Abstraction-Refinement for Hierarchical Probabilistic Models

Sebastian Junges[1(✉)] and Matthijs T. J. Spaan[2]

[1] Radboud University, Nijmegen, The Netherlands
sjunges@cs.ru.nl
[2] Delft University of Technology, Delft, The Netherlands

Abstract. Markov decision processes are a ubiquitous formalism for modelling systems with non-deterministic and probabilistic behavior. Verification of these models is subject to the famous state space explosion problem. We alleviate this problem by exploiting a hierarchical structure with repetitive parts. This structure not only occurs naturally in robotics, but also in probabilistic programs describing, e.g., network protocols. Such programs often repeatedly call a subroutine with similar behavior. In this paper, we focus on a local case, in which the subroutines have a limited effect on the overall system state. The key ideas to accelerate analysis of such programs are (1) to treat the behavior of the subroutine as uncertain and only remove this uncertainty by a detailed analysis if needed, and (2) to abstract similar subroutines into a parametric template, and then analyse this template. These two ideas are embedded into an abstraction-refinement loop that analyses hierarchical MDPs. A prototypical implementation shows the efficacy of the approach.

1 Introduction

Markov Decision Processes (MDPs) are *the* model for sequential decision making under probabilistic uncertainty, and as such are central in modelling of randomized algorithms, distributed systems with lossy channels, or as the underlying formalism in reinforcement learning. A key question in the verification of MDPs is: *What is the maximal probability that some error state is reached?* In this question, one accounts for the probabilistic nature as well as the inherit (potentially adversarial) nondeterminism of the system. Various state-of-the-art probabilistic model checkers, such as Storm [20], Prism [27] and Modest [17] implement a variety of methods that automatically compute such maximal probabilities. Most widespread are variations of value-iteration that iteratively apply a transition function to converge towards the requested probability.

Hierarchical Structure. Despite various successes, the state space explosion remains a significant challenge to the model-based analysis of MDPs. To overcome this challenge, some approaches exploit symmetries or the parallel composition of a system. Other approaches exploit that typically not all paths through a system are equally likely and thus aim to find the essential or critical subsystem.

© The Author(s) 2022
S. Shoham and Y. Vizel (Eds.): CAV 2022, LNCS 13371, pp. 102–123, 2022.
https://doi.org/10.1007/978-3-031-13185-1_6

```
p = 0.5; time = 0; N=3;
repeat N times {
   time += passToken(p);
   if flip(0.5) {p = 0.8p}
   else         {p = 1.25p}
}; return time
```

(a) Repeated invocation of `passToken(p)`

```
passToken(p):
  t = 1;
  while (not flip(p)) {t++};
  t++;
  while (not flip(p)) {t++};
  return t
```

(b) `passToken(p)`: Pass succeed twice.

Fig. 1. Simplified example for sending a token over an unreliable channel.

While we exploit related ideas—a detailed comparison is given in the related work, cf. Sect. 7—our approach is fundamentally different and instead exploits a *hierarchical decomposition* natural in many system models. This decomposition is captured naturally by probabilistic programs (over discrete bounded variables) with non-nested subroutines, where some subroutines are called repeatedly with *similar* arguments. Figure 1 shows an example in which we demonstrate our approach in Sect. 2. More generally, we are interested in systems with an overall task that is achieved by a suitable combination of a limited number of sub-tasks. Such a setting occurs naturally, e.g. (i) in robotics, when multiple rooms in a floor need to be inspected, or (ii) in routing, when multiple packets need to be routed sequentially. The underlying problem structure is also exploited in hierarchical planning [5,19,30], where the goal is to find a good but not necessarily optimal policy (and induced value). *We combine insights from hierarchical planning with an abstraction-refinement perspective and then construct an anytime algorithm with strict guarantees on the result.*

Local Model-Based Analysis. An adequate operational model for the model-based analysis of hierarchical systems is given by a *hierarchical MDP*, where the state space of a hierarchical MDP can be partitioned into *subMDPs*. Abstractly, one can represent a hierarchical MDP by the collection of subMDPs and a *macro-level MDP* [19] where the probabilities of outgoing transitions at a state are described by a corresponding subMDP, cf. Sect. 3.2. In this paper, we focus on a hierarchical MDPs where the policies that are optimal in (only) a subMDP are optimal (partial) policies in the hierarchical MDP. More intuitively, we can solve the subMDPs individually, i.e., the solution (w.r.t. the fixed measure) for the subMDP is part of the globally optimal solution. While this assumption is restrictive, it is satisfied in various interesting settings. The assumption allows us to analyse subMDPs out-of-context, i.e., we can first analyse the subMDPs and then construct the correct macro-MDP, i.e., extract transition probabilities and rewards from the subMDP analysis. This approach already improves the maximal memory consumption and allows for additional speed-ups if the *same* subMDP occurs multiple times.

Epistemic Uncertainty During Computation. The key insight to accelerate the outlined approach further is to avoid analysing all subMDPs precisely, while still providing sound guarantees on the obtained results. Therefore, consider that even

before analysing the subMDPs we can analyse an uncertain variant of the macro-level MDP where we do not yet know the associated transition probabilities and rewards but instead only know intervals. We may then do two things: First, we can identify the subMDPs which are most critical, i.e., where replacing the interval by a concrete value yields most benefits. Second, and more importantly, we can analyse a set of subMDPs and refine the associated uncertainties, i.e., tighten the associated intervals. To support the analysis of sets of subMDPs, we observe that often, these subMDPs are slight variations. In this paper, we represent them as parameterised instances of a particular templates that we define using parametric MDPs (pMDPs). The resulting intervals can be used to create an (interval-valued version of the) macro-level MDP. Analysing this gives bounds on the expected reward in the hierarchical MDP, and the bounds can be refined by analysing the subMDPs more precisely.

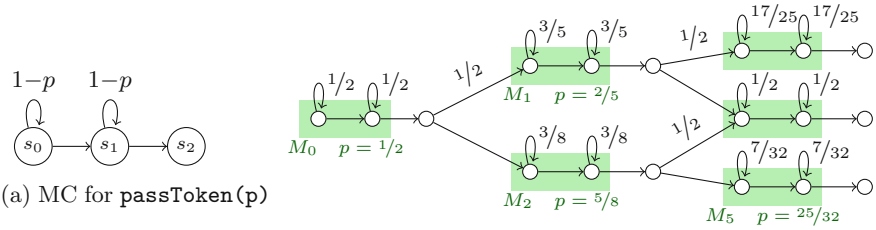
Contributions. In a nutshell, we explicitly allow for *uncertainty* during the solving process to speed up the analysis of hierarchical MDPs. Concretely, we contribute a scalable approach to solve hierarchical MDPs with many different subMDPs, in particular when these subMDPs are similar, but not the same. The approach resembles an abstraction-refinement loop where we abstract the hierarchical MDP in two layers and then refine the analysis of the lower layer to get a refined representation of the complete MDP. In every step, we can provide absolute error bounds. Our approach interprets the different subMDPs as a form of uncertainty. The efficient analysis originates from progress made in the analysis of uncertain (or parametric) MDPs, and brings that progress to a novel setting. The empirical evaluation with a prototype called LEVEL-UP shows the efficacy of the approach.

2 Overview

We clarify the approach and its applicability with a motivating example that drastically abstracts a token passing process where the channel quality varies [12].

Setting. Consider the protocol in Fig. 1a which sends a token N times via a channel. That channel successfully transmits packets with probability p, where p varies over time. The subroutine takes t amount of time, depending on p. Specifically, in the model, we alternate between accumulating the required time and updating the channel quality for N token transmissions and then return the accumulated time. We aim to compute the expected return value. For the subroutine, we assume that sending a token is repeated until an acknowledgement is received, which is abstractly modelled in Fig. 1b and corresponds to the small Markov chain in Fig. 2a. First, the file must successfully be sent ($s_0 \to s_1$), then we start sending acknowledgements. The process terminates ($s_1 \to s_2$) once an acknowledgement is received. The complete protocol from Fig. 1 including the subroutine is reflected by the large Markov chain in Fig. 2b that repeats the

small Markov chain (with different probabilities). This model may be analysed with standard tools, but for large N (and larger subroutines), the state space explosion must be alleviated.



(a) MC for `passToken(p)`

(b) Hierarchical MDP, rewards of 1 at states with loops

Fig. 2. Ingredients for hierarchical MDPs with the Example from Fig. 1. Annotations reflect subMDPs within the macro-MDPs in Fig. 3.

Macro-MDPs and Enumeration. We thus suggest to abstract the hierarchical model into the macro-level MDP in Fig. 3a. Here, every state corresponds to an invocation of the subprocess. The reward at the states corresponds to the expected reward for the complete subprocess. Thus, naively, one may construct the macro-MDP, analyse all (reachable) subMDPs independently and annotate the macro-MDP states with the appropriate rewards, and finally analyse the macro-MDP to obtain a result of ≈12.3. This approach avoids representing the complete hMDP in the memory, but it is still restricted to analysing systems with a limited number of subMDPs.

