This is a repository copy of *Evolutionary-Guided Synthesis of Verified Pareto-Optimal MDP Policies*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/178508/

Version: Accepted Version

Proceedings Paper:
Gerasimou, Simos orcid.org/0000-0002-2706-5272, Camara Moreno, Javier orcid.org/0000-0001-6717-4775, Calinescu, Radu orcid.org/0000-0002-2678-9260 et al. (3 more authors) (2022) Evolutionary-Guided Synthesis of Verified Pareto-Optimal MDP Policies. In: 36th IEEE/ACM International Conference on Automated Software Engineering. IEEE.

https://doi.org/10.1109/ASE51524.2021.9678727

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Evolutionary-Guided Synthesis of Verified Pareto-Optimal MDP Policies

Simos Gerasimou
Department of Computer Science
University of York, UK
simos.gerasimou@york.ac.uk

Javier Cámara
Department of Computer Science
University of York, UK
javier.camaramoreno@york.ac.uk

Radu Calinescu
Department of Computer Science
University of York, UK
radu.calinescu@york.ac.uk

Naif Alasmari
Department of Computer Science
University of York, UK
nnma500@york.ac.uk

Faisal Alhwikem
Department of Computer Science
University of York, UK
faisal.alhwikem@york.ac.uk

Xinwei Fang
Department of Computer Science
University of York, UK
xinwei.fang@york.ac.uk

***Abstract*—We present a new approach for synthesising Pareto-optimal Markov decision process (MDP) policies that satisfy complex combinations of quality-of-service (QoS) software requirements. These policies correspond to optimal designs or configurations of software systems, and are obtained by translating MDP models of these systems into parametric Markov chains, and using multi-objective genetic algorithms to synthesise Pareto-optimal parameter values that define the required MDP policies. We use case studies from the service-based systems and robotic control software domains to show that our MDP policy synthesis approach can handle a wide range of QoS requirement combinations unsupported by current probabilistic model checkers. Moreover, for requirement combinations supported by these model checkers, our approach generates better Pareto-optimal policy sets according to established quality metrics.**

I. Introduction

Markov decision processes (MDPs) provide a powerful mathematical framework for modelling and analysing sequential decision-making problems under uncertainty [1], [2]. Their ability to capture the complexity and uncertainty of modern software-intensive systems has led to numerous MDP applications for stochastic control and dynamic optimisation, in domains ranging from software product lines [3] and service-based systems [4] to self-adaptive systems [5] and robotics [6].

Software engineers can employ MDPs both during system design to analyse different system architectures [4], [7] and at runtime to support system reconfiguration [5], [8]. Consider a service-based system whose operations can be performed by alternative combinations of functionally equivalent third-party services that operate with different reliability, response time and cost. Modelling this service orchestration problem as an MDP allows engineers to analyse how the use of different service combinations affects the quality attributes of the system. The solution to the MDP is a policy that determines which concrete services should be selected so that a given objective, such as the overall system reliability or operational cost, is optimised. Given the MDP representation of such a system and a temporal logic specification [9] that formally defines the objective to be optimised, probabilistic model checkers like PRISM [10] and Storm [11] can automatically synthesise an optimal policy for the specification.

Software systems often require the simultaneous optimisation of multiple objectives whilst also satisfying a set of strict constraints. In a service-based system, software engineers may be interested in policies corresponding to services orchestration that minimise the system operation cost and response time, subject to keeping the system reliability above a critical threshold. This is an instance of a multi-objective optimisation problem [12]. In software-intensive systems, these objectives are typically conflicting, e.g., a more reliable or responsive service tends to be more expensive. As such, the MDP policy synthesis needs to generate *Pareto-optimal policy sets*, i.e., sets of policies that (i) satisfy all constraints, and (ii) for which no policy exists that also satisfies the system constraints and achieves better values for all the optimisation objectives [13].

Executing multi-objective model checking on MDPs for the synthesis of Pareto-optimal policies is an important and non-trivial problem [14]. Despite recent advances [13], [15], [16], [17], [18], existing approaches either use simple iterative methods, or rely on reductions and simplifications to solve the problem using linear programming. This limits their applicability to (i) single-objective problems with multiple strict constraints (for which a single best policy exists); or (ii) unconstrained problems with up to three optimisation objectives. Accordingly, these approaches support only a small fragment of the multi-objective MDP model checking spectrum, and cannot synthesise Pareto-optimal policies for many practical problems encountered, for instance, in software product lines [3], [19].

Our paper introduces EvoPoli, an approach that supports the synthesis of Pareto-optimal policies for MDPs with arbitrary combinations of constraints and optimisation objectives. EvoPoli uses evolutionary algorithms [12] to synthesise policies that cover sufficiently the policy space enabling decision-makers to obtain a holistic view of the tradeoffs between the policies in the objective space and make an informed decision. The crux of our approach is to cast the synthesis of Pareto-

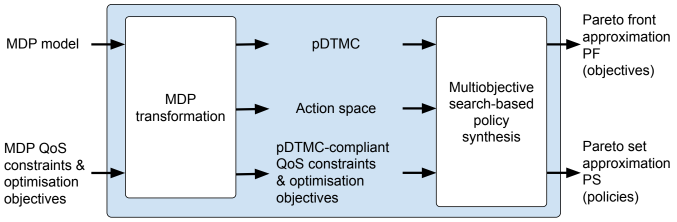

MDP model
MDP QoS constraints & optimisation objectives
MDP transformation
pDTMC
Action space
pDTMC-compliant QoS constraints & optimisation objectives
Multiobjective search-based policy synthesis
Pareto front approximation PF (objectives)
Pareto set approximation PS (policies)

Fig. 1: EvoPoli high-level workflow.

optimal policies for MDPs as a multi-objective search-based problem [20] and leverage the power of evolutionary algorithms [12] to compute the required Pareto-optimal policies.

As shown in Figure 1, EvoPoli takes as inputs an MDP model and a set of quality-of-service (QoS) constraints and optimisation objectives formally defined in probabilistic computational tree logic (PCTL) [9]. Through an MDP analysis and transformation step, EvoPoli produces a parametric discrete-time Markov chain (pDTMC) in which the model parameters encode the actions of the original MDP, and extracts the action space, i.e., the set of possible actions modelled in the MDP. During this step, EvoPoli also converts the constraints and optimisation objectives into equivalent PCTL specifications that comply with the pDTMC representation. Next, EvoPoli executes a multi-objective search-based policy synthesis procedure that successively evolves a population of candidate policies until a termination criterion is met (either the search budget is exhausted or no improvement occurs over a specified number of evolution rounds). The result is an approximate Pareto optimal set of policies, along with the associated approximate *Pareto front* of QoS attribute values.

The main contributions of our paper are:

- The EvoPoli approach for the synthesis of Pareto-optimal policies that extends the multi-objective model checking on MDPs to a much broader spectrum of QoS software requirement combinations than currently possible;
- An extensive EvoPoli evaluation on several variants of two MDPs modelling real-world problems, for a wide variety of constraints and optimisation objectives. Our experiments show that EvoPoli can handle multiple QoS requirement combinations unsupported by current probabilistic model checkers. Moreover, for requirement combinations supported by these model checkers, EvoPoli produces much better Pareto-optimal policy sets according to established quality indicators [21] and statistical analyses [22].
- A prototype open-source EvoPoli tool and case study repository, both available from our project web page at https://github.com/gerasimou/MDPSynthesis.

II. Preliminaries

A. Discrete-time Markov Chains

Definition 1 (Discrete-time Markov chain). *A discrete-time Markov chain (DTMC) over a set of atomic propositions AP is a tuple $\mathcal{D} = (S, s_I, P, L, R)$, where $S \neq \emptyset$ is a finite set of states; $s_I \in S$ is the initial state; $P : S \times S \rightarrow [0, 1]$ is a transition probability matrix such that, for any states $s, s' \in S, P(s, s')$ gives the probability of transitioning from s to s', and $\sum_{s' \in S} P(s, s') = 1$ for any $s \in S$; $L : S \rightarrow 2^{AP}$ is a labelling function that maps every state $s \in S$ to the atomic propositions from AP that hold in that state; and R is a (possibly empty) set of functions $\rho : S \rightarrow \mathbb{R}_{\geq 0}$ that associate non-negative values with the DTMC states.*

A parametric DTMC (pDTMC) is a discrete-time Markov chain whose transition probabilities $P(s, s')$ are specified as rational functions over a set of parameters [23], [24], [25].

B. Markov Decision Processes

Markov decision processes generalise DTMCs with the ability to model nondeterminism.

Definition 2 (Markov decision process). *A Markov decision process (MDP) over a set of atomic propositions AP is a tuple $\mathcal{M} = (S, s_I, A, \Delta, L, R)$, where S, s_I, L and R are defined as for a DTMC; $A \neq \emptyset$ is a finite set of actions; and $\Delta : S \times A \rightarrow Dist(S)$ is a partial probabilistic transition function that maps state-action pairs to discrete probability distributions over S.*

In each state $s \in S$, the set of actions $a \in A$ for which $\Delta(s, a)$ is defined contains the actions *enabled* in state s, and is denoted by $A(s)$. The choice of which action from $A(s)$ to take in every state s is assumed to be nondeterministic. We reason about the behaviour of MDPs using policies. A policy resolves the nondeterministic choices of an MDP, selecting the action taken in every state. MDP policies can be classified into infinite-memory, finite-memory and memoryless policies (depending on whether the action selected in a state depends on all, a finite number, or none of the previously visited states and on the actions selected in those states). Our work, and probabilistic model checkers such as PRISM and Storm, consider memoryless policies. Memoryless policies can be further classified into deterministic (when the same action is selected each time when a state is reached) and randomised (when the action selected in a state is given by a discrete probability distribution over the feasible actions). In this work, we use deterministic memoryless policies (called simply 'policies' in the rest of the paper).

