Magnifying-Lens Abstraction for Markov Decision Processes*

Luca de Alfaro and Pritam Roy

Computer Engineering Department
University of California, Santa Cruz, USA

Abstract. We present a novel abstraction technique which allows the analysis of reachability and safety properties of Markov decision processes with very large state spaces. The technique, called *magnifying-lens abstraction,* (MLA) copes with the state-explosion problem by partitioning the state-space into regions, and by computing upper and lower bounds for reachability and safety properties on the regions, rather than on the states. To compute these bounds, MLA iterates over the regions, considering the concrete states of each region in turn, as if one were sliding across the abstraction a magnifying lens which allowed viewing the concrete states. The algorithm adaptively refines the regions, using smaller regions where more detail is needed, until the difference between upper and lower bounds is smaller than a specified accuracy. We provide experimental results on three case studies illustrating that MLA can provide accurate answers, with savings in memory requirements.

1 Introduction

Markov decision processes (MDPs) provide a model for systems with both probabilistic and nondeterministic behavior, and they are widely used in probabilistic verification, planning, optimal control, and performance analysis [13,4,26,8,10]. MDPs that model realistic systems tend to have very large state spaces, and the main challenge in their analysis consists in devising algorithms that work efficiently on such large state spaces. In the non-probabilistic setting, abstraction techniques have been successful in coping with large state-spaces: abstraction enables to answer questions about a system by considering a smaller, more concise abstract model. This has spurred research into the use of abstraction techniques for probabilistic systems [7,18,22,19]. We present a novel abstraction technique, called *magnifying-lens abstraction* (MLA), for the analysis of reachability and safety properties of MDPs with very large state spaces. We show that the technique can lead to substantial space savings in the analysis of MDPs.

An MDP is defined over a state space S. At every state $s \in S$, one or more *actions* are available; with each action is associated a probability distribution over the successor states. We focus on *safety* and *reachability* properties of MDPs. A safety property specifies that the MDP's behavior should not leave a *safe* subset of states $T \subseteq S$; a reachability property specifies that the behavior should reach a set $T \subseteq S$ of *target* states. A controller can choose the actions available at each state so as to maximize, or minimize, the probability of satisfying reachability and safety properties. MLA computes

* This research was sponsored in part by the grant NSF-CCR-0132780.

W. Damm and H. Hermanns (Eds.): CAV 2007, LNCS 4590, pp. 325–338, 2007.

converging upper and lower bounds for the maximal reachability or safety probability; the minimal probabilities can be obtained by duality. In its ability to provide both upper and lower bounds for the quantities of interest, MLA is similar to [19].

In the analysis of large MDPs, the main challenge lies in the representation of the value $v(s)$ of the reachability or safety probability at all $s \in S$. In contrast, actions and transition probabilities from each state s can usually be either computed on the fly, or represented in a compact fashion, via Kronecker representations or probabilistic guarded commands [23,10,17]. The goal of MLA is to reduce the space required for storing v and, secondarily, the running time of the analysis. To this end, MLA partitions the state space S of the MDP into *regions;* for each region r, it stores upper and lower bounds $v^{+}(r)$, $v^{-}(r)$ for the maximal reachability or safety probability. The values $v^{+}(r)$, $v^{-}(r)$ constitute bounds for all states $s \in r$. In order to update these estimates, MLA iterates over the regions, "magnifying" one of them at a time. When the region r is magnified, MLA computes $v^{+}(s)$, $v^{-}(s)$ at all concrete states $s \in r$ via value iteration, and then summarizes these results by setting $v^{+}(r) = \max_{s \in r} v^{+}(s)$ and $v^{-}(r) = \min_{s \in r} v^{-}(s)$. Figuratively, MLA slides a magnifying lens across the abstraction, enabling the algorithm to see the concrete states of one region at a time when updating the region values. Given a desired accuracy ε for the answer, MLA periodically splits regions r with $v^{+}(r) - v^{-}(r) > \varepsilon$ into smaller regions. In this way, the abstraction is refined in an adaptive fashion: smaller regions are used where finer detail is needed, guaranteeing the convergence of the bounds, and larger regions are used elsewhere, saving space. When splitting regions, MLA takes care to re-use information gained in the analysis of the coarser abstraction in the evaluation of the finer one. MLA can be adapted to the problem of computing a control strategy by recording the optimal actions for the concrete states of interest, when they are magnified.

Related work on MDP abstraction. Compared with other approaches to MDP abstraction, MLA has two distinctive features:

1. it clusters states based on value, rather than based on the similarity in their transition function;
2. it updates the valuation of abstract states by considering the concrete states associated with the abstract states, rather than by considering an abstract model only.

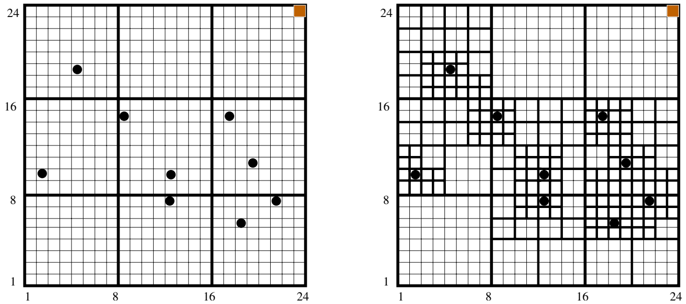
The second of the above points underlines how MLA is a semi-abstract, rather than fully abstract, approach to verification: the abstract computation still involves consideration of the concrete states, even though this is done in a way that provides space savings.

