STAMINA 2.0: Improving Scalability of Infinite-State Stochastic Model Checking

Riley Roberts[1](✉), Thakur Neupane[1], Lukas Buecherl[2], Chris J. Myers[2], and Zhen Zhang[1]

[1] Utah State University, Logan, UT, USA
{riley.roberts,A02262317,zhen.zhang}@usu.edu
[2] University of Colorado Boulder, Boulder, CO, USA
{lukas.buecherl,chris.myers}@colorado.edu

Abstract. Stochastic model checking (SMC) is a formal verification technique for the analysis of systems with probabilistic behavior. Scalability has been a major limiting factor for SMC tools to analyze real-world systems with large or infinite state spaces. The infinite-state Continuous-time Markov Chain (CTMC) model checker, STAMINA, tackles this problem by selectively exploring only a portion of a model's state space, where a majority of the probability mass resides, to efficiently give an accurate probability bound to properties under verification. In this paper, we present two major improvements to STAMINA, namely, a method of calculating and distributing estimated state reachability probabilities that improves state space truncation efficiency and combination of the previous two CTMC analyses into one for generating the probability bound. Demonstration of the improvements on several benchmark examples, including hazard analysis of infinite-state combinational genetic circuits, yield significant savings in both run-time and state space size (and hence memory), compared to both the previous version of STAMINA and the infinite-state CTMC model checker INFAMY. The improved STAMINA demonstrates significant scalability to allow for the verification of complex real-world infinite-state systems.

Keywords: Stochastic Model Checking · Infinite-state systems · Markov chains · Synthetic biology

1 Introduction

Stochastic model checking (SMC) is a formal verification technique to analyze systems that possess probabilistic characteristics. In order to perform SMC, the state space of the system must be generated and stored. Many real-world systems can be modeled as *Continuous-Time Markov Chains* (CTMCs) with large or infinite state spaces. In particular, synthetic biological circuits have become a topic of interest recently, and can be modeled well by CTMCs. However, traditional SMC tools cannot directly analyze them due to the possibly infinite amount of memory required to store their state spaces. Many approaches, such as symbolic model checking [14], attempt to alleviate this issue by compactly representing states symbolically. However, these methods are inefficient in representing states with many probabilistic transitions [14], and still cannot handle infinite-state systems. Satisfiability Modulo Theories-based approaches to model checking large Discrete-Time Markov Chains have recently emerged [15]. However, they are not yet extended to analyzing infinite-state CTMCs. The STAR tool [10] primarily focuses on state reachability probability analysis, instead of checking a given probabilistic property, for infinite-state bio-chemical reaction networks by combining moment-based and state-based representations of probability distributions. Similarly, the SeQuaiA tool [3] analyzes state reachability probabilities for chemical reaction networks using accelerated abstraction techniques to preserve the most probable behavior of a CTMC model. The INFAMY model checker [6] was among the first tools to quantitatively verify infinite-state CTMCs. It truncates the model's state space on-the-fly after exploring it up to a certain finite depth. STAMINA [11] was created to model check transient *Continuous Stochastic Logic* (CSL) [1,8] properties on infinite-state CTMCs. It selectively explores a portion of the model's state space to efficiently give an accurate probability window in which the true probability of the property lies. Rather than exploring all state-transition paths up to the same fixed depth, STAMINA estimates state reachability probabilities during state expansion and uses them to determine paths to either further explore or terminate, effectively exploring the part of the state space where the probability mass lies. STAMINA was shown to outperform INFAMY in [11].

In this work, we present algorithmic improvements to STAMINA that result in significant gains in both state space size and runtime, with improved precision of the results. These algorithmic improvements include a new method of calculating and distributing predicted state reachability probabilities, as well as a method for analyzing the truncated state space using only one CTMC analysis rather than two. For highly complex models, the achieved reduction in both state-space size and runtime is observed to be as large as 90%. We present results from a case study of a synthetic biological circuit and from the benchmarks previously used for STAMINA [11].

2 Overview of the STAMINA Tool

The STAMINA tool takes in a CTMC model, specified in the PRISM modeling language, and a CSL property, and outputs an upper and lower bound for the probability of the property being satisfied for that model. It operates on the basis that it preserves, within an extremely large or infinite state space, a small subset of the states where a majority of the probability mass is located. STAMINA determines and explores this small subset and interfaces with the PRISM probabilistic model checker [9] to obtain a probability window that encloses the true probability of the property under verification.

As STAMINA expands a model's state space using breath-first search, it terminates state expansion if the estimated state reachability probability of the next state along a state exploration path drops below a pre-defined *state reachability probability threshold* κ. We denote the estimated state reachability probability (reachability probability, for short) for a state s as $\hat{\pi}(s)$ and assume $\hat{\pi}(s_0) = 1$ for initial state s_0. It is an estimation because STAMINA computes the probability of choosing a particular next state, but does not consider the time-dependent probability of remaining in each state. The reachability probability to reach from s to s' is defined as $p(s, s') = \frac{\mathbf{R}(s,s')}{\mathbf{E}(s)}$, where the *exit rate* $\mathbf{E}(s) = \sum_{s' \in post(s)} \mathbf{R}(s, s')$ is the sum of all outgoing transition rates $\mathbf{R}(s, s')$ for state s. The probability of leaving state s is $1 - e^{\mathbf{E}(s) \cdot t}$, a function of real time t. STAMINA estimates reachability probability *on-the-fly* during state expansion. It computes $\hat{\pi}(s')$ by summing up reachability probabilities from all *explored* predecessor states of s', denoted as $pre(s')$, as $\hat{\pi}(s') = \sum_{s \in pre(s')} (\hat{\pi}(s) \cdot p(s, s'))$, and $\hat{\pi}(s)$ is computed similarly. Whenever $\hat{\pi}(s') < \kappa$, it stops generating successor states of s'. Instead, it redirects outgoing transitions destined to these unexplored successor states to an artificially created absorbing state, $\hat{s}$, that is not part of the original model. We refer to states that have their transitions routed to $\hat{s}$ as *terminal states*.