Our Approach. We improve scalability by constructing a parameterized macro-MDP. Reconsider the rewards for Fig. 3a. The values can be computed via the graph in Fig. 3d, where we pick for each value for p (x-axis) and compute the corresponding expected reward $\mathbb{E}$ (y-axis) obtained by analysing the subMDP in Fig. 2a. Intuitively, in our abstraction, we annotate the rewards with lower- and upper bounds rather than exact values. Therefore, we compute bounds on the rewards by selecting an interval for the values $p \in [8/25, 25/32]$, as shown in Fig. 3e. Conceptually, this means that we analyse a set of subMDPs at once, namely all subMDPs with $p \in [8/25, 25/32]$. Annotating the corresponding expected rewards, in this case $[64/25, 25/4]$, then yields the macro-MDP in Fig. 3b. Analysis of this MDP yields that overall expected time is in $[7.68, 18.75]$. We refine these bounds by analysing subsets of the subMDPs. We may split the values for p into two sets $[8/25, 2/5]$ and $[1/2, 25/32]$. Then, we obtain two corresponding intervals on the expected time in the subMDP as shown in Fig. 3f. Model checking the associated macro-MDP, in Fig. 3c, bounds to expected time by $[10.12, 14.25]$. Technically, we realize this reasoning using parameter lifting [33].

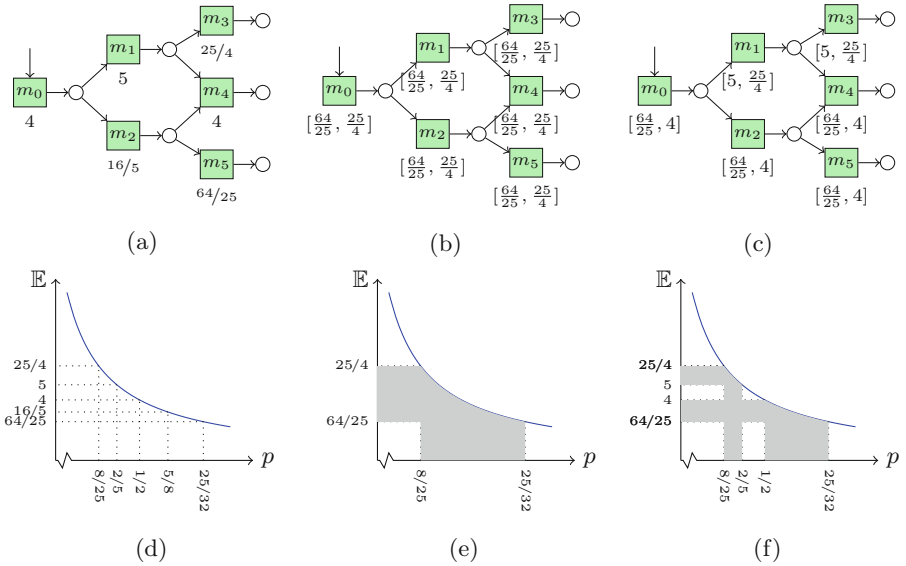


Fig. 3. Visualising the computation of expected rewards for the hMDP from Fig. 2b using a macro-MDP and interval-based abstractions.

Supported Extensions. For conciseness, this example is necessarily simple. Our approach allows nondeterminism, i.e., action-choices, in the macro-MDP *and* in the subMDPs. The subMDPs may have multiple outgoing transitions, but this must be combined with a restricted type of nondeterminism in the subMDP: If multiple outgoing transitions are present, the macro-MDP has transition probabilities that depend on the subMDPs. We present a useful extension for reachability probabilities, see the discussion at the bottom of Sect. 3.3.

More Examples. Key ingredient to models where the approach excels are a repetitive task whose characteristics depend on some global state. Two variations are the expected energy consumption of a robot with slowly degrading components that, e.g., can be improved by maintenance or for job scheduling with periodically changing distribution of tasks (e.g., day vs. night).

3 Formal Problem Statement

We formalize MDPs and *hierarchical MDPs* (hMDPs) to pose the problem statement, then identify a subclass of hMDPs which we call *local-policy hMDPs* and restrict our problem on computing optimal expected rewards in local-policy hMDPs. Furthermore, we introduce parametric MDPs as they are key to the abstraction-refinement procedure later in the paper.

3.1 Background

Definition 1 (Parametric MDP). *A parametric MDP (pMDP) is a tuple* $\mathcal{M} = \langle S_\mathcal{M}, A_\mathcal{M}, \iota_\mathcal{M}, \vec{x}, P_\mathcal{M}, r_\mathcal{M}, T_\mathcal{M} \rangle$ *where* $S_\mathcal{M}$ *is a finite set of* states, $A_\mathcal{M}$ *is a finite set of* actions, $\iota_\mathcal{M} \in S_\mathcal{S}$ *is the* initial state, $\vec{x} = \langle x_0, \dots x_n \rangle$ *is a vector of* parameters, $P_\mathcal{M} \colon S_\mathcal{M} \times A_\mathcal{M} \times S_\mathcal{M} \to \mathbb{Q}[\vec{x}]$ *are the* transition probabilities, $r_\mathcal{M} \colon S \to \mathbb{Q}[\vec{x}]$ *the* state rewards, *and* $T_\mathcal{M}$ *is a set of* target states.

We drop the subscripts whenever possible. MDPs are *parametric* if $\vec{x} \neq \langle\rangle$ and *parameter-free* otherwise. We omit parameters for parameter-free MDPs. We recap some standard notions on pMDPs (and MDPs):

For a (parameter) *valuation* $u \in \mathbb{R}^{\vec{x}}$, the *instantiation* $\mathcal{M}[u]$ globally substitutes $P_\mathcal{M}(s,a,s')$ with $P_\mathcal{M}(s,a,s')(u)$ and $r_\mathcal{M}(s)$ with $r_\mathcal{M}(s)(u)$. An assignment u is well-defined, if $\mathcal{M}(u)$ constitutes an MDP, i.e., if $\sum_{s'} P_\mathcal{M}(s,\alpha,s')(u) \in \{0,1\}$ and $r_\mathcal{M}(s)(u) \geq 0$ for each $s \in S$, $\alpha \in A$. We denote the set of all well-defined assignments with $U_\mathcal{M}$. The set $\mathsf{Act}(s)$ denotes the enabled actions at state s, $\mathsf{Act}(s) = \{\alpha \mid \sum_{s'} P_\mathcal{M}(s,\alpha,s') \neq 0\}$. If $|\mathsf{Act}(s)| = 1$ for every $s \in S$, then the (parametric) MDP is a (parametric) *Markov chain* (MC). A path π is an (in)finite sequence of states $s_0 \xrightarrow{\alpha_0} s_1 \dots$, with $s_i \in S$, $\alpha_i \in \mathsf{Act}(s_i)$, $P(s_i,\alpha_i,s_{i+1}) \neq 0$. For finite π, $\mathsf{last}(\pi)$ denotes the last state of π. We use $[s \to \Diamond T]$ to denote the set of (finite) paths T only at the end. The reward $r(\pi)$ along a finite path π is the sum of the state rewards $r(\pi) := \sum r(s_i)$.

Specifications. We consider indefinite horizon expected reward, i.e., the expected accumulated reward until reaching the target states. We refer to [3,32] for a formal treatment and only introduce notation. Therefore, the unique probability measure Pr for a set of paths in a parameter-free *Markov chain* $\mathcal{M}$ reaching state T can be defined using the usual cylinder set construction. We define $Pr_\mathcal{M}(s \to \Diamond T)$ as the probability to reach a state in T, $\int_{\pi \in [s \to \Diamond T]} Pr(\pi) d\pi$. We then define the expected reward until hitting T, $\mathsf{ER}_\mathcal{M}(s \to \Diamond T) = \int_{\pi \in [s \to \Diamond T]} Pr(\pi) \cdot r(\pi) d\pi$. In both definitions, if s is the initial state, we simply write $\dots(\Diamond T)$. For technical conciseness, we make the standard assumption that target states are reached with probability 1, which ensures that the integral exists and is finite. (Arbitrary) reachability probabilities can be nevertheless be modelled using rewards.