Definition 3 (MDP policy). *A (deterministic memoryless) policy of an MDP is a function $\sigma : S \rightarrow A$ that maps each state $s \in S$ to an action from $A(s)$.*

C. Probabilistic Computation Tree Logic

Probabilistic computation tree logic (PCTL) [9], [26] is used to quantify properties related to probabilities and rewards in system specifications modelled by DTMCs and MDPs.

Definition 4 (PCTL formulae). State PCTL formulae Φ *and* path PCTL formulae Ψ *over an atomic proposition set AP are defined by the grammar:*

$$\begin{aligned} \Phi &::= true \mid \alpha \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\sim p}[\Psi] \mid \mathcal{R}_{\sim r}[C^{\leq k}] \mid \mathcal{R}_{\sim r}[F\Phi] \\ \Psi &::= X\Phi \mid \Phi \text{ U } \Phi \mid \Phi \text{ U}^{\leq k} \Phi \end{aligned} \tag{1}$$

TABLE I: Quality requirements for TAS

ID	Type	Description	PCTL
R1	Constraint	Workflow executions must succeed with probability at least 95%	$\mathsf{P}_{\geq 0.95}[\mathsf{F\ wOK}]$
R2	Objective	Minimise the average response time	$\mathsf{R}^{\mathsf{time}}_{\mathsf{min}=?}[\mathsf{C}]$
R3	Objective	Minimise the average operation cost	$\mathsf{R}^{\mathsf{cost}}_{\mathsf{min}=?}[\mathsf{C}]$

where $\alpha \in AP$ *is an atomic proposition,* $\sim \in \{\geq, >, <, \leq\}$ *is a relational operator,* $p \in [0,1]$ *is a probability bound,* $r \in \mathbb{R}_0^+$ *is a reward bound, and* $k \in \mathbb{N}_{>0}$ *is a timestep bound.*

The PCTL semantics is defined using a satisfaction relation $\models$ over the states S. Given a state s of an MDP $\mathcal{M}$, $s \models \Phi$ means "Φ holds in state s", and we have: always $s \models true$; $s \models \alpha$ iff $\alpha \in L(s)$; $s \models \neg\Phi$ iff $\neg(s \models \Phi)$; and $s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$. The *time-bounded until formula* $\Phi_1 \, \mathrm{U}^{\leq k} \, \Phi_2$ holds for a path iff Φ_1 holds in the first $i < k$ path states and Φ_2 holds in the $(i+1)$-th path state; and the *unbounded until formula* $\Phi_1 \, \mathrm{U} \, \Phi_2$ removes the bound k from the time-bounded until formula. The *next formula* $X\Phi$ holds if Φ is satisfied in the next state. The semantics of the probability $\mathcal{P}$ and reward $\mathcal{R}$ operators are defined over all policies σ of M as follows: $\mathcal{P}_{\sim p}[\Psi]$ specifies that the probability that paths starting at a chosen state s satisfy a path property Ψ is $\sim p$ for all policies; $\mathcal{R}_{\sim r}[C^{\leq k}]$ holds if the expected cumulated reward up to time-step k is $\sim r$ for all policies; and $\mathcal{R}_{\sim r}[F\Phi]$ holds if the expected reward cumulated before reaching a state satisfying Φ is $\sim r$ for all policies. Replacing $\sim p$ (or $\sim r$) from (1) with min $=?$ or max $=?$ specifies that the calculation of the minimum/maximum probability (or reward) over all MDP policies is required. For a full description of the PCTL semantics, see [9], [26].

III. Running Example

We illustrate EvoPoli on a service-based Tele Assistance System (TAS) introduced in [27]. The TAS continually tracks a patient's vital parameters, adapts the drug type or dose whenever needed, and takes action in case of emergency.

TAS combines three service types in a workflow (Figure 2). When the system receives a request that includes the patient's vital parameters, a *Medical Service* analyses the data and replies with instructions to (i) change the patient's drug type, (ii) change the drug dose, or (iii) trigger an alarm for first responders.When changing the drug type or dose, TAS notifies a local pharmacy using a *Drug Service*, and the alarm to notify the first responders is executed via an *Alarm Service*.

The functionality of each service type can be fulfilled by multiple service providers that offer functionally equivalent service implementations with different levels of reliability, performance, and cost. Reliability is given by the percentage of service failures over a predefined time period, performance is given by the service's mean response time, and cost is the price per service invocation.

At run time, the quality attributes of the services can vary, so TAS periodically reconfigures its workflow service bindings to select the combination of service implementations that optimises its operation, based on the requirements in Table I.

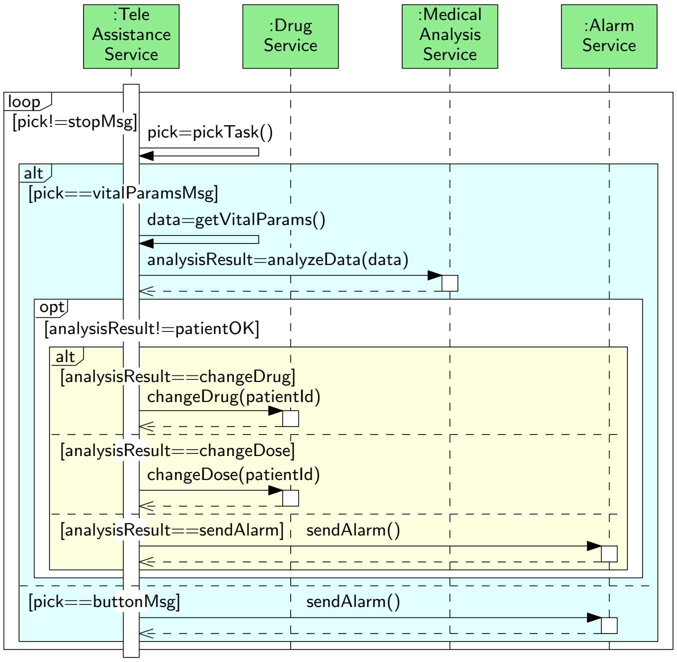
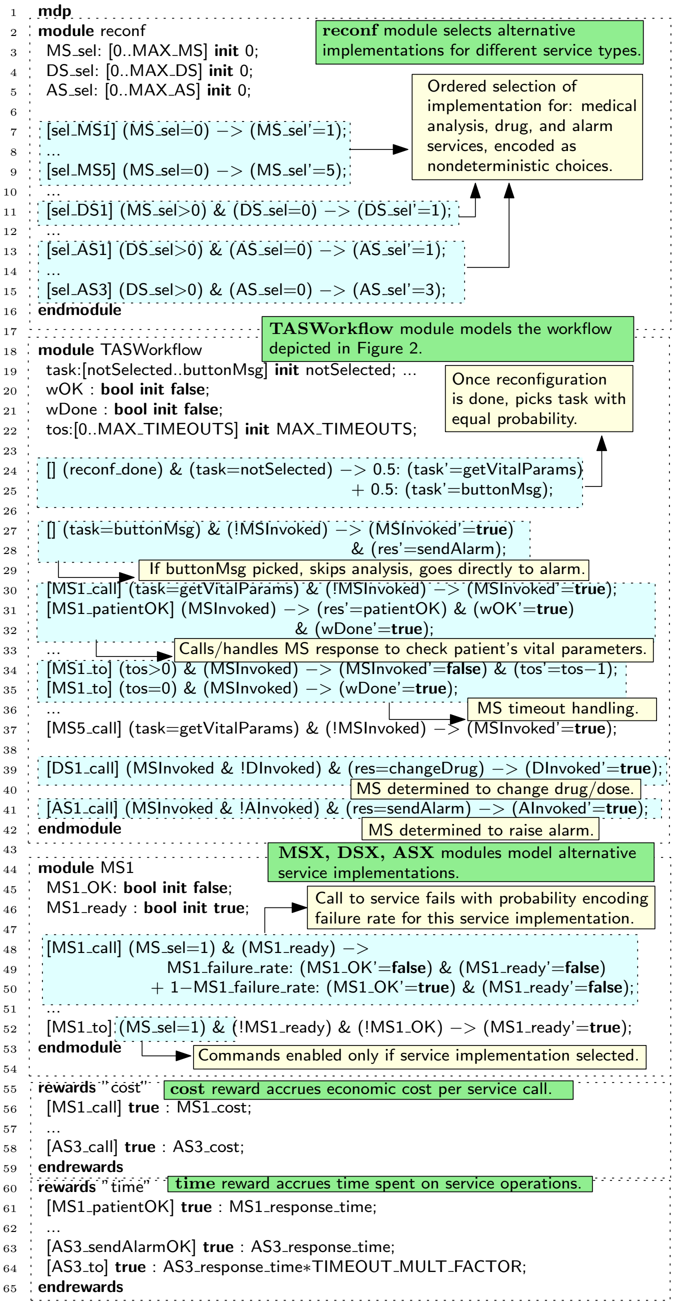


Fig. 2: TAS service workflow (adapted from [27]).

