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

BEN WOODING AND ABOLFAZL LAVAEI

SCHOOL OF COMPUTING, NEWCASTLE UNIVERSITY, UNITED KINGDOM

{BEN.WOODING,ABOLFAZL.LAVAEI}@NEWCASTLE.AC.UK

Abstract. This paper is concerned with developing a 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 on 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 can be formally expressed using high-level logic specifications, typically formulated as *linear temporal logic (LTL)* expressions [BK08]. 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 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* [Tab09, BYG17]. 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 [APLS08, JP09, ZMEM$^+$14, TMKA13, HHHK13, Nej23, NSZ21, LF22, LSAZ22].

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. (2.4)), *rendering MDPs impractical* for infinite-horizon properties.

As a promising alternative, *interval Markov decision processes* (IMDPs) [SVA06, GLD00] 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 (multi-player) 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 for 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 modern ISO 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 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 can be located at:

https://github.com/Kiguli/IMPaCT

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 [CHK13, DC20, HHS$^+$16, WMK19, DLML23] as well as more recently being used for data-driven learning problems [JZC22, LSFZ22, RAM23]. There exists a limited set of software tools available for the formal verification and controller synthesis of stochastic systems, encompassing various classes of stochastic models, using abstraction-based techniques. FAUST^2 [SGA15] 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 [CA19] 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

[1] https://github.com/AdaptiveCpp/AdaptiveCpp/

[2] https://www.khronos.org/sycl/

on the *value iteration* algorithm [LAB15], which *lacks convergence guarantees* when dealing with infinite-horizon specifications. This limitation has been widely pointed out in the relevant literature (see *e.g.,* [HM14, BKL$^+$17, HM18]). 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 sets IMPaCT apart from StocHy, where value iteration is the default choice without any general parallelization capability. 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 [KNP11] supports *robust* verification of uncertain models including IMDPs. However, PRISM requires the states and transitions of the IMDP to be described explicitly, which is cumbersome when the state and input spaces are large. Furthermore, our tool introduces parallel implementations of IMDP abstraction and synthesis algorithms, efficiently automating the IMDP construction process. It is noteworthy to mention that AMYTISS [LKSZ20] also employs parallel implementations but exclusively for constructing MDPs using PFaces [KZ19], 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) [AH20, AH23], employed in IMPaCT, provides a sturdy foundation for flexible cross-platform parallel processing on CPUs, GPUs and hardware accelerators (HWAs) from all major hardware vendors such as Intel, NVIDIA, etc., which is not the case in AMYTISS.

There exist a few more software tools for formal verification and controller synthesis of stochastic systems within diverse model classes, all without employing IMDP approaches. In this regard, SReachTools [VGO19] conducts stochastic reachability analysis specifically for linear (time-varying) discrete-time stochastic systems. ProbReach [SZ15] is developed for the probabilistic reachability verification of stochastic hybrid systems. SReach [WZK$^+$15] addresses probabilistic bounded reachability problems specifically within nonlinear hybrid automata affected by parametric uncertainty. Modest Toolset [HH14] enables modeling and analysis of distributed stochastic hybrid systems. SySCoRe [VHSSH23] is a MATLAB toolbox that derives simulation relations and presents an alternative methodology for the controller synthesis of stochastic systems satisfying co-safe specifications over infinite horizons. The last three ARCH workshops focused on friendly tool competitions over stochastic models are reported in [ABC$^+$20, ABD$^+$22, ABC$^+$23].

1.3. **Paper Organization.** This paper is structured as follows. In Section 2, we define discrete-time stochastic control systems and their equivalent representations as continuous-space MDPs. In Section 3, we define IMDPs, while presenting a serial algorithm for IMDP construction. Section 4 is dedicated to the parallel algorithms of IMDP construction. In Section 5, we discuss serial algorithms for controller synthesis via constructed IMDPs, enforcing safety, reachability, and reach-while-avoid properties. Section 6 offers the parallel synthesis algorithms using constructed IMDPs. Section 7 elaborates on the process of loading and saving respective files throughout various stages of the abstraction and synthesis procedure. Section 8 showcases the outcomes of IMPaCT when implemented on well-known case studies and benchmark scenarios.

1.4. **Preliminaries.** We consistently employ the following notation throughout this work. The set of real numbers is denoted by $\mathbb{R}$ and the set of natural numbers including zero by $\mathbb{N}$. The empty set is denoted by $\emptyset$. We denote the cardinality of a set A as $|A|$ and the power set as 2^A. The Euclidean norm is denoted by $\|\cdot\|_2$, while the infinity norm is represented by $\|\cdot\|_\infty$. We use the notation $\mathsf{diag}(a_1, ..., a_n)$ to represent a diagonal matrix in $\mathbb{R}^{n\times n}$, with its diagonal entries $a_1, ..., a_n$ placed from the upper left corner. We consider a moment in time k to be from the time horizon belonging to $\mathbb{N}$. Where it is clear in context, time will be omitted for simplicity, *e.g.* $x(k) \to x$. We consider a probability space $(\Omega, \mathcal{F}_\Omega, \mathbb{P}_\Omega)$, where Ω is the sample space, $\mathcal{F}_\Omega$ is the sigma-algebra of Ω with elements of Ω called events, and $\mathbb{P}_\Omega$ is the probability measure that assigns a probability to each event. A topological space X is called a Borel space if there exists a metric on X that makes it a separable and complete metrizable space; X is then endowed with a Borel sigma-algebra $\mathscr{B}(X)$. The measurable space of X is $(X, \mathscr{B}(X))$. Other notation will be introduced when required.

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

$$\Sigma = (X, U, W, \varsigma, f), \tag{2.1}$$

where,

- $X \subseteq \mathbb{R}^n$ *is a Borel space as the state set, with* $(X, \mathscr{B}(X))$ *being its measurable space;*
- $U \subseteq \mathbb{R}^m$ *is a Borel space as the input set;*
- $W \subseteq \mathbb{R}^p$ *is a Borel space as the disturbance set;*
- 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

$$\Sigma\colon x(k+1) = f(x(k), u(k), w(k), \varsigma(k)), \quad k \in \mathbb{N}. \tag{2.2}$$

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 [Kal21] 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 (2.1).

Definition 2.2 (Continuous-Space MDPs)**.** *A continuous-space Markov decision process (MDP) is a quadtuple*

$$\Sigma = (X, U, W, T), \tag{2.3}$$

where,

- *X, U, and W are as in Definition 2.1;*
- *$T\colon \mathscr{B}(X) \times X \times U \times W \to [0,1]$ 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 $A \in \mathscr{B}(X)$,*

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

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 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 [LSZ18, 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:

$$\left|\mathbb{P}(\Sigma \models \psi) - \mathbb{P}(\hat{\Sigma} \models \psi)\right| \leq \rho = \mathcal{K}\mathcal{H}\mathscr{L}\eta_x. \tag{2.4}$$

3. Interval Markov Decision Processes

The finite MDP presented in Section 2 is not suitable for infinite-horizon problems due to the inclusion of the time horizon $\mathcal{K}$ in (2.4). In particular, as $\mathcal{K} \to \infty$ then $\rho \to \infty$, the distance between probabilities of satisfaction over concrete system Σ and its finite MDP $\hat{\Sigma}$ converges to infinity, implying that the abstraction will be of no use to providing satisfaction guarantees. To alleviate this, a continuous-space MDP Σ in (2.3) 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 3.1 (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,

- $\hat{X} = \{\hat{x}_0, \hat{x}_1, \dots \hat{x}_{n_x}\}$*, with* $\hat{x}_i$ *being the representative point within* $\boldsymbol{X}^i$*, where* $X = \cup_{i=0}^{n_x} \boldsymbol{X}^i$*;*
- $\hat{U} = \{\hat{u}_0, \hat{u}_1, \dots \hat{u}_{n_u}\}$*, with* $\hat{u}_i$ *being the representative point within* $\boldsymbol{U}^i$*, where* $U = \cup_{i=0}^{n_u} \boldsymbol{U}^i$*;*
- $\hat{W} = \{\hat{w}_0, \hat{w}_1, \dots \hat{w}_{n_w}\}$*, with* $\hat{w}_i$ *being the representative point within* $\boldsymbol{W}^i$*, where* $W = \cup_{i=0}^{n_w} \boldsymbol{W}^i$*;*
- $\hat{T}_{\min}$ *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*

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

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

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.

3.1. **Temporal Logic Specifications.** Here, we define the specifications of interest handled by IMPaCT. The desired specification ψ is codified using *linear temporal logic* (LTL) [BK08]. Of particular interest are the properties described via logical operators including *always* $\Box$, *eventually* $\Diamond$, and *until* U.

Definition 3.2 (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 $\mathcal{S} \subseteq X \backslash \mathcal{A}$ until it reaches the target region $\mathcal{T} \subseteq X$, with $\mathcal{A}$ being an avoid region.*

When constructing IMCs/IMDPs, we aim at labeling the states within the state space based on the specification, especially beneficial for later-stage verification or controller synthesis processes. To do so, with a slight abuse of notation, we consider states $\hat{x} \in \mathcal{S}$ to refer explicitly to states in the state space considered to be safe, *i.e.,* for safety specifications these states belong to the safe region $\mathcal{S}$. We now relabel states from the state space that belong to the target set as target states $\hat{r} \in \mathcal{T}$ and assume that all of these states are absorbing states, *i.e.,* if the system reaches the target region, it will remain within that region. Similarly, we relabel any states from the state space that also belong to the avoid region as avoid states $\hat{a} \in \mathcal{A}$, treating the avoid region as an absorbing area. 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. According to this implementation technique, we introduce static vectors that capture the minimum and maximum probabilities of transitions to the target region, with row entries calculated by

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

Similarly, for transitions to the avoid region, we define

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

Therefore, one can redefine the IMDP depending on the specification as $\hat{\Sigma} = (\hat{X}, \hat{U}, \hat{W}, \hat{T}_{\min}, \hat{T}_{\max}, \hat{R}_{\min}, \hat{R}_{\max}, \hat{A}_{\min}, \hat{A}_{\max})$. It is worth noting that users do not need to be aware of these additional vectors, as they are automatically computed during the abstraction process from the state labeling, as detailed in Algorithm 1.