Policies. In pMDPs, we resolve nondeterminism with policies. In this paper, it suffices to consider *memoryless policies* $\sigma \colon S \to A$. The set of such policies is denoted $\Sigma(\mathcal{M})$. We omit $\mathcal{M}$ if it is clear from the context. It is helpful to also consider *partial* policies $\hat{\sigma} \colon S \nrightarrow A$. For an pMDP $\mathcal{M}$ and a (partial) policy $\hat{\sigma}$, the induced dynamics are described by the *induced pMDP* $\mathcal{M}[\hat{\sigma}]$, defined as $\langle S_\mathcal{M}, A_\mathcal{M}, \iota_\mathcal{M}, \vec{x}, P, r_\mathcal{M}, T_\mathcal{M} \rangle$, where the transition probabilities are given as

$$P(s,\alpha,s') = \begin{cases} P_\mathcal{M}(s,\alpha,s') & \text{if } \hat{\sigma}(s) = \alpha, \\ 0 & \text{otherwise.} \end{cases}$$

If σ is total (not partial), then $\mathcal{M}$ is a MC. We define the maximal expected reward $\mathsf{ER}^{\max}_\mathcal{M}(\Diamond T) = \max_{\sigma \in \Sigma} \mathsf{ER}_{\mathcal{M}[\sigma]}(\Diamond T)$, and say that a policy σ is optimal, if $\mathsf{ER}^{\max}_\mathcal{M}(\Diamond T) = \mathsf{ER}_{\mathcal{M}[\sigma]}(\Diamond T)$.

Regions and Parametric Model Checking. A set of valuations described by is called a (rectangular) *region*, if $R = \{u \mid u^- \leq u \leq u^+\}$ for adequate bounds $u^-, u^+ \in \mathbb{R}^{\vec{x}}$ and using pointwise inequalities, i.e., R is a Cartesian product of intervals of parameter values. We denote this region also with $[[u^-, u^+]]$. For regions, we may compute a lower bound on $\min_{u \in R} \mathsf{ER}^{\max}_{\mathcal{M}[u]}(\Diamond T)$ and an upper bound on $\max_{u \in R} \mathsf{ER}^{\max}_{\mathcal{M}[u]}(\Diamond T)$ via *parameter lifting* [33,36].

3.2 Hierarchical MDPs

We concentrate on solving hierarchical MDPs (hMDPs). We assume that hMDPs are parameter-free and that their topology has some additional known structure.

Definition 2 (Hierarchical MDPs). *A MDP $\mathcal{M}$ with a partitioning of its states $S_\mathcal{M} = \bigcup \mathbf{S}_i$ is a hierarchical MDP, if for all i,*

- *there exists a unique $s^i_\iota \in \mathbf{S}_i$ such that $s^i_\iota = \iota_\mathcal{M}$ or $\mathsf{pred}_\mathcal{M}(s^i_\iota) \not\subseteq \mathbf{S}_i$, and*
- *for all $s \in \mathbf{S}_i \setminus \{s^i_\iota\}$, it holds that $s^i_\iota \neq \iota_\mathcal{M}$ and $\mathsf{pred}_\mathcal{M}(s) \subseteq \mathbf{S}_i$.*

The state s_ι is called the *entry state*, which we denote entry_i. States with $\mathsf{succ}_\mathcal{M}(s) \cap \mathbf{S}_i = \emptyset$ are called *exit-states*. The set $\mathsf{succ}(i) := \mathsf{succ}_\mathcal{M}(\mathbf{S}_i) \setminus \mathbf{S}_i$ are the *successor states* of the partition i. Let $Y = \max_i |\mathsf{succ}(i)|$. By adding auxiliary states, we can assume that $|\mathsf{succ}(i)| = Y$ for all i. We call partitions with $|\mathbf{S}_i| = 1$ *trivial*. We use $\mathbb{I} := \{i \mid |\mathbf{S}_i| > 1\}$ to denote the indices of the non-trivial partitions. We remark that every MDP can be considered as an hMDP with only trivial partitions.

Problem: Given a (hierarchical) MDP $\mathcal{M}$ with target states T and $\eta \in [0,1]$, compute bounds lb, ub with $\mathsf{lb} \leq \mathsf{ER}^{\max}_\mathcal{M}(\Diamond T) \leq \mathsf{ub}$ and $\eta \cdot \mathsf{ub} \leq \mathsf{lb}$.

The naive solution to this problem is to ignore the hierarchical structure and solve the MDP monolithically. In this paper, we contribute methods that actively exploit the structure of the hierarchical MDPs with $|\mathbb{I}| \gg 1$. We will make an additional assumption on the structure of the hierarchical MDP.

3.3 Optimal Local Subpolicies and Beyond

Intuitively, we want to ensure that the optimal policy within the partitions can be computed locally, i.e., on partition without taking into account the complete MDP. Therefore, each partition within the MDP can be considered as an individual MDP. In particular, each $\mathbf{S}_i$ induces a subMDP as follows:

Definition 3 (subMDP). *Given a hierarchical MDP $\mathcal{M}$ and partition $\boldsymbol{S}_i$, the corresponding subMDP is an MDP $\mathcal{M}_i := \langle S_i := \boldsymbol{S}_i \cup \mathsf{succ}_\mathcal{M}(\boldsymbol{S}_i) \cup \{\bot\}, A_\mathcal{M} \cup \{\alpha_\bot\}, \iota := \mathsf{entry}_i, P_i, r_i, G_i\rangle$ with P_i defined by*

$$P_i(s,\alpha,s') := \begin{cases} P_\mathcal{M}(s,\alpha,s') & \text{if } s \in \boldsymbol{S}_i \text{ and } \alpha \in A_\mathcal{M}, \\ 1 & \text{else if } s \notin \boldsymbol{S}_i, \alpha = \alpha_\bot, \text{ and } s' = \bot \\ 0 & \text{otherwise.} \end{cases}$$

r_i is defined as $r_i(s) = r_{\mathcal{M}}(s)$ if $s \in \boldsymbol{S}_i$, $r_i(s) = 0$ otherwise, and $G_i := \{\bot_i\}$.

Thus, for every partition of the hierarchical MDP, the corresponding subMDP contains additionally the successor states, and a unique bottom state that is a target state and simplifies our construction later.

Likewise, we can (de)compose memoryless policies for the hierarchical MDP as a union of policies on the individual subMDPs. We do this only for nontrivial partitions. Let $\sigma_i \colon S_i \mapsto A$ denote memoryless policies for $\mathcal{M}_i$ and σ_i' the restriction of σ_i to $\mathbf{S}_i$, then $(\bigsqcup_{\mathbb{I}} \sigma_i) : S \nrightarrow A$ is the unique partial policy such that

$$\big(\bigsqcup_{\mathbb{I}} \sigma_i\big)(s) := \sigma_i'(s) \text{ if } s \in \mathbf{S}_i, i \in \mathbb{I} \quad \text{and} \quad \big(\bigsqcup_{\mathbb{I}} \sigma_i\big)(s) := \bot \text{ otherwise.}$$

Intuitively, we want that the union of locally optimal policies, a partial policy, can be completed to a total policy that is optimal.

Definition 4 (Optimal local subpolicies). *Given a hierarchical MDP $\mathcal{M}$ with target states T and optimal policies $\sigma_i \in \Sigma(\mathcal{M}_i)$ for all $i \in \mathbb{I}$. The hierarchical MDP has* optimal local subpolicies*, if for $\hat{\sigma} = \bigsqcup_{\mathbb{I}} \sigma_i$ it holds that $\mathsf{ER}^{max}_{\mathcal{M}[\hat{\sigma}]} = \mathsf{ER}^{max}_{\mathcal{M}}$.*

That is, if we collect (locally) optimal policies σ_i and apply them to $\mathcal{M}$, we obtain the MDP $\mathcal{M}[(\bigsqcup_{\mathbb{I}} \sigma_i)]$. In that MDP, we can pick an optimal policy, and together with $(\bigsqcup_{\mathbb{I}} \sigma_i)$ this constitutes an optimal and total policy for $\mathcal{M}$.

Assumption: The hierarchical MDP has optimal local subpolicies.

Roughly, the idea now becomes that rather than solving one large MDP with S states, we solve $|\mathbb{I}|$ MDPs with $S/|\mathbb{I}|$ states and one MDP with $\mathbb{I}$ states (assuming equally-sized and only nontrivial partitions).

The assumption is restrictive, but not unreasonable: A subroutine may not have any nondeterminism, or a finished task will have no influence on any future task. The following proposition, while obvious, formalizes that:

Proposition 1 (Sufficient criterion). *Let $\mathcal{M}$ be a hierarchical MDP. The MDP has optimal local subpolicies, if for each $i \in \mathbb{I}$ either*

- *there is a single successor for the partition, i.e., $|\mathsf{succ}_{\mathcal{M}}(\boldsymbol{S}_i) \setminus \boldsymbol{S}_i| = 1$, or*
- *there are no choices, i.e., $|\mathsf{Act}(s)| = 1$ for all $s \in \boldsymbol{S}_i$,*

Beyond Optimal Local Subpolicies. The efficiency of our approach is partly due to the assumption in Definition 4. We observe that adapting this definition allows for a spectrum of specific yet useful cases. In particular, say that our system describes a protocol in which we must optimize the probability to satisfy N tasks all may fail – the subMDPs will have two successor states. Often, it is then easy to see (and model) that a locally optimal policy will aim to satisfy each task and that thus, the locally optimal policy optimizes the probability to

reach the corresponding successor state. Then, by adopting the target states in Definition 3 to be the successor state where the task is successful, the notion of an optimal policy—and thus of an optimal local subpolicy—changes. These changes are minimal and everything that follows below is easily adapted to this setting as demonstrated by the prototypical implementation.