The reconfiguration decision can be cast as an MDP policy synthesis problem and modeled using high-level specification languages employed by commonly used probabilistic model checkers. Figure 3 illustrates the encoding of a TAS problem instance in Prism, which contains a reconfiguration module (`reconf`, lines 2-16) in charge of selecting the alternative service implementations (one per service type) at the start of the execution. Each of the implementation selections is underspecified in the model and encoded as a nondeterministic choice that will be resolved by the policy synthesis process (lines 7-15). Once reconfiguration is complete, the `TASWorkflow` module executes the workflow, communicating with the different service implementations selected via synchronous actions with shared labels (between "[]" in each command). If a service invocation fails, the workflow can handle timeouts by retrying calls (line 34). The number of retries is configurable via parameter `MAX_TIMEOUTS` (line 22). Due to space constraints, we only represent a subset of commands that bind workflow calls with alternative service implementations. Below the workflow module, the figure shows an excerpt of one of the modules that encode service implementations (medical analysis service `MS1`), which accrues cost and time rewards (lines 55-59, 60-65, respectively), whenever a synchronization with `TASWorkflow` actions occurs, e.g., `MS1_call` (lines 48, 30).

The problem instance presented here is deliberately small for illustration purposes. However, the solution space can grow exponentially as alternative service implementations are added, resulting in situations in which finding optimal policies for service selection cannot be achieved using exhaustive search.



```
mdp
module reconf
 MS_sel: [0..MAX_MS] init 0;
 DS_sel: [0..MAX_DS] init 0;
 AS_sel: [0..MAX_AS] init 0;

 [sel_MS1] (MS_sel=0) −> (MS_sel'=1);
 ...
 [sel_MS5] (MS_sel=0) −> (MS_sel'=5);
 ...
 [sel_DS1] (MS_sel>0) & (DS_sel=0) −> (DS_sel'=1);
 ...
 [sel_AS1] (DS_sel>0) & (AS_sel=0) −> (AS_sel'=1);
 ...
 [sel_AS3] (DS_sel>0) & (AS_sel=0) −> (AS_sel'=3);
endmodule

module TASWorkflow
 task:[notSelected..buttonMsg] init notSelected; ...
 wOK : bool init false;
 wDone : bool init false;
 tos:[0..MAX_TIMEOUTS] init MAX_TIMEOUTS;

 [] (reconf_done) & (task=notSelected) −> 0.5: (task'=getVitalParams)
                                        + 0.5: (task'=buttonMsg);

 [] (task=buttonMsg) & (!MSInvoked) −> (MSInvoked'=true)
                                        & (res'=sendAlarm);

 [MS1_call] (task=getVitalParams) & (!MSInvoked) −> (MSInvoked'=true);
 [MS1_patientOK] (MSInvoked) −> (res'=patientOK) & (wOK'=true)
                                  & (wDone'=true);
 ...
 [MS1_to] (tos>0) & (MSInvoked) −> (MSInvoked'=false) & (tos'=tos−1);
 [MS1_to] (tos=0) & (MSInvoked) −> (wDone'=true);
 ...
 [MS5_call] (task=getVitalParams) & (!MSInvoked) −> (MSInvoked'=true);

 [DS1_call] (MSInvoked & !DInvoked) & (res=changeDrug) −> (DInvoked'=true);

 [AS1_call] (MSInvoked & !AInvoked) & (res=sendAlarm) −> (AInvoked'=true);
endmodule

module MS1
 MS1_OK: bool init false;
 MS1_ready : bool init true;

 [MS1_call] (MS_sel=1) & (MS1_ready) −>
              MS1_failure_rate: (MS1_OK'=false) & (MS1_ready'=false)
            + 1−MS1_failure_rate: (MS1_OK'=true) & (MS1_ready'=false);
 ...
 [MS1_to] (MS_sel=1) & (!MS1_ready) & (!MS1_OK) −> (MS1_ready'=true);
endmodule

rewards "cost"
 [MS1_call] true : MS1_cost;
 ...
 [AS3_call] true : AS3_cost;
endrewards
rewards "time"
 [MS1_patientOK] true : MS1_response_time;
 ...
 [AS3_sendAlarmOK] true : AS3_response_time;
 [AS3_to] true : AS3_response_time∗TIMEOUT_MULT_FACTOR;
endrewards
```

reconf module selects alternative implementations for different service types.

Ordered selection of implementation for: medical analysis, drug, and alarm services, encoded as nondeterministic choices.

TASWorkflow module models the workflow depicted in Figure 2.

Once reconfiguration is done, picks task with equal probability.

If buttonMsg picked, skips analysis, goes directly to alarm.

Calls/handles MS response to check patient's vital parameters.

MS timeout handling.

MS determined to change drug/dose.

MS determined to raise alarm.

MSX, DSX, ASX modules model alternative service implementations.

Call to service fails with probability encoding failure rate for this service implementation.

Commands enabled only if service implementation selected.

cost reward accrues economic cost per service call.

time reward accrues time spent on service operations.

Fig. 3: MDP model of the Tele Assistance System [27] encoded in the high-level modelling language of PRISM [10].

IV. EvoPoli

A. Problem Definition

EvoPoli is applicable to systems whose behaviour can be modelled by MDPs, with the action set $A(s)$ from Definition 2 encoding the *choices* (e.g., of functionally equivalent services that can be invoked to perform an operation) available when the system state is modelled by state $s \in S$ of the MDP.

Definition 5 (Policy Decision Space)**.** *The policy decision space of an MDP $\mathcal{M} = (S, s_I, A, \Delta, L, R)$ is the set of all valid MDP policies, $DS = \{\sigma : S \to A \mid \sigma(s) \in A(s)\}$. The number of such policies is $\#DS = \prod_{s\in S} \#A(s)$.*

In line with the standard practice in the engineering of software-intensive systems [20], EvoPoli considers systems with $n_1 \geq 0$ *constraints* and $n_2 \geq 1$ *optimisation objectives*. A constraint specifies a bound for the acceptable values of a quality attribute, while an optimisation objective specifies whether a quality attribute should be maximised or minimised subject to satisfying all n_1 constraints.

Given an MDP $\mathcal{M} = (S, s_I, A, \Delta, L, R)$, $n_1 \geq 0$ PCTL-encoded *constraints* of the form

$$C_i ::= \mathsf{P}_{\sim p_i}[\cdot] \mid \mathsf{R}_{\sim r_i}[\cdot],\ 1 \leq i \leq n_1, \tag{2}$$

and $n_2 \geq 1$ PCTL-encoded *optimisation objectives* of the form

$$O_i ::= \mathsf{P}_{\max}[\cdot] \mid \mathsf{P}_{\min}[\cdot] \mid \mathsf{R}_{\max}[\cdot] \mid \mathsf{R}_{\min}[\cdot], 1 \leq i \leq n_2, \tag{3}$$

where '·' is a placeholder for the set of PCTL probability and reward properties supported by (1), the *constrained multi-objective policy synthesis problem* solved by EvoPoli is to find the Pareto-optimal set PS of MDP policies that satisfy the n_1 constraints and are Pareto-optimal with respect to the n_2 optimisation objectives. Formally,

$$PS = \{\sigma \in DS \mid \textstyle\bigwedge_{i=1}^{n_1} B\left(M, \sigma, C_i\right) \wedge (\nexists\, \sigma' \in DS \bullet \sigma' \prec \sigma)\} \tag{4}$$

where $B\left(M, \sigma, C_i\right) \in \mathbb{B}$ is $True$ if the constraint C_i is satisfied for the MDP model M and policy σ, and $False$ otherwise. The dominance relation $\prec$: $DS \times DS \to \mathbb{B}$, assuming minimisation of the optimisation objectives $O_1, O_2, \cdots, O_{n_2}$, is given by

$$\forall \sigma, \sigma' \in DS \bullet \sigma \prec \sigma' \equiv \forall 1 \leq i \leq n_2 \bullet Q\left(M, \sigma, O_i\right) \leq Q\left(M, \sigma', O_i\right) \wedge \exists 1 \leq i \leq n_2 \bullet Q\left(M, \sigma, O_i\right) < Q\left(M, \sigma', O_i\right) \tag{5}$$

where $Q\left(M, \sigma, O_i\right) \in \mathbb{R}$ denotes the value of the optimisation objective O_i for policy σ on model $\mathcal{M}$.

Finally, given the Pareto-optimal policies set PS, the Pareto-optimal front PF is defined by

$$PF = \{\left(Q\left(M, \sigma, O_1\right), \ldots, Q\left(M, \sigma, O_{n_2}\right)\right) \mid \sigma \in PS\}. \tag{6}$$

Example 1. *Requirements R1–R3 from Table I define a constrained multi-objective optimisation problem for the MDP modelling the TAS system from our running example (Figure 3), where n_1=1, C_1=R1, n_2=2, O_1=R2 and O_2=R3.*

Solving the *constrained multi-objective policy synthesis problem* to establish the set PS of Pareto-optimal policies (4) and the Pareto front PF (6) is complex and non-trivial [13]. Existing research [28], [18], [16], [17] can only solve simpler forms of this problem, i.e., those for which n_2=1 (i.e., *numerical queries*) or n_1=0 (i.e., *unconstrained Pareto queries*).

We explain next how our EvoPoli approach supports the synthesis of Pareto-optimal policies for an arbitrary number of constraints and optimisation objectives. Furthermore, through experiments detailed in Section VI, we illustrate how EvoPoli subsumes the policies produced by the current state-of-the-art techniques for the simpler problem variants they can solve.