For the most part, approaches to MDP abstraction in the literature have followed another route, which we call very broadly the *full abstraction* approach: an abstract model is constructed, and then analyzed on the basis of an abstract transition structure, without further reference to the concrete model. These fully abstract approaches generally rely on clustering states that are similar not only in value, but also in transition structure: in this way, every region of concrete states can be summarized via an abstract state with an associated abstract transition structure. The abstract transition structure may, or may not, be similar to the concrete one. For instance, [19] bases the abstract transition structure on games, rather than MDPs: in this fashion, player 1 can represent the choice of action of the MDP, and player 2 can represent the uncertainty about the concrete state corresponding to the abstract state. This approach enables the computation of lower and upper bounds for properties of interest, similarly to MLA. In a somewhat related spirit, but using entirely different technical means, [14] proposes to abstract Markov chains into *abstract Markov chains* whose transitions are labeled with intervals of probability, representing the uncertainty about the concrete state. Clustering states based on the similarity in their transition probabilities has also been used in [12], which proposes to find the coarsest refinement of an MDP where for each action, states in the same region have the same probability of going to other regions. An approach for the verification of probabilistic reachability properties via abstraction has been proposed in [7]. The abstraction is built through successive refinements starting from a coarse partition based on the property. Several other approaches also, in fact, rely on constructing MDP abstractions based on simulation or abstract interpretation [18,22,21]; all of these approaches rely on clustering states with similar transition structure, and representing these clusters of states, and their transition structures, via compact abstract representations.

The full-abstraction approach outlined above, and the partial value-based approach followed by MLA, each have advantages. The full-abstraction result can handle unbounded, and (depending on the specific approach) even infinite state spaces. In contrast, the space savings afforded by MLA are limited to a square-root factor (a system of size n can be studied in $O(\sqrt{n})$ space), due to the need to consider the concrete states corresponding to each abstract one. Furthermore, the full-abstraction approaches typically need to construct the abstract model only once; in contrast, MLA needs to refer to concrete states (albeit not all of them at once) during the computation.

On the other hand, the ability of MLA to cluster states based on value only, disregarding differences in their transition relation, can lead to compact abstractions for systems where full abstraction provides no benefit. We will give below an example supporting this. Furthermore, in MLA the abstraction is refined dynamically, depending on the required accuracy of the analysis; there is no need to "guess" the right state partition in advance. In our experience, MLA is particularly well-suited to problems where there is a notion of *locality* in the state space, so that it makes sense to cluster states based on variable values — even though their transition relations may not be similar. Many planning and control problems are of this type. MLA instead is not as well-suited to problems where clustering states based on variable values is less effective. Approaches based on predicate abstraction could lend the MLA approach more generality.

An example of Magnifying-Lens Abstraction. To illustrate MLA, and its potential benefits, we give a simple example. We consider the problem of navigating an $n \times n$ minefield. The minefield contains m mines, each with coordinates (x_i, y_i), for $1 \leq i \leq m$, where $1 \leq x_i < n$, $1 \leq y_i < n$. We consider the problem of computing the maximal probability with which a robot can reach the target corner (n, n), from all $n \times n$ states. At interior states of the field, the robot can choose among four actions: *Up, Down, Left, Right;* at the border of the field, actions that lead outside of the field are missing. From a state $s = (x, y) \in \{1, \dots, n\}^2$ with coordinates (x, y), each action causes the robot to move to square (x', y') with probability $q(x', y')$, and to "blow up" (move to an additional sink state) with probability $1 - q(x', y')$. For action *Right,* we have $x' = x + 1$, $y' = y$; similarly for the other actions. The probability $q(x', y')$ depends on the proximity to mines, and is given by



(a) Initial Abstraction (b) Final Abstraction

Fig. 1. Initial, and final refined abstraction, for the problem of motion planning in a 24×24 minefield. The circles denote the mines

$$q(x', y') = \prod_i^m \exp\big(-0.7 \cdot \big((x' - x_i)^2 + (y' - y_i)^2\big)\big).$$

The problem, for $n = 24$, is illustrated in Figure 1.

Intuitively, it is desirable to group the 8×8 states in the top-middle area into a single region r_0: since no mines are nearby, the robot can freely roam in r_0, so that the maximal probability of reaching the target corner is essentially constant across r_0. Indeed, to a human trying to determine a best path to the target corner, the states in r_0 are essentially equivalent. When the 8×8 concrete states are grouped in r_0, MLA leads to accurate results, since it can analyze the dynamics inside r_0 when r_0 is magnified. We also note how, in this example, the ability of MLA to refine the abstraction adaptively is crucial. As shown in Figure 1(b), MLA is able to use small regions close to mines, and large regions elsewhere. If we insisted on a uniform region size, then we would have to adopt the smallest size throughout, and no space savings would be possible.

