen from Table 4, the transitions in DTMC are 98% of that in MDP, which indicates that the MDP model is low uncertainty. Tables 5 and 6 list very low probabilities because these methods fail to provide counterexamples for large probability boundaries in real time. This is caused by the very low probability of a single diagnostic path being carried in this model, which means that many diagnostic paths must be found and processed to increase the probability of counterexamples. The run time and memory consumption of HGA performs best, especially for large probability bounds. HGA has a greater advantage in memory consumption because more efficient paths are found with a heuristic, and for large probability boundaries, the number of diagnostic paths found is very large, so there is a great advantage in storing them in an AND/OR tree.

Table 4. Model information of MDP in PAT.

N	640	3200
States	103,962	518,682
Transitions	139,888	697,968
Time	347.055	5023.233
Memory	4403.2	20,992.0

Table 5. Run time of counterexample generation.

N	Probability	Eppstein	XUZ	GA	HGA
640	0.01	132.721	120.835	111.443	109.234
	0.04	633.893	626.542	496.856	409.764
	0.08	5328.984	4996.132	1975.362	1153.762
	0.10	Failed	Failed	4206.716	2043.571
3200	0.005	1402.754	1603.239	1085.234	987.365
	0.008	5405.326	5564.271	2354.331	1801.259
	0.016	Failed	Failed	5979.845	2969.267
	0.02	Failed	Failed	Failed	8597.813

Table 6. Memory consumption of counterexample generation.

N	Probability	Eppstein	XUZ	GA	HGA
640	0.01	13,009.5	13,447.4	12,950.1	12,908.3
	0.04	23,768.1	24,647.8	23,840.4	21,987.2
	0.08	514,153.4	528,356.3	49,453.2	77,623.2
	0.10	Failed	Failed	Failed	893,615.8
3200	0.005	549,063.4	43,252.3	30,432.2	29,987.5
	0.008	793,109.2	797,602.6	618,023.1	50,568.2
	0.016	Failed	Failed	896,581.1	93,298.4
	0.02	Failed	Failed	Failed	897,536.8

4.5. Analysis

Compared with the existing works, we can see that HGA can generate a counterexample effectively for all 4 cases, even if the Eppstein algorithm, XUZ algorithm, and Genetic algorithm cannot get the counterexample for large state space of MCPSs, which are noted as "failed" in the corresponding tables. The bigger the state space of MCPSs, the more the effectiveness of our method is apparent, compared with the related works. This is rooted in the advantages of HGA: firstly, under the action of the genetic operator, HGA has a strong search ability, which can find the global optimal solution of the counterexample with a large probability; secondly, inherent in the parallelism of HGA, it can effectively deal with the large state space of the MCPSs model MDP. Compared with GA, HGA also performs better in counterexample generation for MDP. It is because that we adopt the indirect path coding in HGA to extend the search range of state space, and the heuristic crossover operator to generate more effective diagnostic paths.

5. Conclusions

We propose an HGA-based counterexample generation for PCTL probabilistic model checking MCPSs with nondeterministic and discrete-time stochastic behaviors, i.e., MDP.

As far as we know, it is the first counterexample generation method that employs the Genetic algorithm to search the diagnostic subgraph of MDP. We encode paths into chromosomes to generate more efficient paths through priority-based indirect coding. The heuristic crossover operator guarantees that a number of populations are excellent individuals, which helps to improve the average value of the population and speed up the convergence. We use the diagnostic subgraph to represent the counterexample, which is more compact. Experimental results show that the time and memory consumption are less than existing works. In the future, we will continue to optimize HGA to generate counterexamples for MCPSs against requirements in PCTL* which is a super set of PCTL. At the same time, we would like to extend HGA to generate counterexamples for MCPSs models with continuous-time or hybrid-time stochastic behaviors, such as CTMDPs (Continuous-time Markov Decision Processes), PHA (Probabilistic Hybrid Automata), against CSL (Continuous-time Stochastic Logic) stochastic communication logics [44,45].