B. MDP to pDTMC Transformation

To solve the constrained multi-objective policy synthesis problem for an MDP $\mathcal{M} = (S, s_I, A, \Delta, L, R)$, we construct a parametric DTMC $\mathcal{D}(\mathcal{M}) = (S, s_I, P, L, R)$ with the same state space, initial state, labelling function and reward function set as $\mathcal{M}$. For any pDTMC states $s, s' \in S$ with actions $A(s)=\{a_1, a_2, \ldots, a_n\}$ enabled in state s, the transition probability $P(s, s')$ is defined over a parameter $x(s) \in \{1, 2, \ldots, n\}$:

$$P(s, s') = \Delta\big(s, a_{x(s)}\big)(s'). \tag{7}$$

We use the shorthand notation $x : S \to \mathbb{N}$ to refer to all the parameters of this pDTMC. Next, we define n_2 pDTMC optimisation objectives $O'_1, O'_2, \ldots, O'_{n_2}$ analogous to the MDP optimisation objectives from (3) such that, if the i-th MDP optimisation objective is $\mathsf{P}_{\max=?}[\cdot]$, then O'_i is 'maximise $P_{=?}[\cdot]$', etc. The next result shows that solving the MDP policy synthesis problem from the previous section is equivalent to solving a similar problem for this pDTMC.

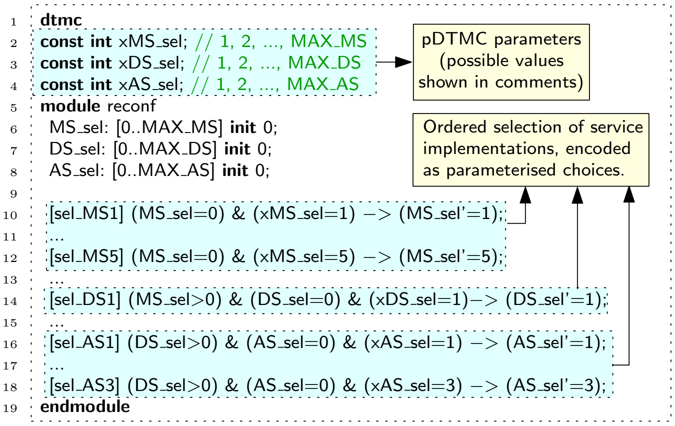
Theorem 1. *If PS' is the set of combinations of parameter values for which $\mathcal{D}(\mathcal{M})$ satisfies the constraints* (2) *and is Pareto-optimal with respect to the objectives O'_1, O'_2, $\ldots$, O'_{n_2}, the solution PS of the constrained multi-objective policy synthesis problem for the MDP $\mathcal{M}$ is given by*

$$\mathit{Policies}(PS') = \{\sigma : S \to A \mid \exists x \in PS' . (\forall s \in S . \sigma(s) = a_{x(s)})\}. \tag{8}$$

Proof. We prove the theorem by contradiction. First, suppose that $\mathit{Policies}(PS')$ contains a policy $\sigma \notin PS$, and let $x \in PS'$ be the combination of pDTMC parameter values associated with this policy. As $x \in PS'$, the $\mathcal{D}(\mathcal{M})$ instance associated with x satisfies the constraints (2). Also, according to (7), the $\mathcal{D}(\mathcal{M})$ instance associated with x and the MDP $\mathcal{M}$ under policy σ have identical transition probabilities, so $\mathcal{M}$ must also satisfy these constraints under policy σ. As such, $\sigma \notin PS \implies \exists \sigma' \in PS \,.\, \sigma' \prec \sigma$. Additionally, for all $1 \le i \le n_2$, $\mathcal{D}(\mathcal{M})$ instance associated with x and the MDP $\mathcal{M}$ under policy σ must yield the same values for the properties evaluated for the optimisation objectives O_i and O'_i, respectively.

Consider now the $\mathcal{D}(\mathcal{M})$ parameter combination x' that satisfies $\forall s \in S \,.\, \sigma'(s) = a_i \implies x'(s) = i$. As before, since $\sigma' \in PS$, both the MDP $\mathcal{M}$ under policy σ' and the $\mathcal{D}(\mathcal{M})$ instance associated with x' must satisfy the constraints (7) *and* must yield identical values for the properties evaluated for the optimisation objectives O_i and O'_i, respectively, for all $1 \le i \le n_2$. It follows that x' dominates x, and therefore $x \notin PS'$, which contradicts the assumption we started from. Accordingly, $\mathit{Policies}(PS') \setminus PS = \emptyset$. The same reasoning can be used to show that $PS \setminus \mathit{Policies}(PS') = \emptyset$, and therefore we must have $PS = \mathit{Policies}(PS')$. □

Example 2. *Figure 4 shows the result of applying the MDP to pDTMC transformation described above to the* `reconf` *module from the TAS system MDP in Figure 3. This pDTMC fragment shows how the nondeterministic choices from the MDP are replaced by choices parameterised by the three pDTMC parameters defined in lines 2–4.*



```
1  dtmc
2  const int xMS_sel; // 1, 2, ..., MAX_MS
3  const int xDS_sel; // 1, 2, ..., MAX_DS
4  const int xAS_sel; // 1, 2, ..., MAX_AS
5  module reconf
6   MS_sel: [0..MAX_MS] init 0;
7   DS_sel: [0..MAX_DS] init 0;
8   AS_sel: [0..MAX_AS] init 0;
9
10  [sel_MS1] (MS_sel=0) & (xMS_sel=1) −> (MS_sel'=1);
11  ...
12  [sel_MS5] (MS_sel=0) & (xMS_sel=5) −> (MS_sel'=5);
13  ...
14  [sel_DS1] (MS_sel>0) & (DS_sel=0) & (xDS_sel=1)−> (DS_sel'=1);
15  ...
16  [sel_AS1] (DS_sel>0) & (AS_sel=0) & (xAS_sel=1) −> (AS_sel'=1);
17  ...
18  [sel_AS3] (DS_sel>0) & (AS_sel=0) & (xAS_sel=3) −> (AS_sel'=3);
19 endmodule
```

pDTMC parameters (possible values shown in comments)
Ordered selection of service implementations, encoded as parameterised choices.

Fig. 4: pDTMC encoding of the reconf module from the TAS MDP in Figure 3.

C. Evolutionary-based Policy Synthesis

Using exhaustive analysis to solve the *constraint multi-objective synthesis problem* is unfeasible since the policy decision space DS (cf. Def 5) is typically extremely large. For instance, for the MDP model of our TAS running example from Section III $|DS| \approx 10^{65}$, while for the systems considered in our experimental evaluation $|DS| > 10^{1000}$. Clearly, enumerating and evaluating all possible policies is both time-consuming and computationally-prohibitive.

EvoPoli reformulates the policy synthesis problem as a search-based optimisation problem [20] and uses multi-objective genetic algorithms (MOGA) [12], like the widely-used NSGA-II [29] and SPEA2 [30] algorithms, to intelligently navigate the decision space. EvoPoli iteratively evolves a population of candidate policies to identify promising regions in the decision space and synthesise a close approximation of the Pareto-optimal policies set PS. EvoPoli encodes each candidate policy (i.e., solution) as a tuple of *genes*. Each state $s \in S$ for which the cardinality of its set of enabled actions $|A(s)| \ge 2$ is mapped to a gene. For any state s, the corresponding gene can take values from the set $\{1, 2, \cdots, |A(s)|\}$. We refer interested readers to [7], [31] for a detailed description of this encoding.

Algorithm 1 shows the high-level process underpinning EvoPoli for the synthesis of the Pareto-optimal policies set PS and the corresponding Pareto front set PF. Given as inputs the DTMC $D(M)$ induced by the MDP $\mathcal{M}$, the decision space DS, and the lists of constraints $(C_1, C_2, \cdots, C_{n_1})$ and optimisation objectives $(O_1, O_2, \cdots, O_{n_2})$, EvoPoli starts with empty PS and PF sets (line 2) and iteratively evolves them through the loop (lines 3-25) until a termination criterion is met. The function TERMINATE(PS, DS) holds when the maximum number of candidate policy evaluations has been carried out (i.e., budget exhausted), or when no new updates have been made in PS over a fixed number of successive iterations (i.e., the decision space has been explored sufficiently yielding diverse and Pareto-optimal poli-

Algorithm 1 Evolutionary-based Pareto Optimal Policy Synthesis

```
1: function SYNTHESIS ($D(M), DS, (C_i)_{1\leq i\leq n_1}, (O_i)_{1\leq i\leq n_2}$)
2:     $PS \leftarrow \emptyset, PF \leftarrow \emptyset$
3:     while $\neg$TERMINATE($PS, DS$) do
4:         $\mathcal{G} \leftarrow$ GENERATECANDIDATEPOLICIES($DS, PS$)
5:         for all $\sigma \in \mathcal{G}$ do
6:             $\{(C_{i,\sigma})_{1\leq i\leq n_1}, (O_{i,\sigma})_{1\leq i\leq n_2}\} \leftarrow$
               EVALUATEPOLICY($D(M), \sigma, (C_i)_{1\leq i\leq n_1}, (O_i)_{1\leq i\leq n_2}$)
7:             if $\wedge_{1\leq i\leq n_1}\{C_{i,\sigma}\}$ then
8:                 $dominated \leftarrow false$
9:                 for all $\sigma' \in PS$ do
10:                    if $\sigma \prec \sigma'$ then
11:                        $PS = PS \setminus \{\sigma'\}$
12:                        $PF = PF \setminus \{(Q(D(M), \sigma', O_i))_{1\leq i\leq n_2}\}$
13:                    else if $\sigma' \prec \sigma$ then
14:                        $dominated \leftarrow true$
15:                        break
16:                    end if
17:                end for
18:                if $\neg dominated$ then
19:                    $PS = PS \cup \{\sigma\}$
20:                    $PF = PF \cup \{(O_{i,\sigma})_{1\leq i\leq n_2}\}$
21:                end if
22:            end if
23:        end for
24:        $PS, PF \leftarrow$ DIVERSIFYPOLICIES($PS, PF$)
25:    end while
26:    return $PS, PF$
27: end function
```