STAMINA's algorithm computes $\mathbf{S}$, the set of all explored states, and $\mathbf{T} \subseteq \mathbf{S}$, the set of all terminal states. By utilizing PRISM's state space construction and model checking methods through subclassing, STAMINA performs reachability analysis and state-space truncation before invoking PRISM to perform the state-space construction, overriding certain methods so as to only generate the states in $\mathbf{S}$ and to route all outgoing transitions from states in $\mathbf{T}$ to $\hat{s}$. After state space construction, STAMINA again utilizes PRISM's API to compute a probability window that encloses the true probability for the CSL property under verification [12]. Figure 1 illustrates a simple overview of STAMINA's architecture.

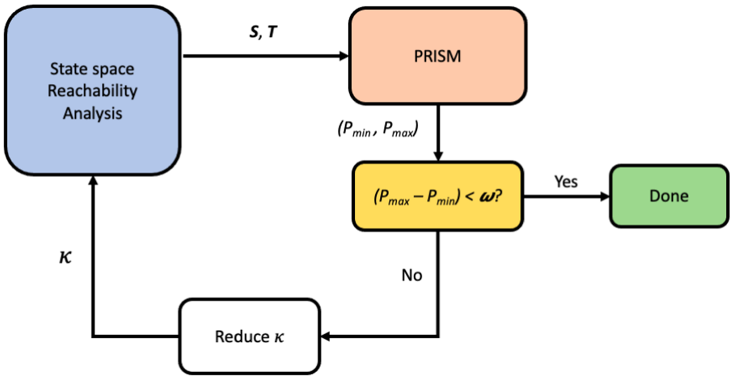

Fig. 1. High-level overview of STAMINA's architecture.

When checking a non-nested CSL property $\mathsf{P}_{=?}(\phi)$, which queries the probability that the path formula ϕ holds, the lower bound $\mathsf{P}_{\mathsf{min}}$ is the probability of ϕ being satisfied within $\mathbf{S}$ and the true probability is at least $\mathsf{P}_{\mathsf{min}}$. In the extreme case, all unexplored states abstracted by $\hat{s}$ satisfy ϕ, and therefore, the upper bound probability $\mathsf{P}_{\mathsf{max}}$ is the sum of $\mathsf{P}_{\mathsf{min}}$ and the probability (as determined by PRISM) of reaching $\hat{s}$ within the time bounds designated in ϕ. In the previous STAMINA implementation [11], it invokes PRISM twice to check two separate modified properties, namely, $\mathsf{P}_{=?}(\phi \wedge \neg\hat{s})$ and $\mathsf{P}_{=?}(\phi \vee \hat{s})$, to obtain $\mathsf{P}_{\mathsf{min}}$ and $\mathsf{P}_{\mathsf{max}}$, respectively. If $\mathsf{P}_{\mathsf{max}} - \mathsf{P}_{\mathsf{min}} > w$, where w is a user-defined tightness of the probability window, STAMINA would reduce κ by a reduction factor (default 1000) so that it can continue state space expansion; after which, it invokes PRISM to check the two properties again. It repeats this procedure until the probability window is tight enough, the machine runs of out memory, or an upper bound on iterations (default 10) is reached. During state expansion, STAMINA applies property-guided early path termination if the CSL property under verification is, or can be converted to, a non-nested and time-bounded "until" formula $\mathsf{P}_{=?}(\Phi \mathcal{U}^{[0,t]}\ \Psi)$. A path satisfies $\Phi \mathcal{U}^{[0,t]}\ \Psi$, if Φ holds in every state of the path from the initial state up until a state where Ψ evaluates to true within t time units. For time-abstract state exploration carried out by STAMINA, it terminates state expansion along a path when encountering a state s known to satisfy or dissatisfy $\Phi \mathcal{U}\ \Psi$, i.e., $s \models (\neg \Phi \vee \Psi)$. Instead, it makes s absorbing to contain probabilities flowing into it from its incoming paths. For detailed algorithms, readers are encouraged to read [11,12]. As the improvements to STAMINA are set forth, we will refer to the original algorithm as STAMINA 1.0, and the new algorithm presented in this paper as STAMINA 2.0, for clarity.

