IMPaCT: Interval MDP Parallel Construction for Controller Synthesis of Large-Scale STochastic Systems

Ben Wooding(✉) and Abolfazl Lavaei

School of Computing, Newcastle University,
Newcastle upon Tyne, UK
{ben.wooding,abolfazl.lavaei}@newcastle.ac.uk

Abstract. This paper is concerned with developing an open-source software tool, called IMPaCT, for the parallelized verification and controller synthesis of large-scale stochastic systems using interval Markov chains (IMCs) and interval Markov decision processes (IMDPs), respectively. The tool serves to (i) construct IMCs/IMDPs as finite abstractions of underlying original systems, and (ii) leverage *interval iteration* algorithms for formal verification and controller synthesis over *infinite-horizon* properties, including safety, reachability, and reach-avoid, while offering *convergence guarantees*. IMPaCT is developed in C++ and designed using AdaptiveCpp, an independent open-source implementation of SYCL, for adaptive parallelism over CPUs and GPUs of all hardware vendors, including Intel and NVIDIA. IMPaCT stands as the first software tool for the *parallel construction* of IMCs/IMDPs, empowered with the capability to leverage high-performance computing platforms and cloud computing services. Specifically, parallelism offered by IMPaCT effectively addresses the challenges arising from the state-explosion problem inherent in discretization-based techniques applied to large-scale stochastic systems. We benchmark IMPaCT across several physical case studies, adopted from the ARCH tool competition for stochastic models, including a 2-dimensional robot, a 3-dimensional autonomous vehicle, a 5-dimensional room temperature system, and a 7-dimensional building automation system. To show the scalability of our tool, we also employ IMPaCT for the formal analysis of a 14-dimensional case study.

Keywords: Interval Markov chain · interval Markov decision process · automated controller synthesis · large-scale stochastic systems · parallel construction · cloud computing

1 Introduction

Large-scale stochastic systems serve as a crucial modeling framework for characterizing a wide array of real-world safety-critical systems, encompassing domains such as power grids, autonomous vehicles, communication networks, smart buildings, energy systems, and so on. The intended behavior of such complex systems

J. Hillston et al. (Eds.): QEST+FORMATS 2024, LNCS 14996, pp. 249–267, 2024.
https://doi.org/10.1007/978-3-031-68416-6_15

can be formally expressed using high-level logic specifications, *e.g. linear temporal logic (LTL)* expressions [8]. Automating the formal verification and controller synthesis for these complex systems that fulfill LTL specifications is an exceedingly formidable challenge (if not impossible), primarily due to the uncountable nature of states and actions in continuous spaces.

To tackle the computational complexity challenges that arise, one promising solution is to approximate original (*a.k.a.* concrete) systems with simpler models featuring finite state sets, commonly referred to as *finite abstractions* [10,39]. When the underlying models are stochastic, these simplified representations commonly adopt the structure of Markov decision processes (MDPs), where discrete states mirror sets of continuous states in the concrete model (similarly for inputs). In practical implementation, constructing such finite abstractions usually entails partitioning the state and input sets of concrete models according to predefined discretization parameters, as discussed in various works including [3,18,22,28,41,45].

Within the finite MDP scheme, one can (i) initially leverage it as an appropriate substitute for the original system, (ii) proceed to synthesize controllers for the abstract system, and (iii) ultimately refine the controller back over the concrete model, facilitated by an interface map. Given that the disparity between the output of the original system and that of its abstraction is accurately quantified, it becomes feasible to ensure that the concrete system fulfills the same specification as the abstract counterpart under some quantified accuracy level. However, this accuracy level is only acceptable for finite-horizon specifications since it converges to infinity as time approaches infinity (cf. (4)), *rendering MDPs impractical* for infinite-horizon properties in this setting.

As a promising alternative, *interval Markov decision processes* (IMDPs) [15, 37] have emerged in the literature as a potential solution for formal verification and controller synthesis of stochastic systems fulfilling *infinite-horizon specifications*. Specifically, IMDPs offer a comprehensive approach by incorporating both *upper and lower* bounds on the transition probabilities among finite abstraction cells. This is accomplished by solving a (multiplayer) game which allows to extend satisfaction guarantees to infinite time horizons. However, the construction of IMDPs is more complicated compared to traditional MDPs as it necessitates the computation of both lower and upper bounds for transition probabilities among partition cells.

Abstraction-based techniques, including those used for constructing either MDPs or IMDPs, encounter a significant challenge known as the *curse of dimensionality*. This phenomenon refers to the exponential growth in computational complexity as the number of state dimensions increases. To alleviate this, we offer scalable parallel algorithms and efficient data structures designed for constructing IMCs/IMDPs and automating the process of verification and controller synthesis over infinite time horizons. In particular, by dividing the computations into smaller concurrent operations, we effectively mitigate the overall complexity by a factor equivalent to the number of threads available. This approach not only enhances computational efficiency of IMC/IMDP construction, but also facilitates

the practical application of these techniques in real-world scenarios with high-dimensional spaces.

1.1 Original Contributions

The primary contributions and noteworthy aspects of our tool paper include:

(i) We propose the first tool that constructs IMCs/IMDPs as abstractions of large-scale discrete-time stochastic systems while providing *convergence guarantees*. Our tool leverages the constructed IMCs/IMDPs for formal verification and controller synthesis ensuring the fulfillment of desired *infinite-horizon* temporal logic specifications, encompassing *safety, reachability, and reach-while-avoid properties*.
(ii) IMPaCT is implemented in C++ and runs in parallel using AdaptiveCpp[1] based on SYCL[2]. AdaptiveCpp eliminates from the user the need to implement cross-platform flexibility manually, serving as a strong foundation for CPU and GPU implementations.
(iii) IMPaCT leverages *interval iteration* algorithms [17] to provide *convergence guarantees* to an optimal controller in scenarios with *infinite time horizons*.
(iv) IMPaCT accepts bounded disturbances and natively supports additive noises with different *practical distributions* including normal and *user-defined* distributions.
(v) We leverage IMPaCT across diverse real-world applications such as autonomous vehicles, room temperature systems, and building automation systems. This broadens the scope of formal method techniques to encompass safety-critical applications that require satisfaction within *infinite time horizons*. The outcomes demonstrate significant efficiency in computational time.

The source code for IMPaCT along with comprehensive instructions on how to build and operate it, including YouTube video guides, can be located at:

https://github.com/Kiguli/IMPaCT

Due to space constraints, comprehensive information on traditional serial algorithms, proposed parallel algorithms, case studies, etc. can be found in the extended version of the paper [44].