cies). Within each iteration, EvoPoli initially employs the GENERATECANDIDATEPOLICIES function (line 4) to create a population $\mathcal{G}$ of plausible policies using MOGA-specific crossover and mutation operators. Crossover randomly chooses two fit policies from the current Pareto-optimal set PS and exchanges their genes to produce new policies. Mutation, on the other hand, creates a new policy by randomly changing a subset of the genes of a policy based on its value range encoded in the decision space DS. Next, the for loop (lines 5-23) evaluates each policy $\sigma \in \mathcal{G}$ and establishes its dominance relation (cf. Eq. 5) with respect to the policies in PS. To this end, the EVALUATEPOLICY function (line 6) uses a probabilistic model checker to determine the satisfaction condition of the n_1 constraints and obtain the values for the n_2 optimisation objectives. The policy σ and the objectives tuple are added to PS and PF, respectively, only if σ satisfies all constraints and is not dominated by any other policy in PS. Similarly, policies dominated by σ are removed from PS along with their associated objectives tuple (lines 7-22). The execution of DIVERSIFYPOLICIES (line 24) uses MOGA-specific mechanisms for diversity preservation to select policies from PS that will participate in the next iteration. These mechanisms maintain diversity in the population and generate a PF that covers sufficiently the objective space. For instance, the diversity mechanism used by NSGA-II [29] combines the non-domination level of each evaluated policy and a crowding distance metric, i.e., the population density in its area of the search space. Once the evolution terminates, the Pareto-optimal set approximation PS is returned along with the Pareto-optimal front approximation PF (line 26).

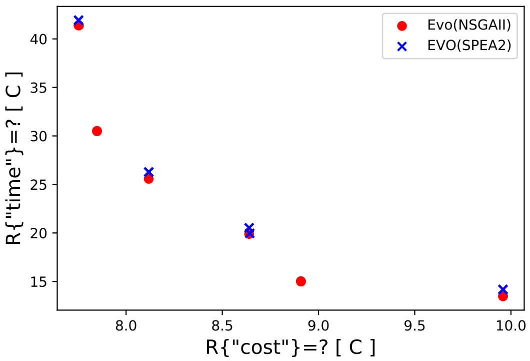


Fig. 5: Pareto front of policies for the TAS quality requirements from Table I synthesised using EvoPoli instrumented with NSGA-II [29] and SPEA2 [30].

Example 3. *Figure 5 shows two Pareto front PF sets obtained for our TAS running example using the quality requirements from Table I. As shown, the NSGA-II-instrumented EvoPoli produces more policies than its SPEA2-instrumented counterpart. Both MOGAs had the same experimental setup, i.e., 1000 evaluations and a population of 20. We should also highlight that neither PRISM [10] nor Storm [11] can produce a Pareto front for this combination of objectives and constraints.*

V. IMPLEMENTATION

To ease the evaluation and adoption of EvoPoli, we have implemented a prototype tool in Java that realises the high-level EvoPoli workflow from Figure 1. The MDP transformation component consumes an MDP model specified in the high-level modelling language of the PRISM model checker [10] and the PCTL-encoded constraints (2) and optimisation objectives (3), and applies the process described in Section IV-B to produce the pDTMC and the pDTMC-compliant constraints and optimisation objectives. We have developed the synthesis method from Algorithm 1 on top of the search-based software engineering tool EvoChecker [7], [31]. The open-source code of EvoPoli, the full experimental results summarised in the following section, additional information about EvoPoli and the case studies used for its evaluation are available at https://github.com/gerasimou/MDPSynthesis.

VI. EVALUATION

A. Research Questions

RQ1 (Validation): How does our approach perform compare to existing probabilistic model checkers? We analyse if our approach can synthesise policies of similar quality to those produced by the probabilistic model checkers PRISM [10] and Storm [11] for the simpler class of problems (i.e., unconstrained Pareto queries) that these model checkers can solve.

RQ2 (Effectiveness): How do EvoPoli instances instrumented with different MOGAs compare to each other? We used this research question to analyse the impact of different MOGAs in the performance of EvoPoli. To this end, we study

TABLE II: Quality requirements for Ocean Worlds

ID	Type	Description	PCTL
R1	Constraint	The robotic lander must complete its mission within 30 mins	$\mathsf{R}^{\mathsf{T}}_{\leq 30}[\mathsf{C}]$
R2	Objective	Maximise science value	$\mathsf{R}^{\mathsf{SV}}_{=\mathsf{max}?}[\mathsf{C}]$
R3	Objective	Maximise probability of success	$\mathsf{P}_{\mathsf{max}=?}[\mathsf{F\ done}]$
R4	Objective	Minimise energy consumption	$\mathsf{R}^{\mathsf{EC}}_{\mathsf{min}=?}[\mathsf{C}]$

TABLE III: System variants analysed using EvoPoli

Variant	Details	$\#DS$	T_{run}:mean($\pm SD$)
TAS2	MAX_Timeout=2	10^{50}	9647.21($\pm$601.56)
TAS3	MAX_Timeout=3	10^{67}	16827.73($\pm$598.35)
TAS4	MAX_Timeout=4	10^{84}	26519.87($\pm$1016.59)
OW4	`xloc`=4, `dloc`=4	10^{72}	668.57($\pm$31.15)
OW5	`xloc`=5, `dloc`=5	10^{98}	3756.21($\pm$230.38)
OW6	`xloc`=6, `dloc`=6	10^{138}	16362.04($\pm$688.61)

the quality of EvoPoli-synthesised policies when our approach uses the established MOGAs NSGA-II [29] and SPEA2 [32].

RQ3 (Decision support): Can EvoPoli provide useful insights into the trade-offs between the quality attributes values produced by different policies? To support decision making and help software engineers to make informed decisions, EvoPoli must synthesise policies with different trade-offs. Hence, we assessed the trade-offs in policies produced by EvoPoli for the software systems analysed in our evaluation.

B. Evaluation Methodology

Software Systems. We performed a wide range of experiments to evaluate EvoPoli using multiple variants of two software systems derived from different application domains: (1) the service-based Tele Assistance System (TAS) adapted from [27] and described in Section III; and (2) a prototype robotic planner software component for ocean world (OW) exploration [33] which we describe next.

Ocean Worlds (OW). The Ocean Worlds Autonomy Testbed for Exploration Research and Simulation project led by NASA Ames Research Center is developing an autonomy software testbed to spur the development of autonomy technologies for surface missions [33]. This testbed is conceived for missions in which a robotic lander collects and analyses samples, and then sends relevant data back to Earth. To complete the mission, the robot must choose among `xloc` alternative excavation locations each of which has an associated *science value* (a measurement of the potential interest of samples in that location) and an *excavatability risk* (signifying the safety and difficulty of excavating in that part of the terrain). For each successful excavation, the robot must choose where to dump the resulting rubble by selecting among `dloc` available dumping locations. Excavating and moving around the robot's arm consumes a corresponding amount of energy. Data is sent back to Earth during a specific time window for processing and further analysis. Table II shows the OW mission requirements.

The autonomy software on the robotic lander includes a facility to replace existing plans as the mission progresses, with updated plans coming from Earth or generated by automated planners on-board and/or on Earth. One of such planners employs MDP policy synthesis to make high-level decisions about excavation and dumping location selections, which are encoded as nondeterministic choices in a MDP model. For the excavation location selection, each alternative is encoded as a command in which a failure to excavate is associated with a probability that encodes the *excavatability risk*. Three reward structures capture the *science value*, *time*, and *energy consumption* associated with each selection. Due to space constraints we omit the full details of the MDP model of this system; we refer interested readers to our project webpage.

Experimental Setup. We performed a wide range of experiments using the TAS and OW system variants from Table III. The 'Details' column lists the values specified for the variables of each system variant, i.e., the maximum timeout (`MAX_Timeout`) for the TAS, and the number of excavation (`xloc`) and dumping (`dloc`) locations for the OW system. The column '$\#DS$' reports the search space of the policies according to Def. 5. Finally, the column T_{run} reports the average running time (and standard deviation in parenthesis) for completing a policy synthesis run per system variant.

We instrumented the evolutionary-based policy synthesis algorithm of EvoPoli using the established MOGAs NSGA-II [29] and SPEA2 [32]. We also used the following configuration to evaluate our approach: 5,000 evaluations with an initial population of 100 individuals (i.e., 50 generations in total), and default values for single-point crossover probability $p_c = 0.9$ and uniform polynomial mutation probability $p_m = 0.8$. We selected these values based on our experience in the field [7], [15], [31] and after performing a set of preliminary experiments.

To alleviate the potential impact of randomness in the performance and effectiveness of MOGAs (e.g., when choosing the crossover point, when sampling randomly to execute the mutation operation), we followed the established procedure in search-based software engineering [20]. We executed 30 independent runs per system variant from Table III and each multiobjective optimisation algorithm [22]. All the experiments were run on a CentOS Linux 6.5 64bit server with two 2.6GHz Intel Xeon E5-2670 processors and 32GB of memory.

Statistical analysis. For real-world systems such as those used in our experimental evaluation the policy decision space DS (Def. 5) is extremely large. Thus, producing the actual Pareto front is typically unfeasible. Aligned with the standard practice [21], for each system variant we produce the *reference front* comprising the nondominated policies from all the runs executed across all MOGA-based EvoPoli instances and the policies produced by the probabilistic model checkers Storm and PRISM (for the simple class of multi-objective policy synthesis problems that these models checkers can handle). We used this reference front and the widely-used Pareto-front quality indicators below to quantify the 'goodness of fit' of Pareto front approximations synthesised by EvoPoli instances, Storm and PRISM. For each quality indicator, we use a box plot to present its central tendency and distribution.

