Magnifying-Lens Abstraction for Markov Decision Processes

Luca de Alfaro and Pritam Roy

Computer Engineering Department
University of California, Santa Cruz, USA

Technical Report ucsc-crl-06-15
School of Engineering
University of California, Santa Cruz
October 17, 2006

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,* 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, magnifying-lens abstraction 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 illustrating that magnifying-lens abstractions can provide accurate answers, with drastic savings in memory requirements, in many cases where previous abstraction techniques yield no benefit.

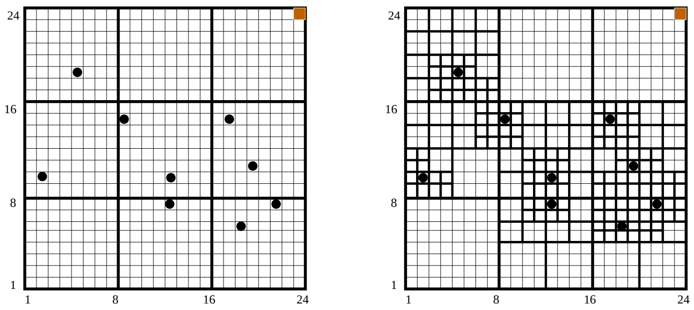
1 Introduction

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. In probabilistic verification and performance evaluation, the nondeterminism can be used to model concurrency and external inputs, while probability captures randomization or unpredictability [13, 23]. In planning and optimal control, the nondeterminism represents the choice of control action, while probability models the uncertainty in the evolution of the controlled system [10, 4].

Markov decision processes 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, 15, 19, 16]. 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. MLA starts from a coarse, and concise, abstraction of an MDP, and gradually refines the abstraction in an adaptive (and automatic) fashion, adding detail where most needed, until reachability and safety questions can be answered to the desired degree of accuracy. We show that the technique can lead to substantial space and time savings in the analysis of MDPs, and we show that the savings extend to cases where previous abstraction techniques yield no benefit.

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. In this paper, 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. Magnifying-lens abstraction enables the computation of 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 [16].

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 [20, 8, 14]. 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



(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.

the problem of computing a control strategy by recording the optimal actions for the concrete states of interest, when they are magnified.

The ability of MLA to group states based on value, and the ability to split regions individually, tailoring the detail to the needs of different portions of the state space, make MLA effective in cases where previous abstraction techniques yield no benefits. This is best illustrated via a simple example. We consider the problem of navigating an $n \times n$ minefield; the robot starts at the corner $(1, 1)$, and must reach the target corner (n, n); the problem, for $n = 24$, is illustrated in Figure 1. Some mines are distributed randomly in the minefield. At each internal state s, there are four actions available: *Up, Down, Left, Right;* border states lack the actions that would lead out of the minefield. When an action is taken, the robot moves one square in the chosen direction with probability $1-p(s)$, and blows up with probability $p(s)$; the probability $p(s)$ is high close to mines, and very low elsewhere (we will provide the precise equations for this model later in the paper).

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. Previous abstraction techniques for probabilistic systems, such as [7, 19, 16], are based on probabilistic simulation [24], and they are unable to provide accurate results. Such techniques associate with 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. We note that iterative abstraction refinement does not help: it would repeatedly strip off the borders from regions, until there are as many regions as there are states. The precision of traditional abstraction approaches can be improved by relying on *weak* simulation [24, 2, 21] rather than regular, or strong, simulation. The stumbling block, in this approach, turns out to be the high computational cost of building the abstractions; we discuss this in detail in the conclusions. We also note how, in this example, the ability of MLA to refine the abstraction adaptively is crucial, as depicted 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. One of the benefits of MLA is that 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.

The memory requirements of MLA are proportional not only to the number of regions, but also to the maximum number of states in a region, due to the "magnifying-lens" computation. This makes it impossible to use MLA for the analysis of infinite-state systems: indeed, for a system with $|S|$ states, the best we can achieve is memory consumption proportional to $\sqrt{8|S|}$. This contrasts with the technique of [16], where the space savings are in principle unbounded, and with [19], where abstract-interpretation techniques are applied to the analysis of infinite-state systems. We experimented with MLA on 256×256 and 512×512 instances of the minefield example. In the 512×512 case, MLA reduced space usage by a factor between 30 and 60, approximately, depending on the desired accuracy of the analysis (10^{-1} to 10^{-3}).