1.2 Related Literature

IMDPs have become the source of significant interest inside the formal methods community over the past few years. In particular, IMDPs have been used for model-based verification and control problems [12–14,19,43] as well as more recently being used for data-driven learning problems [20,29,33]. There exists a limited set of software tools available for the formal verification and controller

[1] https://github.com/AdaptiveCpp/AdaptiveCpp/.
[2] https://www.khronos.org/sycl/.

synthesis of stochastic systems over (in)finite horizons, encompassing various classes of stochastic models, using abstraction-based techniques.

FAUST^2 [38] constructs finite MDPs for continuous-space discrete-time stochastic processes and conducts formal analysis for safety and reachability specifications. Nonetheless, the original MATLAB implementation of FAUST^2 encounters scalability issues, particularly for large models, due to the curse of dimensionality. StocHy [11] offers formal verification and synthesis frameworks for discrete-time stochastic hybrid systems via finite MDPs. While a small segment of StocHy addresses IMDPs, its primary implementation relies on the *value iteration* algorithm [26], which *lacks convergence guarantees* when dealing with infinite-horizon specifications. This limitation has been widely pointed out in the relevant literature (see *e.g.,* [9,16,17]). For the assurance of convergence to an optimal controller, it is essential to utilize *interval iteration* algorithms. This is a key feature of our tool on top of offering parallelization, which both set IMPaCT apart from StocHy. In particular, IMPaCT offers a notably more comprehensive and versatile approach for addressing *infinite-horizon* specifications and stands as the first tool dedicated to IMC/IMDP construction with *convergence guarantees*.

A recent update to PRISM [25] supports *robust* verification of uncertain models including IMDPs. However, PRISM language describes the IMDP states and transitions symbolically for *discrete-space* systems, rather than continuous-space ones as considered by IMPaCT. Furthermore, our tool introduces parallel implementations of IMDP abstraction and synthesis algorithms, efficiently automating the IMDP construction process. SySCoRe [42] is a MATLAB toolbox that derives simulation relations and presents an alternative methodology for the controller synthesis of stochastic systems satisfying co-safe specifications over infinite horizons. A recent tool, IntervalMDP.jl [32], implemented in Julia, was developed concurrently to IMPaCT, providing controller synthesis on CPU/GPU platforms. However, while IMPaCT offers both parallel IMC/IMDP *construction* and controller synthesis capabilities, IntervalMDP.jl is primarily focused on synthesis. More importantly, IMPaCT leverages the essential *interval iteration* algorithm to offer convergence guarantees, while IntervalMDP.jl relies on the *value iteration* algorithm, which *lacks convergence guarantees* when dealing with infinite-horizon specifications [17]. IMPaCT also handles safety specifications in addition to reachability and reach-avoid properties. These features distinguish IMPaCT from IntervalMDP.jl, as well as a few other tools like StocHy [11], which provide support for IMCs/IMDPs with value iteration as the default choice and lack general parallelization capabilities. Another tool, AMYTISS [27], also employs parallel implementations but exclusively for constructing MDPs using PFaces [24], built upon OpenCL, without any support for IMDP models. In addition, our tool is developed using SYCL, which facilitates parallel processing and offers broader utility than PFaces. In particular, AdaptiveCpp (formerly OpenSYCL) [5,6], employed in IMPaCT, provides a sturdy foundation for flexible cross-platform parallel processing on CPUs and GPUs from the major hardware vendors such as Intel, NVIDIA, etc., which is not the case in AMYTISS.

1.3 Overview of IMPaCT

IMPaCT is an object-oriented software tool developed in C++, serving as an Application Programming Interface (API), which facilitates the construction and utilization of IMDP objects for addressing verification and synthesis challenges. The GitHub repository includes numerous examples of configuration files showcasing the tool's versatility across different contexts. The tool takes as input a description of the discrete-time stochastic control system (dt-SCS) Σ and the specification ψ for an infinite-time (or finite-time) horizon, chosen from the options of *safety, reachability, and reach-while-avoid* properties. It then generates an HDF5 [40] file as output, containing a lookup table $\mathcal{C}$ with a list of states $\hat{x}$ along with the corresponding minimal ($\mathbb{P}_{\psi_{\min}}$) and maximal ($\mathbb{P}_{\psi_{\max}}$) probabilities of satisfying the specification. In synthesis cases, the tool will synthesize the optimal control policy π for each state. IMPaCT automatically determines whether the user intends to solve a synthesis problem using an IMDP or a verification problem using an IMC, based on whether the input space is defined. Likewise, it automatically identifies whether the problem pertains to reachability or reach-while-avoid, depending on whether the avoid region is defined.

2 Discrete-Time Stochastic Control Systems

A formal description of discrete-time stochastic control systems, serving as the underlying dynamics of our tool, is presented in the following definition.

Definition 1 (dt-SCS). *A discrete-time stochastic control system (dt-SCS) is a quintuple*

$$\Sigma = (X, U, W, \varsigma, f), \tag{1}$$

where

- $X \subseteq \mathbb{R}^n$, $U \subseteq \mathbb{R}^m$, $W \subseteq \mathbb{R}^p$ *are Borel spaces as, respectively, the state, input, and disturbance sets;*
- ς *is a sequence of independent and identically distributed (i.i.d.) random variables from a sample space* Ω *to a measurable set* $\mathcal{V}_\varsigma$

$$\varsigma := \{\varsigma(k) \colon \Omega \to \mathcal{V}_\varsigma,\ k \in \mathbb{N}\};$$

- $f \colon X \times U \times W \times \mathcal{V}_\varsigma \to X$ *is a measurable function characterizing the state evolution of the system.*

For a given initial state $x(0) \in X$, an input sequence $u(\cdot) : \Omega \to U$, and a disturbance sequence $w(\cdot) : \Omega \to W$, the state evolution of Σ is characterized by

$$\Sigma \colon x(k+1) = f(x(k), u(k), w(k), \varsigma(k)), \quad k \in \mathbb{N}. \tag{2}$$

To facilitate a more straightforward presentation of our contribution, we illustrate our algorithms by incorporating stochasticity with normal distributions. Nevertheless, our tool is capable of handling problems involving *any arbitrary distributions* via a custom *user-defined* distribution.

A dt-SCS has been shown to be equivalent to a continuous-space Markov decision process [23] as the following definition, where a conditional stochastic kernel T captures the evolution of Σ and can be uniquely determined by the pair (ς, f) from (1).