3.2. **Construction of IMDP.** We now describe the approach taken to construct an IMDP $\hat{\Sigma}$, including how to compute minimal and maximal transition probability matrices $\hat{T}_{\min}$, $\hat{T}_{\max}$, $\hat{R}_{\min}$, $\hat{R}_{\max}$, $\hat{A}_{\min}$, and $\hat{A}_{\max}$, which is essential for later-stage verification or controller synthesis procedures. We first present a serial algorithm for the construction of IMDPs, as Algorithm 1, considering stochasticity with normal distributions

```
Algorithm 1: Serial construction of IMDP
Input: Continious MDP $\Sigma = (X, U, W, T)$ and specification $\psi$
1  Select finite partition of sets $X, U, W$ as $X = \cup_{i=0}^{n_x} \mathbf{X}^i$, $U = \cup_{i=0}^{n_u} \mathbf{U}^i$, $W = \cup_{i=0}^{n_w} \mathbf{W}^i$;
2  For each $\mathbf{X}^i$ select a representative point $\hat{x}^i \in \mathbf{X}^i$. Similarly, for each $\mathbf{U}^i$ and $\mathbf{W}^i$, select a representative
     point $\hat{u}^i \in \mathbf{U}^i$ and $\hat{w}^i \in \mathbf{W}^i$;
3  Define $\hat{X} := \{\hat{x}^i, i = 0, \dots, n_x\}$ as the finite state set of $\hat{\Sigma}$ with input set $\hat{U} := \{\hat{u}^i, i = 0, \dots, n_u\}$ and
     disturbance set $\hat{W} := \{\hat{w}^i, i = 0, \dots, n_w\}$;
4  Define the map $\Xi : X \rightarrow 2^X$ that assigns to any $x \in X$, the corresponding partition set it belongs to, i.e.,
     $\Xi(x) = \mathbf{X}^i$ if $x \in \mathbf{X}^i$;
5  Label states as safe states $\hat{x} \in \mathcal{S}$, target states $\hat{r} \in \mathcal{T}$, or avoid states $\hat{a} \in \mathcal{A}$, based on desired specification
     $\psi$;
6  for $\hat{w}_i \in \hat{W}, i = \{0, \dots, n_w\}$ do
7      for $\hat{u}_j \in \hat{U}, j = \{0, \dots, n_u\}$ do
8          for $\hat{x}_k \in \mathcal{S}, k = \{0, \dots, n_s\}$ do
9              for $\hat{x}'_l \in \mathcal{S}, l = \{0, \dots, n_s\}$ do
10                 $\hat{T}_{\min}(\hat{x}'_l \,|\, \hat{x}_k, \hat{u}_j, \hat{w}_i) = \min_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{x}'_l) \,|\, x, \hat{u}_j, \hat{w}_i)$;
11                 $\hat{T}_{\max}(\hat{x}'_l \,|\, \hat{x}_k, \hat{u}_j, \hat{w}_i) = \max_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{x}'_l) \,|\, x, \hat{u}_j, \hat{w}_i)$;
12             end
13             $\hat{R}_{\min}(\hat{x}_k, \hat{u}_j, \hat{w}_i) = \sum_{\forall \hat{r} \in \mathcal{T}} \min_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{r}) \,|\, x, \hat{u}_j, \hat{w}_i)$;
14             $\hat{R}_{\max}(\hat{x}_k, \hat{u}_j, \hat{w}_i) = \sum_{\forall \hat{r} \in \mathcal{T}} \max_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{r}) \,|\, x, \hat{u}_j, \hat{w}_i)$;
15             $\hat{A}_{\min}(\hat{x}_k, \hat{u}_j, \hat{w}_i) = \sum_{\forall \hat{a} \in \mathcal{A}} \min_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{a}) \,|\, x, \hat{u}_j, \hat{w}_i)$;
16             $\hat{A}_{\max}(\hat{x}_k, \hat{u}_j, \hat{w}_i) = \sum_{\forall \hat{a} \in \mathcal{A}} \max_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{a}) \,|\, x, \hat{u}_j, \hat{w}_i)$;
17         end
18     end
19 end
Output: IMDP $\hat{\Sigma} = (\hat{X}, \hat{U}, \hat{W}, \hat{T}_{\min}, \hat{T}_{\max}, \hat{R}_{\min}, \hat{R}_{\max}, \hat{A}_{\min}, \hat{A}_{\max})$
```

which are the default noise type in IMPaCT, while providing a custom user-defined distribution to support *any arbitrary distributions.*

In Algorithm 1, Steps $1 - 2$ first construct finite partitions and obtain the representative points within X, U, and W. In Step 3, the finite set of states $\hat{X}$, inputs $\hat{U}$, and disturbances $\hat{W}$ are defined, where $n_x = |\hat{X}|$,

$n_u = |\hat{U}|$, and $n_w = |\hat{W}|$. In Step 5, we filter the states based on their labeling for the specification ψ. In Steps $10-11$, the transition probability matrices $\hat{T}_{\min}$ and $\hat{T}_{\max}$ are computed by iterating through all the combinations of safe states $\hat{x} \in \mathcal{S}$, where $n_s = |\mathcal{S}|$, inputs $\hat{u} \in \hat{U}$, and disturbances $\hat{w} \in \hat{W}$. To do so, a nonlinear optimization is required to find $x \in \Xi(\hat{x})$ corresponding to either the lower bound or upper bound probability of transitioning from $\Xi(\hat{x})$ to $\Xi(\hat{x}')$. Similarly, Steps $13-16$ perform the same procedure for one-step transitions going to either the target or avoid regions. The output of the algorithm is therefore IMDP $\hat{\Sigma} = (\hat{X}, \hat{U}, \hat{W}, \hat{T}_{\min}, \hat{T}_{\max}, \hat{R}_{\min}, \hat{R}_{\max}, \hat{A}_{\min}, \hat{A}_{\max})$.

Remark 3.3. *One can readily construct interval Markov chains (IMCs) for verification purposes via Algorithm 1 by setting the control input to zero within the concrete dynamics,* i.e., $\Sigma = (X, W, T)$.

3.3. **Complexity Analysis for Serial Construction of IMDP.** To conduct a computational complexity analysis for the serial construction of IMDP, we adopt a slight abuse of notation by defining n_{x_i}, n_{u_j}, and n_{w_k} as the dimension-wise counts of partitions within $\hat{X}$, $\hat{U}$, and $\hat{W}$, respectively, where $i = 1, \dots, n$, $j = 1, \dots, m$, and $k = 1, \dots, p$. 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 $d = (n_{x_i}^n \times n_{u_j}^m \times n_{w_k}^p) \times n_{x_i}^n$. The value d represents the total count of rows in the static transition matrix, *i.e.,* $n_x \times n_x \times n_u$, multiplied by the number of columns, *i.e.,* n_x. Given that the construction of IMDP abstraction involves both lower bound 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 within Steps $10-11$ in Algorithm 1, represented as $\mathcal{O}(\kappa)$. The level of complexity κ hinges on the algorithm chosen by the user, detailed in Section 4.3. Consequently, the overall complexity of Algorithm 1 amounts to $\mathcal{O}(2\kappa d)$. This accounts for both the space explosion (d) due to the system dimensions and the computational load (κ) associated with nonlinear optimization.

4. Parallel Construction of IMDP

In this section, we describe how we update Algorithm 1 for parallel processing as offered by IMPaCT. We also here present the user functions of our tool corresponding to various subsections. Firstly, an IMDP object should be constructed by defining the dimensions of state, input and disturbance:

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

Listing 1. Creating IMDP object.

4.1. **Setting Noise Distribution.** IMPaCT by default supports normal distributions but also allows a user-defined distribution to be provided:

```
enum class NoiseType {NORMAL, CUSTOM};
```

Listing 2. Define noise distribution.

We consider here an i.i.d. noise where the tool receives a vector of standard deviations. In this setup, the covariance matrix is diagonal, and each diagonal entry represents the variance:

```
void setStdDev(vec sig);
void setNoise(NoiseType n, bool diagonal = true);
```

Listing 3. Diagonal covariance matrix.

IMPaCT also accommodates full covariance matrix, with nonzero off-diagonal elements, when provided with the inverse covariance matrix and its determinant. For multi-dimensional noise distributions, Monte Carlo integration is employed instead of a closed-form solution, offering advantages for higher dimension integrals [NB99], requiring the user to specify the number of samples for the integration process. We utilize the GNU Scientific Library for this integration process [Gou09]. In a broader scope, matrix and vector manipulations are implemented using the C++ library Armadillo [SC16, SC18]:

```
void setInvCovDet(mat inv_cov, double det);
void setNoise(NoiseType n, bool diagonal, size_t monte_carlo_samples);
```

Listing 4. Full covariance matrix.

We facilitate the utilization of a user-defined custom distribution, prompting the user to define the specific desired distribution. The code will provide a struct `customParams` containing data for the variables `state_start` $\hat{x}$, `input` $\hat{u}$, `disturb` $\hat{w}$, `eta` η_x, `mean` $f(\hat{x}, \hat{u}, \hat{w})$, `lb` lower bound of the integration, `ub` upper bound of the integration, and `dynamics{1,2,3}` dynamics with numbers representing the number of function parameters. The `customPDF` function intended for integration can be coded within the file `src/custom.cpp`, and the parameters for the Monte Carlo integration–`CUSTOM` noise and the number of samples–can be configured using:

```
void setNoise(NoiseType n);
void setCustomDistribution(size_t monte_carlo_samples);
```

Listing 5. Custom noise distributions.