In addition to the space savings, MLA is also faster than traditional value iteration: for the 512×512 example, the speedup factor was between 5 and 7. The reason for the speedup is that MLA avoids magnifying and re-analyzing a region, when none of its neighbors have changed. Moreover, MLA can perform different numbers of value iterations for each region. While the speedup is relatively minor, it shows that the space savings of MLA are not associated with a speed penalty, compared to classical value iteration.

Related work. We have already discussed the differences between MLA and methods for MDP abstraction based on simulation or abstract interpretation [7, 15, 19, 18, 16]. 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. Due to the continuous nature of differential equations, the computation over a coarse mesh can be performed independently of the computation for finer meshes, and splitting is done by comparing the results for different mesh sizes. In particular, 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. MLA then summarizes the computation over the individual states into lower and upper bounds, and uses the difference between these bounds to decide which regions to split. 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.

Paper organization. The paper is organized as follows. In Section 2, we give preliminary definitions. In Section 3 we recall the standard value-iteration algorithms for solving MDPs, and in Section 4 we present magnifying-lens abstraction. In Section 5, we provide detailed experimental results on MLA applied to the minefield navigation problem, and we compare MLA with the game-abstraction method of [16] over a model of the ZeroConf protocol, which is the case study considered in [16]. We conclude with some final observations, and considerations on future work.

2 Preliminary Definitions

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 valuations v, u over S, we define operators and inequalities in pointwise fashion: for instance, we define $v+u$ by $(v+u)(s) = v(s)+u(s)$ for all $s \in S$, and we write $v \leq u$ if $v(s) \leq u(s)$ at all $s \in S$. 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 $||v - u|| = \max_{s \in S} |v(s) - u(s)|$.

A *partition* of a set S is a set $R \subseteq 2^S$, such that $\bigcup\{r | r \in R\} = S$, 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'.

Definition 1 (Markov decision process). *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 $p \ : S \times A \to D(S)$.*

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

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

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* [23] or *policies* [10]). 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 $b \in A \backslash \Gamma(s)$. 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 [25], so that given a strategy $\pi \in \Pi$, we can define the probabilities $\mathrm{Pr}_s^\pi(\Diamond T)$, $\mathrm{Pr}_s^\pi(\Box T)$ of following a path in these sets from an initial state $s \in S$ under strategy π. By choosing appropriate strategies, the controller can maximize or minimize these probabilities. Thus, we consider the problem of computing, at all $s \in S$, the quantities:

$$V_{\Box T}^{\max}(s) = \max_{\pi \in \Pi} \mathrm{Pr}_s^\pi(\Box T) \qquad V_{\Diamond T}^{\max}(s) = \max_{\pi \in \Pi} \mathrm{Pr}_s^\pi(\Diamond T)$$

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

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 [10]. In the remainder of the paper, unless explicitly noted, we present algorithms and definitions for a fixed MDP $M = \langle S, A, \Gamma, p \rangle$.

3 Value Iteration Algorithms for Reachability and Safety

Reachability and safety probabilities on an MDP can be computed via a classical value-iteration scheme [10, 4, 9]. The algorithm, depicted as Algorithm 1, is parameterized 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 parameterized by $\varepsilon_{\mathit{float}} > 0$: this is the threshold below which we consider value iteration to have converged. The following theorem states the correctness of the algorithm.

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

1. Termination. *For all $\varepsilon_{\mathit{float}} > 0$ and for all $f, g \in \{\min, \max\}$, the call ValIter$(T, f, g, \varepsilon_{\mathit{float}})$ terminates.*
2. (Partial) correctness. *Consider any $g \in \{\max, \min\}$ and any $\triangle \in \{\Box, \Diamond\}$, and let $f = \min$ if $\triangle = \Box$, and $f = \max$ if $\triangle = \Diamond$. The following holds. 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, in a call to ValIter(T, f, g), the value-iteration converges to the limit from below if $f = \max$, and from above if $f = \min$ (see, e.g., the fixpoints in [9]). Consequently, we can replace statement 1 with the following initialization:

if $f = \max$ **then** $v := \mathbf{0}$ **else** $v := \mathbf{1}$

We will use this observation in the presentation of magnifying-lens abstraction.