On the other hand, the full-abstraction approaches described earlier, such as [7,22,19], based on probabilistic simulation [27], are not well suited to this example. Such techniques would associate with an abstract state, such as r_0, a summary of the transition structure from states $s \in r_0$, and use that summary to analyze the abstraction. The problem is that the states in r_0, while similar in value, are not similar in transition structure: the states on the border of r_0 can transition outside of r_0, while those in the interior cannot. In the abstraction, the probability of going from r_0 to the region at the right hand side will be modeled as being in an interval $[0, q]$, for some q close to 1 (all mines are far away). Consequently, previous techniques would have yielded a lower bound of 0, and an upper bound close to 1, for the maximum probability of reaching the target corner. Similarly, the technique of [12] would lead to recursively splitting the MDP, until the regions consisted of only one concrete state each.

Other related work. MLA is reminiscent to methods that represent value functions via ADDs or MTBDDs [6,1] with an approximation factor used to merge leaves. The similarity, however, is superficial: MLA leads to far more precise results in the analysis; we discuss this in the conclusions, where the appropriate notation will be available.

MLA is also loosely reminiscent of *adaptive mesh refinement* (AMR) methods used in the solution of partial differential equations [3]. There are, however, two important differences between MLA and AMR. In AMR, separate lower and upper bounds are not kept. AMR methods perform computation at the finest mesh sizes only where needed. In MLA, due to the discrete nature of MDPs, we have no way of computing over a "coarse mesh" only: to update valuations over a region, we need to "magnify" the region to its individual states. Thus, MLA is forced to consider the individual states over the whole system, and it summarizes and returns the results in terms of lower and upper bounds, which are well-suited to answering verification questions.

2 Preliminary Definitions and Algorithms

For a countable set S, a *probability distribution* on S is a function $p : S \mapsto [0, 1]$ such that $\sum_{s \in S} p(s) = 1$; we denote the set of probability distributions on S by $D(S)$. A *valuation* over a set S is a function $v : S \mapsto \mathbb{R}$ associating a real number $v(s)$ with every $s \in S$. For $x \in \mathbb{R}$, we denote by $\mathbf{x}$ the valuation with constant value x; for $T \subseteq S$, we indicate by $[T]$ the valuation having value 1 in T and 0 elsewhere. For two valuations v, u on S, we define $||v - u|| = \sup_{s \in S} |v(s) - u(s)|$.

A *partition* of a set S is a set $R \subseteq 2^S$, such that $\bigcup\{s | s \in R\} = S$, and such that for all $r, r' \in R$, if $r \neq r'$ then $r \cap r' = \emptyset$. For $s \in S$ and a partition R of S, we denote by $[s]_R$ the element $r \in R$ with $s \in r$. We say that a partition R' is *finer* than a partition R if the elements of R can be written as unions of the elements of R'.

A *Markov decision process* (MDP) $M = \langle S, A, \Gamma, p \rangle$ consists of the following components:

- A finite state space S.
- A finite set A of actions (moves),
- A move assignment $\Gamma : S \to 2^A \setminus \emptyset$.
- A probabilistic transition function $p \ : S \times A \to D(S)$.

At every state $s \in S$, the controller can choose an action $a \in \Gamma(s)$; the MDP then proceeds to the successor state t with probability $p(s, a, t)$, for all $t \in S$. A *path* of G is an infinite sequence $\overline{s} = s_0, s_1, s_2, \ldots$ of states of S; we denote by S^ω the set of all paths, and we denote by $\overline{s}_k$ the k-th state s_k of $\overline{s} = s_0, s_1, s_2, \ldots$.

We model the choice of actions, on the part of the controller, via a *strategy* (strategies are also variously called *schedulers* [26] or *policies* [13]). A *strategy* is a mapping $\pi : S^+ \mapsto D(A)$: given a past history $\sigma s \in S^+$ for the MDP, a strategy π chooses each action $a \in \Gamma(s)$ with probability $\pi(\sigma s)(a)$; we obviously require $\pi(\sigma s)(b) = 0$ for all $b \in A \setminus \Gamma(s)$. Thus, strategies can be both history-dependent, and randomized. We denote by Π the set of all strategies.

We consider *safety* and *reachability* goals. Given a subset $T \subseteq S$ of states, the reachability goal $\Diamond T = \{\overline{s} \in S^\omega \mid \exists k.\overline{s}_k \in T\}$ consists in the paths that reach T, and the safety goal $\Box T = \{\overline{s} \in S^\omega \mid \forall k.\overline{s}_k \in T\}$ consists in the paths that stay always in T. These sets of paths are measurable [28], so that given a strategy $\pi \in \Pi$, we can define the probabilities $\Pr_s^\pi(\Diamond T)$, $\Pr_s^\pi(\Box T)$ of following a path in these sets from an initial state $s \in S$ under strategy π. By choosing appropriate strategies, the

Algorithm 1. ValIter$(T, f, g, \varepsilon_{\mathit{float}})$ Value iteration

```
1. $v := [T]$
2. repeat
3.     $\hat{v} := v$
4.     for all $s \in S$ do $v(s) := f\Big([T](s),\ g\{\sum_{s' \in S} p(s, a, s') \cdot \hat{v}(s') \mid a \in \Gamma(s)\}\Big)$
5. until $||v - \hat{v}|| \leq \varepsilon_{\mathit{float}}$
6. return $v$
```

controller can maximize or minimize these probabilities. Thus, we consider the problem of computing, at all $s \in S$, the quantities:

$$V^{\max}_{\Box T}(s) = \max_{\pi \in \Pi} \Pr^{\pi}_{s}(\Box T) \qquad V^{\max}_{\Diamond T}(s) = \max_{\pi \in \Pi} \Pr^{\pi}_{s}(\Diamond T)$$

$$V^{\min}_{\Box T}(s) = \min_{\pi \in \Pi} \Pr^{\pi}_{s}(\Box T) \qquad V^{\min}_{\Diamond T}(s) = \min_{\pi \in \Pi} \Pr^{\pi}_{s}(\Diamond T).$$