4 Solving hMDPs with Abstraction-Refinement

In this section, we consider hMDPs with optimal local subpolicies. We step-wise develop a sketch of an anytime algorithm that provides lower and upper bounds on the expected reward in this hMDP. In Sect. 4.1, we introduce an alternative representation of our problem that formalizes the idea of individually computing subMDPs. We then formalize the ideas that allow to construct an anytime algorithm in Sect. 4.2. In Sect. 4.3, we introduce the abstract requirements for analysing sets of subMDPs into the algorithm, and finally, in Sect. 4.4 we introduce a method that realises this using pMDPs.

4.1 The Macro-MDP Formulation

We adapt macro-MDPs [5] which summarize the subMDPs by single states.

Definition 5 (Macro-MDP). *Let $\mathcal{M}$ be a hMDP with n non-trivial $\boldsymbol{S}_i$ partitions and $S_{\mathcal{M}}$ partitioned as $S_{\mathcal{M}} = \bigcup \boldsymbol{S}_i \cup S'$. The* macro-MDP *is defined as $\mu(\mathcal{M}) := \langle S' \cup \{\mathsf{entry}_i \mid 1 \leq i \leq n\}, A_{\mathcal{M}}, \iota_{\mathcal{M}}, \emptyset, P, r, T_{\mathcal{M}} \rangle$ with P and r given by*

$$P(s,\alpha,s') = \begin{cases} \mathsf{Pr}_{\mathcal{M}_i[\sigma_i]}(\Diamond\{s'\}) & \text{if } s \in \boldsymbol{S}_i, \\ P_{\mathcal{M}}(s,\alpha,s') & \text{otherwise,} \end{cases} \quad r(s) = \begin{cases} \mathsf{ER}^{\max}_{\mathcal{M}_i}(\Diamond\{\bot\}) & \text{if } s \in \boldsymbol{S}_i, \\ r_{\mathcal{M}}(s) & \text{otherwise.} \end{cases}$$

where $\mathcal{M}_i$ is the corresponding subMDP (see Definition 3) and σ_i is an arbitrary but fixed optimal policy, i.e., a policy such that $\mathsf{ER}_{\mathcal{M}_i[\sigma_i]}(\Diamond G_i) = \mathsf{ER}^{\max}_{\mathcal{M}_i}(\Diamond G_i)$.

Intuitively, we replace the transitions within $\mathbf{S}_i$ by a 'big-step semantics' that aggregates the transitions within $\mathbf{S}_i$ by single transitions such that the probability to reach any successor matches the probability to do so within $\mathbf{S}_i$ under a specific –optimal– policy. Likewise, the expected reward matches the expected reward collected in $\mathbf{S}_i$[1].

Remark 1. To define a *unique* macro-MDP, we can take the lexicographically smallest policy σ_i among the optimal policies. Furthermore, we observe that for the cases covered by Proposition 1, it is not necessary to compute σ_i at all: Either there is a single successor—implying $\mathsf{Pr}_{\mathcal{M}_i[\sigma_i]}(\Diamond\{s'\}) = 1$ for any σ_i—or $|\Sigma(\mathcal{M}_i)| = 1$.

The following theorem formalises that, given the assumptions, taking the big-step semantics is adequate when optimizing for an expected reward.

[1] Due to the additive nature of expected rewards, we can annotate the state with the expected reward even though it may differ over the different paths to an exit of $\mathbf{S}_i$.

Theorem 1. *Let $\mathcal{M}$ be a hMDP with optimal local subpolicies and let $\mu(\mathcal{M})$ be the corresponding macro-MDP. Then: $\mathsf{ER}^{max}_{\mu(\mathcal{M})}(\Diamond T) = \mathsf{ER}^{max}_{\mathcal{M}}(\Diamond T)$.*

The important ingredient are the optimal local subpolicies that ensure that we aggregate behavior within the partitions by behavior that agrees with a (globally) optimal policy. We give a proof in the appendix[2].

Naive Algorithm. Algorithmically, we first compute $\mathsf{ER}^{\mathsf{max}}_{\mathcal{M}_i}(\Diamond T_i)$ and the associated policy σ_i, then compute the reachability probabilities on the induced Markov chain. We collect these results in a vector res_i, which is helpful to construct the macro-MDP. To clarify further constructions in this paper, we make res_i explicit. Recall that $|\mathsf{succ}_{\mathcal{M}}(\mathbf{S}_i)| = Y$ for all i.

Definition 6 (Results for subMDP). *Let $\mathcal{M}_i$ be a subMDP for the partition $\boldsymbol{S}_i$ of a hMDP $\mathcal{M}$. Let $\mathsf{succ}_{\mathcal{M}}(\boldsymbol{S}_i)$ be ordered. We define $\mathsf{res}_i \in \mathbb{R}^{Y+1}$ s.t.*

$$\mathsf{res}_i(j) := Pr_{\mathcal{M}_i[\sigma_i]}(\Diamond\{\mathsf{succ}_{\mathcal{M}}(\boldsymbol{S}_i)_j\}) \text{ for } 0 \leq j < Y \text{ and } \mathsf{res}_i(Y) := \mathsf{ER}^{max}_{\mathcal{M}_i}(\Diamond G_i),$$

where σ_i is an arbitrary but fixed policy such that $\mathsf{ER}_{\mathcal{M}_i[\sigma_i]}(\Diamond G_i) = \mathsf{ER}^{max}_{\mathcal{M}_i}(\Diamond G_i)$.

This allows us to reformulate the macro-MDP, in particular, the following two identities do hold:

$$P(s,\alpha,s') = \begin{cases} \mathsf{res}_i(j) & \text{if } s \in \mathbf{S}_i \text{ and} \\ & s' = \mathsf{succ}_{\mathcal{M}}(\mathbf{S}_i)_j \\ P_{\mathcal{M}}(s,\alpha,s') & \text{otherwise,} \end{cases} \quad r(s) = \begin{cases} \mathsf{res}_i(Y) & \text{if } s \in \mathbf{S}_i, \\ r_{\mathcal{M}}(s) & \text{otherwise.} \end{cases} \tag{1}$$

The identities trivialize that constructing the macro-MDP can be done by precomputing the necessary result-vectors.

Enumeration baseline: With macro-MDPs, we reduce the computation of $\mathsf{ER}^{\mathsf{max}}_{\mathcal{M}}(\Diamond T)$ to (1) analysing all subMDPs $\mathcal{M}_i$ and (2) analysing $\mu(\mathcal{M})$.

This rather naive algorithm already limits memory and may exploit similarities between subMDPs during the analysis, e.g., based on the structure discussed in Sect. 4.4. It performs well if the number $|\mathbb{I}|$ of subMDPs is sufficiently small. We are interested in considering methods that allow for larger $\mathbb{I}$ or larger subMDPs. In particular, we want to avoid analysing all subMDPs, all individually.

4.2 The Uncertain Macro-MDP Formulation

Uncertainty Before Computation. We start introducing a method that allows providing bounds on the expected rewards after individually analysing a subset of the subMDPs. Before computing the individual probabilities in $\mathcal{M}_i$, we are *uncertain* about the probabilities and rewards in the MDP $\mu(\mathcal{M})$. Under this

[2] See: https://doi.org/10.48550/arXiv.2206.02653.

uncertainty, we may not be able to compute $\mathsf{ER}^{\max}_{\mu(\mathcal{M})}(\lozenge T)$ precisely. However, we may solve the problem statement by *bounding* the expected reward. Thus, the goal is to compute values lb, ub s.t.

$$\mathsf{lb} \leq \mathsf{ER}^{\max}_{\mathcal{M}}(\lozenge T) = \mathsf{ER}^{\max}_{\mu(\mathcal{M})}(\lozenge T) \leq \mathsf{ub}. \tag{2}$$

Uncertain Macro-MDPs. We capture the a-priori uncertainty about the sub-MDP results in an uncertain macro-MDP, a particularly shaped *parametric* MDP.

Definition 7 (Uncertain macro-MDP). *Let $\mathcal{M}$ be a hMDP with n non-trivial $\boldsymbol{S}_i$ partitions and $S_{\mathcal{M}}$ partitioned as $S_{\mathcal{M}} = \bigcup \boldsymbol{S}_i \cup S'$. The* uncertain macro-MDP *is defined as $\nu(\mathcal{M}) := \langle S' \cup \{\mathsf{entry}_i \mid 1 \leq i \leq n\}, A_{\mathcal{M}}, \iota_{\mathcal{M}}, \vec{x}, P, r, T_{\mathcal{M}}\rangle$ with parameters $\vec{x} := \{p_{i,j}, q_i \mid 1 \leq i \leq n, 1 \leq j \leq Y\}$ where $Y = |\mathsf{succ}_{\mathcal{M}}(\boldsymbol{S}_i)|$. P and r given by*

$$P(s,\alpha,s') := \begin{cases} p_{i,j} & \text{if } s \in \boldsymbol{S}_i \text{ and} \\ & s' = \mathsf{succ}_{\mathcal{M}}(\boldsymbol{S}_i)_j, \\ P_{\mathcal{M}}(s,\alpha,s') & \text{otherwise,} \end{cases} \quad r(s) := \begin{cases} q_i & \text{if } s \in \boldsymbol{S}_i, \\ r_{\mathcal{M}}(s) & \text{otherwise.} \end{cases}$$

Remark 2. Whenever $\mathcal{M}_i$ and $\mathcal{M}_{i'}$ are isomorphic, we may reduce the parameters and replace each occurrence of $p_{i',j}$ with $p_{i,j}$ and each occurrence of $q_{i'}$ with q_i.