4 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 $\varepsilon_{abs} > 0$, MLA enables the computation of upper and lower bounds for v^*, spaced less than ε_{abs}. To save space, rather than using a valuation v over S, MLA starts from an initial partition R of S, and computes the lower and upper bounds as valuations u^- and u^+ over R. To compute u^- and u^+, MLA iteratively considers each r in turn, and improves the estimates for $u^-(r)$ and $u^+(r)$ using value iteration on all $s \in r$. Once u^- and u^+ are computed, if $u^+(r) - u^-(r) \leq \varepsilon_{abs}$ for all $r \in R$, the algorithm terminates. Otherwise, the algorithm refines some of the partitions in R, and starts over. The algorithm is guaranteed to terminate: in the worst case, the partition is refined to the point in which every region consists of a single state, so that u^+ and u^- coincide.

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 $\varepsilon_{abs} > 0$. Parameter ε_{abs} indicates the maximum difference between the lower and upper bounds returned by MLA.

Algorithm 2 MLA($T, f, g, \varepsilon_{\mathit{float}}, \varepsilon_{abs}$)   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** $||u^+ - u^-|| \geq \varepsilon_{abs}$
12.         **then** $R, u^-, u^+ :=$ SplitRegions$(R, u^-, u^+, \varepsilon_{abs})$
13.         **else return** $R, u^-, u^+$
14.     **end if**
15. **end loop**
```

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

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

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 ε_{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 [9]. 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 $\varepsilon_{\mathit{float}}$.

4.1 Magnified Iteration

The algorithm for value iteration on the magnified region 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 $f = \min$. This is an important optimization. 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 "abstract" values of the regions to which they belong. To this end, we need to be able to efficiently find the "abstract" counterpart $[s]_R$ of a state $s \in S$. A general scheme, which works well for a wide variety of models, is as follows. Most commonly, the state-space S of a system 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 decision tree. The nodes of the tree are labeled by variables in X; the edges outgoing a node with label x are labeled with conditions on x. For instance, the top node for the partition of Figure 1(a) could be labeled with x (the horizontal coordinate), and have three descendants, with the edges labeled $x \leq 4$, $5 \leq x \leq 8$, $9 \leq x$. 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 following the tree, from root to leaf. If the splits are balanced (see the section below on partition refinement), this 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.

4.2 Adaptive Abstraction Refinement

If the lower and upper bounds u^- and u^+ computed for a partition R differ by more than ε_{abs} at some $r \in R$, MLA refines the partition R by splitting some regions. We have compared experimentally various criteria for selecting which regions to split. In our experiments, the best criterion (in terms of running time and space) was also the simplest: split all regions $r \in R$ with $u^+(r) - u^-(r) > \varepsilon_{abs}$. We considered an alternative criterion that split the regions where the precision of the computation suddenly degrades — the regions r that have large spread $u^+(r) - u^-(r)$, even though their neighbours have on average low spread. This alternative criterion yielded a slight decrease in the number of regions in the final abstraction, but caused an increase in the running time of the algorithm. Once the regions to be split have been selected, it remains to decide how to split them. 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 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 defining the state-space S: this defines a priority order for splitting regions. As previously mentioned, we represent a partition R via a decision tree, whose leaves correspond to the regions. To split a region r, we look at the label