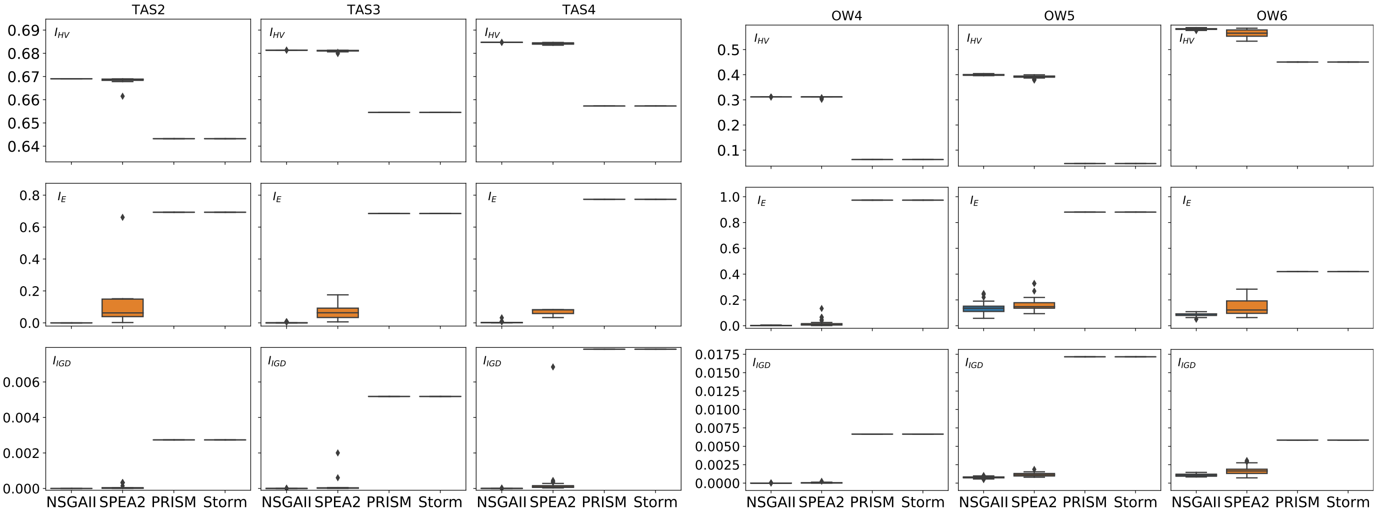


Fig. 6: Boxplots comparing EvoPoli (NSGA-II), EvoPoli (SPEA2), PRISM and Storm for the TAS (left) and OW (right) system variants and for unconstrained Pareto queries (i.e., $n_1\!=\!0, n_2\!=\!2$), evaluated using quality indicators I_{HV}, I_ϵ and I_{IGD}.

- The I_{HV} *(Hypervolume)* indicator uses a reference front and measures the volume in the objective space consumed by a Pareto front approximation. I_{HV} measures both diversity and convergence, and is strictly Pareto compliant[1] [21]. Better Pareto front approximations have larger I_{HV} values.
- The I_ϵ *(Unary additive epsilon)* denotes the minimum additive term needed to alter the objective vector from a Pareto front approximation to dominate the corresponding objective vector of the reference front. This indicator shows convergence to the reference front and is Pareto compliant. Smaller I_ϵ values mean better Pareto front approximations.
- The I_{IGD} *(Inverted Generational Distance)* indicator measures the Euclidean distance in the objective space between the reference front and the Pareto front approximation. I_{IGD} signifies an "error measure", and indicates both diversity and convergence to the reference front. Smaller I_{IGD} values signify better Pareto front approximations.

Following the recommended practice [22], we used inferential statistical tests to compare the quality indicator values obtained by EvoPoli instances and the values obtained by PRISM and Storm. We employed the Shapiro-Wilk test and confirmed that the quality indicator values do not follow a normal distribution. Thus, we used the Mann-Whitney and Kruskal-Wallis non-parametric tests with 95% confidence level ($\alpha = 0.05$) to analyse the results without making assumptions about the data distribution or the homogeneity of its variance. We also performed a post-hoc analysis with pairwise comparisons between the algorithms, using the conservative Bonferroni correction $p_{crit} = \alpha/k$ (k is the number of comparisons) to control the family-wise error rate.

When statistical significance exists, we use Cohen's d to quantify the importance of the observed effect [22]. Cohen's d score summarises the difference between two groups as the number of standard deviations with d=0.2, d=0.5 and d=0.8 denoting a small, medium and large effect size, respectively.

C. Results & Discussion

RQ1 (Validation). Since neither PRISM nor Storm can solve the constrained multi-objective policy synthesis problem from Section IV-A, we can ensure a fair comparison only by transforming the problem into an *unconstrained Pareto query* (i.e., $n_1 = 0, n_2 = 2$) that both model checkers can handle[2]. To achieve this, we removed constraint $R1$ from both systems and retained requirements $R2, R3$ (minimise response time, minimise cost) and $R2, R4$ (maximise science value, minimise energy) for TAS and OW, respectively.

Figure 6 shows the boxplots for the I_{HV}, I_ϵ and I_{IGD} quality indicators for all six system variants from Table III. Undoubtedly, for all quality indicators and across all system variants there is a clear distance between the quality indicator values obtained by EvoPoli instrumented with NSGA-II or SPEA2 and those produced by PRISM and Storm. We confirmed our findings from the visual inspection of the boxplots by using the Kruskal-Wallis test which showed statistical significance (p-value $<$ 0.05) for all system variants and for all quality indicators. We also ran a post-hoc analysis of pairwise comparisons between the EvoPoli instances, and PRISM and Storm using the Mann-Whitney test. For all comparisons, we observed statistically significant differences in favour of EvoPoli, with the p-value being in the range $[6.2473E-15, 1.0016E-13]$ and with a high effect size ($d > 0.8$). This is a key result of our validation experiments that indicates EvoPoli's capacity to produce Pareto fronts of higher quality than those produced by PRISM and Storm.

We support further our findings through the Pareto front approximations produced by NSGA-II-based EvoPoli, PRISM and Storm for the OW system variants (Figure 7). Evidently,

[1] Pareto compliant indicators conform to the order specified by the Pareto dominance relation on Pareto front approximations [21]

[2] We selected the maximum number of objectives that both model checkers support; Storm can handle up to three objectives.

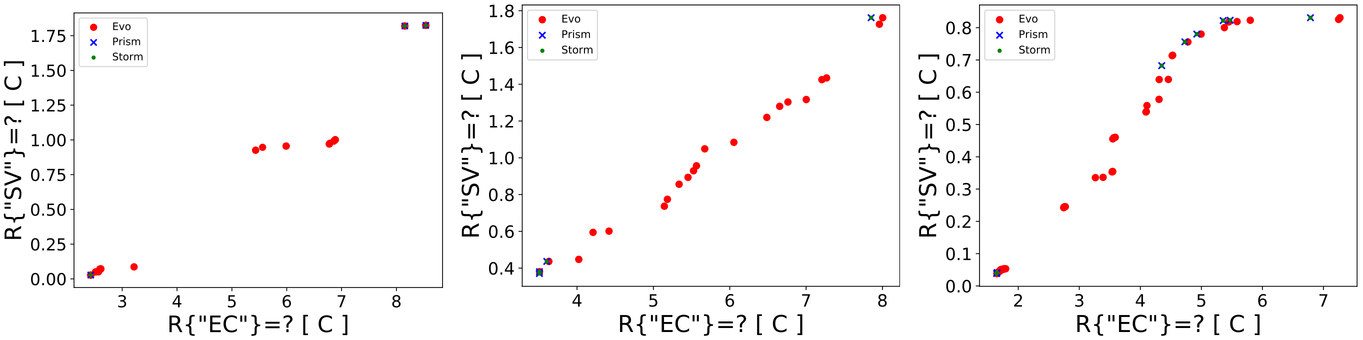


Fig. 7: Pareto front approximation for the OW system variants (OW4 left, O5 middle, OW6 right) and objectives R2 (maximise science value) and R4 (minimise energy) from Table II using PRISM, Storm and NSGA-II-based EvoPoli.