3 Improvements over STAMINA 1.0

Combined Analysis. When benchmarking STAMINA 1.0, we observed that a significant portion of the runtime was spent on performing CTMC analysis. One reason for this is that two separate CTMC analyses had to be carried out to calculate $\mathsf{P}_{\mathsf{min}}$ and $\mathsf{P}_{\mathsf{max}}$. If the property being checked is a non-nested CSL property of the form $\mathsf{P}_{=?}(\Phi \mathcal{U}^{[0,t]}\ \Psi)$, we have been able to improve this by combining the two analyses into a single analysis. The transient analysis performed by PRISM yields $P_t(s)$ to indicate the probability of being in state s at time t. Due to the property-guided early path termination described in Sect. 2, we obtain $\mathsf{P}_{\mathsf{min}} = \sum P_t(s_i)$ for all states s_i satisfying Ψ, excluding $\hat{s}$. The transient analysis also returns $P_t(\hat{s})$ for the absorbing state $\hat{s}$, $\mathsf{P}_{\mathsf{max}}$ is simply $\mathsf{P}_{\mathsf{min}} + P_t(\hat{s})$. This combined analysis results in significant time savings.

Re-exploration of States. We observe that re-visiting a previously explored state can cause its reachability probability to become trapped. In STAMINA 1.0, the tool does not re-explore an already explored state, to avoid never-ending state re-exploration within cycles, which represent one example of state re-visitation. However, this strategy causes the following issue. Suppose s_i is explored for the first time, its reachability probability $\hat{\pi}(s_i)$ is below κ, but at a later step, it discovers a new incoming transition to s_i, which brings $\hat{\pi}(s_i)$ to be equal to or above κ. Since s_i is not re-explored, it traps $\hat{\pi}(s_i)$, even if it increases again in future state exploration steps. Figure 2 illustrates a situation where this problem can manifest. Each state is labeled with its name and reachability probability. We consider the situation starting with the state shown in Fig. 2a: s_3 and s_4 are the next to be explored, $\hat{\pi}(s_3) = 0.1$, and $\hat{\pi}(s_4) = 0.9$. Then, s_5 is visited, resulting in $\hat{\pi}(s_5) = 0.1$ in Fig. 2b. Then, s_4 has a transition returning to s_3, which causes $\hat{\pi}(s_3)$ to increase to 1.0. However, since s_3 is not re-explored, the updated $\hat{\pi}(s_3)$ is never passed on to s_5. Instead, it has become trapped in s_3, as shown in Fig. 2c. If s_5 had some successor states s_i that were truncated due to $\hat{\pi}(s_i) < \kappa$, they will not be explored, even though their reachability probabilities would be sufficiently high to be explored if the reachability could properly pass through s_5.

The STAMINA 1.0 algorithm attempted to solve this problem in the following way: after finishing an iteration of state expansion, it walks through the entire explored state space to find all terminal states to be re-explored. Note that this step was not described in the original algorithm presented in [11,12]. Once it finishes exploration again, it will repeat the process of re-exploring all terminal states until the change in state space size between iterations becomes sufficiently small. The two main drawbacks of this strategy are (1) the non-trivial time complexity required to repeatedly search the state space for the terminal states; and (2) its inability to release trapped probabilities in non-terminal states as they are not re-explored. In Fig. 2, if s_5 was not a terminal state, but its successors were, the STAMINA 1.0 algorithm would not alleviate this issue.

Calculating Reachability. In order to set the stage for how STAMINA 2.0 solves this problem, we must first define a new method of calculating $\hat{\pi}(s)$, as the previous method does not allow for the re-exploration of a previously visited non-terminal state, because doing so would cause reachability probability that has already been passed on to the successors to be passed on again. To alleviate this issue, $\hat{\pi}(s)$ is now calculated in the following way: when a particular state s is explored, we first update the reachability probability for every successor state s' as follows: $\hat{\pi}(s') = \hat{\pi}(s') + \hat{\pi}(s) \cdot p(s, s')$. Then, $\hat{\pi}(s)$ is assigned to zero, indicating it has passed all of its reachability on to the successor states. By reducing $\hat{\pi}(s)$ to zero after exploration, if s is re-explored in the future and $\hat{\pi}(s) > 0$, we know that this non-zero reachability probability must have come from a transition that has returned to s since the last time it was explored. In this way, re-exploring s will only pass on the probability flowing into it since its most recent visit. As an additional benefit, this method is much less computationally expensive, as it can be performed as each state is visited, rather than needing to iterate over the predecessors.

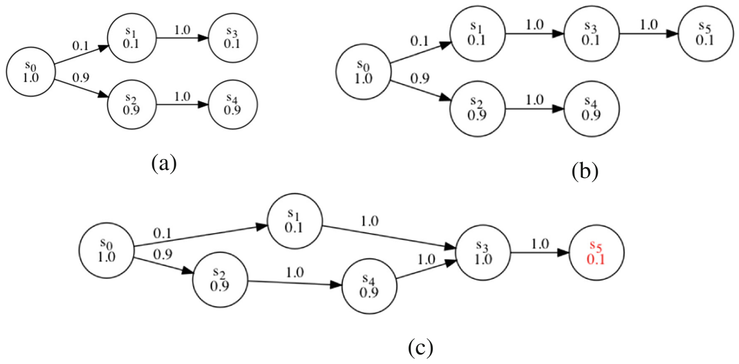

Fig. 2. Example of STAMINA 1.0 state exploration resulting in trapped reachability.