x_i of its parent, and at the conjunction ϕ of all conditions on x_i on a path to r. If ϕ is satisfied only by one value of x_i, then we split r according to the values of x_{i+1}: we label r with x_{i+1}, and we create children of r in the tree, labeling the edges from r to the children with conditions on x_{i+1}. If ϕ is satisfied by multiple values of x_i, then we label r with x_i, and we create children of r in the tree, labeling the edges from r to the children with conditions on x_i.

Once a region r has been split into regions $r_1, \ldots, r_k$, we set $u^-(r_j) = u^-(r)$ and $u^+(r_j) = u^+(r)$ for all $1 \leq j \leq k$. 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.

4.3 Correctness

The following theorem summarizes MLA correctness.

Theorem 2. *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:*

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

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 (cfr. Theorem 1). If linear programming [10, 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 $\varepsilon_{float} > 0$.

5 Experimental Results

In order to evaluate the time and space performance of MLA, we have implemented the algorithm, and we have used it for two case studies: a minefield navigation problem, and the ZeroConf protocol for the autonomous configuration of IP addresses [5]. This latter example had been used as the test-case in [16], and it will enable us to compare MLA with the game-based abstraction algorithms presented there.

When comparing MLA to ValIter, we compute the space and time 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. A simple calculation yields that for a system with N states, the space usage is at least $\sqrt{8N}$. We measure the running time of the algorithms in terms of *valuation updates,* where a valuation update is the act of updating the value of v at a state, or the values of u^- or u^+ at a region, in any of the algorithms. Updating these valuations is where the algorithms spend most of the time. We measure the running time for a version of MLA that includes the optimization of re-evaluating a region, only when the value in some neighboring region changes by more than ε_{float}.

5.1 Minefield Navigation

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. We model this example by an MDP with state space $S = \{1, \dots, n\}^2 \cup \{s_{sink}\}$. 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 the sink state s_{sink}) 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

$$q(x', y') = \prod_{i-1}^{m} \exp\bigl(-0.7 \cdot \bigl((x' - x_i)^2 + (y' - y_j)^2\bigr)\bigr).$$

We experimented with two minefields: one of size 256×256, with 20 mines in it, and the other of size 512×512, with 100 mines in it. In both cases, the mines were distributed in a pseudo-random fashion across the field. The performance of algorithms ValIter and MLA are compared in Table 2. From the tables, in the 512×512 case we see that the savings range from 61.45, when $\varepsilon_{abs} = 10^{-1}$, to 28.33, when $\varepsilon_{abs} = 10^{-3}$. The theoretical minimum, when no region needs splitting, is $\sqrt{8 \cdot 512^2} = 1448$, and we see how our results are not too far off from this bound. For this same example, the number of valuation updates required by MLA is less than that required by ValIter by a factor of 5 to 7, approximately. While this is in part offset by other bookkeeping calculations performed by MLA, we see that the space savings come at no performance hit.

5.2 The ZeroConf Protocol

MLA and *game-based abstraction* [16] have different strengths and weaknesses: game-based abstraction is applicable to some systems with extremely large, and

$n = 256,\ \ \epsilon_{abs} = 10^{-1}$
$m = 20,\ \ \epsilon_{float} = 10^{-2}$

Algorithm	Space	Updates
ValIter	65,536	33,488,896
MLA	1,248	5,764,830

MLA Iteration Details			
#Abs	$\|R\|$	max Δ	Updates
1	256	0.528	3,097,996
2	316	0.528	215,943
3	376	0.528	101,924
4	436	0.497	79,538
5	496	0.067	57,829

$n = 256,\ \epsilon_{abs} = 10^{-2}$
$m = 20,\ \epsilon_{float} = 10^{-4}$

Algorithm	Space	Updates
ValIter	65,536	33,488,896
MLA	1,872	3,712,081

MLA Iteration Details			
#Abs	$\|R\|$	max Δ	Updates
1	256	0.528	3,164,812
2	325	0.528	246,678
3	412	0.528	118,825
4	568	0.497	104,647
5	808	0.001	77,119

$n = 256,\ \epsilon_{abs} = 10^{-3}$
$m = 20,\ \epsilon_{float} = 10^{-6}$

Algorithm	Space	Updates
ValIter	65,536	33,488,896
MLA	2262	3,827,678

MLA Iteration Details			
#Abs	$\|R\|$	max Δ	Updates
1	256	0.528	3,232,396
2	334	0.528	273,694
3	442	0.528	137,897
4	643	0.497	101,761
5	1,003	0.0009	81,930