The fact that on the right-hand side we have $\max$, $\min$ rather than $\sup$, $\inf$ is a consequence of the existence of optimal (and memoryless) strategies [13]. In the remainder of the paper, unless explicitly noted, we present algorithms and definitions for a fixed MDP $M = \langle S, A, \Gamma, p \rangle$.

Reachability and safety probabilities on an MDP can be computed via a classical value-iteration scheme [13,4,11]. The algorithm, depicted as Algorithm 1, is parametrized by two operators $f, g \in \{\max, \min\}$. The operator f specifies how to merge the valuation of the current state with the expected next-state valuation; we use $f = \max$ for reachability goals, and $f = \min$ for safety ones. The operator g specifies whether to select the action that maximizes, or minimizes, the expected next-state valuation; we use $g = \max$ to compute maximal probabilities, and $g = \min$ to compute minimal probabilities, The algorithm is also parametrized by $\varepsilon_{\mathit{float}} > 0$: this is the threshold below which we consider value iteration to have converged. The following facts are well-known (see, e.g., [13,8,9]). For all $\varepsilon_{\mathit{float}} > 0$ and for all $f, g \in \{\min, \max\}$, the call ValIter$(T, f, g, \varepsilon_{\mathit{float}})$ terminates. Moreover, consider any $g \in \{\max, \min\}$ and any $\triangle \in \{\Box, \Diamond\}$, and let $f = \min$ if $\triangle = \Box$, and $f = \max$ if $\triangle = \Diamond$. Then, for all $\delta > 0$, there is $\varepsilon_{\mathit{float}} > 0$ such that, at all $s \in S$:

$$v(s) - \delta \ \leq \ V^{g}_{\triangle T}(s) \ \leq \ v(s) + \delta$$

where $v =$ ValIter$(T, f, g, \varepsilon_{\mathit{float}})$. We note that can replace statement 1 of Algorithm 1 with the following initialization: **if** $f = \max$ **then** $v := \mathbf{0}$ **else** $v := \mathbf{1}$.

3 Magnifying-Lens Abstraction

Magnifying-lens abstractions (MLA) is a technique for the analysis of reachability and safety properties of MDPs. Let v^* be the valuation on S that is to be computed: v^* is one of $V^{\min}_{\Box T}$, $V^{\max}_{\Box T}$, $V^{\min}_{\Diamond T}$, $V^{\max}_{\Diamond T}$. Given a desired accuracy $\varepsilon_{\mathit{abs}} > 0$, MLA computes upper

Algorithm 2. MLA$(T, f, g, \varepsilon_{\mathit{float}}, \varepsilon_{\mathit{abs}})$ Magnifying-Lens Abstraction

```
1.  $R :=$ some initial partition.
2.  **if** $f = \max$ **then** $u^- := \mathbf{0}$; $u^+ := \mathbf{0}$ **else** $u^- := \mathbf{1}$; $u^+ := \mathbf{1}$
3.  **loop**
4.      **repeat**
5.          $\hat{u}^+ := u^+; \hat{u}^- := u^-;$
6.          **for** $r \in R$ **do**
7.              $u^+(r) :=$ MagnifiedIteration$(r, R, T, \hat{u}^+, \hat{u}^-, \hat{u}^+, \max, f, g, \varepsilon_{\mathit{float}})$
8.              $u^-(r) :=$ MagnifiedIteration$(r, R, T, \hat{u}^-, \hat{u}^-, \hat{u}^+, \min, f, g, \varepsilon_{\mathit{float}})$
9.          **end for**
10.     **until** $||u^+ - \hat{u}^+|| + ||u^- - \hat{u}^-|| \leq \varepsilon_{\mathit{float}}$
11.     **if** $||u^+ - u^-|| \geq \varepsilon_{\mathit{abs}}$
12.         **then** $R, u^-, u^+ :=$ SplitRegions$(R, u^-, u^+, \varepsilon_{\mathit{abs}})$
13.         **else return** $R, u^-, u^+$
14.     **end if**
15. **end loop**
```

and lower bounds for v^*, spaced less than $\varepsilon_{\mathit{abs}}$. MLA starts from an initial partition R of S, and computes the lower and upper bounds as valuations u^- and u^+ over R. The partition is refined, until the difference between u^- and u^+, at all regions, is below a specified threshold. To compute u^- and u^+, MLA iteratively considers each r in turn, and performs a *magnified iteration:* it improves the estimates for $u^-(r)$ and $u^+(r)$ using value iteration on the concrete states $s \in r$.