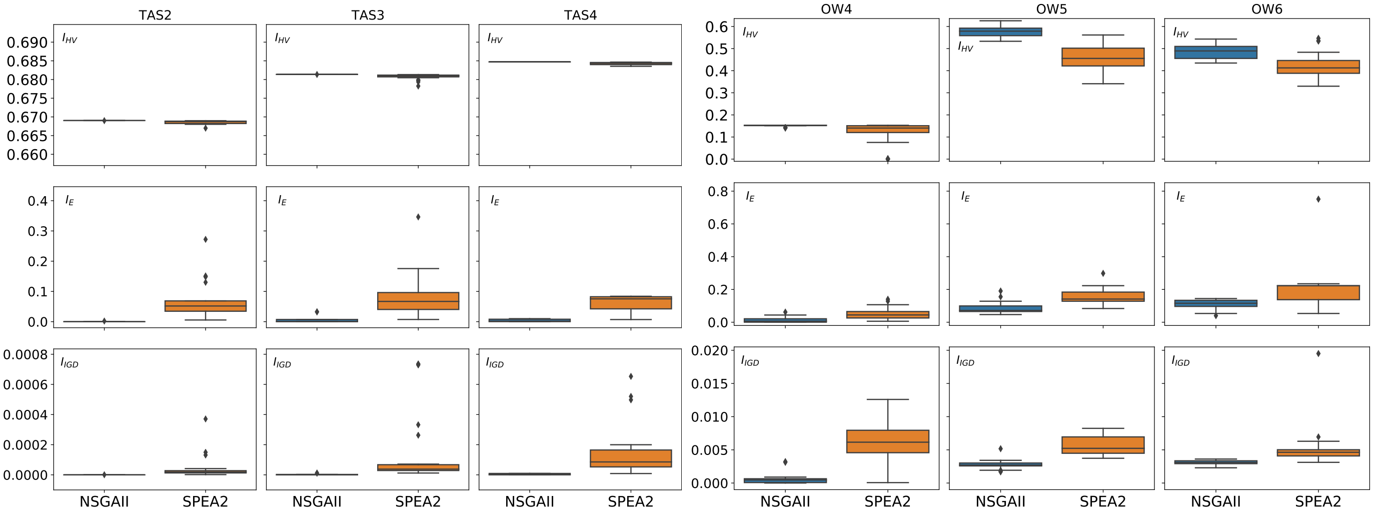


Fig. 8: Boxplots comparing EvoPoli (NSGA-II) and EvoPoli (SPEA2) for the TAS (left) and OW (right) system variants and requirements from Tables I and II, respectively, evaluated using quality indicators I_{HV}, I_ϵ and I_{IGD}.

the policies synthesised by EvoPoli for this typical run closely approximate those produced by the model checkers while also covering a larger spectrum of the objective space. In general, both model checkers found the same policies as shown by the identical Pareto fronts (Figure 7) and the almost identical quality indicator values (Figure 6) – in few problem instances Storm produced more solutions than PRISM. Irrespective of the system variant, however, the produced policies are constrained to the boundaries of the objective space. Since the model checkers employ linear programming, they, unsurprisingly, have difficulties finding useful policies when the objective space is non-convex [17]. In contrast, EvoPoli with its MOGA-based specialisation is not sensitive to the shape or continuity of the Pareto front, and, thus, can synthesise policies when the objective space is also discontinuous or concave [12]. We note that due to the iterative nature of MOGAs used in EvoPoli, our approach takes more time than PRISM or Storm. We have demonstrated, however, that EvoPoli produces a richer and more diversified set of solutions than the other model checkers. Investigating mechanisms to improve the scalability of EvoPoli is part of our future work.

These findings clearly demonstrate that EvoPoli can synthesise policies of equivalent quality to those produced by PRISM and Storm for the simpler class of problems (i.e., unconstrained Pareto queries) that these model checkers can solve. Also, the EvoPoli-produced Pareto front is greatly more diverse and covers a wider spectrum of the objective space.

RQ2 (Effectiveness). We answer this research question by comparing the quality of the Pareto fronts synthesised by two EvoPoli instances using NSGA-II [29] and SPEA2 [32] for the TAS and OW system variants and the full set of requirements from Tables I and II, respectively. Figure 5 shows two derived Pareto fronts for a typical run using these two EvoPoli instances. As shown in Figure 8, the distributions of the I_{HV}, I_ϵ and I_{IGD} quality indicators for the SPEA2-instrumented EvoPoli have a larger overall variability. In contrast, the NSGA-II-based boxplots are more concentrated as indicated by the smaller whiskers and the very few points above or below them. Since both MOGAs generally follow the same evolutionary algorithm principles and apply elitism, i.e., they propagate the best policies across generations, this behaviour could occur due to the different diversity preservation mechanisms used; NSGA-II employs a crowding distance while SPEA2 invokes an archive truncating procedure [12].

The statistical comparison using the Mann-Whitney test showed statistical significance across all system variants, with p-value ranging $[1.716E{-}12, 2.599E{-}10]$ and $[6.255E{-}10, 2.655E{-}06]$ for TAS and OW, respectively. The effect size was large in all system variant-quality indicator combinations except from the TAS4-I_{HV} pair where the effect was medium.

These results provide strong empirical evidence that EvoPoli with NSGA-II can synthesise policies that achieve better quality indicator values than policies synthesised by EvoPoli using SPEA2. More importantly, we have shown that EvoPoli can form effective Pareto optimal policies sets using alternative MOGAs, thus demonstrating the ability of EvoPolito solve the constraint multi-objective policy synthesis problem.

RQ3 (Decision Support). We answer this research question by qualitatively analysing the Pareto front approximations to identify actionable insights concerning the trade-offs between the quality attributes encoded by the synthesised policies. First, through the use of MOGAs, EvoPoli can examine efficiently the discontinuous, and likely non-convex, policy decision space to produce Pareto front policy approximations that cover sufficiently the space. Given this information, software engineers can have a more informed view of the different quality attributes trade-off for their system.

Second, EvoPoli enables the identification of the "points of diminishing returns" where every increase in the value of a quality attribute incurs a disproportional deterioration to the other quality attributes. For the OW6 system variant, for instance, one such point is approximately located at (5,0.75) signifying that policies which contribute higher science values consume significantly more energy. Depending on the system-specific preferences, software engineers can use this information to eliminate such policies (if a balance in quality attributes is preferred) or analyse further these policies (e.g., equip the robot with a larger battery to accommodate the increased energy consumption and enable to use this policy).

Finally, a closer inspection of the Pareto policies set revealed multiple policies that yielded the same quality attribute values. From a planning perspective, these alternative policies (cannot be shown on the Pareto fronts as their values overlap) are very useful as they can support fast system reconfiguration without the need to perform another policy synthesis operation. Having, for instance, a repository of policies with the same quality attributes enables the quick selection of the functioning policies when a malfunction renders the currently active policy unusable. This is a unique feature of EvoPoli that does not exist in either Storm or PRISM.

D. Threats to Validity

We limit **construct validity** threats that could occur due to simplifications in the adopted experimental methodology using the widely-studied TAS case study [27]. We obtained the information for the OW system from the literature [33].

We mitigate internal **validity threats** that could introduce bias when establishing the causality between our findings and the experimental study by assessing EvoPoli using independent research questions. We reported results over 30 independent runs per system variant, thus reducing threats due to the stochasticity of the employed multi-objective evolutionary algorithms. Also, we used the inferential statistical tests Mann-Whitney and Kruskal-Wallis to check for statistical significance ($\alpha = 0.05$), supported by post-hoc analysis using Mann-Whitney's test and Bonferroni's correction to control the family-wise error rate. Finally, we employed Cohen's d to assess the effect size and calculate the amount of improvement.

We mitigate **external validity** threats that could affect the generalisation of our approach by developing EvoPoli on top of the search-based software engineering tool EvoChecker [31] that uses MDP models encoded in the high-level modelling language of PRISM [10]. The experimental evaluation using multiple variants of two software systems reduces further the risk that EvoPoli may be difficult to use in practice. However, further experiments are needed to establish the applicability, feasibility and scalability of EvoPoli in domains and applications with characteristics different from those used in our evaluation.

VII. Related Work

Markov decision processes (MDP) have a wide range of applications in software systems across many domains [8], [34], [35], [36]. MDP models can leave nondeterministic choices underspecified, which can be resolved in disparate ways by different control policies that can balance multiple, potentially conflicting, objectives [3], [37], [38]. In a high optimisation space, there is typically no single policy that optimises all objectives, but rather, a set of Pareto optimal policies with different tradeoffs that form a Pareto front. For existing model checkers, Pareto fronts are often obtained by either using linear programming [16], [13] or iterating over weighted sums of objectives [17], [39], [40]. Employing these methods lead to limited applicability and scalability due to the computational cost involved, constrained search space and the limited number of optimisation objectives supported [7], [31], [18], [28]. PRISM and Storm, for instance, are two of the most advanced probabilistic model checkers currently available, and they are limited to synthesis of MDP multi-objective policies that can consider up to two and three optimisation objectives without constraints (in Storm and PRISM, respectively), or only one objective if the problem contains constraints. In contrast, EvoPoli can handle an arbitrary combination of any number of constraints and objectives. Also, Pareto fronts generated by our approach contain much more diversity because, unlike other approaches, the applicability of evolutionary algorithms is not constrained to convex optimisation problems, where the set of achievable values for a Pareto query is also convex [17].

Multi-objective Reinforcement learning (RL) is a technique, orthogonal to model checking, for obtaining Pareto optimal policies. A major research direction of multi-objective RL is currently on improving the efficiency of training [41], [42], [43]. The approximation of Pareto fronts using RL is determined by minimising the difference between sampling actions and feedback signals. In contrast to conventional RL, multi-objective RL uses one scalar feedback signal per objective, which amplifies training complexity and makes it less efficient [44]. Another issue of using RL for obtaining Pareto optimal policies is that it does not always guarantee safety properties, although recent works started introducing extra mechanisms to mitigate this issue (e.g., by integrating human or domain knowledge in the training) [45]. However, these approaches have several limitations, i.e., do not support multi-objective optimisation [46], make strong assumptions [47], or need complex preprocessing [43].

Search-based software engineering (SBSE), has been extensively studied in various applications and domains, including project management [48], [49], [50], software testing [51], [52], [53], model checking [20], [54], [55], [56] and feature selection in software product lines [57], [58]. SBSE has also been extended to synthesising Pareto-optimal sets of probabilistic models [7], [31], [59], [60]. EvoChecker [7], [31] uses multiobjective optimisation (i.e., genetic algorithms) to automatically produce approximate Pareto-optimal probabilistic model sets with respect to given requirements or constraints. In our work, we leverage EvoChecker as a means of supporting the synthesis method from Algorithm 1.