Algorithm Improvements. Using the improved method, the trapped reachability problem was solved by restructuring the STAMINA 1.0 algorithm in the following way: after state exploration finishes with a particular κ, we begin a re-exploration of the state space, starting from the initial state, in order to push the reachability probabilities of all states toward the outer boundaries of the explored state space. In STAMINA 1.0, κ starts from a small value of 1.0e-6 and is reduced infrequently, with the state space being verified between each reduction. STAMINA 2.0 changes this strategy, alleviating another challenge of STAMINA 1.0, which is the determination of a proper κ for a given PRISM model. In STAMINA 2.0, κ starts at its maximal value 1.0. When there are no more states to explore with the current κ, it is divided by a reduction factor, r_κ (default of 1.25), and the exploration repeats from s_0. Note that r_κ is much smaller than that used in STAMINA 1.0, which has a default value of 1000. Therefore, it causes κ to be reduced more frequently but by a smaller amount each time, allowing state reachability probabilities to properly pass through explored states, which in turn results in improved choice of state exploration paths with a larger portion of the total probability mass.

With significantly increased frequency of reducing κ, it is no longer reasonable to perform CTMC analysis prior to each reduction. Instead, to determine termination, we define $\hat{\Pi} = \sum_{s_i \in \mathbf{T}} \hat{\pi}(s_i)$ as an estimate of $(\mathsf{P}_{\mathsf{max}} - \mathsf{P}_{\mathsf{min}})$, where $\mathbf{T}$ is the terminal state set. Heuristically, we find that $(\mathsf{P}_{\mathsf{max}} - \mathsf{P}_{\mathsf{min}})$ tends to be smaller than w when $\hat{\Pi}$ becomes less than half of w. Thus, we specify a user-defined parameter *misprediction factor* m (default of 2 to match heuristic). Prior to each reduction of κ, we compute $\hat{\Pi}$ and terminate exploration when $\hat{\Pi} < \frac{w}{m}$. The state space is then passed to PRISM to compute the probability window. If it does not meet the desired tightness, i.e., $(\mathsf{P}_{\mathsf{max}} - \mathsf{P}_{\mathsf{min}}) > w$, m is increased in a manner proportional to the relative error between $(\mathsf{P}_{\mathsf{max}} - \mathsf{P}_{\mathsf{min}})$ and w. Specifically m is multiplied by 4 times $\frac{\mathsf{P}_{\mathsf{max}} - \mathsf{P}_{\mathsf{min}}}{w}$, except that if $\frac{\mathsf{P}_{\mathsf{max}} - \mathsf{P}_{\mathsf{min}}}{w} > 100$, 100 is used instead. The multiplier 4 is an additional heuristic. It is worth noting that STAMINA 2.0's algorithm contains several parameters that were determined heuristically, the majority of which can be tuned by the user if necessary, but are set by default to a value that tended to perform well across many different case studies. The reason for including these heuristics is that each particular model has a state space structure that will affect STAMINA's probability reachability estimations in different ways. To prevent the user from having to tune many different parameters for a particular model in order to get STAMINA to perform well, the parameter defaults were chosen heuristically to perform well across a large set of models. To the best of our knowledge gained through testing the tool across various use cases, there does not seem to be a strong theoretical basis for why certain values for these parameters would perform better on one model than another, so optimizing them for the general cases appears the best course of action. This new algorithm fully automates the choice of an accurate κ for STAMINA, in order to optimize runtime and state-space size, relieving the user of making such a choice. In addition, it allows a tighter probability window to be found with reduced states (and hence memory) using less time in almost all tested case studies. The improved accuracy of choosing portions of the state space to explore far outweighs the added computational complexity of re-exploring the state space.

Convergence of the STAMINA Algorithm. Algorithm 1 shows the full STAMINA 2.0 algorithm. The set $post(s)$ is defined as the set of successor states of state s and is generated from the input PRISM CTMC model. Additionally, note that for the notation of the algorithm, $\mathbf{S}$ is the set of all states explored up to the current execution point in the algorithm, while $explored$ is the set of states that have been explored using the current value of κ and is emptied after κ gets updated. In order to reason about the convergence of the algorithm, we first define $emb(C)$ as the embedded Discrete-Time Markov Chain (DTMC) of the PRISM CTMC model C under verification, as our estimates of reachability are calculated based on the transition probabilities of this embedded DTMC. Note that the entire embedded DTMC is never generated, we simply compute the transition probabilities during exploration of a particular state. We then define a path, pa, as a sequence of states that start from the initial state s_0, that can be traversed in $emb(C)$. Denote $P(s_i, s_j)$ as the probability of transitioning from s_i to s_j as encoded in the transition probability matrix of $emb(C)$. Let $pa(j)$ be the j-th state in pa and $len(pa)$ be the length of pa. We also define $paths(s_i)$ as the set of all paths whose *last* state is s_i, and $prob(pa)$ as the probability of the path pa, which is equal to $\prod_{i=0}^{len(pa)-2} P(s_i, s_{i+1})$. Finally, we let $pa_m <| pa_n$ indicate that pa_m is a *subpath* of pa_n, i.e. $\forall j = 0, 1, ..., len(pa_m) - 1,\ pa_m(j) = pa_n(j)$ and $len(pa_m) < len(pa_n)$.