Definition 2 (Continuous-Space MDPs). *A continuous-space Markov decision process (MDP) is a quadtuple*

$$\Sigma = (X, U, W, T), \tag{3}$$

where

- X, U, *and* W *are as in Definition 1;*
- $T\colon \mathscr{B}(X) \times X \times U \times W \to [0,1]$ *is a conditional stochastic kernel that assigns any* $x \in X$, $u \in U$, *and* $w \in W$, *a probability measure* $T(\cdot|x,u,w)$, *on the measurable space* $(X, \mathscr{B}(X))$ *so that for any set* $A \in \mathscr{B}(X)$,

$$\mathbb{P}\Big\{x(k+1) \in A \,\Big|\, x(k), u(k), w(k)\Big\} = \int_A T(dx(k+1) \,|\, x(k), u(k), w(k)).$$

To construct a traditional finite MDP (*i.e.*, an MDP with finite spaces), the continuous spaces X, U, W are constructed from a finite number of partitions $\mathbf{X}^i, \mathbf{U}^i, \mathbf{W}^i$, where $X = \cup_{i=0}^{n_u} \mathbf{X}^i, U = \cup_{i=0}^{n_x} \mathbf{U}^i, W = \cup_{i=0}^{n_w} \mathbf{W}^i$. The number of partitions n_x, n_u, n_w can be computed based on discretization parameters η_x, η_u, η_w, chosen by the user, which defines the size of regions $\mathbf{X}^i, \mathbf{U}^i, \mathbf{W}^i$, respectively. Each finite partition can be identified by some representative points $\hat{x}_i \in \mathbf{X}^i$, $\hat{u}_i \in \mathbf{U}^i$, $\hat{w}_i \in \mathbf{W}^i$; the collections of all representative points can now be considered as the finite sets $\hat{X}, \hat{U}, \hat{W}$ of the finite MDP. Using a map $\Xi : X \to 2^X$, one can assign any continuous state $x \in X$ to the partition $\mathbf{X}^i$, where $x \in \mathbf{X}^i$. Additionally, a map $\Pi_x : X \to \hat{X}$ assigns any continuous state $x \in X$ to the representative point $\hat{x} \in \hat{X}$ of the corresponding partition containing x. The map Π_x satisfies the inequality

$$\|\Pi_x(x) - x\|_2 \leq \eta_x, \qquad \forall x \in X,$$

with η_x being a state discretization parameter. Using these mappings, the transition function $\hat{f}$ and the transition probability matrix $\hat{T}$ within the finite MDP construction are defined as $\hat{f}(\hat{x}, \hat{u}, \hat{w}, \varsigma) = \Pi_x(f(\hat{x}, \hat{u}, \hat{w}, \varsigma))$ and $\hat{T}(\hat{x}'|\hat{x}, \hat{u}, \hat{w}) = T(\Xi(\hat{x}')|\hat{x}, \hat{u}, \hat{w})$, respectively [30, Alg. 1 & Thm. 2.2].

Of particular importance to this work, an accuracy level ρ regards the difference between probabilities of satisfaction of the desired specification ψ over the continuous-space system Σ and its finite MDP counterpart $\hat{\Sigma}$. This accuracy level ρ can be computed a-priori as the product of the Lipschitz constant $\mathcal{H}$ of the stochastic kernel, the Lebesgue measure $\mathscr{L}$ of the state space, the state discretization parameter η_x, and the finite horizon $\mathcal{K}$, as the following:

$$\big|\mathbb{P}(\Sigma \models \psi) - \mathbb{P}(\hat{\Sigma} \models \psi)\big| \leq \rho = \mathcal{K}\mathcal{H}\mathscr{L}\eta_x. \tag{4}$$

3 Interval Markov Decision Processes

The finite MDP presented in Sect. 2 is not suitable for infinite-horizon problems due to the inclusion of the time horizon $\mathcal{K}$ in (4). In particular, as $\mathcal{K} \to \infty$ then $\rho \to \infty$, implying that the abstraction will be of no use to providing satisfaction guarantees. To alleviate this, a continuous-space MDP Σ in (3) can be *finitely abstracted* by an interval Markov decision process. Specifically, IMDPs provide bounds over the transition probability of the stochastic kernel T, offering a reliable model for analyzing infinite-horizon specifications, as outlined in the subsequent definition.

Definition 3 (IMDPs). *An interval Markov decision process (IMDP) is defined as a quintuple*

$$\hat{\Sigma} = (\hat{X}, \hat{U}, \hat{W}, \hat{T}_{\min}, \hat{T}_{\max}),$$

where

- $\hat{X} = \{\hat{x}_0, \hat{x}_1, \dots \hat{x}_{n_x}\}$, $\hat{U} = \{\hat{u}_0, \hat{u}_1, \dots \hat{u}_{n_u}\}$, $\hat{W} = \{\hat{w}_0, \hat{w}_1, \dots \hat{w}_{n_w}\}$, *with* $\hat{x}_i$, $\hat{u}_i$, *and* $\hat{w}_i$ *being the representative points within* $\boldsymbol{X}^i$, $\boldsymbol{U}^i$, $\boldsymbol{W}^i$, *respectively;*
- $\hat{T}_{\min}$ *is a conditional stochastic kernel for the minimal transition probability, computed as*

$$\hat{T}_{\min}(\hat{x}' \,|\, \hat{x}, \hat{u}, \hat{w}) = \min_{x \in \Xi(\hat{x})} T(\Xi(\hat{x}') \,|\, x, \hat{u}, \hat{w}), \quad \forall \hat{x}, \hat{x}' \in \hat{X},\ \forall \hat{u} \in \hat{U},\ \forall \hat{w} \in \hat{W},$$

  with $x \in \Xi(\hat{x})$, *and where the map* $\Xi : X \to 2^X$ *assigns to any* $x \in X$, *the corresponding partition element it belongs to, i.e.,* $\Xi(x) = \boldsymbol{X}^i$ *if* $x \in \boldsymbol{X}^i$*;*
- $\hat{T}_{\max}$ *is a conditional stochastic kernel for the maximal transition probability, computed similarly as*

$$\hat{T}_{\max}(\hat{x}' \,|\, \hat{x}, \hat{u}, \hat{w}) = \max_{x \in \Xi(\hat{x})} T(\Xi(\hat{x}') \,|\, x, \hat{u}, \hat{w}),\ \forall \hat{x}, \hat{x}' \in \hat{X},\ \forall \hat{u} \in \hat{U},\ \forall \hat{w} \in \hat{W},$$

$$\text{where } \hat{T}_{\min} \leq \hat{T}_{\max}, \text{ and } \sum_{\hat{x}' \in \hat{X}} \hat{T}_{\min}(\hat{x}' \,|\, \hat{x}, \hat{u}, \hat{w}) \leq 1 \leq \sum_{\hat{x}' \in \hat{X}} \hat{T}_{\max}(\hat{x}' \,|\, \hat{x}, \hat{u}, \hat{w}).$$

Although IMDPs incur higher computational costs compared to traditional MDPs, leveraging lower and upper bound probabilities for state transitions facilitates the resolution of infinite-horizon control problems. It is worth mentioning that since $\hat{X}, \hat{U}, \hat{W}$ are all finite sets, $\hat{T}_{\min}$ and $\hat{T}_{\max}$ can be represented by static matrices of size $(n_x \times n_u \times n_w)$ by n_x. This enables the use of powerful iterative algorithms. In the code, an IMDP object should be constructed by defining the dimension of state, input and disturbance:

```
IMDP(const int x, const int u, const int w);
```

Listing 1.1. Creating IMDP object.