Author Contributions: Conceptualization, Y.L. and Y.M.; methodology, Y.L. and Y.Y.; software, T.Z.; validation, Y.L., Y.M., Y.Y. and T.Z.; formal analysis, Y.L.; investigation, Y.L.; resources, T.Z.; data curation, T.Z.; writing—original draft preparation, T.Z.; writing—review and editing, Y.L.; visualization, T.Z.; supervision, Y.M.; project administration, Y.M.; funding acquisition, Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the MOE Humanities and Social Sciences Foundation of China under Grant Nos. 20YJCZH102, and Shanghai Science and Technology Commission under Grant Nos. 19595810700.

Acknowledgments: Thanks to Marta Kwiatkowska at the University of Oxford, UK, who has inspired us a lot through her books and papers, especially the direct discussion with her.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

Full Name	Abbreviation
Micro-scale Cyber-Physical Systems	MCPSs
Micro-scale Cyber-Physical System	MCPS
Discrete-time Markov Chains	DTMCs
Markov Decision Processes	MDPs
Probabilistic tic Timed Automata	PTA
Probabilistic Computation Tree Logic	PCTL
Liner Temporal logic	LTL
Probabilistic Timed Computation Tree Logic	PTCTL
Bounded Model Checking	BMC
Best-Frist	BF
Genetic Algorithm	GA
Heuristic Genetic Algorithm	HGA
Bounded Retransmission Protocol	BRP
Continuous-time Markov Decision Processes	CTMDPs
Probabilistic Hybrid Automata	PHA
Continuous-time Stochastic Logic	CSL

References

1. Lee, E.A.; Seshia, S.A. *Introduction to Embedded Systems, a Cyber-Physical Systems Approach*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2017.
2. Mulet Alberola, J.A.; Fassi, I. Cyber-Physical Systems for Micro-/Nano-assembly Operations: A Survey. *Curr. Robot. Rep.* **2021**, *2*, 33–41. [CrossRef]
3. Trunzer, E.; Vogel-Heuser, B.; Chen, J.-K.; Kohnle, M. Model-Driven Approach for Realization of Data Collection Architectures for Cyber-Physical Systems of Systems to Lower Manual Implementation Efforts. *Sensors* **2021**, *21*, 745. [CrossRef] [PubMed]
4. Wang, Y.; Zarei, M.; Bonakdarpoor, B.; Pajic, M. Probabilistic conformance for cyber-physical systems. In Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems (ICCPS'21), Association for Computing Machinery, New York, NY, USA, 19–21 May 2021; pp. 55–66. [CrossRef]