$n = 512,\ \ \epsilon_{abs} = 10^{-1}$
$m = 100,\ \ \epsilon_{float} = 10^{-2}$

Algorithm	Space	Updates
ValIter	262,144	268,173,312
MLA	4276	23,545,536

MLA Iteration Details			
#Abs	$\|R\|$	max Δ	Updates
1	576	0.778	18,879,885
2	876	0.582	2,273,902
3	1,173	0.583	865,780
4	1,470	0.583	562,006
5	1,767	0.524	473,213
6	1,896	0.072	490,750

$n = 512,\ \ \epsilon_{abs} = 10^{-2}$
$m = 100,\ \ \epsilon_{float} = 10^{-4}$

Algorithm	Space	Updates
ValIter	262,144	268,173,312
MLA	7,216	32,350,918

MLA Iteration Details			
#Abs	$\|R\|$	max Δ	Updates
1	576	0.777	26,501,993
2	906	0.583	2,743,462
3	1,278	0.583	974,042
4	1,798	0.583	771,216
5	2,673	0.525	788,025
6	3,366	0.003	572,180

$n = 512,\ \epsilon_{abs} = 10^{-3}$
$m = 100, \epsilon_{float} = 10^{-6}$

Algorithm	Space	Updates
ValIter	262,144	268,173,312
MLA	9136	35,832,505

MLA Iteration Details			
#Abs	$\|R\|$	max Δ	Updates
1	576	0.777	28,374,281
2	939	0.583	3,487,939
3	1,392	0.583	1,420,126
4	2,127	0.583	1,110,869
5	3,431	0.525	806,507
6	4,326	0.0009	632,783

Fig. 2. Comparision between MLA and ValIter for 256 × 256 and 512 × 512 minefields, for $\varepsilon_{abs} = 10^{-1}, 10^{-2}, 10^{-3}$. n is the dimension of the minefield, m the number of mines, #Abs is the number of abstraction steps (number of loops 3–15 of MLA), and $\max \Delta = \max_{r \in R}(u^{+}(r) - u^{-}(r))$.

even infinite, state spaces, but can fail to provide benefits in cases where MLA is effective, as in the minefield example. In order to compare the two approaches in a case where they are both applicable, we considered the *ZeroConf* protocol case study considered in [16]. The ZeroConf protocol is used for the dynamic self-configuration of a host joining a network. We consider a network with n existing hosts, and m total IP addresses; protocol messages have a certain probability of being lost during transmission. We use the same model as in [16], taking $n = 4$ and $m = 32$. The model has the following state variables:

- $s \in [0..4]$ denoting the protocol state of the process joining the network;
- $ip0 \in [0..m]$ is the IP address chosen by the new host;
- $x0 \in [0..20]$ is a local clock;
- $probes \in [0..4]$ is the number of IP-probing messages sent.

In game-based abstraction, the authors of [16] manually selected the abstract state space, which consisted of 737 states. The most important protocol property they analyzed was the probability that the new host would eventually be correctly configured (this is a reachability property, where the target set T consists of the states where the new host has been configured. In addition, [16] considered various time-bounded properties.

To construct the abstraction in MLA, it sufficed to choose a variable priority ordering for splitting regions: from high to low priority, we chose s, $ip0$, $x0$, $probes$. We used MLA to compute the probability that the new host would eventually be correctly configured. MLA needed to split according to s and $ip0$, thus considering $5 \cdot 33 = 165$ regions (the actual count was lower, since some of these regions are unreachable from the initial state of interest).

While it may appear that MLA is more economical in constructing the abstraction, the difference is explained by the fact that MLA did not need to split according to the values of clock $x0$, as the property considered was not a bounded reachability property. Rather, there were three main differences between the two verification approaches as applied to the ZeroConf protcol.

- The abstraction had to be selected manually in game-based abstraction, whereas it was constructed automatically (except for specifying the variable ordering) in MLA.
- Once an appropriate abstraction is devised, game-based abstraction is able to cope with larger state spaces than MLA, as it is not limited by the square-root lower-bound on the number of states considered by MLA.
- Game-based abstraction requires the use of game-theoretic algorithms for solving the abstract model. On the other hand, MLA can use standard MDP techniques, which are easier to implement and more efficient.

6 Discussion