Next, we reason about the estimated state reachability for state s_i, $\hat{\pi}(s_i)$, in terms of the definitions we have set forth. At *any* time point during the execution of Algorithm 1, $\hat{\pi}(s_i) = \sum_{pa \in \sigma(s_i)} prob(pa)$, where $\sigma(s_i) \subseteq paths(s_i)$ and $\forall pa_x, pa_y \in \sigma(s_i)$, where $x \neq y$, $pa_x \not<| pa_y$. We then denote $\pi(s_i)$ as the true probability (as opposed to the estimate probability $\hat{\pi}(s_i)$ calculated by STAMINA) of eventually reaching s_i for the first time in $emb(C)$. Note that a path can have possibly many revisits to s_i after its first visit, and it is the probability of the first visit to this state considered here. So define $X(s_i) = \{pa \mid pa \in paths(s_i) \ \wedge \ pa(j) \neq s_i,\ \forall 0 \leqslant j < len(pa) - 1\}$ as the set of paths that end with their first visit to state s_i. Then $\pi(s_i) = \sum_{pa \in X(s_i)} prob(pa)$. In other words, $\pi(s_i)$ aggregates the reachability probabilities for *all* paths at their the first visit to s_i. These definitions then allow us to derive the following statement, which we will use as the basis for convergence reasoning: $\hat{\pi}(s_i) \leqslant \pi(s_i)$ is an invariant of Algorithm 1 that holds true during all points of execution. This is due to the fact that *every* path $pa_j \in \sigma(s_i)$ meets one of two following conditions: Either $pa_j \in X(s_i)$ or pa_j is part of a set of paths $e \subseteq \sigma(s_i)$, where the set e satisfies the following condition: there must exist a path $pa_k \in X(s_i)$ such that $\forall pa_l \in e, pa_k <| pa_l$. In this latter case, we know that all paths for which pa_k is a subpath will have a combined probability of $prob(pa_k)$, and the sum of probabilities of all paths in e will be at most $prob(pa_k)$, i.e., $\sum_{pa_l \in e} prob(pa_l) \leqslant prob(pa_k)$. Intuitively, paths belonging to e are explored on-the-fly during STAMINA's state exploration, and it is possible that a path $pa \in paths(s_i)$, for which $pa_k <| pa$, is not added to e because pa gets truncated before s_i appears as its last state. In simpler terms, every probability contributing to the sum of the estimate either contributes directly to the sum of the true reachability, or is part of a set of probabilities contributing to the estimate that are in aggregate less than or equal to a corresponding probability contributing to the true reachability.

Now we can reason about the convergence of this algorithm with respect to the convergence of each of the three while loops contained within it. Note that although the algorithm is guaranteed to eventually converge under the constraints given here, it is not guaranteed to do so within the hardware limits, such as memory of the machine running it. Additionally, note that the conditions given for convergence are sufficient, but not necessary, as the algorithm may converge even when the conditions are not met, depending on the structure of the state space and the property being checked.

In order for the loop beginning on line 6 to terminate, all states that the algorithm encounters that have not yet been explored must have an estimated reachability of less than κ. This can be guaranteed under the following condition: There does not exist an infinitely long path, pa_i, in $emb(C)$ such that $\pi(s_j) \geq \kappa, \forall s_j \in pa_i$ and $\nexists k, l$ where

$k \neq l$ and $pa_i(k) = pa_i(l)$. The final condition regarding k, l comes from the fact that if a path encounters a state that has already been explored, this particular loop will terminate. Then, the loop beginning on line 4 terminates when the estimated reachability of all terminal states sums to less than $\frac{w}{m}$. This can be guaranteed to eventually occur under the following condition: There exists a $\kappa > 0$ such that for all states s in $emb(C)$ with $\pi(s) < \kappa$, $\sum \pi(s) < \frac{w}{m}$. Note that m will get larger until the conditions for the outermost loop of the algorithm are satisfied. The convergence of this outermost loop, beginning on line 2, is somewhat simpler to reason about. We first recognize that $\mathsf{P}_{\mathsf{max}} - \mathsf{P}_{\mathsf{min}}$ is equal to the probability of reaching a state (in the original CTMC) that the algorithm did not explore, within the time constraints specified by the CSL property. Thus, as the algorithm explores more states, $\mathsf{P}_{\mathsf{max}} - \mathsf{P}_{\mathsf{min}}$ must necessarily grow smaller. The inner two loops shown before operate with an increasingly small κ, which causes more states to be explored, and thus the termination of the outermost loop. In future work, we plan to investigate the incorporation of the temporal information available in the CTMC to expand the conditions for convergence to a larger number of models, as well as to further improve STAMINA's performance.

4 Results

We obtained all results on a machine with an AMD Ryzen Threadripper 12-Core 3.5 GHz Processor and 132 GB of RAM, running Ubuntu Linux (v18.04.3). 120 GB of RAM was allocated to the Java Virtual Machine used by STAMINA. Both STAMINA 1.0 and 2.0 utilized PRISM v4.5 and OpenJDK 11.0.10. All INFAMY results use the same parameters as in [11]. STAMINA 2.0 uses the default parameters for all examples. Both STAMINA versions attempted to obtain a user-desired probability window w of at least 1e$-$3, and INFAMY used a precision of 1e$-$3. This w was achieved by STAMINA 2.0 and INFAMY for all models; STAMINA 1.0 failed to achieve it in some cases, which are noted. Because each tool, other than those noted exceptions, obtained the specified w, the tools need only be compared in terms of the runtime and number of states (which translates to memory usage) required to reach the specified window. All benchmarks and case studies presented in this section, detailed tables of results comparison, and its source code can be found at: https://github.com/fluentverification/stamina.