In a broader scope, matrix and vector manipulations are implemented using the C++ library Armadillo [35,36], and the sets $\hat{X}, \hat{U}, \hat{W}$ are defined using the object's function calls:

```
void setStateSpace(vec lb, vec ub, vec eta);
void setInputSpace(vec lb, vec ub, vec eta);
void setDisturbSpace(vec lb, vec ub, vec eta);
```

Listing 1.2. Construction of $\hat{X}, \hat{U}$, and $\hat{W}$

Additionally, users are required to configure the noise by selecting either normal distributions or a user-defined distribution, see the extended version [44, Sec. 4.1] for details.

3.1 Complexity Analysis for Serial Construction of IMDP

The computational complexity of IMDP construction comprises two main components. The first, often termed the *curse of dimensionality*, describes the exponential rise in computational time concerning system dimensions, expressed as $\mathcal{O}(2d)$, where $d = (n_{x_i}^n \times n_{u_j}^m \times n_{w_k}^p) \times n_{x_i}^n$, with n_{x_i}, n_{u_j}, and n_{w_k} being the dimension-wise counts of partitions within $\hat{X}$, $\hat{U}$, and $\hat{W}$, respectively, where $i = 1, \ldots, n$, $j = 1, \ldots, m$, and $k = 1, \ldots, p$. Given that the construction of IMDP abstraction involves both lower and upper bound transition matrices $\hat{T}_{\min}$ and $\hat{T}_{\max}$, respectively, the computational complexity d is doubled.

The second part pertains to the computational complexity of the nonlinear optimization required for the IMDP construction, represented as $\mathcal{O}(\kappa)$. The level of complexity κ hinges on the algorithm chosen by the user. We implement the nonlinear optimization algorithms via NLopt [21], where the default (`nlopt::LN_SBPLEX` [34]) is a variant of the derivative-free Nelder-Mead Simplex algorithm. Consequently, the overall complexity of IMDP construction amounts to $\mathcal{O}(2\kappa d)$.

3.2 Temporal Logic Specifications

IMPaCT inherently handles specifications including safety, reachability, and reach-avoid, as formally articulated via logical operators *always* $\Box$, *eventually* $\Diamond$, and *until* U, in the following definition.

Definition 4 (Specifications). *The specifications of interest ψ, handled by IMPaCT, are defined as*

- $\psi := \Box\mathcal{S}$ *- safety; the system should always remain within a safe region* $\mathcal{S} \subseteq X$.
- $\psi := \Diamond\mathcal{T}$ *- reachability; the system should eventually reach some target region* $\mathcal{T} \subseteq X$.
- $\psi := \mathcal{S}\ \mathsf{U}\ \mathcal{T}$ *- reach-while-avoid; the system should remain within the safe region* $\mathcal{S} = X \backslash (\mathcal{A} \cup \mathcal{T})$ *until it reaches the target region* $\mathcal{T} \subseteq X$*, with* $\mathcal{A} \subseteq X$ *being an avoid region.*

When constructing IMCs/IMDPs, we aim at labeling the states within the state space based on the specification, this is especially beneficial for the verification or controller synthesis. We now relabel states from the state space that

belong to the target set $\hat{x} \in \mathcal{T}$ as target states $\hat{r}$ and assume that all of these states are absorbing states. Similarly, we relabel any states from the state space that also belong to the avoid region $\hat{x} \in \mathcal{A}$ as avoid states $\hat{a}$, treating the avoid region as an absorbing area. We explicitly treat the remaining states $\hat{x}$ as part of the *safe* state space, denoted by $\hat{x} \in \mathcal{S}$. This relabelling is computed by calling the following commands, where `remove` should be `True` to relabel the states:

```
void setTargetSpace(const function<bool(const vec&)>&
    separate_condition, bool remove);
void setAvoidSpace(const function<bool(const vec&)>&
    separate_condition, bool remove);
void setTargetAvoidSpace(const function<bool(const vec&)>&
    target_condition,const function<bool(const vec&)>&
    avoid_condition, bool remove);
```

Listing 1.3. Construction of $\mathcal{S}, \mathcal{T}$ and $\mathcal{A}$.

Due to the absorbing properties, the avoid and target regions are commonly modeled as a single state to simplify the algorithms. Transition probabilities to these states can be summed together for these new states. Accordingly, we introduce static vectors that capture the minimum and maximum probabilities of transitions to target ($\hat{R}_{\min}$ and $\hat{R}_{\max}$) and avoid ($\hat{A}_{\min}$ and $\hat{A}_{\max}$) regions, with row entries calculated by

$$\hat{R}_{\min}(\hat{x},\hat{u},\hat{w}) = \sum_{\forall \hat{r} \in \mathcal{T}} \hat{T}_{\min}(\hat{r} \,|\, \hat{x},\hat{u},\hat{w}), \quad \hat{R}_{\max}(\hat{x},\hat{u},\hat{w}) = \sum_{\forall \hat{r} \in \mathcal{T}} \hat{T}_{\max}(\hat{r} \,|\, \hat{x},\hat{u},\hat{w}),$$

$$\hat{A}_{\min}(\hat{x},\hat{u},\hat{w}) = \sum_{\forall \hat{a} \in \mathcal{A}} \hat{T}_{\min}(\hat{a} \,|\, \hat{x},\hat{u},\hat{w}), \quad \hat{A}_{\max}(\hat{x},\hat{u},\hat{w}) = \sum_{\forall \hat{a} \in \mathcal{A}} \hat{T}_{\max}(\hat{a} \,|\, \hat{x},\hat{u},\hat{w}).$$

$\hat{T}_{\min}$ and $\hat{T}_{\max}$ are then reduced to consider only transitions between states in $\mathcal{S}$. Therefore, one can redefine the IMDP depending on the specification as $\hat{\Sigma} = (\hat{X}, \hat{U}, \hat{W}, \hat{T}_{\min}, \hat{T}_{\max}, \hat{R}_{\min}, \hat{R}_{\max}, \hat{A}_{\min}, \hat{A}_{\max})$. The construction of these vectors is described in detail in the serial algorithm in the extended version [44, Alg. 1].

4 Parallel Construction of IMDP

In this section, we propose the required commands for parallel construction of IMDPs as offered by IMPaCT. This IMDP construction is required for our controller synthesis approach, discussed in Sect. 5. The details of the proposed parallel algorithm can be found in the extended version [44, Alg. 2].

```
void minTransitionMatrix();
void maxTransitionMatrix();
void minTargetTransitionVector();
void maxTargetTransitionVector();
void minAvoidTransitionVector();
void maxAvoidTransitionVector();
```

Listing 1.4. Abstraction of transition matrices.