5. Clarke, E.M.; Henzinger, T.A.; Veith, H.; Bloem, R. (Eds.) *Handbook of Model Checking*; Springer: Cham, Switzerland, 2018.
6. Kwiatkowska, M.; Norman, G.; Parker, D. PRISM 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 585–591.
7. Liu, Y.; Sun, J.; Dong, J.S. PAT 3: An extensible architecture for building multi-domain model checkers. In Proceedings of the 2011 IEEE 22nd International Symposium on Software Reliability Engineering, Hiroshima, Japan, 29 November 2011; pp. 190–199.
8. Lacerda, B.; Faruq, F.; Parker, D.; Hawes, N. Probabilistic Planning with Formal Performance Guarantees for Mobile Service Robots. *Int. J. Robot. Res.* **2019**, *38*, 1098–1123. [CrossRef]
9. Pfeffer, A.; Wu, C.; Fry, G.; Lu, K.; Marotta, S.; Reposa, M.; Shi, Y.; Kumar, T.S.; Knoblock, C.A.; Parker, D.; et al. Software Adaptation for an Unmanned Undersea Vehicle. *IEEE Softw.* **2019**, *36*, 1–96. [CrossRef]
10. Henze, R.; Mu, C.; Puljiz, M.; Henze, R.; Mu, C.; Puljiz, M.; Kamaleson, N.; Huwald, J.; Haslegrave, J.; di Fenizio, P.S.; et al. Multi-scale Stochastic Organization-oriented Coarse-graining Exemplified on the Human Mitotic Checkpoint. *Sci. Rep.* **2019**, *9*, 3902. [CrossRef]
11. Chen, T.; Diciolla, M.; Kwiatkowska, M.; Mereacre, A. Quantitative Verification of Implantable Cardiac Pacemakers. In Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS'12), San Juan, PR, USA, 4–7 December 2012; pp. 263–272.
12. Bernardeschi, C.; Domenici, A.; Masci, P. A PVS-Simulink Integrated Environment for Model-Based Analysis of Cyber-Physical Systems. *IEEE Trans. Softw. Eng.* **2018**, *44*, 512–533. [CrossRef]
13. Češka, M.; Hensel, C.; Junges, S.; Katoen, J.P. Counterexample-guided inductive synthesis for probabilistic systems. *Form. Asp. Comput.* **2021**, *33*, 637–667. [CrossRef]
14. Lal, R.; Prabhakar, P. Counterexample guided abstraction refinement for polyhedral probabilistic hybrid systems. *ACM Trans. Embed. Comput.* **2019**, *18*, 1–23. [CrossRef]
15. Gao, M.; Wang, K.; He, L. Probabilistic model checking and scheduling implementation of an energy router system in energy Internet for green cities. *IEEE Trans. Ind. Inform.* **2018**, *14*, 1501–1510. [CrossRef]
16. Liu, Y.; Ma, L.; Zhao, J. Secure deep learning engineering: A road towards quality assurance of intelligent systems. In Proceedings of the 21st International Conference on Formal Engineering Methods, Shenzhen, China, 5–9 November 2019; Springer: Cham, Switzerland, 2019; pp. 3–15.
17. Han, T.; Katoen, J.P. Counterexamples in probabilistic model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 72–86.
18. Han, T.; Katoen, J.P.; Damman, B. Counterexample generation in probabilistic model checking. *IEEE Trans. Softw. Eng.* **2009**, *35*, 241–257.
19. Daws, C. Symbolic and parametric model checking of discrete-time Markov chains. In *International Colloquium on Theoretical Aspects of Computing*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 280–294.
20. Andrés, M.E.; D'Argenio, P.; van Rossum, P. Significant diagnostic counterexamples in probabilistic model checking. In *Haifa Verification Conference*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 129–148.
21. Jansen, N.; Abrah´am, E.; Katelaan, J.; Wimmer, R.; Katoen, J.P.; Becker, B. Hierarchical counterexamples for discrete-time Markov chains. In *International Symposium on Automated Technology for Verification and Analysis*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 443–452.
22. Hermanns, H.; Wachter, B.; Zhang, L. Probabilistic CEGAR. In Proceedings of the International Conference on Computer Aided Verification, Princeton, NJ, USA, 2 January 2018; pp. 162–175.
23. Chadha, R.; Viswanathan, M. A counterexample-guided abstraction-refinement framework for Markov decision processes. *ACM Trans. Comput. Log.* **2010**, *12*, 1–45. [CrossRef]
24. Češka, M.; Hensel, C.; Junges, S.; Katoen, J.P. Counterexample-driven synthesis for probabilistic program sketches. In *International Symposium on Formal Methods*; Springer: Cham, Switzerland, 2019; pp. 101–120.
25. Jansen, N.; Abraham, E.; Zajzon, B.; Wimmer, R.; Schuster, J.; Katoen, J.P.; Becker, B. Symbolic counterexample generation for discrete-time Markov chains. In *International Workshop on Formal Aspects of Component Software*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 134–151.
26. Jansen, N.; Wimmer, R.; Abrah´am, E.; Zajzon, B.; Katoen, J.P.; Becker, B.; Schuster, J. Symbolic counterexample generation for large discrete-time Markov chains. *Sci. Comput. Program.* **2014**, *91*, 90–114. [CrossRef]
27. Wimmer, R.; Jansen, N.; Ábrahám, E.; Becker, B.; Katoen, J.P. Minimal critical subsystems for discrete-time Markov models. In Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Tallinn, Estonia, 24 March 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 299–314.
28. Wimmer, R.; Becker, B.; Jansen, N.; Abrahám, E.; Katoen, J.P. Minimal Critical Subsystems as Counterexamples for omega-Regular DTMC Properties. In *MBMV*; Kovač: Hamburg, Germany, 2012; pp. 169–180.
29. Aljazzar, H.; Leue, S. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Trans. Softw. Eng.* **2010**, *36*, 37–60. [CrossRef]
30. Ma, Y.; Cao, Z.; Liu, Y. A PSO-Based CEGAR Framework for Stochastic Model Checking. *Int. J. Softw. Eng. Knowl. Eng.* **2019**, *29*, 1465–1495. [CrossRef]
31. Zheng, T.; Liu, Y. Genetic Algorithm for Generating Counterexample in Stochastic Model Checking. In Proceedings of the 2018 VII International Conference on Network, Communication and Computing, Taipei City, Taiwan, 14–16 December 2018; pp. 92–96.
32. Segala, R.; Lynch, N. Probabilistic simulations for probabilistic processes. *Nord. J. Comput.* **1995**, *2*, 250–273.

33. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* **2021**, *80*, 8091–8126. [CrossRef] [PubMed]
34. Beke, L.; Weiszer, M.; Chen, J. A Comparison of Genetic Representations for Multi-objective Shortest Path Problems on Multi-graphs. In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar), Seville, Spain, 15–17 April 2020; pp. 35–50.
35. Ghasemishabankareh, B.; Ozlen, M.; Li, X.; Deb, K. A genetic algorithm with local search for solving single-source single-sink nonlinear non-convex minimum cost flow problems. *Soft Comput.* **2020**, *24*, 1153–1169. [CrossRef]
36. Benini, L.; Bogliolo, A.; Paleologo, G.A.; De Micheli, G. Policy optimization for dynamic power management. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **1999**, *18*, 813–833. [CrossRef]
37. Aljazzar, H.; Leitner-Fischer, F.; Leue, S. Dipro-a tool for probabilistic counterexample generation. In *International SPIN Workshop on Model Checking of Software*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 183–187.
38. Arnaboldi, L.; Czekster, R.M.; Morisset, C.; Metere, R. Modelling Load-Changing Attacks in Cyber-Physical Systems. *Electron. Notes Theor. Comput. Sci.* **2020**, *353*, 39–60. [CrossRef]
39. Itai, A.; Rodeh, M. Symmetry breaking in distributed networks. *Inf. Comput.* **1990**, *88*, 60–87. [CrossRef]
40. Srinivasan, S.; Kandukoori, R. A synod based deterministic and indulgent leader election protocol for asynchronous large groups. *Int. J. Parallel Emergent Distrib. Syst.* **2021**, 1–28. [CrossRef]
41. Norman, G.; Parker, D.; Zou, X. Verification and control of partially observable probabilistic systems. *Real-Time Syst.* **2017**, *53*, 354–402. [CrossRef]
42. Kwiatkowska, M.; Norman, G.; Parker, D.; Sproston, J. Performance analysis of probabilistic timed automata using digital clocks. *Form. Methods Syst. Des.* **2006**, *29*, 33–78. [CrossRef]
43. Aarts, F.; Kuppens, H.; Tretmans, J.; Vaandrager, F.; Verwer, S. Learning and testing the bounded retransmission protocol. In Proceedings of the International Conference on Grammatical Inference, College Park, MD, USA, 12–15 September 2012; pp. 4–18.
44. Guo, X.; Kurushima, A.; Piunovskiy, A.; Zhang, Y. On gradual-impulse control of continuous-time Markov decision processes with exponential utility. *Adv. Appl. Probab.* **2021**, *53*, 301–334. [CrossRef]
45. Sproston, J. Verification and control for probabilistic hybrid automata with finite bisimulations. *J. Log. Algebraic Methods Program.* **2019**, *103*, 46–61. [CrossRef]