4.2. **Parallel Construction of Transition Probability Matrices.** We propose in Algorithm 2 the parallel approach for constructing the transition matrices $\hat{T}_{\min}$, $\hat{T}_{\max}$, $\hat{R}_{\min}$, $\hat{R}_{\max}$, $\hat{A}_{\min}$, and $\hat{A}_{\max}$. Steps $1-5$ are equivalent to Algorithm 1. In IMPaCT, these Steps $1-5$, can be computed using the following user functions. The dimensions and representative points within each space can be specified by setting the lower bounds, upper bounds, and discretization parameters. As elaborated in Subsection 3.1, we consider states within the state space as safe states by default, constituting the safe region. Subsequently, the target and avoid regions can be delineated by employing Boolean expressions that assess the state space. These expressions facilitate the relabeling of states, transitioning them from safe states to either target or avoid states based on desired specifications:

```
void setStateSpace(vec lb, vec ub, vec eta);
void setInputSpace(vec lb, vec ub, vec eta);
void setDisturbSpace(vec lb, vec ub, vec eta);
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 6. Construction of $\hat{X}, \hat{U}, \hat{W}, \mathcal{T}$, and $\mathcal{A}$.

In Steps $8-9$, transition probabilities are computed *in parallel* for the value $x \in \Xi(\hat{x})$ corresponding to either the minimal (lower bound) or maximal (upper bound) probability of transitioning from $\Xi(\hat{x})$ to $\Xi(\hat{x}')$. In Step $11-12$, target vector bounds $\hat{R}_{\min}$ and $\hat{R}_{\max}$ calculate the one-step transition probability of the state $\hat{x}$ transitioning to $\mathcal{T}$. Similarly, avoid vector bounds $\hat{A}_{\min}$ and $\hat{A}_{\max}$ calculate the transition probability of the state $\hat{x}$ transitioning to $\mathcal{A}$. For each state-input-disturbance triple, we sum the probabilities together resulting in the respective target vector or avoid vector. In IMPaCT, Steps $8-9$ and $11-14$, can be computed using the following function calls:

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

Listing 7. Abstraction of transition matrices.

Remark 4.1. *In IMPaCT, given that the state space is bounded, some case studies may involve potential transitions outside the defined state space. This is captured in the functions* `minAvoidTransitionVector()`

Algorithm 2: *Parallel* Construction of IMDP

```
Input: MDP $\Sigma = (X, U, W, T)$ and specification $\psi$
Select finite partition of sets $X, U, W$ as: $X = \cup_{i=0}^{n_x} \mathbf{X}^i$, $U = \cup_{i=0}^{n_u} \mathbf{U}^i$, $W = \cup_{i=0}^{n_w} \mathbf{W}^i$;
For each partition $\mathbf{X}^i$ select a representative point $\hat{x}^i \in \mathbf{X}^i$ used to describe the partition, e.g., centre of
  the partition. Similarly for $\mathbf{U}^i$ and $\mathbf{W}^i$, select representative points $\hat{u}^i \in \mathbf{U}^i$ and $\hat{w}^i \in \mathbf{W}^i$;
Define $\hat{X} := \{\hat{x}^i, i = 0, \dots, n_x\}$ as the finite state set of $\hat{\Sigma}$ with input set $\hat{U} := \{\hat{u}^i, i = 0, \dots, n_u\}$ and
  disturbance set $\hat{W} := \{\hat{w}^i, i = 0, \dots, n_w\}$;
Define the map $\Xi : X \to 2^X$ that assigns to any $x \in X$, the corresponding partition set it belongs to, i.e.,
  $\Xi(x) = \mathbf{X}^i$ if $x \in \mathbf{X}^i$;
Relabel states as safe states $\hat{x} \in \mathcal{S}$, target states $\hat{r} \in \mathcal{T}$, or avoid states $\hat{a} \in \mathcal{A}$ based on desired
  specification $\psi$;
forall combinations of $\hat{x}_k \in \mathcal{S}, \hat{u}_j \in \hat{U}$, and $\hat{w}_i \in \hat{W}$ in parallel do
    for $\hat{x}'_l \in \mathcal{S}, l = \{0, \dots, n_x\}$ do
        $\hat{T}_{\min}(\hat{x}'_l | \hat{x}_k, \hat{u}_j, \hat{w}_i) = \min_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{x}'_l) | x, \hat{u}_j, \hat{w}_i)$;
        $\hat{T}_{\max}(\hat{x}'_l | \hat{x}_k, \hat{u}_j, \hat{w}_i) = \max_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{x}'_l) | x, \hat{u}_j, \hat{w}_i)$;
    end
    $\hat{R}_{\min}(\hat{x}_k, \hat{u}_j, \hat{w}_i) = \sum_{\forall \hat{r} \in \mathcal{T}} \min_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{r}) | x, \hat{u}_j, \hat{w}_i)$;
    $\hat{R}_{\max}(\hat{x}_k, \hat{u}_j, \hat{w}_i) = \sum_{\forall \hat{r} \in \mathcal{T}} \max_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{r}) | x, \hat{u}_j, \hat{w}_i)$;
    $\hat{A}_{\min}(\hat{x}_k, \hat{u}_j, \hat{w}_i) = \sum_{\forall \hat{a} \in \mathcal{A}} \min_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{a}) | x, \hat{u}_j, \hat{w}_i)$;
    $\hat{A}_{\max}(\hat{x}_k, \hat{u}_j, \hat{w}_i) = \sum_{\forall \hat{a} \in \mathcal{A}} \max_{x \in \Xi(\hat{x}_k)} T(\Xi(\hat{a}) | x, \hat{u}_j, \hat{w}_i)$;
end
Output: IMDP $\hat{\Sigma} = (\hat{X}, \hat{U}, \hat{W}, \hat{T}_{\min}, \hat{T}_{\max}, \hat{R}_{\min}, \hat{R}_{\max}, \hat{A}_{\min}, \hat{A}_{\max})$
```

as $\hat{A}_{\min}(\hat{x}, \hat{u}, \hat{w}) = 1 - \hat{T}_{\max}(\hat{X} | \hat{x}, \hat{u}, \hat{w})$ *and with a similar construction for* `maxAvoidTransitionVector()`*. These functions also consider the minimal and maximal transition probabilities to any additional avoid states that are inside the state space by summing them to* $\hat{A}_{\min}(\hat{x}, \hat{u}, \hat{w})$ *and* $\hat{A}_{\max}(\hat{x}, \hat{u}, \hat{w})$*, respectively.*

4.3. **Complexity Analysis for Parallel Construction of IMDP.** Running these steps in parallel enhances the performance of solving the abstraction and synthesis problems. Using $\mathcal{O}(\kappa)$ to represent the optimization complexity, the complexity of Algorithm 2 is therefore $\mathcal{O}(\frac{2\kappa d}{\text{THREADS}})$, 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. We implement the optimization algorithms using

NLopt [Joh07], by default we select `nlopt::LN_SBPLX` [Row90], which is a variant of the derivative-free Nelder-Mead Simplex algorithm. It is worth highlighting that the chosen algorithm in IMPaCT can be readily swapped to any other nonlinear optimization algorithm from NLopt[3] using the following function:

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

Listing 8. Set NLopt algorithm.

4.4. **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 the *low-cost abstraction*. This speed up can be used by calling the following functions for the transition matrices and the target transition vectors, respectively:

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

Listing 9. Low-cost abstractions.

5. Controller Synthesis with Interval Iteration

We now synthesize a controller, via constructed IMDP, to enforce infinite-horizon properties over dt-SCS. When dealing with infinite-horizon problems, the *interval iteration* algorithm is capable of providing convergence guarantees unlike the common *value iteration* [HM18]. In particular, the interval iteration algorithm converges to an under-approximation and over-approximation of the 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 [HM18]. In finite MDP construction, a state within a partition is predetermined beforehand, and the probability transitions to all other partitions are computed based on this state. However, in IMDP construction, each partition might utilize a distinct state when calculating probability transitions to another partition, contingent upon the destination partition, *i.e.,* the partition where the system lands

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

after a one-step evolution. Consequently, solving the dynamic program can be seen as akin to determining a partition's state that could universally serve for computing probability transitions to other partitions. This solution is referred to as a *feasible* distribution. In particular, the feasible distribution is determined to optimize (either minimize or maximize) the decision variables given the desired property of interest. Subsequently, the corresponding optimization is minimized with respect to the disturbance $\hat{w}$ and maximized concerning the input $\hat{u}$:

$$\max_{\hat{u}\in\hat{U}}\min_{\hat{w}\in\hat{W}}\operatorname*{optimize}_{\hat{R},\hat{A},\hat{T}}\ \delta_1\hat{R}(\hat{x},\hat{u},\hat{w})+\delta_2\hat{A}(\hat{x},\hat{u},\hat{w})+\sum_{\forall\hat{x}'\in\hat{X}}\delta_3(\hat{x}')\hat{T}(\hat{x}'|\hat{x},\hat{u},\hat{w}) \tag{5.1}$$

with weight functions δ_1, δ_2, and $\delta_3(\cdot)$, and being subject to the following constraints [HM18]:

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

In particular, $\hat{R}$ mimics s^+ and $\hat{A}$ mimics s^- in the formulation of [HM18]. The corresponding weights are attributed to the dynamic program and are updated at each iteration step. The controller synthesis here uses a linear program (LP) to converge to an optimal solution, where the weights are the probability of satisfying the specification and δ_3 is derived from either V_0 or V_1. Specifically, the optimization in (5.1) takes into account the probability from transitioning from a partition with representative state $\hat{x}$ to another partition with representative state $\hat{x}'$ multiplied by some weight. This weight corresponds to the probability of satisfying the specification when initiating from a state within the partition indicated by $\hat{x}'$. The affine part of the optimization depends on the specification for the weights attributed to $\hat{R}$ and $\hat{A}$. For reachability specifications, $\delta_1 = 1$ and $\delta_2 = 0$, whereas for safety, the inverse holds true. Additionally, the probability of fulfilling the specification complements the resulting optimal values for V_0 and V_1.

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

$$\begin{cases}V_0'=\delta_1\hat{R}+\delta_2\hat{A}+\hat{T}V_0,\\V_1'=\delta_1\hat{R}+\delta_2\hat{A}+\hat{T}V_1,\end{cases} \tag{5.2}$$

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 resulting controller $\mathcal{C} = (\hat{X}, \pi, \mathbb{P}_{\psi_{\min}}, \mathbb{P}_{\psi_{\max}})$ consists of a lookup table with the optimal control policy π for each representative point $\hat{x} \in \mathcal{S}$, the minimal probability of satisfying the specification $\mathbb{P}_{\psi_{\min}}$ to and the maximal probability of satisfying the specification $\mathbb{P}_{\psi_{\max}}$. When synthesizing the controller, either $\mathbb{P}_{\psi_{\min}}$ or $\mathbb{P}_{\psi_{\max}}$ is prioritized for finding π, giving a *pessimistic or optimistic* control policy, respectively. In particular, a pessimistic policy is when the policy π is optimized using (5.1) that minimizes $\hat{A}, \hat{R}$ and $\hat{T}$, while an optimistic policy maximizes $\hat{A}, \hat{R}$ and $\hat{T}$ [DLML23]. The policy π is then fixed when calculating the other bound, *e.g.,* π synthesized from calculations on $\mathbb{P}_{\psi_{\min}}$ is fixed when finding $\mathbb{P}_{\psi_{\max}}$.

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 represents the range across the targeted partition addressed by the optimization program, acting as an adversary in computing $\mathbb{P}_{\psi_{\min}}$ and an ally when computing $\mathbb{P}_{\psi_{\max}}$.

5.1. **Safety Specifications.** Safety specifications seem to be typically the simplest of the three specification types handled by IMPaCT. However, the interval iteration algorithm for safety specifications is not entirely straightforward. In particular, for controller synthesis via finite MDPs, value iteration is used where the transition probability matrix $\hat{T}$ is multiplied by a vector of ones V_1 at each iteration backward over time, *e.g.* $V_1' = \hat{T}V_1$. In contrast, interval iteration in IMDP requires a static affine vector (*i.e.,* either $\hat{A}$ or $\hat{R}$), otherwise V_0' will always remain $\mathbf{0}_n$ as $V_0' = \hat{T}V_0$ with $V_0 = \mathbf{0}_n$, and $V_1' = \hat{T}V_1$ with $V_0 = \mathbf{1}_n$ at the start of the Bellman equation. Hence, the algorithm cannot converge, except when V_1 converges to $\mathbf{0}_n$, indicating zero safety guarantee.

To resolve this challenge, given that safety and reachability are complement of each other, we fulfill our safety specification by solving a reachability problem over the complement of the safe set. Logically speaking, $\square\mathcal{S}$ is equivalent to $\neg\lozenge\neg\mathcal{S}$, where $\neg$ refers to the LTL term "not". Therefore, the dynamic program to be solved is (5.1), where $\delta_1 = 0$, $\delta_2 = 1$, and $\delta_3 = V_1$, and the set $\mathcal{T} = \emptyset$. Additionally, we minimize with respect to the input $\hat{u}$ and maximize with respect to the disturbance $\hat{w}$. This approach stems from the intuition behind solving a reachability problem toward an *undesired* area. Consequently, we aim to select the input least likely to reach the avoid region, which equates to choosing the input most likely to remain within the safe region. Similarly, the adversary selects the disturbance most likely to reach the avoid region. When the algorithm converges, the complement is taken to give the resulting $\mathbb{P}_{\psi_{\min}}$ and $\mathbb{P}_{\psi_{\max}}$. It is notable that for safety specifications over an *infinite time horizon*, it is not uncommon for the solution to converge to the trivial result where $\mathbb{P}_{\psi_{\min}} = 0$ and $\mathbb{P}_{\psi_{\max}} = 1$, or for both bounds to converge to 0.

Algorithm 3: Serial (Pessimistic) Safety Controller Synthesis