The MLA algorithm is presented as Algorithm 2. The algorithm has parameters T, f, g, which have the same meaning as in Algorithm ValIter. The algorithm also has parameters $\varepsilon_{\mathit{float}} > 0$ and $\varepsilon_{\mathit{abs}} > 0$. Parameter $\varepsilon_{\mathit{abs}}$ indicates the maximum difference between the lower and upper bounds returned by MLA. Parameter $\varepsilon_{\mathit{float}}$, as in ValIter, specifies the degree of precision to which the local, magnified value iteration should converge. MLA should be called with $\varepsilon_{\mathit{abs}}$ greater than $\varepsilon_{\mathit{float}}$ by at least one order of magnitude: otherwise, errors in the magnified iteration can cause errors in the estimation of the bounds. Statement 2 initializes the valuations u^- and u^+ according to the property to be computed: reachability properties are computed as least fixpoints, while safety properties are computed as greatest fixpoints [11]. A useful time optimization, not shown in Algorithm 2, consists in executing the loop at lines 6–9 only for regions r where at least one of the neighbor regions has changed value by more than $\varepsilon_{\mathit{float}}$.

Magnified iteration. The algorithm performing the magnified iteration is given as Algorithm 3. The algorithm is very similar to Algorithm 1, except for three points.

First, the valuation v (which here is local to r) is initialized not to $[T]$, but rather, to $u^-(r)$ if $f = \max$, and to $u^+(r)$ if $f = \min$. Indeed, if $f = \max$, value iteration converges from below, and $u^-(r)$ is a better starting point than $[T]$, since $[T](s) \leq u^-(r) \leq v^*(s)$ at all $s \in r$. The case for $f = \min$ is symmetrical.

Second, for $s \in S \setminus r$, the algorithm uses, in place of the value $v(s)$ which is not available, the value $u^-(r')$ or $u^+(r')$, as appropriate, where r' is such that $s \in r'$. In other words, the algorithm replaces values at concrete states outside r with the

Algorithm 3. MagnifiedIteration$(r, R, T, u, u^-, u^+, h, f, g, \varepsilon_{\mathit{float}})$

```
$v$: a valuation on $r$
1. if $f = \max$
2.     then for $s \in r$ do $v(s) = u^-(r)$
3.     else for $s \in r$ do $v(s) = u^+(r)$
4. repeat
5.     $\hat{v} := v$
6.     for all $s \in r$ do
           $v(s) = f\left([T](s),\ g\left\{\sum_{s' \in r} p(s,a,s') \cdot \hat{v}(s') + \sum_{s' \in S \setminus r} p(s,a,s') \cdot u([s]_R) \;\middle|\; a \in \Gamma(s)\right\}\right)$
7. until $||v - \hat{v}|| \leq \varepsilon_{\mathit{float}}$
8. return $h\{v(s) \mid s \in r\}$
```

"abstract" values of the regions to which the states belong. To this end, we need to be able to efficiently find the "abstract" counterpart $[s]_R$ of a state $s \in S$. We use the following scheme, similar to schemes used in AMR [3]. Most commonly, the state-space S of the MDP consists in value assignments to a set of variables $X = \{x_1, x_2, \ldots, x_l\}$. We represent a partition R of S, together with the valuations u^+, u^-, via a binary decision tree. The nodes of the tree are labeled by $\langle y, i \rangle$, where $y \in X$ is the variable according to which we split, and i is the position of the bit (0 =LSB) of the variable according to whose value we split. The leaves of the tree correspond to regions, and they are labeled with u^-, u^+ values. Given s, finding $[s]_R$ in such a tree requires time logarithmic in $|S|$.

Third, once the concrete valuation v is computed at all $s \in r$, Algorithm 3 returns the minimum (if $h = \min$) or the maximum (if $h = \max$) of $v(s)$ at all $s \in r$, thus providing a new estimates for $u^-(r)$, $u^+(r)$, respectively.

Adaptive abstraction refinement. We denote the *imprecision* of a region r by $\Delta(r) = u^+(r) - u^-(r)$. MLA adaptively refines a partition R by splitting all regions r having $\Delta(r) > \varepsilon_{\mathit{abs}}$. This is perhaps the simplest possible refinement scheme. We experimented with alternative refinement schemes, but none of them gave consistently better results. In particular, we considered splitting the regions with high Δ-value, all whose successors, according to the optimal moves, have low Δ-value: the idea is that such regions are the ones where precision degrades. While this reduces somewhat the number of region splits, the total number of refinements is increased, and the resulting algorithm is not clearly superior, at least in the examples we considered. We also experimented with splitting all regions $r \in R$ with $\Delta(r) > \delta$, for a threshold δ that is initially set to $\frac{1}{2}$, and that is then gradually decreased to $\varepsilon_{\mathit{abs}}$. This approach, inspired by simulated annealing, also failed to provide consistent improvements.