Hazard Analysis in Genetic Circuits. Recent efforts in synthetic biology work towards applying principles from electric circuit design to genetic circuit design. One example is the *genetic design automation* (GDA) tool Cello [13] that was designed to accelerate and simplify the genetic design process. To verify the functionality of the tool, 60 combinational genetic were generated and tested in *Escherichia coli*. One of the generated circuits, circuit 0×8E, showed an unwanted switching behavior *in vivo*. Namely, in response to an input change, the output of the circuit was supposed to remain high, but it glitched low for a short time. In [5], it was demonstrated that this glitch was due to a *function hazard* (i.e., a property of the function being implemented). In [2] a stochastic analysis of the circuit was performed using both simulation and STAMINA 1.0 to evaluate the robustness of this design. The glitching behavior of this circuit is investigated under 12 possible transition patterns, where the transitions indicate a change in

Algorithm 1: Improved state re-exploration algorithm in STAMINA 2.0.

```
Input  : A PRISM CTMC model file, a CSL property, and $w$.
Output: $\mathsf{P_{min}}$ and $\mathsf{P_{max}}$.
1  $\mathsf{P_{min}} := 0.0$; $\mathsf{P_{max}} := 1.0$; $\hat{\pi}(s_0) := 1.0$; $\mathbf{S} := \{s_0\}$; $\mathbf{T} := \{s_0\}$;
2  while $\mathsf{P_{max}} - \mathsf{P_{min}} > w$ do
3      $\hat{\Pi} := 1.0$;
4      while $\hat{\Pi} > \frac{w}{m}$ do
5          $enqueue(queue, s_0)$; $explored := \emptyset$;
6          while $queue \neq \emptyset$ do
7              $s := dequeue(queue)$;
8              if $s \notin \mathbf{T} \ \vee \ \hat{\pi}(s) \geqslant \kappa$ then
9                  if $\hat{\pi}(s) = 0$ then
10                     forall the $s' \in post(s)$ do
11                         $enqueue(queue, s')$;
12                 else
13                     if $s \in \mathbf{T}$ then
14                         $\mathbf{T}.remove(s)$;
15                     forall the $s' \in post(s)$ do
16                         $\hat{\pi}(s') := \hat{\pi}(s') + \hat{\pi}(s) \cdot p(s, s')$;
17                         if $s' \notin explored$ then
18                             $explored := explored \cup \{s'\}$;
19                             $enequeue(queue, s')$;
20                             if $s' \notin \mathbf{S}$ then
21                                 $\mathbf{T} := \mathbf{T} \cup \{s'\}$; $\mathbf{S} := \mathbf{S} \cup \{s'\}$;
22                     $\hat{\pi}(s) := 0$;
23         $\hat{\Pi} := \sum_{s_i \in \mathbf{T}} \hat{\pi}(s_i)$;
24         $\kappa := \frac{\kappa}{r_\kappa}$;
25     Instruct PRISM to build the proper statespace based on the states in $\mathbf{S}$ and $\mathbf{T}$, and the
       original inputted PRISM model;
26     Compute $\mathsf{P_{min}}$ and $\mathsf{P_{max}}$ of the inputted CSL property, using PRISM;
27     if $\mathsf{P_{max}} - \mathsf{P_{min}} > w$ then
28         $m := m * 4 * min(100,\ (\frac{\mathsf{P_{max}} - \mathsf{P_{min}}}{w}))$
```

the amount of each of the circuit's three inducer molecules: *IPTG*, *aTc*, and *Ara*. Transitions are labeled as a set of 3 digits, each a 0 (low) or 1 (high) representing the amount of *IPTG*, *aTc*, and *Ara*, respectively. Since this genetic circuit is inherently noisy and has an infinite state space, it is an excellent candidate to be checked by STAMINA.

Originally, STAMINA 1.0 performed poorly when attempting to model check the genetic hazard circuit. Through a study of STAMINA's behavior when checking this circuit, we discovered the inefficiencies of the original algorithms as described in Sect. 3, and in particular, the issue showcased in Fig. 2, and optimized these algorithms in STAMINA 2.0. Figure 3 shows a comparison of the two versions of STAMINA's performance on the hazard genetic circuit model. STAMINA 1.0 was initially tested with its default value for κ, 1e$-$6, and then was reduced to 1e$-$20, but failed to compute an adequately small probability window for both values. The results presented here are for an initial κ of 1e$-$35. We can see that even after manually searching for a proper κ value, STAMINA 1.0 still cannot outperform 2.0. On those transitions that STAMINA 1.0 is able to compute bounds with the desired tightness, the improved algorithms implemented in STAMINA 2.0 achieved the same with approximately 90% less states (and by extension less memory) and 90% less time. STAMINA 1.0 was capped to a maximum of 10 iterations where κ is reduced before forced termination in order to avoid spending excessive time. In addition, STAMINA 1.0 failed to achieve the desired probability window for some transitions due to running out of memory. This does not affect the result comparison, because all runs that were either stopped after 10 iterations or ran out of memory had far surpassed STAMINA 2.0 in state-space size and runtime, despite not yet achieving the desired probability window size. In reality, if STAMINA 1.0 were allowed to run to completion, assuming no bound on runtime or memory, the improvements for both state space size and runtime would be greater than those reported for the models STAMINA 1.0 could not complete. Table 1 shows a comparison of the probability windows for examples that STAMINA 1.0 did not obtain an adequate probability window. From this table, we can observe the drastically tightened probability window STAMINA 2.0 was able to obtain despite it's lower runtime and state-space sizes. We were unable to obtain results for INFAMY on this 0$\times$8E genetic hazard circuit model, as its PRISM parser could not parse the model's transition rate formulas.