```
Input: IMDP $\hat{\Sigma}$, stopping condition $\varepsilon$
1  $V_0 := \mathbf{0}_n,\ V_1 := \mathbf{1}_n,\ \pi := \mathbf{0}_n$;
2  while $\|V_1 - V_0\|_\infty > \varepsilon$ do
3      for $\hat{x}_k \in \mathcal{S}, k = \{0, \dots, |\mathcal{S}|\}$ do
4          Solve max LP (5.1) for $\hat{A}$ and $\hat{T}$;
5          $\pi(\hat{x}_k) := \operatorname*{argmin}_{\hat{u} \in \hat{U}} \max_{\hat{w} \in \hat{W}} \max_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k, \cdot, \cdot) + \sum_{\forall \hat{x}' \in \mathcal{S}} \hat{T}(\cdot | \hat{x}_k, \cdot, \cdot) V_0(\hat{x}')\}$;
6          $V_0'(\hat{x}_k) := \min_{\hat{u} \in \hat{U}} \max_{\hat{w} \in \hat{W}} \max_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k, \cdot, \cdot) + \sum_{\forall \hat{x}' \in \mathcal{S}} \hat{T}(\cdot | \hat{x}_k, \cdot, \cdot) V_0(\hat{x}')\}$;
7          $V_1'(\hat{x}_k) := \min_{\hat{u} \in \hat{U}} \max_{\hat{w} \in \hat{W}} \max_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k, \cdot, \cdot) + \sum_{\forall \hat{x}' \in \mathcal{S}} \hat{T}(\cdot | \hat{x}_k, \cdot, \cdot) V_1(\hat{x}')\}$;
8      end
9      $V_0 := V_0',\ V_1 := V_1'$;
10 end
11 $\mathbb{P}_{\psi_{\min}} = \mathbf{1}_n - V_0$;
12 $V_0 := \mathbf{0}_n,\ V_1 := \mathbf{1}_n$;
13 while $\|V_1 - V_0\|_\infty > \varepsilon$ do
14     for $\hat{x}_k \in \mathcal{S}, k = \{0, \dots, |\mathcal{S}|\}$ do
15         Solve min LP (5.1) for $\hat{A}$ and $\hat{T}$;
16         fix $\hat{u}$ from $\pi(\hat{x}_k)$;
17         $V_0'(\hat{x}_k) := \max_{\hat{w} \in \hat{W}} \min_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k, \hat{u}, \cdot) + \sum_{\forall \hat{x}' \in \mathcal{S}} \hat{T}(\cdot | \hat{x}_k, \hat{u}, \cdot) V_0(\hat{x}')\}$;
18         $V_1'(\hat{x}_k) := \max_{\hat{w} \in \hat{W}} \min_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k, \hat{u}, \cdot) + \sum_{\forall \hat{x}' \in \mathcal{S}} \hat{T}(\cdot | \hat{x}_k, \hat{u}, \cdot) V_1(\hat{x}')\}$;
19     end
20     $V_0 := V_0',\ V_1 := V_1'$;
21 end
22 $\mathbb{P}_{\psi_{\max}} = \mathbf{1}_n - V_1$;
Output: controller $\mathcal{C} = (\mathcal{S}, \pi, \mathbb{P}_{\psi_{\min}}, \mathbb{P}_{\psi_{\max}})$
```

In Algorithm 3, controller synthesis for a pessimistic safety specification in serial is displayed. Steps $6-7$, and Steps $17-18$ show the interval iteration algorithm where the first loops find the optimal *pessimistic* control policy π using the lower bound and the second loops fix the control policy to find the IMDP upper bound. Notice that when calculating $\mathbb{P}_{\psi_{\min}}$, we solve the max LP, and for $\mathbb{P}_{\psi_{\max}}$ we solve the min LP. Step 12 and Step 22 show the complement is taken, giving us the safety probability bounds.

Algorithm 4: Serial (Pessimistic) Reach-Avoid Controller Synthesis

```
Input: IMDP $\hat{\Sigma}$, stopping condition $\varepsilon$
$V_0 := \mathbf{0}_n,\ V_1 := \mathbf{1}_n,\ \pi := \mathbf{0}_n$;
while $\|V_1 - V_0\|_\infty > \varepsilon$ do
    for $\hat{x}_k \in \mathcal{S}, k = \{0, \dots, |\mathcal{S}|\}$ do
        Solve min LP (5.1) for $\hat{R}$ and $\hat{T}$;
        $\pi(\hat{x}_k) := \operatorname*{argmax}_{\hat{u} \in \hat{U}} \min_{\hat{w} \in \hat{W}} \min_{\hat{R}, \hat{T}} \{\hat{R}(\hat{x}_k, \cdot, \cdot) + \sum_{\forall \hat{x}' \in \hat{X}} \hat{T}(\cdot|\hat{x}_k, \cdot, \cdot) V_0(\hat{x}')\}$;
        $V_0'(\hat{x}_k) := \max_{\hat{u} \in \hat{U}} \min_{\hat{w} \in \hat{W}} \min_{\hat{R}, \hat{T}} \{\hat{R}(\hat{x}_k, \cdot, \cdot) + \sum_{\forall \hat{x}' \in \hat{X}} \hat{T}(\cdot|\hat{x}_k, \cdot, \cdot) V_0(\hat{x}')\}$;
        $V_1'(\hat{x}_k) := \max_{\hat{u} \in \hat{U}} \min_{\hat{w} \in \hat{W}} \min_{\hat{R}, \hat{T}} \{\hat{R}(\hat{x}_k, \cdot, \cdot) + \sum_{\forall \hat{x}' \in \hat{X}} \hat{T}(\cdot|\hat{x}_k, \cdot, \cdot) V_1(\hat{x}')\}$;
    end
    $V_0 := V_0',\ V_1 := V_1'$;
end
$\mathbb{P}_{\psi_{\min}} = V_0$;
$V_0 := \mathbf{0}_n,\ V_1 := \mathbf{1}_n$;
while $\|V_1 - V_0\|_\infty > \varepsilon$ do
    for $\hat{x}_k \in \mathcal{S}, k = \{0, \dots, |\mathcal{S}|\}$ do
        Solve max LP (5.1) for $\hat{R}$ and $\hat{T}$;
        fix $\hat{u}$ from $\pi(\hat{x}_k)$;
        $V_0'(\hat{x}_k) := \min_{\hat{w} \in \hat{W}} \max_{\hat{R}, \hat{T}} \{\hat{R}(\hat{x}_k, \hat{u}, \cdot) + \sum_{\forall \hat{x}' \in \hat{X}} \hat{T}(\cdot|\hat{x}_k, \hat{u}, \cdot) V_0(\hat{x}')\}$;
        $V_1'(\hat{x}_k) := \min_{\hat{w} \in \hat{W}} \max_{\hat{R}, \hat{T}} \{\hat{R}(\hat{x}_k, \hat{u}, \cdot) + \sum_{\forall \hat{x}' \in \hat{X}} \hat{T}(\cdot|\hat{x}_k, \hat{u}, \cdot) V_1(\hat{x}')\}$;
    end
    $V_0 := V_0',\ V_1 := V_1'$;
end
$\mathbb{P}_{\psi_{\max}} = V_1$;
Output: controller $\mathcal{C} = (\mathcal{S}, \pi, \mathbb{P}_{\psi_{\min}}, \mathbb{P}_{\psi_{\max}})$
```

5.2. **Reachability and Reach-while-Avoid Specifications.** Reachability specifications typically introduce additional complexities compared to safety specifications, primarily due to the set of target states that the control policy must guide the system toward. Specifically, states that are not in the target (or avoid) region are considered safe, but the system should make progress towards the target set $\mathcal{T}$. Interval iteration for reachability is relatively straightforward. In our setting, $\hat{R}$ represents the transition probability of a state to the target region $\mathcal{T}$.

Reach-while-avoid specifications consider the case where $\hat{a} \in \mathcal{A}$ in the state space should be avoided. Then the dynamic program to be solved is (5.1), where $\delta_1 = 1$, $\delta_2 = 0$, and $\delta_3 = V_1$. When the algorithm converges, the resulting solution is $\mathbb{P}_{\psi_{\min}}$ and $\mathbb{P}_{\psi_{\max}}$ for each bound. It is worth noting that the synthesis of a reachability specification is equivalent to a reach-while-avoid specification where the avoid region $\mathcal{A} = \emptyset$.

A serial pessimistic reach-while-avoid controller approach is outlined in Algorithm 4, in which the interval iteration occurs in Steps $6-7$ and Steps $17-18$. The first **while** loop iterates to synthesize a control policy π, and the second **while** loop uses π by fixing the input $\hat{u}$ for each state $\hat{x}_k$. Steps 3 and 14 show the reach-avoid specification where the avoid set is removed from the state space leaving only $\hat{x} \in \mathcal{S}$. As earlier mentioned, the algorithm remains equivalent for the simpler reachability specification when $\mathcal{A} = \emptyset$. Steps 11 and 22 store the minimal and maximal probabilities of satisfaction corresponding to the policy π. Compared with the safety algorithm, no complement is required when calculating those probabilities.

Remark 5.1. *IMPaCT can also support finite-horizon specifications within IMDP construction by replacing the **while** loop in Algorithms 3 and 4 with a **for** loop over a finite number of time intervals. If this scenario applies, we still calculate each bound separately, fixing the policy π for the second bound. In this case, interval iteration can be reduced to value iteration, enabling the solution of finite-horizon problems, albeit with the trade-off of sacrificing convergence guarantees* [HM18].

Remark 5.2. *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 might not converge. In this case, calculating the number of time intervals k for which each independent bound of the interval iteration algorithm converges to itself (with a small enough threshold) can be beneficial. The larger k value can then be employed by the value iteration algorithm as its time horizon to hopefully ascertain a converged solution, ensuring the correctness of convergence. IMPaCT offers detailed guidance on the required steps to follow in such cases, including the example* `ex_load_safe`.

6. Parallel Controller Synthesis with Interval Iteration

As offered in our approach, we can improve the computational efficiency of solving synthesis problems to acquire the resulting controller $\mathcal{C} = (\mathcal{S}, \pi, \mathbb{P}_{\psi_{\min}}, \mathbb{P}_{\psi_{\max}})$ by using parallel processing. The synthesis includes solving several dynamic programs and two interval iteration algorithms. We use the GNU linear programming kit [Mak08] for solving the linear programs.

Algorithm 5: *Parallel* (Pessimistic) Safety Controller Synthesis