Remark 1. In IMPaCT, given that the state space is bounded, certain case studies may entail the possibility of transitions extending beyond the defined state space. Such transitions are considered as transitions to the avoid region, since leaving the state space should be avoided. This behavior is captured in the functions for transitions to avoid states, and therefore in such scenarios, even if no avoid region is present inside the state space, the functions `minAvoidTransitionVector()` and `maxAvoidTransitionVector()` should still be called.

4.1 Complexity Analysis for Parallel Construction of IMDP

Parallelization offered by IMPaCT enhances the performance of solving both abstraction and synthesis problems. Using $\mathcal{O}(\kappa)$ to represent the optimization complexity, the computational complexity of our proposed parallel algorithm is $\mathcal{O}(\frac{2\kappa d}{\text{THREADS}})$, where $d = (n_{x_i}^n \times n_{u_j}^m \times n_{w_k}^p) \times n_{x_i}^n$, and THREADS is the number of parallel threads running. It is worth highlighting that the chosen algorithm in IMPaCT can be readily swapped to any other nonlinear optimization algorithm from NLopt[3] using a dedicated function inside the tool:

```
void setAlgorithm(nlopt::algorithm alg);
```

Listing 1.5. Set NLopt algorithm.

4.2 Low-Cost Abstraction

The primary bottleneck in the IMDP abstraction arises from the necessity of computing two transition probability matrices, coupled with the additional computational load of min/max optimization. To enhance performance, by computing the matrix $\hat{T}_{\max}$ first, the computations required for $\hat{T}_{\min}$ can be reduced. This is due to the fact that for any matrix entry $\hat{T}_{\max}(i,j) = 0$, one has $\hat{T}_{\min}(i,j) = 0$. The sparser the matrix, the more efficient the abstraction computation; we refer to this as *low-cost abstraction*. The corresponding commands for this are:

```
void transitionMatrixBounds();
void targetTransitionMatrixBounds();
```

Listing 1.6. Low-cost abstractions.

5 Controller Synthesis with Interval Iteration

We now synthesize a controller, via the constructed IMDP, to enforce *infinite-horizon* properties over the dt-SCS. When dealing with infinite-horizon problems, the *interval iteration* algorithm is capable of providing convergence guarantees unlike the common *value iteration* [17]. In particular, the interval iteration algorithm converges to an under-approximation and over-approximation of the

[3] https://nlopt.readthedocs.io/en/latest/NLopt_Algorithms/.

satisfaction probability associated with a temporal logic specification. The trade-off for achieving such a guarantee is the doubled computational load compared to the value iteration algorithm.

In essence, the interval iteration algorithm iterates over two Bellman equations simultaneously; one assuming an initial probability vector of zeros, denoted by $V_0 = \mathbf{0}_n$, and the other an initial vector of ones, represented as $V_1 = \mathbf{1}_n$. When calculating V_0' and V_1' as the next iteration step, a dynamic program with decision variables $\hat{R}$, $\hat{A}$, and $\hat{T}$ needs to be solved [17]. These algorithm parameters are known as the *feasible distribution*. We provide two ways to solve this dynamic program, described in Sect. 6. To do so, the following optimization should be minimized with respect to disturbance $\hat{w}$ and maximized concerning the input $\hat{u}$:

$$\max_{\hat{u}\in\hat{U}} \min_{\hat{w}\in\hat{W}} \underset{\hat{R},\hat{A},\hat{T}}{\text{optimize}}\ \delta_1\hat{R}(\hat{x},\hat{u},\hat{w}) + \delta_2\hat{A}(\hat{x},\hat{u},\hat{w}) + \sum_{\forall\hat{x}'\in\hat{X}} \delta_3(\hat{x}')\hat{T}(\hat{x}'|\hat{x},\hat{u},\hat{w}), \quad (5)$$

with weight functions δ_1, δ_2, and $\delta_3(\cdot)$, and being subject to the following constraints [17]:

$$\hat{T}_{\min}(\hat{x}'|\hat{x},\hat{u},\hat{w}) \le \hat{T}(\hat{x}'|\hat{x},\hat{u},\hat{w}) \le \hat{T}_{\max}(\hat{x}'|\hat{x},\hat{u},\hat{w}),$$
$$\hat{R}_{\min}(\hat{x},\hat{u},\hat{w}) \le \hat{R}(\hat{x},\hat{u},\hat{w}) \le \hat{R}_{\max}(\hat{x},\hat{u},\hat{w}),\ \hat{A}_{\min}(\hat{x},\hat{u},\hat{w}) \le \hat{A}(\hat{x},\hat{u},\hat{w}) \le \hat{A}_{\max}(\hat{x},\hat{u},\hat{w}),$$
$$\hat{R}(\hat{x},\hat{u},\hat{w}) + \hat{A}(\hat{x},\hat{u},\hat{w}) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\hat{x}'|\hat{x},\hat{u},\hat{w}) = 1.$$

For further details, $\hat{R}$ mimics s^+ and $\hat{A}$ mimics s^- in [17]. The corresponding weights are updated at each iteration step, where δ_1 and δ_2 depend on the specification and δ_3 is derived from either V_0 or V_1. For reachability (and reach-avoid) specifications, we set $\delta_1 = 1$ and $\delta_2 = 0$. For safety, in order to ensure convergence of the algorithm (described in detail in the extended version of this paper [44, Sec. 5.1]), we frame the problem as the complement of reaching the avoid region. Therefore, we set $\delta_1 = 0$ and $\delta_1 = 1$, and subsequently derive the complement of the converged solution from V_0 and V_1. The following equations that describe the interval iteration process. When the dynamic program is solved and the optimal feasible solutions $\hat{T}$, $\hat{R}$, and $\hat{A}$ are found, the interval iteration algorithm solves the two equations

$$\begin{cases} V_0' = \delta_1\hat{R} + \delta_2\hat{A} + \hat{T}V_0, \\ V_1' = \delta_1\hat{R} + \delta_2\hat{A} + \hat{T}V_1, \end{cases}$$

to find the new probabilities of satisfying the specification for the state-input-disturbance triples. Prior to the subsequent iteration, V_0 and V_1 are, respectively, updated to V_0' and V_1'. The interval iteration algorithm terminates when the two vectors converge, $\|V_1 - V_0\|_\infty \le \varepsilon$, where ε is set by default to a small enough threshold.