In the minefield example, each region is *squarish* (horizontal and vertical sizes differ by at most 1); we split each such squarish region into 4 smaller squarish regions. In more general cases, the following heuristic for splitting regions is widely applicable, and has worked well for us. The user specifies an ordering $x_0, x_1, \ldots, x_l$ for the state variables X defining S: this specifies a priority order for splitting regions. As previously mentioned, we represent a partition R via a decision tree, whose leaves correspond to the regions. In the refinement phase, we split a leaf according to the value of a new variable (not present in that leaf), following the variable ordering given by the user. Precisely, to split a region r, we look at the label $\langle x_j, i\rangle$ of its parent node. If $i > 0$, we split according to bit $i - 1$ of x_j; otherwise, we split according to the MSB of x_{j+1}. A refinement of this technique allows the specification of groups of variables, whose ranges are split in interleaved fashion. Once a region r has been split into regions r_1, r_2, we set $u^-(r_j) = u^-(r)$ and $u^+(r_j) = u^+(r)$ for all $j = 1, 2$. A call to SplitRegions$(R, u^+, u^-, \varepsilon_{abs})$ returns a triple $\tilde{R}, \tilde{u}^-, \tilde{u}^+$, consisting of the new partition with its upper and lower bounds for the valuation.

Correctness. The following theorem summarizes MLA correctness.

Theorem 1. *For all MDPs $M = \langle S, A, \Gamma, p\rangle$, all $T \subseteq S$, and all $\varepsilon_{abs} > 0$, the following assertions hold.*

1. Termination. *For all $\varepsilon_{float} > 0$, and for all $f, g \in \{\min, \max\}$, the call MLA $(T, f, g, \varepsilon_{float}, \varepsilon_{abs})$ terminates.*
2. (Partial) correctness. *Consider any $g \in \{\max, \min\}$, any $\varepsilon_{abs} > 0$, and any $\triangle \in \{\Box, \Diamond\}$, and let $f = \min$ if $\triangle = \Box$, and $f = \max$ if $\triangle = \Diamond$. The following holds. For all $\delta > 0$, there is $\varepsilon_{float} > 0$ such that:*

$$\begin{aligned} \forall r \in R: &\qquad u^+(r) - u^-(r) \;\le\; \varepsilon_{abs} \\ \forall s \in S: &\qquad u^-([s]_R) - \delta \;\le\; V^g_{\triangle T}(s) \;\le\; u^+([s]_R) + \delta \end{aligned}$$

where $(R, u^-, u^+) = MLA(T, f, g, \varepsilon_{float}, \varepsilon_{abs})$.

We note that the theorem establishes the correctness of lower and upper bounds only within a constant $\delta > 0$, which depends on ε_{float}. This limitation is inherited from the value-iteration scheme used over the magnified regions. If linear programming [13,4] were used instead, then MLA would provide true lower and upper bounds. However, in practice value iteration is preferred over linear programming, due to its simplicity and great speed advantage, and the concerns about δ are solved — in practice, albeit not in theory — by choosing a small $\varepsilon_{float} > 0$.

4 Experimental Results

In order to evaluate the time and space performance of MLA, we have implemented a prototype, and we have used it for three case studies: the minefield navigation problem, the Bounded Retransmission Protocol [7], and the ZeroConf protocol for the autonomous configuration of IP addresses [5,19].

When comparing MLA to ValIter, we compute the space needs of the algorithms as follows. For ValIter, we take the space requirement to be equal to $|S|$, the domain of v. For MLA, we take the space requirement to be the maximum value of $2 \cdot |R| + \max_{r \in R} |r|$ that occurs every time MLA is at line 4 of Algorithm2: this gives the maximum space required to store the valuations u^+, u^-, as well as the values v for the largest magnified region. Since $\max_{r \in R} |r| \ge (|S|/|R|)$, the space complexity of the algorithm is (lower) bounded by a square-root function $\sqrt{8 \cdot |S|}$.

Algorithm	Space	Time
ValIter	16,384	21.97
MLA	7,926	123.54

MLA Iteration Details			
#Abs	$\lvert R\rvert$	D	Time
1	144	0.994	9.21
2	576	0.837	38.48
3	2,312	0.663	47.36
4	3,256	0.645	11.39
5	3,566	0.020	14.59
6	3,899	0.007	2.52

(a) $n = 128$, $m = 128$

Algorithm	Space	Time
ValIter	65,536	130.18
MLA	7,944	185.13

MLA Iteration Details			
#Abs	$\lvert R\rvert$	D	Time
1	256	0.983	49.48
2	985	0.656	76.27
3	1,513	0.776	12.61
4	2,341	0.605	17.58
5	3,844	0.007	29.19

(b) $n = 256$, $m = 128$

Algorithm	Space	Time
ValIter	262,144	1,065.36
MLA	30,180	3,199.31

MLA Iteration Details			
#Abs	$\lvert R\rvert$	D	Time
1	576	0.999	299.02
2	2,295	0.777	1648.67
3	4,347	0.777	206.64
4	7,171	0.659	228.95
5	11,678	0.525	362.70
6	14,862	0.007	453.33

(c) $n = 512$, $m = 512$

Fig. 2. Comparison between MLA and ValIter for $n \times n$ minefields with m mines, for $\varepsilon_{abs} = 10^{-2}$ and $\varepsilon_{float} = 10^{-4}$. Mine densities (m/n^2) are (a) $1/64$, (b) $1/512$, and (c) $1/512$. All times are in seconds. #Abs is the number of abstraction steps (number of loops 3–15 of MLA), and $D = \max_{r \in R}(u^+(r) - u^-(r))$.