```
Input: IMDP $\hat{\Sigma}$, stopping condition $\varepsilon$
1  $V_0 := \mathbf{0}_n,\ V_1 := \mathbf{1}_n,\ \pi := \mathbf{0}_n$;
2  while $\|V_1 - V_0\|_\infty > \varepsilon$ do
3      for $x \in \mathcal{S}$ in parallel do
4          Solve max LP (5.1) for $\hat{A}$ and $\hat{T}$;
5          $\pi(\hat{x}_k) := \operatorname*{argmin}_{\hat{u}\in\hat{U}} \max_{\hat{w}\in\hat{W}} \max_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k,\cdot,\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\cdot,\cdot)V_0(\hat{x}')\}$;
6          $V_0'(\hat{x}_k) := \min_{\hat{u}\in\hat{U}} \max_{\hat{w}\in\hat{W}} \max_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k,\cdot,\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\cdot,\cdot)V_0(\hat{x}')\}$;
7          $V_1'(\hat{x}_k) := \min_{\hat{u}\in\hat{U}} \max_{\hat{w}\in\hat{W}} \max_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k,\cdot,\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\cdot,\cdot)V_1(\hat{x}')\}$;
8      end
9      $V_0 := V_0',\ V_1 := V_1'$;
10 end
11 $\mathbb{P}_{\psi_{\min}} = \mathbf{1}_n - V_0$;
12 $V_0 := \mathbf{0}_n,\ V_1 := \mathbf{1}_n$;
13 while $\|V_1 - V_0\|_\infty > \varepsilon$ do
14     for $x \in \mathcal{S}$ in parallel do
15         Solve min LP (5.1) for $\hat{A}$ and $\hat{T}$;
16         fix $\hat{u}$ from $\pi(\hat{x}_k)$;
17         $V_0'(\hat{x}_k) := \max_{\hat{w}\in\hat{W}} \min_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k,\hat{u},\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\hat{u},\cdot)V_0(\hat{x}')\}$;
18         $V_1'(\hat{x}_k) := \max_{\hat{w}\in\hat{W}} \min_{\hat{A},\hat{T}} \{\hat{A}(\hat{x}_k,\hat{u},\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\hat{u},\cdot)V_1(\hat{x}')\}$;
19     end
20     $V_0 := V_0',\ V_1 := V_1'$;
21 end
22 $\mathbb{P}_{\psi_{\max}} = \mathbf{1}_n - V_1$;
Output: controller $\mathcal{C} = (\mathcal{S}, \pi, \mathbb{P}_{\psi_{\min}}, \mathbb{P}_{\psi_{\max}})$
```

6.1. **Implementations.** The performance of the safety, reachability, and reach-avoid specifications can be enhanced by running in parallel all of the nested **for** loops from Algorithms 3 and 4. These can be seen in the updated Algorithm 5 in Step 3, and Step 14 for safety specifications. Once more, it is crucial to highlight that we aim to synthesize the minimal $\hat{u}$ as we seek the input that minimizes the likelihood of reaching the region to avoid (*i.e.*, complement of the safe set). Algorithm 6 presents an updated parallel implementation

Algorithm 6: *Parallel* (Pessimistic) Reach-Avoid Controller Synthesis

```
Input: IMDP $\hat{\Sigma}$, stopping condition $\varepsilon$
1  $V_0 := \mathbf{0}_n,\ V_1 := \mathbf{1}_n,\ \pi := \mathbf{0}_n$;
2  while $\|V_1 - V_0\|_\infty > \varepsilon$ do
3      for $x \in \mathcal{S}$ in parallel do
4          Solve min LP (5.1) for $\hat{R}$ and $\hat{T}$;
5          $\pi(\hat{x}_k) := \operatorname{argmax}_{\hat{u}\in\hat{U}} \min_{\hat{w}\in\hat{W}} \min_{\hat{R},\hat{T}} \{\hat{R}(\hat{x}_k,\cdot,\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\cdot,\cdot)V_0(\hat{x}')\}$;
6          $V_0'(\hat{x}_k) := \max_{\hat{u}\in\hat{U}} \min_{\hat{w}\in\hat{W}} \min_{\hat{R},\hat{T}} \{\hat{R}(\hat{x}_k,\cdot,\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\cdot,\cdot)V_0(\hat{x}')\}$;
7          $V_1'(\hat{x}_k) := \max_{\hat{u}\in\hat{U}} \min_{\hat{w}\in\hat{W}} \min_{\hat{R},\hat{T}} \{\hat{R}(\hat{x}_k,\cdot,\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\cdot,\cdot)V_1(\hat{x}')\}$;
8      end
9      $V_0 := V_0',\ V_1 := V_1'$;
10 end
11 $\mathbb{P}_{\psi_{\min}} = V_0$;
12 $V_0 := \mathbf{0}_n,\ V_1 := \mathbf{1}_n$;
13 while $\|V_1 - V_0\|_\infty > \varepsilon$ do
14     for $x \in \mathcal{S}$ in parallel do
15         Solve max LP (5.1) for $\hat{R}$ and $\hat{T}$;
16         fix $\hat{u}$ from $\pi(\hat{x}_k)$;
17         $V_0'(\hat{x}_k) := \min_{\hat{w}\in\hat{W}} \max_{\hat{R},\hat{T}} \{\hat{R}(\hat{x}_k,\hat{u},\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\hat{u},\cdot)V_0(\hat{x}')\}$;
18         $V_1'(\hat{x}_k) := \min_{\hat{w}\in\hat{W}} \max_{\hat{R},\hat{T}} \{\hat{R}(\hat{x}_k,\hat{u},\cdot) + \sum_{\forall\hat{x}'\in\hat{X}} \hat{T}(\cdot|\hat{x}_k,\hat{u},\cdot)V_1(\hat{x}')\}$;
19     end
20     $V_0 := V_0',\ V_1 := V_1'$;
21 end
22 $\mathbb{P}_{\psi_{\max}} = V_1$;
Output: controller $\mathcal{C} = (\mathcal{S}, \pi, \mathbb{P}_{\psi_{\min}}, \mathbb{P}_{\psi_{\max}})$
```

of Algorithm 4 for reach-while-avoid specifications. In particular, Step 3 and Step 14 replace the previous **for** loops in parallel. We reiterate that when $\mathcal{A} = \emptyset$, the reach-avoid algorithm is reduced to a simpler reachability algorithm.

IMPaCT automatically detects whether to use reachability or reach-avoid specification algorithms based on whether an avoid region has been set by the user. Additionally 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.

```
void setStoppingCondition(double eps);
void infiniteHorizonReachController(bool IMDP_lower);
void infiniteHorizonSafeController(bool IMDP_lower);
void finiteHorizonReachController(bool IMDP_lower, size_t timeHorizon);
void finiteHorizonSafeController(bool IMDP_lower, size_t timeHorizon);
```

Listing 10. Synthesis Algorithms.

6.2. **Controller Synthesis over GPU.** IMPaCT enables the utilization of GPUs for the verification and synthesis purposes via the constructed IMCs/IMDPs. The implementation leverages the *sorting method* from [SVA06, Lemma 7] and provides the same guarantees as the LP solving approach described in Section 6.1. The primary computational advantage lies in the absence of external library functions such as GLPK. Consequently, lines 4 and 15 in Algorithm 5 and Algorithm 6 can be substituted with a sorting method to compute the feasible distribution. Since the algorithms can be implemented manually without relying on external functions, the GPU can be leveraged for these computations. Intuitively, the sorting method solves the LP from (5.1) by initially defining the feasible distribution as the minimum transition probabilities, employing $\hat{T}_{\min}(\hat{x}'|\hat{x},\hat{u},\hat{w})$, $\hat{A}_{\min}(\hat{x},\hat{u},\hat{w})$ and $\hat{R}_{\min}(\hat{x},\hat{u},\hat{w})$. If the sum of these probabilities is less than 1, then each value of the feasible distribution is sequentially updated until the sum equals 1. The probability of a single value can be increased until it reaches $\hat{T}_{\max}(\hat{x}'|\hat{x},\hat{u},\hat{w})$, $\hat{A}_{\min}(\hat{x},\hat{u},\hat{w})$ or $\hat{R}_{\min}(\hat{x},\hat{u},\hat{w})$, respectively. Subsequently, the next value in the ordering will be incremented until the total sum equals 1. The ordering is determined by the values δ_1, δ_2, and $\delta_3(\hat{x}')$, with the maximal LP employing a *descending* order and the minimal LP employing an *ascending* order.

```
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 11. Sorted 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.