The underlying synthesis technique is equivalent to a three-and-a-half player game with a max-min optimization problem, where the input and disturbance are

two players. In particular, treating the disturbance as an adversary to the system entails minimizing the probability concerning the disturbance while maximizing it concerning the control inputs. The third player ("optimize" in (5)) represents the range across the targeted partition addressed by the optimization program, acting adversarially (minimization) in computing $\mathbb{P}_{\psi_{\min}}$ and angelically (maximization) when computing $\mathbb{P}_{\psi_{\max}}$. The serial algorithms for synthesis of safety and reach-while-avoid specifications are detailed in the extended version [44, Alg. 3 & Alg. 4].

Remark 2. In certain scenarios, such as the presence of absorbing states beyond those already outlined in the target and/or avoid regions, the two bounds of the interval iteration algorithm may be mathematically impossible to converge, see extended version [44] for details. IMPaCT offers detailed guidance via output to the terminal for resolving such scenarios, including the example `ex_load_safe`.

6 Parallel Controller Synthesis with Interval Iteration

We propose parallel algorithms for safety (extended version [44, Alg. 5]) and reach-while-avoid (extended version [44, Alg. 6]) to enhance the computational efficiency of controller synthesis procedure acquiring the resulting controller $\mathcal{C} = (\mathcal{S}, \pi, \mathbb{P}_{\psi_{\min}}, \mathbb{P}_{\psi_{\max}})$, where π is the optimal control policy. In particular, the synthesis process includes solving several dynamic programs and two interval iteration algorithms in a parallel fashion. To solve the dynamic programs, we either use the GNU linear programming kit (GLPK) [31] to solve a linear program, or use the *sorting approach* for *synthesis* outlined in [37, Lemma 7], which provides a significant increase in synthesis efficiency over CPU as well as removing the reliance on external function libraries to enable synthesis over GPUs. It should be noted that for smaller models, GPU synthesis may not yield substantial performance enhancements compared to CPU synthesis (cf. Table 2).

A Boolean value, `IMDP_lower`, is set as `true` for a pessimistic policy, or `false` for an optimistic policy. For finite-horizon controllers, an additional parameter is needed for the time horizon. The finite-horizon specifications are supported within IMDP synthesis by replacing the **while** loop in the algorithms of the extended version (i.e., [44, Alg. 3 & Alg. 4]) with a **for** loop over a finite number of time intervals. We still compute each bound $\mathbb{P}_{\psi_{\max}}$ and $\mathbb{P}_{\psi_{\min}}$ separately, and find the optimal policy π. It is worth mentioning that for finite-horizon specifications, interval iteration can be reduced to value iteration, since convergence is not required. The synthesis commands using GLPK are:

```
1 void setStoppingCondition(double eps);
2 void infiniteHorizonReachController(bool IMDP_lower);
3 void infiniteHorizonSafeController(bool IMDP_lower);
4 void finiteHorizonReachController(bool IMDP_lower, size_t
    timeHorizon);
5 void finiteHorizonSafeController(bool IMDP_lower, size_t
    timeHorizon);
```

Listing 1.7. Synthesis Algorithms.

With the sorted approach, synthesis can be readily computed by appending `Sorted` to the desired synthesis function. As demonstrated in Table 1 and Table 2, the sorted approach yields significant performance improvements for both CPU and GPU computations across various example classes.

```
void infiniteHorizonReachControllerSorted(bool IMDP_lower);
void infiniteHorizonSafeControllerSorted(bool IMDP_lower);
void finiteHorizonReachControllerSorted(bool IMDP_lower,
    size_t timeHorizon);
void finiteHorizonSafeControllerSorted(bool IMDP_lower,
    size_t timeHorizon);
```

Listing 1.8. Sorted Synthesis Algorithms.

Remark 3. In the current implementation, the sorting process itself is not parallelized and utilizes `std::sort`, while subsequent computations are parallelized. Implementing a parallel sorting algorithm manually could be regarded as a potential future extension to IMPaCT. It is worth noting that for large systems, the sorting method still offers substantial benefits over GLPK.

Remark 4. The functions for sorted synthesis and for saving and loading files, as described in the following section, can be found in the source file `GPU_synthesis.cpp`, which is included by the main source file `IMDP.cpp`. This separation is intentional to prevent compilation errors relating to external function libraries when employing GPU. For GPU synthesis, the configuration file should load all required matrices and vectors before invoking the sorted approach. The Makefile also needs to be modified to ensure that `--acpp-targets` flag specifies the GPU architecture, *e.g.* `cuda:sm_80`. Furthermore, in the Makefile, `IMDP.cpp` should be substituted with `GPU_synthesis.cpp`. For an illustration, see the example `ex_GPU`.

7 Loading and Saving Files

We use HDF5 [40] as the data format for saving and loading files into IMPaCT. In particular, HDF5 is a common widely supported format for large, heterogeneous, and complex data sets. This data structure is self-descriptive, eliminating the need for extra metadata to interpret the files. Additionally, it supports "data slicing", enabling extraction of specific segments from a dataset without the necessity of analyzing the entire set. The primary advantage of HDF5 lies in its open format, which ensures native support across numerous programming languages and tools, such as MATLAB, Python, and R. This should facilitate simpler sharing of synthesized controllers without requiring end-users to install additional programs. Loading the abstraction also facilitates compatibility with *data-driven* approaches for controller synthesis, such as those described in [7]. Details of the commands to load and save files can be found in the extended version [44, Sec. 7].

8 Benchmarking and Case Studies

We illustrate IMPaCT's applications by employing multiple well-known benchmark systems, adopted from the ARCH tool competition for stochastic models [1,2,4], encompassing safety, reachability, and reach-avoidance specifications across infinite horizons. Among these, we leverage several complex physical case studies, encompassing a 2D robot, a 3D autonomous vehicle, 3D and 5D room temperature control systems, as well as 4D and 7D building automation systems. We also consider a 14D case study used for the purposes of showing the scalability of the tool. In Table 1, we showcase the underlying details of these case studies, including memory usage and computation time, all executed over an *infinite time horizon* using the GLPK library utilizing the *sorting method*. It is noteworthy that for the *smallest case studies* the GLPK library outperforms the sorting method, likely due to the overhead associated with sorting the states before solving Eq. (5). In Table 2, we contrast the GLPK library with the *sorting method* from [37, Lemma 7] over both CPU and GPU, demonstrating notable efficiency of the latter in terms of computation time for larger models. Future extensions aimed at enhancing the sorting algorithms, such as implementing *parallel sorting techniques*, could potentially enhance the efficiency of the tool. The description of all case studies together with simulation results can be found in the extended version [44, Sec. 8].