The uncertain macro-MDP can be instantiated to coincide with the macro-MDP by setting the parameters accordingly.

Theorem 2. *Let $\mathcal{M}$ be a hMDP, $\mu(\mathcal{M})$ the associated unique macro-MDP, and $\nu(\mathcal{M})$ the associated uncertain macro-MDP with parameters $p_{i,j}$ and q_i. Let u^* be a parameter valuation with $u^*(p_{i,j}) = \mathsf{res}_i(j)$ and $u^*(q_i) = \mathsf{res}_i(Y)$ for all i, j. Then:*

$$\nu(\mathcal{M})[u^*] = \mu(\mathcal{M})$$

Proof sketch. The construction of the uncertain macro-MDP and the macro-MDP only differs in the assignment of probabilities. We set u here as in the characterisation in (1) and thus the equality follows. □

Computing Bounds. Assume for now that we can derive some (trivial) sound bounds on the results vector for any subMDP $\mathcal{M}_i$[3].

Definition 8 (Sound bounds on results). *For $\mathcal{M}_i$, the vectors lbres_i and ubres_i are* sound bounds *if the following pointwise inequality holds*

$$\mathsf{lbres}_i \leq \mathsf{res}_i \leq \mathsf{ubres}_i. \tag{3}$$

[3] We discuss our approach in Sect. 4.4, alternatively, one may use bounds from, e.g., [4].

These bounds on properties in the subMDP correspond to bounds on the parameters of the uncertain macro-level MDP $\nu(\mathcal{M})$. Let us formalize this idea.

Definition 9 (Suitable parameter region). *Given u^* from Theorem 2. The bounds u^-, u^+ are* suitable *if $u^- \leq u^* \leq u^+$. For suitable u^-, u^+, the region $[[u^-, u^+]]$ is called* suitable.

Using this notion, sound bounds lbres_i and ubres_i thus yield suitable bounds $u^-(x), u^+(x)$ for all $x \in \bigcup_j p_{i,j} \cup \{q_i\}$. Combined, the sound bounds for every i yields a suitable region. Formally:

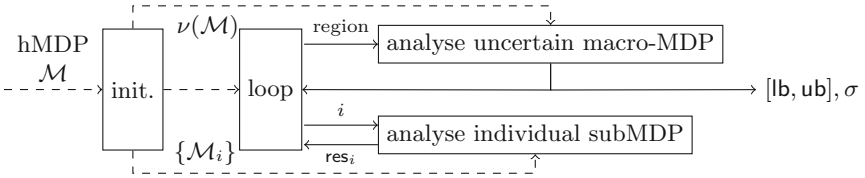


Fig. 4. Analysing hMDPs via uncertain macro-MDPs via individual refinement.

Lemma 1. *Given sound bounds $\mathsf{lbres}_i, \mathsf{ubres}_i$ for each i, there exists a trivial mapping Reg s.t. $\mathsf{Reg}(\mathsf{lbres}_1, \ldots \mathsf{lbres}_n, \mathsf{ubres}_1, \ldots \mathsf{ubres}_n)$ is a suitable region.*

With the suitable region we can apply verification on the parametric MDP.

Lemma 2. *Let R be a suitable region. Then:*

$$\min_{u \in R} \mathsf{ER}^{\mathsf{max}}_{\nu(\mathcal{M})[u]}(\Diamond T) \leq \mathsf{ER}^{\mathsf{max}}_{\mathcal{M}}(\Diamond T) \leq \max_{u \in R} \mathsf{ER}^{\mathsf{max}}_{\nu(M)[u]}(\Diamond T).$$

Proof sketch. We observe that the inequalities follow from the fact that $u^* \in R$ with u^* as in Theorem 2. By that theorem, $\mathsf{ER}^{\mathsf{max}}_{\nu(\mathcal{M})[u^*]}(\Diamond T) = \mathsf{ER}^{\mathsf{max}}_{\mu(\mathcal{M})}(\Diamond T)$. The statement then follows from Theorem 1. □

From the bounds that we can compute using a suitable region, we then set lb and ub for Eq. (2):

$$\mathsf{lb} \leq \min_{u \in R} \mathsf{ER}^{\mathsf{max}}_{\nu(\mathcal{M})[u]}(\Diamond T) \leq \mathsf{ER}^{\mathsf{max}}_{\mathcal{M}}(\Diamond T) \leq \max_{u \in R} \mathsf{ER}^{\mathsf{max}}_{\nu(M)[u]}(\Diamond T) \leq \mathsf{ub}. \tag{4}$$

Computationally, we may use parameter lifting [33] to find these values.

Refinement Loop. The complete anytime algorithm is summarized in Fig. 4. We start with an hMDP $\mathcal{M}$ and extract the uncertain macro-MDP $\nu(\mathcal{M})$ and the subMDPs $\{\mathcal{M}_i\}$[4]. Furthermore we compute (trivial) sound bounds on $\mathsf{lbres}_i \leq \mathsf{res}_i \leq \mathsf{ubres}_i$. This leads to a suitable region $[[u^-, u^+]] = \mathsf{Reg}(\mathsf{lbres}_1, \mathsf{ubres}_1, \ldots)$. Then, we may at any time compute the bounds lb, ub on the expected reward

[4] For efficiency, one must implement extraction without first computing an explicit representation of $\mathcal{M}$.

in the hMDP $\mathcal{M}$ by analysing $\nu(\mathcal{M})$ on the region $[[u^-, u^+]]$. To tighten these bounds, we must first refine the suitable region. Therefore, we analyse individual subMDPs $\mathcal{M}_i$ and compute res_i and thus $u^*(x)$ for $x \in \cup_j p_{i,j} \cup q_i$. This refines the suitable bounds such that $u^-(x) = u^*(x) = u^+(x)$ for $x \in \cup_j p_{i,j} \cup q_i$. We call this refinement *individual refinement*. The new region is suitable and Theorem 2 ensures correctness of the refinement. As we only have finitely many subMDPs, we obtain $\mathsf{lb} = \mathsf{ub}$ after finitely many steps.

Anytime version of the enumeration baseline. Individually refine any subset of subMDPs, then analyse the uncertain macro-MDP $\nu(\mathcal{M})$.

4.3 Set-Based SubMDP Analysis

Next, we aim to provide an alternative refinement procedure that analyses a set of subMDPs at once, i.e., that refines the suitable bounds for a set of parameters at once. We denote the set of goal states for all subMDPs as G[5].

Adequate Abstractions. We aim to compute sound bounds on the results for a set of subMDPs such that the bounds are sound for every individual subMDP in this set. We generalize Definition 8 as follows: The (lower and upper) bounds $\mathsf{lbres}_I, \mathsf{ubres}_I$ are *sound*, if they are sound (lower and upper) bounds for every res_i, $i \in I$.

Lemma 3. *Let lbres_I satisfy the following inequations using $0 \leq j < Y$:*

$$\mathsf{lbres}_I(Y) \leq \min_i ER^{\mathsf{max}}_{\mathcal{M}_i}(\lozenge G) \quad \text{and} \quad \mathsf{lbres}_I(j) \leq \min_i \min_\sigma Pr_{\mathcal{M}_i[\sigma]}(\lozenge G). \tag{5}$$

Then, lbres_I is a sound lower bound.

Proof sketch. We must show $\mathsf{lbres}_I \leq \mathsf{res}_i$ for each $i \in I$. By definition for each $1 \leq j \leq Y$, $\mathsf{lbres}_I(j) \leq \min_{i' \in I} \mathsf{res}_{i'}(j)$ and trivially $\min_{i' \in I} \mathsf{res}_{i'}(j) \leq \mathsf{res}_i(j)$. □

We omit the analogous statement for ubres[6]. In Sect. 4.4, we discuss a particular approach to obtain these bounds, i.e., the right hand sides of the equations in Eq. 5. Here, we update the algorithm sketch to handle this alternative refinement.

Remark 3. We cannot compute the optimal policy σ_i for the subMDP $\mathcal{M}_i$ in this setting. Thus, we must compute probability bounds for all policies, which may make these bounds weak. Some optimizations are possible as some actions can in fact be excluded. More importantly, however, is that for cases within Proposition 1 the policy σ_i is irrelevant.