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, 15, 19, 16]? 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 [17]; this quantity is related to building abstractions via *weak simulation* [24, 2, 21]. 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 [22, 11, 12]. 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. It is unclear whether this approach is beneficial in practice. If the size N of the state space is such that a reduced abstraction of size $\sqrt{8N}$ does not fit in memory, it is likely that even if the space requirements were tamed via hierarchical approaches, the time requirements would make the analysis unfeasible.

Symbolic representations such as ADDs and MTBDDs [6, 1] have been used for representing the value function compactly [8, 14]. The space savings are limited by the fact that the value function is usually slightly different at different states. MLA is loosely reminiscent of approaches that cluster MTBDD leaves with values within a specified $\varepsilon > 0$. However, the similarity is superficial: such leaf-clustering corresponds in MLA to taking $\varepsilon_{abs} = \varepsilon_{\mathit{float}} = \varepsilon$, and yields considerably poorer results than clustering according to ε_{abs}, and computing according to $\varepsilon_{\mathit{float}}$, as MLA does. In particular, MTBDD leaf-clustering approaches do not yield lower and upper bounds for the property of interest. 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.

Acknowledgements. We thank Pascale Garaud (AMS Department, UC Santa Cruz) for an insightful discussion on adaptive mesh refinement algorithms.

References

1. R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Journal of Formal Methods in System Design*, 10(2/3):171–206, 1997.
2. C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. In *CAV 97*, volume 1254 of LNCS, pages 119–130, Springer-Verlag, 1997.
3. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.
4. D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995. Volumes I and II.
5. S. Cheshire, B. Adoba, and E. Gutterman. Dynamic configuration of ipv4 link local addresses (internet draft).
6. E. Clarke, M. Fujita, P. McGeer, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *International Workshop for Logic Synthesis*, 1993.
7. P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proc. of PAPM/PROBMIV*, volume 2165 of LNCS, pages 39–56. Springer-Verlag, 2001.
8. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In *TACAS 00*, volume 1785 of LNCS, pages 395–410. Springer-Verlag, 2000.
9. L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. *Journal of Computer and System Sciences*, 68:374–397, 2004.
10. C. Derman. *Finite State Markovian Decision Processes*. Academic Press, 1970.
11. I. Gilboa. Expected utility with purely subjective non-additive probabilities. *Journal of Mathematical Economics*, 16:65–88, 1987.
12. I. Gilboa and D. Schmeidler. 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.
13. H. Hansson. *Time and Probabilities in Formal Design of Distributed Systems*. Real-Time Safety Critical Systems Series. Elsevier, 1994.
14. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS 06*, volume 3920 of LNCS, pages 441–444. Springer-Verlag, 2006.
15. M. Huth. On finite-state approximations for probabilistic computational-tree logic. *Theor. Comp. Sci.*, 346(1):113–134, 2005.
16. M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for markov decision processes. In *Proc. of QEST: Quantitative Evaluation of Systems*, pages 157–166. IEEE Computer Society, 2006.
17. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.
18. A. McIver and C. Morgan. *Abstraction, Refinement, and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer-Verlag, 2004.
19. D. Monniaux. Abstract interpretation of programs as Markov decision processes. *Science of Computer Programming*, 58(1–2):179–205, 2005.
20. B. Plateau. 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, pages 147–154, New York, NY, USA, 1985.

21. A. Plilippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In *CONCUR 00*, volume 1877 of LNCS, pages 334–349. Springer-Verlag, 2000.
22. D. Schmeidler. Integral representation without additivity. *Proceedings of the American Mathematical Society*, 97:255–261, 1986.
23. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995. Technical Report MIT/LCS/TR-676.
24. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR 94*, volume 836 of LNCS, pages 481–496. Springer-Verlag, 1994.
25. M. Vardi. Automatic verification of probabilistic concurrent finite-state systems. In *Proc. 26th IEEE Symp. Found. of Comp. Sci.*, pages 327–338. IEEE Computer Society Press, 1985.