Algorithm	Space	Time
ValIter	16,384	20.51
MLA	3,672	54.51

(a) $n = 128$, $m = 128$

Algorithm	Space	Time
ValIter	65,536	130.08
MLA	4,548	126.40

(b) $n = 256$, $m = 128$

Algorithm	Space	Time
ValIter	262,144	1,065.65
MLA	15,476	1,853.01

(c) $n = 512$, $m = 512$

Fig. 3. Comparison between MLA and ValIter for $n \times n$ minefields with m mines, for $\varepsilon_{abs} = 10^{-1}$ and $\varepsilon_{float} = 10^{-2}$. Mine densities (m/n^2) are (a) $1/64$, (b) $1/512$, and (c) $1/512$. All times are in seconds.

4.1 Minefield Navigation

We experimented with different-size minefields in the mine-field example. In all cases, the mines were distributed in a pseudo-random fashion across the field. The performance of algorithms ValIter and MLA, for $\varepsilon_{abs} = 0.01$, are compared in Figure 2. As we can see, the space savings are 2.06 for a mine density of 1/64, and an average of 8.47 for a mine density of 1/512. This comes at a cost in running time, which is of 5.67 for a mine density of 1/64, and 1.42 to 3.00 for a mine density of 1/512. Especially for lower mine densities, MLA provides space savings that are larger than the incurred time penalty. The space savings are even more pronounced when we decrease the desired precision of the result to $\varepsilon_{abs} = 0.1$, as indicated in Figure 3.

4.2 The ZeroConf Protocol

The ZeroConf protocol [5] is used for the dynamic self-configuration of a host joining a network; it has been used as a testbed for the abstraction method considered in [19]. We consider a network with 4 existing hosts, and 32 total IP addresses; protocol messages have a certain probability of being lost during transmission. We consider the problem of determining the worst-case probability of a host eventually acquiring an IP address: this is a probabilistic reachability problem.

N	MAX	ValIter time	#Reachable states	MLA space	MLA time
16	3	0.08	1,966	918	27.38
32	5	0.21	5,466	2,604	140.79
64	5	0.40	10,650	5,380	266.53

Fig. 4. Comparison between MLA and ValIter for BRP. N denotes number of chunks and MAX denotes the maximum number of retransmissions. All times are in seconds.

The abstraction approach of [19] reduces the problem from $26,121$ concrete reachable states to 737 abstract states. MLA reduces the problem to 131 regions, requiring a total space of 1267 (including also the space to perform the magnification step) for $\varepsilon_{abs} = 10^{-3}$ and $\varepsilon_{float} = 10^{-6}$. We cannot compare the running times, due to the absence of timing data in [19].

4.3 Bounded Retransmission Protocol

We also considered the Bounded Retransmission Protocol described in [7]. We compared the performance of algorithms ValIter and MLA on "Property 1" from [7], stating that the sender eventually does not report a successful transmission. The results are compared in Figure 4, for $\varepsilon_{abs} = 10^{-2}$ and $\varepsilon_{float} = 10^{-4}$. MLA achieves a space saving of a factor of 2, but at the price of a great increase in running time.

4.4 Discussion

From these examples, it is apparent that MLA does well on problems where there is some notion of "distance" between states, so that "nearby" states have similar values for the reachability or safety property of interest. These problems are common in planning and control. As we discussed in the introduction, many of these problems do not lend themselves to abstraction methods based on the similarity of transition relations, such as [19,7], and other methods based on simulation. We believe the MLA algorithm is valuable for the study of this type of problems. We note that each mine affects a region of size 5×5 by more than the desired precision $\varepsilon_{abs} = 10^{-2}$. Therefore, while the mine density is only 1/512, the ratio of "disturbed" vs. "undisturbed" state space is 25/512, or 1/20. This is a typical value in planning problems with sparse obstacles.

On the other hand, for problems where simulation-based methods can be used, these methods tend to be more effective than MLA, as they can construct, once and for all, a small abstract model on which all properties of interest can be analyzed.

5 Conclusions

A natural question about MLA is the following: why does MLA consider the concrete states at each iteration, as part of the "magnification" steps, rather than constructing an abstract model once and for all, and then analyze it, as other approaches to MDP abstraction do [7,18,22,19]? The answer has two parts. First, we cannot build an abstract model once and for all: our abstraction refinement approach would require the computation of several abstractions. Second, we have found that the cost of building abstractions that are sufficiently precise, without resorting to a "magnification" step, is substantial, negating any benefits that might derive from the ability to perform computation on a reduced system.

To understand the performance issues in constructing precise abstractions, consider the problem of computing the maximal reachability probability. To summarize the maximal probability of a transition from a region r to r_1, we need to compute $P_r^+(r_1) = \min_{s \in r} \max_{\pi \in \Pi} \Pr_s^\pi(r \, \mathcal{U} \, r_1)$, where $\mathcal{U}$ is the "until" operator of linear temporal logic [20]; this quantity is related to building abstractions via *weak simulation* [27,2,24]. These probability summaries are not additive: for $r_1 \neq r_2$, we have that $P_r^+(r_1) + P_r^+(r_2) \leq P^+(r_1 \cup r_2)$, and equality does not hold in general. Indeed, these probability summaries constitute *capacities,* and they can be used to analyze maximal reachability properties via the Choquet integral [25,15,16]. To construct a fully precise abstraction, one must compute $P_r^+(R')$ for all $R' \subseteq R$, clearly a daunting task. In practice, in the minefield example, it suffices to consider those $R' \subseteq R$ that consist of neighbors of r. To further lower the number of capacities to be computed, we experimented with restricting R' to unions of no more than k regions, but for all choices of k, the algorithm either yielded grossly imprecise results, or proved to be markedly less efficient than MLA.