Remark 6.1. *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 the LP solver.*

Remark 6.2. *The functions for synthesis along with the functions for saving and loading files, described in the next section, are located in the source file* `GPU_synthesis.cpp`*, which is included by the main source file* `IMDP.cpp`*. This separation has been intentionally implemented to prevent compilation errors when employing the GPU approach. For GPU utilization, the configuration file should be set to load all required matrices and vectors before invoking the sorted approach. The Makefile needs to be modified to ensure that the* `--acpp-targets` *flag specifies the GPU architecture, such as* `cuda:sm_80`*. Furthermore, in the Makefile, the reference to the file* `IMDP.cpp` *should be substituted with* `GPU_synthesis.cpp`*. For an illustration, we refer to the example* `ex_GPU`*.*

6.3. Verification Analysis via Interval Markov Chains (IMCs).

In striving for maximum user-friendliness, the tool also automatically distinguishes between verification tasks using an IMC and controller synthesis via an IMDP. In particular, for verification analysis, users only need to specify the absence of input parameters, denoted by `dim_u` = 0. Accordingly, functions like `infiniteHorizonReachController`, etc., will identify the lack of input and generate a lookup table represented as $\mathcal{C} := (\mathcal{S}, \mathbb{P}_{\psi_{\min}}, \mathbb{P}_{\psi_{\max}})$, although it will still be saved as `controller.h5`. In fact, the verification's lookup table provides both the minimum and maximum probabilities of fulfilling a desired property corresponding to each partition of the finite system.

7. Loading and Saving Files

We use HDF5 [The23] as the data format for saving and loading 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. In IMPaCT, the subsequent commands are employed to save data in HDF5 format:

```
void saveStateSpace();
void saveInputSpace();
void saveDisturbSpace();
void saveTargetSpace();
void saveAvoidSpace();
void saveMinTargetTransitionVector();
void saveMaxTargetTransitionVector();
void saveMinAvoidTransitionVector();
void saveMaxAvoidTransitionVector();
void saveMinTransitionMatrix();
void saveMaxTransitionMatrix();
void saveController();
```

Listing 12. Save functions.

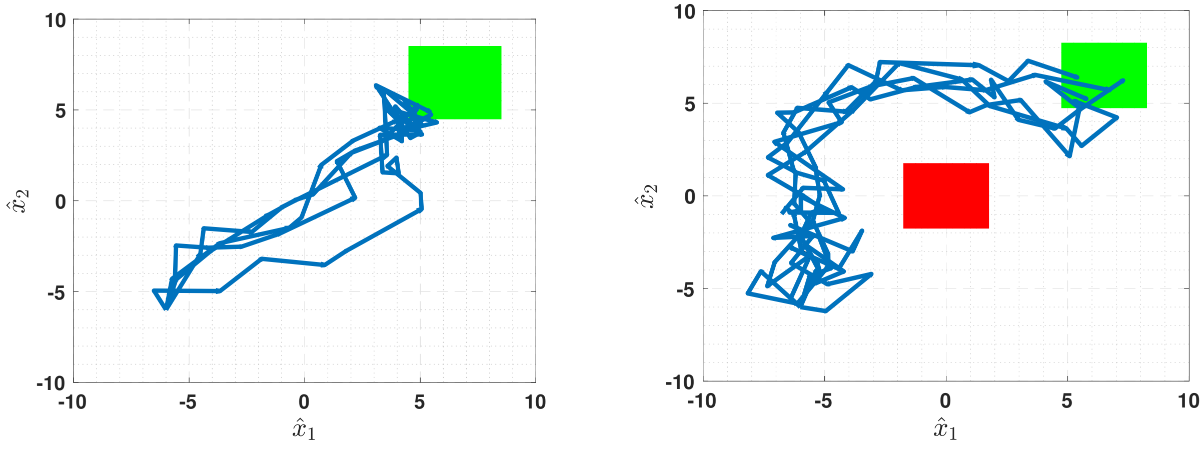
Furthermore, users have the flexibility to build the abstraction using their preferred methods and load the required matrices into IMPaCT for verification or controller synthesis tasks. For an illustration, we refer to the example `ex_load_reach`.

```
void loadStateSpace(string filename);
void loadInputSpace(string filename);
void loadDisturbSpace(string filename);
void loadTargetSpace(string filename);
void loadAvoidSpace(string filename);
void loadMinTargetTransitionVectorx(string filename);
void loadMaxTargetTransitionVector(string filename);
void loadMinAvoidTransitionVector(string filename);
void loadMaxAvoidTransitionVector(string filename);
void loadMinTransitionMatrix(string filename);
void loadMaxTransitionMatrix(string filename);
void loadController(string filename);
```

Listing 13. Load functions.

8. Benchmarking and Case Studies

We illustrate IMPaCT's applications by employing multiple well-known benchmark systems, 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. All the case studies are run on a high performance computer with 2 AMD EPYC 7702 64-Core along with 2TB RAM. In Table 1, we showcase the computational times for these case studies, all executed over an *infinite time horizon.* In Table 2, we provide a



(A) 2D Robot - Reachability

(B) 2D Robot - Reach-Avoid

Figure 1. 2D Robot case study fulfilling reachability and reach-avoid properties with different noise realizations. The green and red boxes are target and avoid regions, respectively.

comparison of various synthesis algorithms' performance on a standard desktop machine Linux, utilizing both CPU (12th Gen Intel Core i9-12900 × 24) and GPU (NVIDIA RTX A4000) configurations.

8.1. **2D Robot.** Consider a robot described by the following *nonlinear* difference equations:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} x_1(k) + 10u_1(k)\cos(u_2(k)) + w(k) + \varsigma_1(k) \\ x_2(k) + 10u_2(k)\sin(u_2(k)) + w(k) + \varsigma_2(k) \end{bmatrix},$$

where $(x_1, x_2) \in X := [-10, 10]^2$ is the spacial coordinate of the location of the robot, $(u_1, u_2) \in U := [-1, 1]^2$ is the input, and $w \in W := [-0.5, 0.5]$ is the disturbance. The noise $(\varsigma_1, \varsigma_2)$ has the covariance matrix $\mathsf{Cov} := \mathsf{diag}(0.75, 0.75)$. The discrete intervals are, respectively, $\eta_x = (0.5, 0.5)$, $\eta_u = (0.1, 0.1)$ and $\eta_w = 0.1$. The target set $\mathcal{T} := [5, 7]^2$ and the avoid set $\mathcal{A} := [-2, 2]^2$.

We consider four scenarios for this case study:

- Reach-avoid specification with no disturbance, see Fig. 1(B), *i.e.*, $\hat{w}(k) = 0$.
- Reach-avoid specification with disturbance.
- Reachability specification with no disturbance with updated $\eta_x = (1, 1)$ and $\eta_u = (0.2, 0.2)$, see Fig. 1(A).
- Reachability specification with disturbance with with updated $\eta_x = (1, 1)$, $\eta_u = (0.2, 0.2)$.

The latter two scenarios are designed for swift simulations on small personal computers.

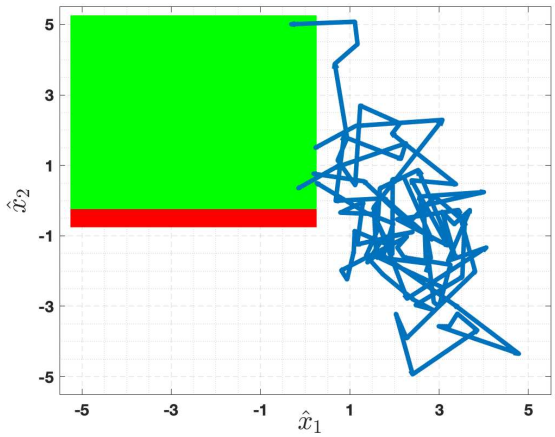


Figure 2. 3D Autonomous Vehicle case study fulfilling reach-while-avoid property, with different noise realizations, starting from an initial condition $[3; -3; 0.6]$. The green and red boxes are target and avoid regions, respectively.

8.2. **3D Autonomous Vehicle.** Consider a 3-dimensional autonomous vehicle described by the following *nonlinear* difference equations:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{bmatrix} = \begin{bmatrix} x_1(k) + (u_1(k)\cos(\alpha + x_3(k))\cos(\alpha)^{-1})T_s + \varsigma_1(k) \\ x_2(k) + (u_1(k)\sin(\alpha + x_3(k))\cos(\alpha)^{-1})T_s + \varsigma_2(k) \\ x_3(k) + (u_1(k)\tan(u_2(k)))T_s + \varsigma_3(k) \end{bmatrix},$$

where $\alpha = \arctan(\frac{\tan(u_2)}{2})$, and $T_s = 0.1$ is the sampling time. The inputs $(u_1, u_2) \in U := [-1, 4] \times [-0.4, 0.4]$ represent the wheel velocity and the steering angle, where $\eta_u = (1, 0.2)$. The states (x_1, x_2) are the spatial coordinates, and x_3 is the orientation, where $(x_1, x_2, x_3) \in X := [-5, 5]^2 \times [-3.4, 3.4]$ and $\eta_x = (0.5, 0.5, 0.4)$. The noise $(\varsigma_1, \varsigma_2, \varsigma_3)$ has covariance matrix $\mathsf{Cov} := \mathsf{diag}(\frac{2}{3}, \frac{2}{3}, \frac{2}{3})$. The target set $\mathcal{T}$ and avoid set $\mathcal{A}$ are described by the hyper-rectangles $[-5.75, 0.25] \times [-0.25, 5.75] \times [-3.45, 3.45]$ and $[-5.75, 0.25] \times [-0.75, -0.25] \times [-3.45, 3.45]$, respectively. For this case study, we consider a reach-while-avoid specification with no disturbances, see Fig. 2. We also examine a larger version of the same case study, incorporating $\eta_u = (0.5, 0.1)$ and $\eta_x = (0.5, 0.5, 0.2)$.