EvoPoli is, to the best of our knowledge, the first that can solve the multi-objective constrained policy synthesis problem. Concretely, EvoPoli can approximate a set of Pareto optimal policies and the Pareto front for an arbitrary combination of any number of optimisation objectives and constraints.

VIII. Conclusion

We presented EvoPoli, a tool-supported approach for the automated synthesis of Pareto-optimal policies for MDPs with complex combinations of constraints and optimisation objectives. We evaluated EvoPoli on two case studies from different domains and demonstrated its ability to synthesise policies for problems that can be handled by the probabilistic model checkers PRISM [10] and Storm [11] as well as for more complex problems that neither of them can support. Our future work includes (1) extending EvoPoli to support policy synthesis on timed MDPs; (2) explore parallelisation methods to improve EvoPoli's scalability; and (3) applying EvoPoli to other applications and scenarios.

Acknowledgements: This research was supported by the European Union's Horizon 2020 project SESAME (grant agreement No 101017258), the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience', the UK EPSRC project EP/R026173/1 'Offshore Robotics for Certification of Assets' (through its PRF project COVE), and the Assuring Autonomy International Programme.

References

[1] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

[2] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[3] P. Chrszon, C. Dubslaff, S. Klüppelholz, and C. Baier, "Profeat: feature-oriented engineering for family-based probabilistic model checking," *Formal Aspects of Computing*, vol. 30, no. 1, pp. 45–75, 2018.

[4] J. Cámara, D. Garlan, and B. Schmerl, "Synthesizing tradeoff spaces with quantitative guarantees for families of software systems," *Journal of Systems and Software*, vol. 152, pp. 33–49, 2019.

[5] G. Su, T. Chen, Y. Feng, D. S. Rosenblum, and P. Thiagarajan, "An iterative decision-making scheme for markov decision processes and its application to self-adaptive systems," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2016, pp. 269–286.

[6] B. Lacerda, D. Parker, and N. Hawes, "Optimal policy generation for partially satisfiable co-safe ltl specifications." in *IJCAI*, 2015, pp. 1587–1593.

[7] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 319–330.

[8] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 1–12.

[9] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[10] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 585–591.

[11] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 592–600.

[12] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen *et al.*, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[13] K. Etessami, M. Kwiatkowska, M. Y. Vardi, and M. Yannakakis, "Multi-objective model checking of Markov decision processes," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2007, pp. 50–65.

[14] C. Baier, H. Hermanns, and J.-P. Katoen, "The 10,000 facets of MDP model checking," in *Computing and Software Science*. Springer, 2019, pp. 420–451.

[15] R. Calinescu, M. Autili, J. Cámara, A. Di Marco, S. Gerasimou, P. Inverardi, A. Perucci, N. Jansen, J.-P. Katoen, M. Kwiatkowska *et al.*, "Synthesis and verification of self-aware computing systems," in *Self-Aware Computing Systems*. Springer, 2017, pp. 337–373.

[16] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Quantitative multi-objective verification for probabilistic systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 112–127.

[17] V. Forejt, M. Kwiatkowska, and D. Parker, "Pareto curves for probabilistic model checking," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2012, pp. 317–332.

[18] F. Delgrange, J.-P. Katoen, T. Quatmann, and M. Randour, "Simple strategies in multi-objective MDPs," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2020, pp. 346–364.

[19] J. R. Harbin, S. Gerasimou, N. Matragkas, A. Zolotas, and R. Calinescu, "Model-driven simulation-based analysis for multi-robot systems," in *ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2021.

[20] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–61, 2012.

[21] E. Zitzler, J. Knowles, and L. Thiele, "Quality assessment of Pareto set approximations," *Multiobjective optimization*, pp. 373–404, 2008.

[22] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 1–10.

[23] C. Daws, "Symbolic and parametric model checking of discrete-time Markov chains," in *First International Conference on Theoretical Aspects of Computing (ICTAC)*, 2005, pp. 280–294.

[24] R. Calinescu, C. Paterson, and K. Johnson, "Efficient parametric model checking using domain knowledge," *IEEE Transactions on Software Engineering*, vol. 47, no. 6, pp. 1114–1133, 2021.

[25] X. Fang, R. Calinescu, S. Gerasimou, and F. Alhwikem, "Fast parametric model checking through model fragmentation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 835–846.

[26] A. Bianco and L. De Alfaro, "Model checking of probabilistic and nondeterministic systems," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 1995, pp. 499–513.

[27] D. Weyns and R. Calinescu, "Tele assistance: A self-adaptive service-based system exemplar," in *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015*. IEEE Computer Society, 2015.

[28] T. Quatmann, S. Junges, and J.-P. Katoen, "Markov automata with multiple objectives," *Formal Methods in System Design*, pp. 1–54, 2021.

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[30] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," in *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems (EUROGEN'01)*, 2001, pp. 95–100.

[31] S. Gerasimou, R. Calinescu, and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering," *Automated Software Engineering*, vol. 25, no. 4, pp. 785–831, 2018.

[32] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[33] L. Edwards, U. Wong, K. Dalal, C. Kulkarni, A. Rogg, A. Tardy, T. Stucky, O. Umurhan, D. Catanoso, and T. Welsh, "An autonomy software testbed simulation for ocean worlds missions," in *Earth and Space 2021*, 2021, pp. 369–380.

[34] T. L. Cheung, K. Okamoto, F. Maker III, X. Liu, and V. Akella, "Markov decision process (MDP) framework for optimizing software on mobile phones," in *Proceedings of the seventh ACM International Conference on Embedded software*, 2009, pp. 11–20.

[35] A. Ksentini, T. Taleb, and M. Chen, "A Markov decision process-based service migration procedure for follow me cloud," in *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014, pp. 1350–1354.

[36] J. Noppen, M. Aksit, B. Tekinerdogan, and V. Nicola, "Market-driven approach based on Markov decision theory for optimal use of resources in software development," *IEE proceedings-Software*, vol. 151, no. 2, pp. 85–94, 2004.

[37] K. Chatterjee, "Markov decision processes with multiple long-run average objectives," in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 2007, pp. 473–484.

[38] E. M. Hahn, V. Hashemi, H. Hermanns, M. Lahijanian, and A. Turrini, "Multi-objective robust strategy synthesis for interval Markov decision processes," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2017, pp. 207–223.

[39] C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk, "The probabilistic model checker storm," *arXiv preprint arXiv:2002.07080*, 2020.

[40] A. Hartmanns, S. Junges, J.-P. Katoen, and T. Quatmann, "Multi-cost bounded reachability in MDP," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2018, pp. 320–339.

[41] G. Tesauro, R. Das, H. Chan, J. O. Kephart, D. Levine, F. L. Rawson III, and C. Lefurgy, "Managing power consumption and performance of computing systems using reinforcement learning." in *NIPS*, vol. 7. Citeseer, 2007, pp. 1–8.

[44] K. Van Moffaert and A. Nowé, "Multi-objective reinforcement learning using sets of pareto dominating policies," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3483–3512, 2014.

[42] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proceedings of the 25th International Conference on Machine Learning*, 2008, pp. 41–47.

[43] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Scalarized multi-objective reinforcement learning: Novel design techniques," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2013, pp. 191–199.

[45] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[46] S. Junges, N. Jansen, C. Dehnert, U. Topcu, and J.-P. Katoen, "Safety-constrained reinforcement learning for MDPs," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2016, pp. 130–146.

[47] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[48] F. Ferrucci, M. Harman, J. Ren, and F. Sarro, "Not going to take this anymore: Multi-objective overtime planning for software engineering projects," in *35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 462–471.

[49] J. Ren, M. Harman, and M. Di Penta, "Cooperative co-evolutionary optimization of software project staff assignments and job scheduling," in *International Symposium on Search Based Software Engineering*. Springer, 2011, pp. 127–141.

[50] C. Stylianou, S. Gerasimou, and A. S. Andreou, "A novel prototype tool for intelligent software project scheduling and staffing enhanced with personality factors," in *24th IEEE International Conference on Tools with Artificial Intelligence*, vol. 1. IEEE, 2012, pp. 277–284.

[51] J. H. Andrews, T. Menzies, and F. C. Li, "Genetic algorithms for randomized unit testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 1, pp. 80–94, 2011.

[52] G. Fraser and A. Arcuri, "Whole test suite generation," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 276–291, 2012.

[53] M. Harman, Y. Jia, and W. B. Langdon, "Strong higher order mutation-based test data generation," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, 2011, pp. 212–222.

[54] G. Katz and D. Peled, "Synthesis of parametric programs using genetic programming and model checking," *arXiv preprint arXiv:1402.6785*, 2014.

[55] E. Alba and F. Chicano, "Finding safety errors with aco," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, 2007, pp. 1066–1073.

[56] ——, "Searching for liveness property violations in concurrent systems with ACO," in *Proceedings of the 10th annual Conference on Genetic and Evolutionary Computation*, 2008, pp. 1727–1734.

[57] L. Ochoa, O. Gonzalez-Rojas, A. P. Juliana, H. Castro, and G. Saake, "A systematic literature review on the semi-automatic configuration of extended product lines," *Journal of Systems and Software*, vol. 144, pp. 511–532, 2018.

[58] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon, "Combining multi-objective search and constraint solving for configuring large software product lines," in *37th IEEE/ACM International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 517–528.

[59] R. Calinescu, M. Češka, S. Gerasimou, M. Kwiatkowska, and N. Paoletti, "Efficient synthesis of robust models for stochastic systems," *Journal of Systems and Software*, vol. 143, pp. 140–158, 2018.

[60] ——, "Designing robust software systems through parametric markov chain synthesis," in *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 131–140.