[5] Formally, we label the goal states and use G to refer to denote those states.
[6] where min becomes max and inequalities flip.

Updated Algorithm. We update the loop from Fig. 4: Rather than refining using a single i, we refine using a set I. Instead of res_i, we use Lemma 3 to compute sound bounds $\mathsf{lbres}_I, \mathsf{ubres}_I$ and call this *set-based refinement*. We may set $\mathsf{lbres}_i = \mathsf{lbres}_I$ for each $i \in I$. Then, we can compute a new suitable region via Lemma 1. With the suitable region, we can still utilise Eq. (4) to compute an approximation $[\mathsf{lb}, \mathsf{ub}]$. However, for completeness we must ensure that if $|I| = 1$, the upper and lower bounds coincide, i.e., $\mathsf{lbres}_{\{i\}} = \mathsf{ubres}_{\{i\}}$ for every i. That can be ensured by using individual subMDP refinement when $|I| = 1$.

Idea: We may improve the anytime algorithm by iteratively considering sets of subMDPs and extract sound bounds.

We now first discuss the set-based analysis of multiple subMDPs $\mathcal{M}_i$. We clarify the realization of the loop box in Sect. 5.

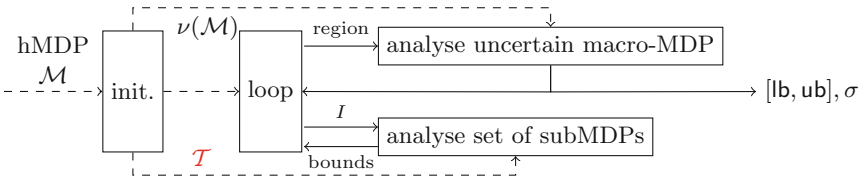


Fig. 5. Analysing hMDPs with set-based refinement on templated subMDPs.

4.4 Templates for Set-Based subMDP Analysis

We present an instance of set-based subMDP analysis where the subMDPs can be described as instantiations of a parametric MDPs.

Parametric Templates. We observe that the subMDPs are often similar, e.g., they define sending a file over a channel, exploring a room, in different conditions. We capture this similarity as follows: Let $\{\mathcal{T}_1, \dots \mathcal{T}_m\}$ define a set of parametric MDPs, where we call each pMDP a *template*. In particular, for a hierarchical MDP $\mathcal{M}$ with partitioning $\mathbf{S}_1, \dots \mathbf{S}_n$ and corresponding subMDPs $\mathcal{M}_1, \dots, \mathcal{M}_n$ a subMDP $\mathcal{M}_i$ is an instantiation of template $\mathcal{T}_j$ and parameter instantiation v[7], if $\mathcal{M}_i = \mathcal{T}_j[v]$. For a concise description, this paper considers hMDPs over a single template $\mathcal{T}$ and, for any $I \subseteq \mathbb{I}$, we denote $V_I := \{v_1, \dots, v_n\}$ the finite (multi)set of parameter instantiations for the pMDP $\mathcal{T}$ such that $\mathcal{T}[v_i] = \mathcal{M}_i$.

Abstractions from Templates. In terms of the templates, Lemma 3 requires us to bound the expected rewards $\mathsf{ER}^{\mathsf{max}}_{\mathcal{T}[v]}(\Diamond G)$ for all $v \in V_I$. We realize this by defining the smallest region $\mathsf{toRegion}(V_I) \supseteq V_I$. For this region, we obtain expected rewards by computing the minimum maximal reward in $\mathsf{toRegion}(V_I)$. That is:

$$\mathsf{lbres}_I(Y) := \min_{v \in \mathsf{toRegion}(V_I)} \mathsf{ER}^{\mathsf{max}}_{\mathcal{T}[v]}(\Diamond G) \quad \leq \quad \min_i \mathsf{ER}^{\mathsf{max}}_{\mathcal{M}_i}(\Diamond G).$$

[7] We use v instead of u to avoid confusion with the instantiations for pMDP $\nu(\mathcal{M})$.

We handle the probabilities equally while taking into account the quantification over the policies. Following Lemma 3, these bounds are sound. Upper bounds are handled analogously. Computationally, we again use parameter lifting [33] to find these bounds. We can easily refine: Whenever we split I (or equally, V_I), we can compute (potentially) smaller regions $\mathsf{toRegion}(V_I)$.

In Fig. 5, we depict our method. In contrast to Fig. 4, we pass the template $\mathcal{T}$ rather than the individual subMDPs. Furthermore, we now compute initial sound bounds via the analysis of the template (i.e., of V_I) and must pass the mapping from I to V_I to clarify the shape of the subMDPs.

Abstraction-Refinement on the subMDPs provides increasingly tight suitable regions for the uncertain macro-MDP from the anytime baseline.

Algorithm 1. Algorithm for Abstraction-Refinement Procedure

Construct macro-MDP $\nu(\mathcal{M})$, class-MDP $\mathcal{T}$, and $V_{\mathbb{I}}$ from high-level description.
$Q \leftarrow \{\langle I = \mathbb{I}, \text{bounds} = [0, \infty), \text{weightedvals} = \mathbb{I} \to \{1\}\rangle\}$
$\mathsf{lb} \leftarrow 0$; $\mathsf{ub} \leftarrow \infty$; #iter $= 0$; Res $\leftarrow \emptyset$
while $\eta \cdot \mathsf{ub} > \mathsf{lb}$ **do**
  $R \leftarrow Q.pop()$ ▷ Use priority
  if $R.I = \{i\}$ **then**
    $\text{Res}[i] \leftarrow check_one(\mathcal{T}[v_i])$ ▷ Computes res_i
  else
    $R.bounds \leftarrow check_set(\mathcal{T}, \mathsf{toRegion}(V_{R.I}))$ ▷ Computes $\mathsf{lbres}_{R.I}, \mathsf{ubres}_{R.I}$
    $Q \leftarrow Q \cup split(R)$ ▷ Split $R.I$, keep bounds and weights
  end if
  if #iter $mod\ k = 1$ or Q is empty **then**
    $R' \leftarrow \mathsf{Reg}(extract(Q, \text{Res}))$ ▷ Compute suitable region via Lem 1
    $\mathsf{lb}, \mathsf{ub} \leftarrow check_set(\nu(\mathcal{M}), R')$
  end if
end while

5 Implementing the Abstraction-Refinement Loop

Algorithm 1 outlines a basic implementation of the idea sketched in Fig. 5. We detail this implementation and then discuss an essential improvement.

We construct $\nu(\mathcal{M})$, $\mathcal{T}$, and (the implicit) mapping $V : \mathbb{I} \to V_{\mathbb{I}}$ to map subMDPs to instantiations of $\mathcal{T}$ from a suitable high-level representation. We initialize a priority queue with triples that represent sets of template instantiations: I such that $V_I := \{v_i := V(i) \mid i \in I\}$ contains all valuations v such that $\mathcal{T}[v]$ is a subMDP of $\mathcal{M}$. We initially store bounds reflecting lbres_I and ubres_I as well as weights for the computation of the priority (see below). Initially, we assume that $\mathsf{lb} = 0$ and $\mathsf{ub} = \infty$, we count the number of iterations in #iter. Res is map for storing result vectors. The algorithm now refines lb and ub until the gap between lb and ub is sufficiently small.

The main loop now iteratively refines lb, ub by first refining lbres_I and ubres_I, by splitting I and model checking $\mathcal{T}$ w.r.t. subsequently smaller regions $\mathsf{toRegion}(V_I)$ (l. 5-11): Therefore, we take a set R from the queue. If $R.I = \{i\}$ is a singleton, we compute $\mathsf{lbres}_{R.I} = \mathsf{res}_i = \mathsf{ubres}_{R.I}$ and store this result. Otherwise, we apply model checking to the pMDP $\mathcal{T}$ w.r.t. the region representation of $R.I$. We then split $R.I$, by splitting I into (here) two subsets. For splitting I, we use the geometric interpretation of $\mathsf{toRegion}(V_I)$ as a subset of $\mathbb{R}^{|\vec{y}|}$, where we then split along one of the axis into two equally large subsets. Every k (we use $k = 8$) iterations, we analyse the macro-MDP (l. 12-15). From Q and Res we extract the proper bounds $\mathsf{lbres}_i, \mathsf{ubres}_i$ from $Res[i]$ if possible and from Q using R.bounds for R such that $i \in R.I$ otherwise. Then via $\mathsf{Reg}(\mathsf{lbres}_1, \mathsf{ubres}_1, \ldots)$ from Lemma 1 we compute a suitable region R'. We analyse the uncertain macro-MDP to obtain lb and ub in accordance with Eq. (4).

Finally, we discuss the priority function: If we a-priori naively assume that each subMDP contributes an equal amount to the overal minimal expected reward in the hMDP (weights are all one) then the following priority function: $|R.\text{bounds}| \cdot \sum_{v \in I} \text{R.weights}(v)$ computes priorities that correlate with how much computing res_i for all $i \in I$ would reduce the gap between lb and ub.