8.1 Comparisons

As previously discussed in the introduction, StocHy utilizes the value iteration algorithm for IMDP construction, which *lacks convergence guarantees* for infinite-horizon specifications. In contrast, IMPaCT employs the *interval iteration* algorithm to guarantee convergence to an optimal controller. This crucial feature, in addition to offering parallelization, distinguishes IMPaCT from StocHy, which defaults to value iteration without general parallelization capabilities. Beyond this guarantee, we aim to execute the 14D case study using StocHy to showcase the efficiency of our tool. While IMPaCT completed the IMDP construction within an hour (53 minutes), as detailed in Table 1, StocHy failed to complete the task within a 24-hour time frame. We have refrained from including comparisons with PRISM or IntervalMDP.jl, given that our focus on demonstrating the applicability of our tool through case studies involves large *continuous-space* transition systems, which are challenging to describe using the PRISM language or IntervalMDP.jl that focuses on *discrete-space models*.

Table 1. Execution times and memory requirements of IMPaCT applied to a set of benchmarks. Computation times are in seconds and memory usages in MB, unless otherwise specified. Specifications: S for safety, R for reachability, and R − A for reach-while-avoid. BAS stands for Building Automation System. We signify the synthesis times using the GLPK Library with "[a]" and the synthesis times based on the sorting method from [37, Lemma 7] with "[b]". Note that "*" indicates possible absorbing states: in this case a finite horizon run is conducted with a convergent number of steps, where algorithm times are aggregated (see Remark 2). In 4D BAS benchmark, using the GLPK library, controller synthesis for a finite horizon of 10 steps is computed in 4.66 seconds, while it takes over 6 hours for the infinite horizon to converge. The total time for the abstraction and synthesis is computed as the sum of the times for $\hat{T}_{\min}, \hat{T}_{\max}, \hat{R}_{\min}, \hat{R}_{\max}, \hat{A}_{\min}, \hat{A}_{\max}$ and one of the two different method times for $\mathcal{C}$.

Case Study	Spec					$\hat{T}_{\min}$	$\hat{T}_{\max}$		$\hat{R}_{\min}$	$\hat{R}_{\max}$		$\hat{A}_{\min}$	$\hat{A}_{\max}$		$\mathcal{C}$		
		$\lvert\hat{X}\rvert$	$\lvert\hat{U}\rvert$	$\lvert\hat{W}\rvert$	$\lvert\hat{X}\times\hat{U}\times\hat{W}\rvert$	time	time	mem	time	time	mem	time	time	mem	time[a]	time[b]	mem
2D Robot	R	441	121	0	53,361	0.60	1.58	174.8	0.09	0.294	4.5	0.016	0.015	4.5	7.34	8.53	0.02
2D Robot	R	441	121	11	586,971	5.9	15.6	1.9 GB	0.251	1.10	6.58	0.04	0.04	6.58	65.7	330	0.02
2D Robot	R − A	1,681	441	0	741,321	20.7	59.8	8.8 GB	0.78	2.51	61.4	1.86	1.88	61.4	1,549	1,047	0.08
2D Robot	R − A	1,681	441	11	8,154,531	227	675	97.2 GB	7.46	22.0	274	21.2	22.6	274	5.66 h	8.46 h	0.08
3D Vehicle	R − A	7,938	30	0	238,140	89.1	114	7.42 GB	44.6	46.6	2.91 GB	4.53	4.97	264	3.69 h	286	0.61
3D Vehicle	R − A	15,435	99	0	1,528,065	1,004	1,340	92.6 GB	534	551	36.3 GB	51.8	46.5	3.3 GB	>24 h	5,933	1.22
3D RoomTemp	S	9,261	36	0	333,396	1.51	78.5	24.7 GB	–	–	–	0.015	0.014	2.7	136*	154*	0.52
4D BAS	S	1,225	4	0	4,900	0.89	1.33	48.02	–	–	–	0.004	0.007	0.04	6.37 h*	3,038*	0.07
5D RoomTemp	S	7,776	36	0	279,936	1.2	167.6	17.4 GB	–	–	–	0.21	0.24	2.24	97.88*	111.5*	0.56
7D BAS	S	107,163	0	0	107,163	1.47 h	2.03 h	91.9 GB	–	–	–	0.501	0.139	0.86	>24 h	>24 h	7.7
14D Case	S	16,384	0	0	16,384	699	987	2.15 GB	–	–	–	0.041	0.201	0.13	623	67.7	2.1

Table 2. Execution times for controller synthesis, comparing the solving of the linear program in (5) using GNU Linear Programming Kit (GLPK) against the sorting method from [37, Lemma 7]. The comparison is conducted on both a CPU (Intel i9-12900) and a GPU (NVIDIA RTX A4000), with times reported in seconds. The symbol "$*$" denotes that a finite horizon of 10 seconds was utilized for the case study synthesis.

Case Study	Spec					GLPK Library	Sorted LP Method	
		$\lvert\hat{X}\rvert$	$\lvert\hat{U}\rvert$	$\lvert\hat{W}\rvert$	$\lvert\hat{X}\times\hat{U}\times\hat{W}\rvert$	CPU i9	CPU i9	GPU RTX
4D BAS*	S	1,225	4	0	4,900	23.2	0.38	0.53
3D RoomTemp*	S	216	36	0	7,776	0.34	0.22	0.29
2D Robot	R	441	121	0	53,361	13.07	2.26	5.2
5D RoomTemp*	S	7,776	9	0	69,984	160	22.2	76.2
3D Vehicle	R − A	7,938	30	0	238,140	>3.0 h	241	490
2D Robot	R − A	1,681	441	0	741,321	1691	577	216

9 Conclusion

In this work, we developed the advanced software tool IMPaCT, which is the first tool to exclusively construct IMC/IMDP abstraction and perform verification and controller synthesis over *infinite-horizon* properties while providing *convergence guarantees*. Developed in C++ using AdaptiveCpp, an independent open-source implementation of SYCL, IMPaCT capitalizes on adaptive parallelism across diverse CPUs/GPUs of the major hardware vendors, including Intel and NVIDIA. We benchmarked IMPaCT across various physical case studies borrowed from the ARCH tool competition, with its scalability further highlighted through a 14D case study.

Data Availibility Statement. The artifact accompanying this paper is available at https://zenodo.org/doi/10.5281/zenodo.11085098.