Table 1. Probability window comparison between STAMINA 2.0 and 1.0 on hazard circuit transitions for which the latter failed to produce a probability window that met the desired tightness.

Transition	STAMINA 2.0	STAMINA 1.0	Transition	STAMINA 2.0	STAMINA 1.0
010 to 111	$[0.0166, 0.0168]$	$[0.0060, 0.9218]$	100 to 111	$[0.0166, 0.0168]$	$[0.0125, 0.5405]$
011 to 101	$[0.9895, 0.9897]$	$[0.8608, 0.9990]$	000 to 011	$[0.8260, 0.8262]$	$[0.6661, 0.9669]$
010 to 101	$[0.9902, 0.9905]$	$[0.9477, 0.9998]$	101 to 011	$[0.9895, 0.9898]$	$[0.8498, 0.9981]$

Other Benchmarks. While the hazard circuit represents one of the more complex systems STAMINA may be used on, it can also perform well on simpler models. We tested STAMINA 2.0 on the same set of benchmarks used to evaluate STAMINA 1.0 in [11], in order to illustrate that STAMINA 2.0 was not simply optimized for the hazard circuit case. These benchmark examples come from both the PRISM benchmark suite [7] and the INFAMY tool's case studies at https://depend.cs.uni-saarland.de/tools/infamy/casestudies/. Many of these case studies are not infinite state models, but contain parameters that can be scaled to increase the state space size to an arbitrarily large size. It should be noted that STAMINA can analyze very large, but finite, state spaces as well as infinite state spaces. These particular case studies were chosen because they have been previously tested using either PRISM or INFAMY, and are accessible on these tools' respective websites for users to test other tools against STAMINA's results. A brief description of each of these benchmark models, the corresponding CSL properties being checked, and the meaning of the parameters can be found in [11]. It is worth mentioning that for the Robot models, the property being checked is a nested CSL formula, so the combined analysis improvement does not apply. All performance

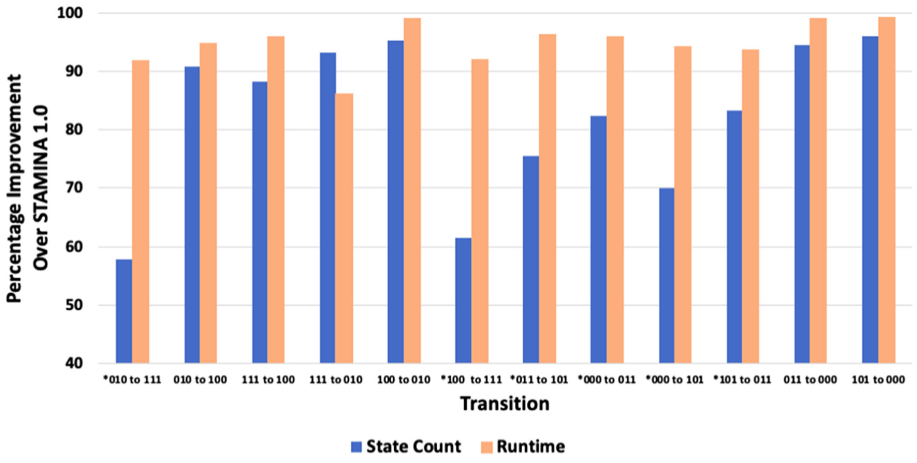

Fig. 3. STAMINA 2.0 improvement over 1.0 on the $0\times8E$ genetic hazard circuit. Columns labeled with a * indicate that STAMINA 1.0 did not achieve the desired probability window due to memory or iteration constraints.

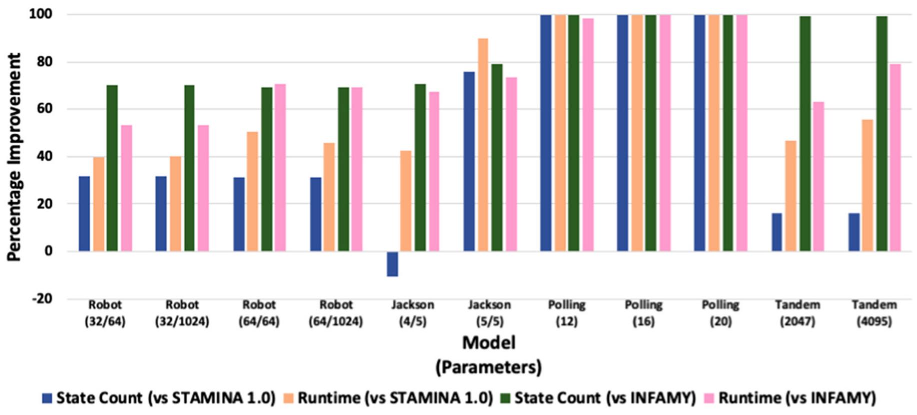

Fig. 4. STAMINA 2.0 improvement on the Benchmark models.