Termination and Correctness Argument. Algorithm 1 terminates. We split in such way that $\max_{I \in Q} |I|$ monotonically decreases. Thus, eventually Q is empty and Res contains results for all subMDPs. Then, R' is a point region and checking $\nu(\mathcal{M})$ with this point region ensures that $\mathsf{lb} = \mathsf{ub}$. Correctness follows as R' is always suitable, see Eq. (4).

Computing Expected Visits. Based on our empirical evaluation we added one crucial improvement: While the algorithm above assumed that all subMDPs (or states in the macro-MDP) are equally important, that assumption is generally inadequate. Roughly, only states reached by the optimal policy contribute at all (provided the bounds are tight enough that we can identify these states). The reachable states are weighted by the expected number of visits of these states. We compute an approximation of this expected number of visit by computing the currently optimizing policy (a by-product of l. 13) and compute the center of R'; this results in a MC for which we can compute the number of expected visits by a standard equation system [32]. Additionally, we update the weights for the regions in the queue based on these new results. We remark that this also makes the priority function more useful.

Interleaving Individual Refinement. Furthermore, for a subMDPs for which the expected number of visits is large[8] are individually analysed (and the points are removed from the region in the queue). This optimization reduces the need to split the corresponding regions until we obtain tight bounds.

[8] In our implementation, we define this as subMDPs where the expected number of visits is in the top $1 + 1/16 \cdot \#\text{iter}$ percent, but not more than 150 at a time.

6 Experiments

Implementation. We implemented LEVEL-UP[9], a prototype on top of the python bindings for Storm [20]. LEVEL-UP analyses hierarchical MDPs by taking two MDPs, each provided as probabilistic program descriptions in the PRISM format: One MDP that encodes the (uncertain) macro-MDP and one that describes the parametric template for the subMDPs. The parameter instance of the subMDP can be deduced as a function of the high-level variable assignment of the macro-MDP states. For technical reasons, the prototype currently provides support for subMDPs with one or two successor states – arguably the setting in which we expect our prototype to perform best. For subMDPs with a single successor state, the uncertain macro-MDP may be represented as an (parameter-free) MDP with interval-valued rewards. For two successors, we include support of the extension of Sect. 3.3 where the successor aims to optimize reaching a fixed successor state.

Table 1. Benchmark statistics, runtimes of the approaches, and details for Algorithm 1.

Name	Inst	$\lvert S_{\mathcal{M}}\rvert$	$\lvert\mathbb{I}\rvert$	$\lvert S_{\mu(\mathcal{M})}\rvert$	$\lvert \mathrm{A}_{\mu(\mathcal{M})}\rvert$	$\lvert S_{\mathcal{T}}\rvert$	$\lvert \mathrm{A}_{\mathcal{T}}\rvert$	t_{init}	t_{enum}	t_{50}	t_{90}	t_{95}	iter.	indrf.	$^{\text{um}}/_{\%}$	$^{\text{sr}}/_{\%}$	$^{\text{ir}}/_{\%}$
corr	11,10,50	10^7	624	255576	541704	15000	65006	<1	16	3	9	**13**	17	14	2	67	2
corr	11,8,100	10^8	624	254376	539040	60000	260006	<1	100	10	45	**45**	9	16	2	80	4
corr	11,8,200	10^8	624	254376	539040	240000	1040006	2	689	51	313	**568**	17	30	0	92	4
corr	13,11,50	10^7	768	1024344	2172432	15000	65006	3	21	8	**18**	25	17	17	5	36	1
corr1	17,14,75	10^8	1056	34200	83160	33750	146256	<1	90	4	21	**38**	17	43	0	84	8
corr1	18,15,75	10^8	1128	39576	96768	33750	146256	<1	98	4	38	**38**	17	45	0	84	8
corr1	25,20,75	10^8	1632	89136	224160	33750	146256	<1	168	5	44	**67**	25	102	1	80	14
mail	10	10^9	173857	793971	1088152	2801	3601	4	552	8	21	**48**	57	658	29	2	4
mail	12	10^9	236802	1446551	2023504	2801	3601	8	738	16	43	**130**	97	703	42	1	2
netw	30,50	10^8	9801	437823	437823	4026	4026	1	23	**8**	33	46	217	150	60	1	1
netw	30,80	10^8	9801	437823	437823	10041	10041	1	62	8	34	**48**	217	150	59	3	3
netw	50,80	10^8	9801	1025883	1025883	10041	10041	2	62	**16**	94	112	225	150	62	1	1
sdn	5,12,4,4	10^8	23375	128386	128386	13506	16855	<1	62	2	**20**	112	289	305	2	17	11
sdn	5,8,4,4	10^8	23375	128386	128386	2802	3455	<1	98	1	5	**15**	281	305	13	17	8
sdn	6,8,4,4	10^9	126337	408227	408227	2802	3455	2	519	5	46	**394**	3057	305	27	7	0

Setup. We investigate the scalability and the quality of the approximation over time. Therefore, we run our prototype on an MacBook 2020 M1 with an 8 GB RAM limit. We compare the enumerative baseline from Sect. 4.1 with Algorithm 1. Both exploit the hierarchical nature of the MDP. We qualitatively compare to standard model checking on the flat MDP, see below. We use a collection of benchmarks reflecting networks, job schedulers and robots.

Results. We consider instances that we summarize in Table 1. In particular, we give the benchmark name and instance for reference, the approximate number of states in the hierarchical MDP (computed from the macro-MDP and the

[9] The source code and executables, the benchmarks, logfiles and utilities are all available in an archived Docker container: https://doi.org/10.5281/zenodo.6524787.

subMDPs), the number of nontrivial partitions, and the number of states and actions in the (uncertain) macro-MDP and subMDPs, respectively. Then, we give the time to setup the data structures from the high-level representation t_{init} in seconds. We highlight that a flat representation of all our benchmarks has at least 10^7, often more, states. As a reference, we present the performance of the enumerative baseline from Sect. 4.1. The performance of this approach is positive as it enables the verification of huge MDPs. A TO indicates >1200 s. To scale to either larger subMDPs or more subMDPs, we use the abstraction-refinement loop. To reflect the anytime nature, we list three run times for terminating when $\eta \cdot \mathsf{ub} \leq \mathsf{lb}$ with $\eta \in \{0.5, 0.9, 0.95\}$ respectively. The largest time faster than the enumerative baseline is highlighted (further to the right is better for the abstraction-refinement). For $\eta = 0.95$, we give details: The number of iterations (iter), the number of individual refinements based on the improvement from Sect. 5, and the fraction of time spent on model checking the uncertain macro-MDPs $\%_{\text{um}}$, the set-refinements $\%_{\text{sr}}$, and the individual refinements $\%_{\text{ir}}$, respectively.

Discussion. Before we discuss details of the results, let us clarify that *exploiting the hierarchical structure is essential.* MDPs with $\approx 10^8$ states are at the limit of what fits in around 8GB of memory[10]. Symbolic methods based on MTBDDs easily scale beyond these sizes, but—noting that the subMDPs are all slightly different—the models we consider lack the necessary symmetry that make MTBDDs compact. Thus, support for hierarchical MDPs is a necessary step forward.

Regarding the abstraction-refinement: While a larger study may be necessary, we can start with two standard observations: The abstraction-refinement loop is significantly faster on $\eta \leq 0.9$. As $\eta \to 1$, coarse abstractions are insufficient. Furthermore, the efficiency of the abstraction-refinement heavily depends on the particular structure. That being said, the approach outperforms the enumerative approach, especially for $\eta = 0.9$, and up to more than an order of magnitude. This happens even if $\mathbb{I}$ is rather small, or if, e.g., $\mathcal{T}$ is small. We furthermore observe that for large $\mathbb{I}$, the bookkeeping in python becomes a bottleneck. We think these observations are promising: we left many options for further optimizations and tweaking towards particular examples on the table. However, for models where most time is spent on model checking the macro-level MDP, the approach is less suitable. We furthermore conjecture that tailored algorithms may exploit some of these dimensions, e.g., when there is the macro-MDP or the subMDPs are indeed MCs or perhaps acyclic, depending on the number of parameters and their influence [36], or based on the relative weight of the uncertain rewards compared to rewards in the macro-MDP.

7 Related Work

In the model-free reinforcement learning (RL) setting, hierarchical models are popular. An excellent, recent survey is given in [29]. Our work generalizes the

[10] Assuming 128 byte per state, i.e., 8 doubles and 16 (32-bit) ints, as used in Storm.

solution techniques on hierarchical MDPs that assume that these subMDPs are the same. In RL, this assumption is treated liberally, and the methods provide only weak error bounds. In contrast, our model-based approach provides error-bounds in every step, and the error disappears in finitely many steps.