References

1. Abate, A., et al.: ARCH-COMP23 category report: stochastic models. In: International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH), EPiC Series in Computing, pp. 126–150. EasyChair (2023)
2. Abate, A., et al.: ARCH-COMP22 category report: stochastic models. EPiC Ser. Comput. **90**, 113–141 (2022)
3. Abate, A., Prandini, M., Lygeros, J., Sastry, S.: Probabilistic reachability and safety for controlled discrete-time stochastic hybrid systems. Automatica **44**(11), 2724–2734 (2008)
4. Abate, A., et al.: ARCH-COMP20 Category Report: Stochastic Models (2020)
5. Alpay, A., Heuveline, V.: SYCL beyond OpenCL: the architecture, current state and future direction of HipSYCL. In: Proceedings of the International Workshop on OpenCL (2020)

6. Alpay, A., Heuveline, V.: One pass to bind them: the first single-pass SYCL compiler with unified code representation across backends. In: Proceedings of the 2023 International Workshop on OpenCL (2023)
7. Badings, T.S., Abate, A., Jansen, N., Parker, D., Poonawala, H.A., Stoelinga, M.: Sampling-based robust control of autonomous systems with non-gaussian noise. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, pp. 9669–9678 (2022)
8. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
9. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: interval iteration for Markov decision processes. In: International Conference on Computer Aided Verification, pp. 160–180 (2017)
10. Belta, C., Yordanov, B., Gol, E.A.: Formal methods for discrete-time dynamical systems, vol. 89 (2017)
11. Cauchi, N., Abate, A.: StocHy: automated verification and synthesis of stochastic processes. In: 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2019)
12. Chen, T., Han, T., Kwiatkowska, M.: On the complexity of model checking interval-valued discrete time Markov chains. Inf. Process. Lett. **113**(7), 210–216 (2013)
13. Delimpaltadakis, G., Lahijanian, M., Mazo, M., Laurenti, L.: Interval Markov decision processes with continuous action-spaces. In: Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control, pp. 1–10 (2023)
14. Dutreix, M., Coogan, S.: Specification-guided verification and abstraction refinement of mixed monotone stochastic systems. IEEE Trans. Autom. Control **66**(7), 2975–2990 (2020)
15. Givan, R., Leach, S., Dean, T.: Bounded-parameter Markov decision processes. Artif. Intell. **122**(1), 71–109 (2000)
16. Haddad, S., Monmege, B.: Reachability in MDPs: refining convergence of value iteration. In: 8th International Workshop on Reachability Problems, pp. 125–137 (2014)
17. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. **735**, 111–131 (2018)
18. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst. Des. **43**(2), 191–232 (2013)
19. Hashemi, V., Hermanns, H., Song, L., Subramani, K., Turrini, A., Wojciechowski, P.: Compositional bisimulation minimization for interval Markov decision processes. In: 10th International Conference on Language and Automata Theory and Applications, pp. 114–126 (2016)
20. Jiang, J., Zhao, Y., Coogan, S.: Safe learning for uncertainty-aware planning via interval MDP abstraction. IEEE Control Syst. Lett. **6**, 2641–2646 (2022)
21. Johnson, S.G.: The NLopt nonlinear-optimization package (2007). https://github.com/stevengj/nlopt
22. Julius, A.A., Pappas, G.J.: Approximations of stochastic hybrid systems. IEEE Trans. Autom. Control **54**(6), 1193–1203 (2009)
23. Kallenberg, O.: Foundations of Modern Probability, vol. 3. Springer, Cham (2021)
24. Khaled, M., Zamani, M.: PFaces: an acceleration ecosystem for symbolic control. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, pp. 252–257 (2019)

25. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: 23rd International Conference on Computer Aided Verification, pp. 585–591 (2011)
26. Lahijanian, M., Andersson, S.B., Belta, C.: Formal verification and synthesis for discrete-time stochastic systems. IEEE Trans. Autom. Control **60**(8), 2031–2045 (2015)
27. Lavaei, A., Khaled, M., Soudjani, S., Zamani, M.: AMYTISS: parallelized automated controller synthesis for large-scale stochastic systems. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 461–474. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53291-8_24
28. Lavaei, A., Soudjani, S., Abate, A., Zamani, M.: Automated verification and synthesis of stochastic hybrid systems: a survey. Automatica **146** (2022)
29. Lavaei, A., Soudjani, S., Frazzoli, E., Zamani, M.: Constructing MDP abstractions using data with formal guarantees. IEEE Control Syst. Lett. **7**, 460–465 (2022)
30. Lavaei, A., Soudjani, S., Zamani, M.: From dissipativity theory to compositional construction of finite Markov decision processes. In: Proceedings of the 21st ACM International Conference on Hybrid Systems: Computation and Control, pp. 21–30 (2018)
31. Makhorin, A.: GLPK (GNU linear programming kit) (2008). http://www.gnu.org/s/glpk/glpk.html
32. Mathiesen, F.B., Lahijanian, M., Laurenti, L.: IntervalMDP.jl: Accelerated Value Iteration for Interval Markov Decision Processes (2024)
33. Rickard, L., Abate, A., Margellos, K.: Learning robust policies for uncertain parametric Markov decision processes. arXiv: 2312.06344 (2023)
34. Rowan, T.H.: Functional stability analysis of numerical algorithms. Ph.D. thesis, Department of Computer Science, University of Texas at Austin (1990)
35. Sanderson, C., Curtin, R.: Armadillo: a template-based C++ library for linear algebra. J. Open Source Softw. **1**(2), 26 (2016)
36. Sanderson, C., Curtin, R.: A user-friendly hybrid sparse matrix class in C++. In: 6th International Conference on Mathematical Software, pp. 422–430 (2018)
37. Sen, K., Viswanathan, M., Agha, G.: Model-checking Markov chains in the presence of uncertainties. In: 12th International Conference Tools and Algorithms for the Construction and Analysis of Systems, pp. 394–410 (2006)
38. Soudjani, S.E.Z., Gevaerts, C., Abate, A.: FAUST^2: formal abstractions of uncountable-state stochastic processes. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 272–286. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_23
39. Tabuada, P.: Verification and Control of Hybrid Systems: A Symbolic Approach. Springer, Cham (2009)
40. The HDF Group: Hierarchical Data Format, version 5 (1997–2023). https://www.hdfgroup.org/HDF5/
41. Tkachev, I., Mereacre, A., Katoen, J.P., Abate, A.: Quantitative automata-based controller synthesis for non-autonomous stochastic hybrid systems. In: Proceedings of the 16th ACM International Conference on Hybrid Systems: Computation and Control, pp. 293–302 (2013)
42. Van Huijgevoort, B., Schön, O., Soudjani, S., Haesaert, S.: SySCoRe: synthesis via stochastic coupling relations. In: Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control, pp. 1–11 (2023)
43. Weininger, M., Meggendorfer, T., Křetínskỳ, J.: Satisfiability bounds for ω-regular properties in bounded-parameter Markov decision processes. In: 2019 IEEE 58th Conference on Decision and Control (CDC), pp. 2284–2291 (2019)

44. Wooding, B., Lavaei, A.: IMPaCT: Interval MDP Parallel Construction for Controller Synthesis of Large-Scale Stochastic Systems (2024)
45. Zamani, M., Mohajerin Esfahani, P., Majumdar, R., Abate, A., Lygeros, J.: Symbolic control of stochastic systems via approximately bisimilar finite abstractions. IEEE Trans. Autom. Control **59**(12), 3135–3150 (2014)