8.3. **3D and 5D Room Temperature Control Systems.** We also employ IMPaCT for the room temperature control systems taken from the ARCH competition [ABC+20], one with 3 dimensions and another one with 5 dimensions. The two case studies both consider two inputs described by $(u_1, u_2) \in U := [0, 1]^2$ with $\eta_u = (0.2, 0.2)$, and have a state space described by $(x_1, \ldots, x_n) \in X := [19, 21]^n$, where n is the state

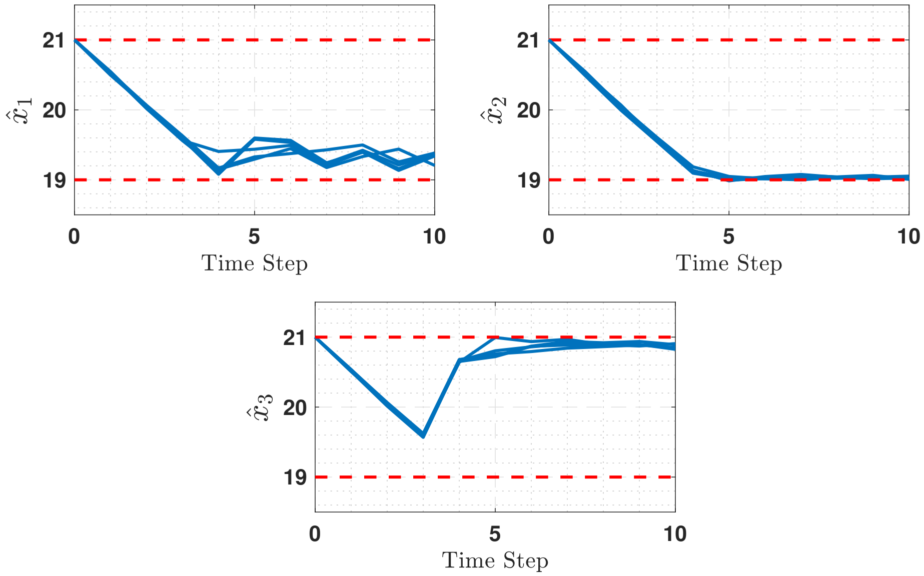


FIGURE 3. 3D Room Temperature fulfilling safety properties over 10 time steps, with different noise realizations, starting from an initial condition [21; 21; 21].

dimension. For the 3D case, the dynamics are described as:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{bmatrix} = \begin{bmatrix} (a-\gamma u_1(k))x_1(k) + \zeta(x_2(k)+x_3(k)) + \gamma T_h u_1(k) + \beta T_e + \varsigma_1(k) \\ a x_2(k) + \zeta(x_1(k)+x_3(k)) + \beta T_e + \varsigma_2(k) \\ (a-\gamma u_2(k))x_3(k) + \zeta(x_1(k)+x_2(k)) + \gamma T_h u_2(k) + \beta T_e + \varsigma_3(k) \end{bmatrix},$$

where $\beta = 0.022, \gamma = 0.05, \zeta = 0.2$, are the conduction factors, respectively, between the external environment and the current room, between the heater and the current room, and between neighbouring rooms. In addition, $T_h = 50°C$ is the heater temperature, $T_e = -1°C$ is the outside temperature, and $a = 1-2\zeta-\beta$. Furthermore, $\eta_x = (0.1, 0.1, 0.1)$ and the noise has covariance matrix $\mathsf{Cov} := \mathsf{diag}(0.02, 0.02, 0.02)$.

The dynamics for the 5D case study are characterized as:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \\ x_5(k+1) \end{bmatrix} = \begin{bmatrix} (a-\gamma u_1(k))x_1(k) + \zeta(x_2(k)+x_5(k)) + \gamma T_h u_1(k) + \beta T_e + \varsigma_1(k) \\ a x_2(k) + \zeta(x_1(k)+x_3(k)) + \beta T_e + \varsigma_2(k) \\ (a-\gamma u_2(k))x_3(k) + \zeta(x_4(k)+x_2(k)) + \gamma T_h u_2(k) + \beta T_e + \varsigma_3(k) \\ a x_4(k) + \zeta(x_3(k)+x_5(k)) + \beta T_e + \varsigma_4(k) \\ a x_5(k) + \zeta(x_4(k)+x_1(k)) + \beta T_e + \varsigma_5(k) \end{bmatrix},$$

where $\zeta = 0.3$ and $\eta_x = (0.4, 0.4, 0.4, 0.4, 0.4)$. The noise has covariance matrix $\mathsf{Cov} := \mathsf{diag}(0.01, 0.01, 0.01, 0.01, 0.01)$.

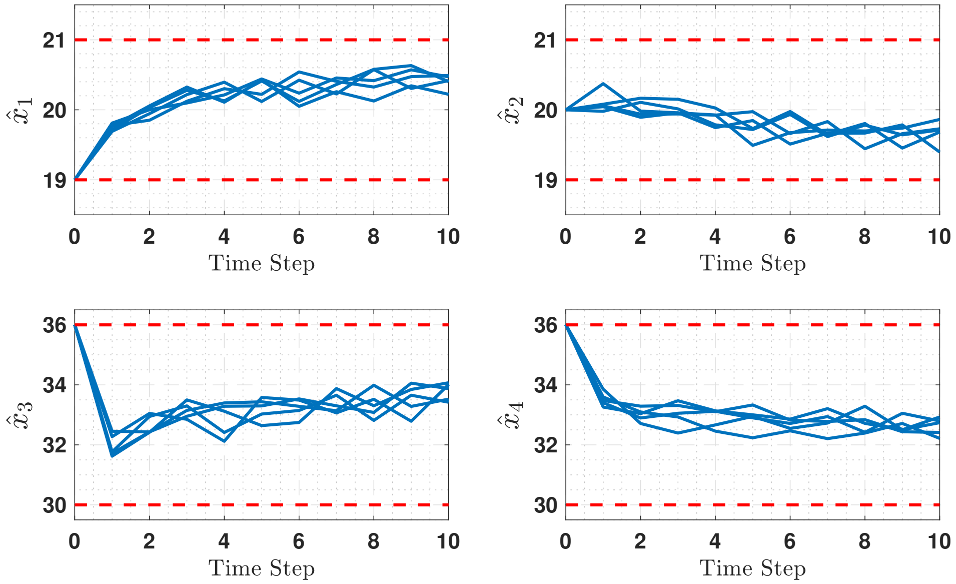


FIGURE 4. 4D Building Automation System fulfilling safety properties within 10 time steps, with 5 different noise realizations, starting from an initial condition $[19; 20; 36; 36]$.

Both case studies examine safety specifications where the safe region $\mathcal{S} = X$. Figure 3 showcases a simulation of the safety controller spanning 10 time steps, while Table 1 presents the synthesis times for this case study within an infinite time horizon.

8.4. **4D and 7D Building Automation Systems.** The following benchmarks are based on a smart building at the University of Oxford [CA18]. We consider two versions of this benchmark with different dimensions. The first one involves a safety synthesis problem rooted in a two-zone configuration, featuring stochastic dynamics:

$$x(k+1) = \begin{bmatrix} 0.6682 & 0 & 0.02632 & 0 \\ 0 & 0.683 & 0 & 0.02096 \\ 1.0005 & 0 & -0.000499 & 0 \\ 0 & 0.8004 & 0 & 0.1996 \end{bmatrix} x(k) + \begin{bmatrix} 0.1320 \\ 0.1402 \\ 0 \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} 3.4378 \\ 2.9272 \\ 13.0207 \\ 10.4166 \end{bmatrix} + \varsigma(k),$$

where the covariance matrix $\mathsf{Cov} := \mathsf{diag}(12.9199, 12.9199, 2.5826, 3.2279)$ and the off-diagonal elements are zero. Moreover, $x \in X := [19,21]^2 \times [30,36]^2$ with $\eta_x := (0.5, 0.5, 1.0, 1.0)$, and $u \in U := [17, 20]$ with $\eta_u := (1.0, 1.0)$, see Fig. 4.

The second version is a safety *verification* problem (*i.e.,* without input variables), in which the dynamics extend the first benchmark to consider a larger number of continuous variables:

$$x(k+1) = \begin{bmatrix} 0.968 & 0 & 0.004 & 0 & 0.004 & 0 & 0.004 \\ 0 & 0.968 & 0 & 0.003 & 0 & 0.003 & 0.003 \\ 0.011 & 0 & 0.949 & 0 & 0 & 0 & 0 \\ 0 & 0.010 & 0 & 0.952 & 0 & 0 & 0 \\ 0.011 & 0 & 0 & 0 & 0.949 & 0 & 0 \\ 0 & 0.010 & 0 & 0 & 0 & 0.952 & 0 \\ 0.011 & 0.010 & 0 & 0 & 0 & 0 & 0.979 \end{bmatrix} x(k) + \varsigma(k),$$

where the noise has covariance $\mathsf{Cov} := \mathsf{diag}(51.3, 50.0, 21.8, 23.5, 25.2, 26.5, 91.7)$, with off-diagonals equal to zero. In addition, $x(k) \in X := [-0.5, 0.5]^7$ with $\eta_x := (0.05, 0.05, 0.5, 0.5, 0.5, 0.5, 0.5)$.

It is noteworthy that, as indicated in Table 1, the construction of the IMDP abstraction for the 7D building automation systems was accomplished within 24 hours. However, the synthesis process extended beyond 24 hours to complete. This prolonged duration is primarily attributed to the expansive size of the state space, comprising $107,163$ finite states, while each state functions as a decision variable within the dynamic program. Considering that the synthesis involves solving the dynamic program for each row, the number of decision variables in one iteration amounts to $107,163^2$, *i.e.,* approximately 11.5 billion decision variables.

8.5. **14D Case Study.** To demonstrate the scalability of IMPaCT, we consider a 14D case study, borrowed from the relevant literature [CA19, LKSZ20], with the following dynamics:

$$x(k+1) = 0.8x(k) + 0.2\varsigma(k), \tag{8.1}$$

where $x = [x_1; \ldots; x_{14}], \varsigma = [\varsigma_1; \ldots; \varsigma_{14}]$, with $X := [-0.5, 0.5]^{14}$. We consider this case study to solve a safety verification problem, as detailed in Table 1.

8.6. **Comparison with StocHy.** As previously highlighted in the introduction, StocHy utilizes the value iteration algorithm for IMDP construction, which *lacks convergence guarantees* when dealing with 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 28.1 minutes, as detailed in Table 1, StocHy failed to complete task within a 24-hour timeframe.

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 all hardware vendors, including Intel and NVIDIA. We have benchmarked IMPaCT across various physical case studies including a 2D robot, a 3D autonomous vehicle, a 5D room temperature control system, and a 7D building automation system, borrowed from the ARCH tool competition, with its scalability further highlighted through a 14D case study.

Acknowledgment

The authors express their gratitude to Sadegh Soudjani for valuable discussions at the early stage of this project. Additionally, the authors would like to acknowledge Tobias Kaufmann in the Max Planck Institute for granting access to the high-performance computer utilized for simulations in Table 1.