Hierarchical abstractions are used to analyse large MDPs in [5]. There, the goal is to find a policy that almost optimizes the reward. Rather than preimposing a hierarchy, the algorithm aims to find a hierarchy and define the goal states of the subMDP such that the model admits local policies. Instead, our solution can find the optimal policy and in particular gives strict error bounds at the cost of requiring a high-level model that induces the hierarchy. An symbolic approach for continuous MDP, where the transition probabilities are the result of an associated LP, has recently been discussed in [24]. An hierarchical SCC-decomposition [1] aims to accelerate the process of solving a (given, monolithic) Markov chain. The computation of reward-bounded properties [18] generalizes topological value iteration and their notion of episodes mildly resembles an hierarchical approach but no uncertainty is assumed or used in the approach. The probabilistic model checker PAT [35] analyses a hierarchical probabilistic timed automaton given as a process algebra. The hierarchy is not exploited in the solving process.

While symbolic approaches, often on decision diagrams, exploit the transition system by compressing the data structures, abstractions aim to yield smaller systems that may assess an approximation for the sought-for values. Abstraction-refinement without an imposed hierarchy is explored in [16,21,25]: Refinement amounts to considering a better approximation of the state space. In contrast, we impose the hierarchy, the abstraction amounts to an imprecise analysis of this fixed state space and we refine by analysing the state space more precisely (by means of analysing subMDPs at a greater level of detail). Contract-based abstractions (in probabilistic systems) are used to decompose the analysis of systems given by parallel running subsystems [14,28,38]. Partial exploration and bounded model checking approaches focus on the most critical paths, i.e., the paths where most of the probability mass lies [7,23,26], but these approaches do generally not exploit the hierarchical and repetitive structure. The observation that many parts of the system are not critical allows us to weigh the potential benefit of refining the intervals in various parts of the macro-MDP.

Parametric MDPs are commonly used to model and analyse the effects of uncertainty in the precise transitions [15,23,31]. The methods presented in [13,22] exploit a repetitive structure in parametric MCs to accelerate the construction of closed form solutions and are not applicable to MDPs. Parametric models have been used to support the design of systems [2,8] or their adaption [6,9], to find policies for partially observable systems [11], to analyse Bayesian networks [34], and to speed up the analysis of, e.g., software product lines [10,37]. On top of technical differences, none of these approaches uses a hierarchical decomposition of an MDP or uses the results of the analysis in the analysis of a larger MDP.

8 Conclusion

This paper presents a first verification approach that exploits a specific hierarchical structure natural in many models to accelerate analysing the underlying MDP. An essential ingredient is to separate the two levels in the hierarchy. Then, when analysing the (toplevel) macro-MDP, we may consider subMDPs that have not yet been analysed as epistemic uncertainty. Analysis techniques for uncertain (more precise: parametric) MDPs then enable an online approximation loop that incrementally removes uncertainty in a targeted fashion by analysing more and more subMDPs (more) precisely. Three clear directions for future work are to (i) consider an approach where one lifts the restrictions to locally-optimal policies, (ii) investigate the applicability to a richer set of temporal properties and (iii) to allow automatic detection of partitions in, e.g., the Prism language.

References

1. Ábrahám, E., Jansen, N., Wimmer, R., Katoen, J.-P., Becker, B.: DTMC model checking by SCC reduction. In: QEST, pp. 37–46. IEEE CS (2010)
2. Andriushchenko, R., Češka, M., Junges, S., Katoen, J.-P.: Inductive synthesis for probabilistic programs reaches new horizons. In: TACAS 2021. LNCS, vol. 12651, pp. 191–209. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72016-2_11
3. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)
4. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: interval iteration for markov decision processes. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 160–180. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_8
5. Barry, J.L., Kaelbling, L.P., Lozano-Pérez, T.: DetH*: approximate hierarchical solution of large Markov decision processes. In IJCAI, pp. 1928–1935. IJCAI/AAAI (2011)
6. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 326–340. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_30
7. Brázdil, T., et al.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_8
8. Calinescu, R., Ceska, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Efficient synthesis of robust models for stochastic systems. J. Syst. Softw. **143**, 140–158 (2018)
9. Chen, T., Hahn, E.M., Han, T., Kwiatkowska, M.Z., Qu, H., Zhang, L.: Model repair for Markov decision processes. In: TASE, pp. 85–92. IEEE CS (2013)
10. Chrszon, P., Dubslaff, C., Klüppelholz, S., Baier, C.: ProFeat: feature-oriented engineering for family-based probabilistic model checking. Formal Aspects Comput. **30**(1), 45–75 (2018)
11. Cubuktepe, M., Jansen, N., Junges, S., Marandi, A., Suilen, M., Topcu, U.: Robust finite-state controllers for uncertain POMDPs. In: AAAI, pp. 11792–11800. AAAI Press (2021)

12. Dombrowski, C., Junges, S., Katoen, J.-P., Gross, J.: Model-checking assisted protocol design for ultra-reliable low-latency wireless networks. In: SRDS, pp. 307–316. IEEE CS (2016)
13. Fang, X., Calinescu, R., Gerasimou, S., Alhwikem, F.: Fast parametric model checking through model fragmentation. In: ICSE, pp. 835–846. IEEE (2021)
14. Feng, L., Han, T., Kwiatkowska, M., Parker, D.: Learning-based compositional verification for synchronous probabilistic systems. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 511–521. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_40
15. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PARAM: a model checker for parametric Markov models. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 660–664. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_56
16. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PASS: abstraction refinement for infinite probabilistic models. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 353–357. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_30
17. Hartmanns, A., Hermanns, H.: The modest toolset: an integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 593–598. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_51
18. Hartmanns, A., Junges, S., Katoen, J.-P., Quatmann, T.: Multi-cost bounded tradeoff analysis in MDP. J. Autom. Reason. **64**(7), 1483–1522 (2020)
19. Hauskrecht, M., Meuleau, N., Kaelbling, L.P., Dean, T.L., Boutilier, C.: Hierarchical solution of Markov decision processes using macro-actions. In: UAI, pp. 220–229. Morgan Kaufmann (1998)
20. Hensel, C., Junges, S., Katoen, J.-P., Quatmann, T., Volk, M.: The probabilistic model checker storm. CoRR, abs/2002.07080 (2020)
21. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 162–175. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70545-1_16
22. Jansen, N., et al.: Accelerating parametric probabilistic verification. In: Norman, G., Sanders, W. (eds.) QEST 2014. LNCS, vol. 8657, pp. 404–420. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10696-0_31
23. Jansen, N., Dehnert, C., Kaminski, B.L., Katoen, J.-P., Westhofen, L.: Bounded model checking for probabilistic programs. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 68–85. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_5
24. Jeong, J., Jaggi, P., Sanner, S.: Symbolic dynamic programming for continuous state MDPs with linear program transitions. In: IJCAI, pp. 4083–4089. ijcai.org (2021)
25. Kattenbelt, M., Kwiatkowska, M.Z., Norman, G., Parker, D.: A game-based abstraction-refinement framework for Markov decision processes. Formal Methods Syst. Des. **36**(3), 246–280 (2010)
26. Kretínský, J., Meggendorfer, T.: Of cores: a partial-exploration framework for Markov decision processes. Log. Methods Comput. Sci. **16**(4) (2020)
27. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47

28. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 23–37. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_3
29. Pateria, S., Subagdja, B., Tan, A.-H., Quek, C.: Hierarchical reinforcement learning: a comprehensive survey. ACM Comput. Surv. **54**(5), 109:1–109:35 (2021)
30. Precup, D., Sutton, R.S.: Multi-time models for temporally abstract planning. In: NIPS, pp. 1050–1056. The MIT Press (1997)
31. Puggelli, A., Li, W., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Polynomial-time verification of PCTL properties of MDPs with convex uncertainties. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 527–542. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_35
32. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, Hoboken (1995)
33. Quatmann, T., Dehnert, C., Jansen, N., Junges, S., Katoen, J.-P.: Parameter synthesis for Markov models: faster than ever. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 50–67. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_4
34. Salmani, B., Katoen, J.-P.: Fine-tuning the odds in Bayesian networks. In: Vejnarová, J., Wilson, N. (eds.) ECSQARU 2021. LNCS (LNAI), vol. 12897, pp. 268–283. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86772-0_20
35. Song, S., Sun, J., Liu, Y., Dong, J.S.: A model checker for hierarchical probabilistic real-time systems. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 705–711. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_53
36. Spel, J., Junges, S., Katoen, J.-P.: Finding provably optimal Markov chains. In: TACAS 2021. LNCS, vol. 12651, pp. 173–190. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72016-2_10
37. ter Beek, M.H., Legay, A.: Quantitative variability modelling and analysis. Int. J. Softw. Tools Technol. Transfer **21**(6), 607–612 (2019). https://doi.org/10.1007/s10009-019-00535-1
38. Xu, D.N., Gössler, G., Girault, A.: Probabilistic contracts for component-based design. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 325–340. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15643-4_24

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.