gains on this model come from the other discussed improvements. Figure 4 shows the performance improvements of STAMINA 2.0 on these benchmarks, relative to both STAMINA 1.0 and INFAMY. Because these models are simpler, there is not as much room for improvement, and the gains tend to be smaller than those for the hazard circuit. However, the average gains remain substantial. Of particular note, the Polling benchmark needed only 1 state with STAMINA 2.0, regardless of the parameters. This is due to the fact that the property under verification is satisfied in the initial state. STAMINA 2.0 is able to recognize this and stop state expansion while STAMINA 1.0 and INFAMY still expand the state space to sizes in the tens of thousands, and even millions, of states. Note that STAMINA 1.0 had the property-guided truncation implemented, but its attempted solution to the problem shown in Fig. 2 caused it to explore additional states anyway. Also of note, the Jackson case study with parameters (4/5) is the only example for which STAMINA 1.0 performs better than STAMINA 2.0 in terms of states generated. In order to understand this, first note that STAMINA 1.0 relied on a user-determined probability window to determine stoppage, rather than using heuristics based on the calculated estimates as STAMINA 2.0 does. In most cases, the heuristic finds a cutoff that is much closer to optimal than the user defined cutoff can; however, in rare cases, such as this particular Jackson example, the arbitrarily chosen cutoff works well with the model structure and stops closer to the optimum cutoff. However, the STAMINA 2.0 algorithm still performs notably better than STAMINA 1.0 in terms of runtime for this example.

5 Conclusion

The algorithmic improvements made to STAMINA 2.0 result in significant savings of both runtime and memory usage. In particular, for highly complex models the new version is able to achieve gains on the order of 90% for both runtime and state space size. Through these improvements, the tool is able to obtain results on models it previously failed on. The STAMINA 2.0 tool allows us to obtain guarantees about the probabilistic behavior of infinite-state systems that would otherwise be impossible. In the future, we plan to create a version of the tool that integrates with the STORM model checker [4]. We also plan to integrate an estimate of state resident time into the STAMINA algorithm, in order to further improve the choice of states to be explored.

Acknowledgement. The authors of this work are supported by the National Science Foundation under Grant Nos. 1856733, 1856740, 1939892 and 1856733, DARPA FA8750-17-C-0229, Dean's Graduate Assistantship at the University of Colorado Boulder, and the University of Colorado Palmer Chair funds. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time Markov chains. ACM Trans. Comput. Logic **1**(1), 162–170 (2000)
2. Buecherl, L., et al.: Genetic circuit hazard analysis using stamina. In: 12th International Workshop on Bio-design Automation, pp. 39–40 (2020)
3. Češka, M., Chau, C., Křetínský, J.: SeQuaiA: a scalable tool for semi-quantitative analysis of chemical reaction networks. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification, pp. 653–666. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_32
4. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification, pp. 592–600. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31

5. Fontanarrosa, P., Doosthosseini, H., Borujeni, A.E., Dorfan, Y., Voigt, C.A., Myers, C.: Genetic circuit dynamics: hazard and glitch analysis. ACS Synth. Biol. **9**(9), 2324–2338 (2020)
6. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: INFAMY: an infinite-state Markov model checker. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 641–647. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_49
7. Kwiatkowsa, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: Quantitative Evaluation of Systems, International Conference on(QEST), pp. 203–204, 09 2012. https://doi.org/10.1109/QEST.2012.14
8. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72522-0_6
9. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
10. Lapin, M., Mikeev, L., Wolf, V.: SHAVE: stochastic hybrid analysis of Markov population models. In: Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control, HSCC 2011, pp. 311–312. ACM, New York (2011)
11. Neupane, T., Myers, C.J., Madsen, C., Zheng, H., Zhang, Z.: STAMINA: stochastic approximate model-checker for infinite-state analysis. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification, pp. 540–549. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_31
12. Neupane, T., Zhang, Z., Madsen, C., Zheng, H., Myers, C.J.: Approximation techniques for stochastic analysis of biological systems. In: Liò, P., Zuliani, P. (eds.) Automated Reasoning for Systems Biology and Medicine. CB, vol. 30, pp. 327–348. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17297-8_12
13. Nielsen, A.A.K., et al.: Genetic circuit design automation. Science **352**(6281), aac7341 (2016). https://doi.org/10.1126/science.aac7341, http://science.sciencemag.org/content/352/6281/aac7341
14. Parker, D.: Implementation of Symbolic Model Checking for Probabilistic Systems. Ph.D. Thesis, University of Birmingham (2002)
15. Rabe, M.N., Wintersteiger, C.M., Kugler, H., Yordanov, B., Hamadi, Y.: Symbolic approximation of the bounded reachability probability in large Markov chains. In: Norman, G., Sanders, W. (eds.) QEST 2014. LNCS, vol. 8657, pp. 388–403. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10696-0_30