References

[ABC$^+$20] Alessandro Abate, Henk Blom, Nathalie Cauchi, Joanna Delicaris, Arnd Hartmanns, Mahmoud Khaled, Abolfazl Lavaei, Carina Pilch, Anne Remke, Stefan Schupp, Fedor Shmarov, Sadegh Soudjani, Abraham Vinod, Ben Wooding, Majid Zamani, and Paolo Zuliani. ARCH-COMP20 Category Report: Stochastic Models. 2020.

[ABC$^+$23] A. Abate, H. Blom, N. Cauchi, J. Delicaris, S. Haesaert, B. van Huijgevoort, A. Lavaei, A. Remke, O. Schön, S. Schupp, et al. ARCH-COMP23 Category report: stochastic models. In *International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH), EPiC Series in Computing*, pages 126–150. EasyChair, 2023.

[ABD$^+$22] A. Abate, H. Blom, J. Delicaris, S. Haesaert, A. Hartmanns, B. van Huijgevoort, A. Lavaei, H. Ma, M. Niehage, A. Remke, et al. ARCH-COMP22 category report: stochastic models. 90:113–141, 2022.

[AH20] A. Alpay and V. Heuveline. SYCL beyond OpenCL: The Architecture, Current State and Future Direction of HipSYCL. In *Proceedings of the International Workshop on OpenCL*, 2020.

[AH23] A. Alpay and V. Heuveline. 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.

[APLS08] A. Abate, M. Prandini, J. Lygeros, and S. Sastry. Probabilistic reachability and safety for controlled discrete-time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008.

[BK08] C. Baier and J.P. Katoen. *Principles of model checking.* MIT Press, 2008.

[BKL$^+$17] C.l Baier, J. Klein, L. Leuschner, D. Parker, and S. Wunderlich. Ensuring the reliability of your model checker: Interval iteration for Markov decision processes. In *International Conference on Computer Aided Verification*, pages 160–180, 2017.

[BYG17] C. Belta, B. Yordanov, and E. A. Gol. *Formal methods for discrete-time dynamical systems*, volume 89. 2017.

[CA18] N. Cauchi and A. Abate. Benchmarks for cyber-physical systems: A modular model library for building automation systems. *IFAC-PapersOnLine*, 51(16):49–54, 2018.

[CA19] N. Cauchi and A. Abate. 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.

[CHK13] T. Chen, T. Han, and M. Kwiatkowska. On the complexity of model checking interval-valued discrete time Markov chains. *Information Processing Letters*, 113(7):210–216, 2013.

[DC20] M. Dutreix and S. Coogan. Specification-guided verification and abstraction refinement of mixed monotone stochastic systems. *IEEE Transactions on Automatic Control*, 66(7):2975–2990, 2020.

[DLML23] G. Delimpaltadakis, M. Lahijanian, M. Mazo, and L. Laurenti. Interval Markov Decision Processes with Continuous Action-Spaces. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2023.

[GLD00] R. Givan, S. Leach, and T. Dean. Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122(1):71–109, 2000.

[Gou09] B. Gough. *GNU scientific library reference manual*. Network Theory Ltd., 2009.

[HH14] A. Hartmanns and H. Hermanns. Modest Toolset: An integrated environment for quantitative modelling and verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 593–598, 2014.

[HHHK13] E. M. Hahn, A. Hartmanns, H. Hermanns, and J.-P. Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2013.

[HHS$^+$16] V. Hashemi, H. Hermanns, K. Song, L.and Subramani, A. Turrini, and P. Wojciechowski. Compositional bisimulation minimization for interval Markov decision processes. In *10th International Conference on Language and Automata Theory and Applications*, pages 114–126, 2016.

[HM14] S. Haddad and B. Monmege. Reachability in MDPs: Refining convergence of value iteration. In *8th International Workshop on Reachability Problems*, pages 125–137, 2014.

[HM18] S. Haddad and B. Monmege. Interval iteration algorithm for MDPs and IMDPs. *Theoretical Computer Science*, 735:111–131, 2018.

[Joh07] S. G. Johnson. The NLopt nonlinear-optimization package. `https://github.com/stevengj/nlopt`, 2007.

[JP09] A. A. Julius and G. J. Pappas. Approximations of stochastic hybrid systems. *IEEE Transactions on Automatic Control*, 54(6):1193–1203, 2009.

[JZC22] J. Jiang, Y. Zhao, and S. Coogan. Safe Learning for Uncertainty-Aware Planning via Interval MDP Abstraction. *IEEE Control Systems Letters*, 6:2641–2646, 2022.

[Kal21] O. Kallenberg. *Foundations of Modern Probability*, volume 3. Springer, 2021.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *23rd International Conference on Computer Aided Verification*, pages 585–591, 2011.

[KZ19] M. Khaled and M. Zamani. PFaces: An Acceleration Ecosystem for Symbolic Control. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 252–257, 2019.

[LAB15] M. Lahijanian, S. B. Andersson, and C. Belta. Formal verification and synthesis for discrete-time stochastic systems. *IEEE Transactions on Automatic Control*, 60(8):2031–2045, 2015.

[LF22] A. Lavaei and E. Frazzoli. Scalable synthesis of finite MDPs for large-scale stochastic switching systems. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 7510–7515, 2022.

[LKSZ20] A. Lavaei, M. Khaled, S. Soudjani, and M. Zamani. AMYTISS: Parallelized automated controller synthesis for large-scale stochastic systems. In *Proc. 32nd International Conference on Computer Aided Verification (CAV)*, LNCS, 2020.

[LSAZ22] A. Lavaei, S. Soudjani, A. Abate, and M. Zamani. Automated verification and synthesis of stochastic hybrid systems: A survey. *Automatica*, 146, 2022.

[LSFZ22] A. Lavaei, S. Soudjani, E. Frazzoli, and M. Zamani. Constructing MDP abstractions using data with formal guarantees. *IEEE Control Systems Letters*, 7:460–465, 2022.

[LSZ18] A. Lavaei, S. Soudjani, and M. Zamani. 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*, pages 21–30, 2018.

[Mak08] A. Makhorin. GLPK (GNU linear programming kit). *http://www. gnu. org/s/glpk/glpk. html*, 2008.

[NB99] Mark EJ Newman and Gerard T Barkema. *Monte Carlo methods in statistical physics*. Clarendon Press, 1999.

[Nej23] A. Nejati. *Formal Verification and Control of Stochastic Hybrid Systems: Model-based and Data-driven Techniques*. PhD thesis, Department of Electrical Engineering, Technische Universität München, 2023.

[NSZ21] A. Nejati, S. Soudjani, and M. Zamani. Compositional abstraction-based synthesis for continuous-time stochastic hybrid systems. *European Journal of Control*, 57:82–94, 2021.

[RAM23] L. Rickard, A. Abate, and K. Margellos. Learning robust policies for uncertain parametric markov decision processes. *arXiv: 2312.06344*, 2023.

[Row90] T. Harvey Rowan. *Functional stability analysis of numerical algorithms*. PhD thesis, Department of Computer Science, University of Texas at Austin, 1990.

[SC16] C. Sanderson and R. Curtin. Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016.

[SC18] C. Sanderson and R. Curtin. A user-friendly hybrid sparse matrix class in C++. In *6th International Conference on Mathematical Software*, pages 422–430, 2018.

[SGA15] S. Soudjani, C. Gevaerts, and A. Abate. FAUST^2: Formal abstractions of uncountable-state stochastic processes. In *TACAS'15*, volume 9035 of *Lecture Notes in Computer Science*, pages 272–286. 2015.

[SVA06] K. Sen, M. Viswanathan, and G. Agha. Model-checking Markov chains in the presence of uncertainties. In *12th International Conference Tools and Algorithms for the Construction and Analysis of Systems*, pages 394–410, 2006.

[SZ15] F. Shmarov and P. Zuliani. ProbReach: verified probabilistic delta-reachability for stochastic hybrid systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 134–139, 2015.

[Tab09] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[The23] The HDF Group. Hierarchical Data Format, version 5. https://www.hdfgroup.org/HDF5/, 1997-2023.

[TMKA13] I. Tkachev, A. Mereacre, Joost-Pieter Katoen, and A. Abate. 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*, pages 293–302, 2013.

[VGO19] A. P. Vinod, J. D. Gleason, and M. M. Oishi. SReachTools: A MATLAB stochastic reachability toolbox. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 33–38, 2019.

[VHSSH23] B. Van Huijgevoort, O. Schön, S. Soudjani, and S. Haesaert. SySCoRe: Synthesis via stochastic coupling relations. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2023.

[WMK19] M. Weininger, T. Meggendorfer, and J. Křetínskỳ. Satisfiability bounds for ω-regular properties in bounded-parameter Markov decision processes. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 2284–2291, 2019.

[WZK$^+$15] Q. Wang, P. Zuliani, S. Kong, S. Gao, and E. M. Clarke. SReach: A probabilistic bounded delta-reachability analyzer for stochastic hybrid systems. In *Proceedings of the International Conference on Computational Methods in Systems Biology*, pages 15–27, 2015.

[ZMEM$^+$14] M. Zamani, P. Mohajerin Esfahani, R. Majumdar, A. Abate, and J. Lygeros. Symbolic control of stochastic systems via approximately bisimilar finite abstractions. *IEEE Transactions on Automatic Control*, 59(12):3135–3150, 2014.

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 [SVA06, 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 5.2). 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.

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	1.9GB	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	8.8GB	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	97.2GB	7.46	22.0	274	21.2	22.6	274	5.66hr	8.46hr	0.08
3D Vehicle	R – A	7,938	30	0	238,140	89.1	114	7.42GB	44.6	46.6	2.91GB	4.53	4.97	264	3.69hr	286	0.61
3D Vehicle	R – A	15,435	99	0	1,528,065	1,004	1,340	92.6GB	534	551	36.3GB	51.8	46.5	3.3GB	>24hr	5,933	1.22
3D RoomTemp	S	9,261	36	0	333,396	1.51	78.5	24.7GB	-	-	-	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	6.37hr*	3,038*	0.07
5D RoomTemp	S	7,776	36	0	279,936	1.2	167.6	17.4GB	-	-	-	0.21	0.24	2.24	97.88*	111.5*	0.56
7D BAS	S	107,163	0	0	107,163	1.47hr	2.03hr	91.9GB	-	-	-	0.501	0.139	0.86	>24hr	>24hr	7.7
14D Case	S	16,384	0	0	16,384	699	987	2.15GB	-	-	-	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 (5.1) using GNU Linear Programming Kit (GLPK) against the sorting method from [SVA06, 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	>3.0hr	241	490
2D Robot	R – A	1,681	441	0	741,321	1691	577	216