The space savings provided by MLA are bounded by a square-root function of the state space. We could improve this bound by applying MLA hierarchically, so that each magnified region is studied, in turn, with a nested application of MLA.

Symbolic representations such as ADDs and MTBDDs [6,1] have been used for representing the value function compactly [10,17]. The decision-tree structure used by MLA to represent regions and abstract valuations is closely related to MTBDDs, and in future work we intend to explore symbolic implementations of MLA, where separate MTBDDs will be used to represent lower and upper bounds.

References

1. Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. Journal of Formal Methods in System Design 10(2/3), 171–206 (1997)
2. Baier, C., Hermanns, H.: Weak bisimulation for fully probabilistic processes. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 119–130. Springer, Heidelberg (1997)
3. B., J., Berger, J.O.: Adaptive mesh refinement for hyperbolic partial differential equations. Journal of Computational Physics 53, 484–512 (1984)
4. Bertsekas, D.: Dynamic Programming and Optimal Control. Athena Scientific, Volumes I and II (1995)
5. Cheshire, S., Adoba, B., Gutterman, E.: Dynamic configuration of ipv4 link local addresses (internet draft).
6. Clarke, E., Fujita, M., McGeer, P., Yang, J., Zhao, X.: Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In: International Workshop for Logic Synthesis (1993)

7. D'Argenio, P., Jeannet, B., Jensen, H., Larsen, K.: Reachability analysis of probabilistic systems by successive refinements. In: de Luca, L., Gilmore, S.T. (eds.) PROBMIV 2001, PAPM-PROBMIV 2001, and PAPM 2001. LNCS, vol. 2165, pp. 39–56. Springer, Heidelberg (2001)
8. de Alfaro, L.: Formal Verification of Probabilistic Systems. PhD thesis, Stanford University, Technical Report STAN-CS-TR-98-1601 (1997)
9. Alfaro, L.d.: Computing minimum and maximum reachability times in probabilistic systems. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 66–81. Springer, Heidelberg (1999)
10. L.,, Kwiatkowska, M., Norman, G., Parker, D., Alfaro, R.S.d.: Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In: Schwartzbach, M.I., Graf, S. (eds.) ETAPS 2000 and TACAS 2000. LNCS, vol. 1785, pp. 395–410. Springer, Heidelberg (2000)
11. Alfaro, L.d., Majumdar, R.: Quantitative solution of omega-regular games. Journal of Computer and System Sciences 68, 374–397 (2004)
12. Dean, T., Givan, R.: Model minimization in markov decision processes. In: AAAI/IAAI, pp. 106–111 (1997)
13. Derman, C.: Finite State Markovian Decision Processes. Academic Press, London (1970)
14. Fecher, H., Leucker, M., Wolf, V.: Don't know in probabilistic systems. In: Valmari, A. (ed.) Model Checking Software. LNCS, vol. 3925, Springer, Heidelberg (2006)
15. Gilboa, I.: Expected utility with purely subjective non-additive probabilities. Journal of Mathematical Economics 16, 65–88 (1987)
16. Gilboa, I., Schmeidler, D.: Additive representations of non-additive measures and the choquet integral. Discussion Papers 985, Northwestern University, Center for Mathematical Studies in Economics and Management Science (1992), available at `http://ideas.repec.org/p/nwu/cmsems/985.html.`
17. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006 and ETAPS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
18. Huth, M.: On finite-state approximations for probabilistic computational-tree logic. Theor. Comp. Sci. 346(1), 113–134 (2005)
19. Kwiatkowska, M., Norman, G., Parker, D.: Game-based abstraction for markov decision processes. In: Proc. of QEST: Quantitative Evaluation of Systems, pp. 157–166. IEEE Computer Society Press, Los Alamitos (2006)
20. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, New York (1991)
21. McIver, A., Morgan, C.: Abstraction, Refinement, and Proof for Probabilistic Systems. In: Monographs in Computer Science, Springer, Heidelberg (2004)
22. Monniaux, D.: Abstract interpretation of programs as Markov decision processes. Science of Computer Programming 58(1–2), 179–205 (2005)
23. Plateau, B.: On the stochastic structure of parallelism and synchronization models for distributed algorithms. In: SIGMETRICS '85: Proceedings of the 1985 ACM SIGMETRICS conference on Measurement and modeling of computer systems, pp. 147–154. ACM Press, New York, NY, USA (1985)
24. Plilippou, A., Lee, I., Sokolsky, O.: Weak bisimulation for probabilistic systems. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 334–349. Springer, Heidelberg (2000)
25. Schmeidler, D.: Integral representation without additivity. Proceedings of the American Mathematical Society 97, 255–261 (1986)

26. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, MIT, Technical Report MIT/LCS/TR-676 (1995)
27. Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. In: Jonsson, B., Parrow, J. (eds.) CONCUR 1994. LNCS, vol. 836, pp. 481–496. Springer, Heidelberg (1994)
28. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state systems. In: Proc. 26th IEEE Symp. Found. of Comp. Sci. pp. 327–338. IEEE Computer Society Press, Los Alamitos (1985)