Solving Markov Decision Processes in Metric spaces

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present an approximation scheme for solving Markov Decision Processes (MDPs) in which the states are embedded in a metric space. Our algorithm has a time bound of $\tilde{O}(n^{\frac{\log\log n}{\epsilon}})$, where n is the number of states and ϵ the approximation factor. This bound is independent of the numerical size of the input and the discount factor and is hence *strongly* polynomial. We present the result for deterministic MDPs and discuss extensions to stochastic domains.

Unlike most MDP algorithms, ours does not make local iterative updates, but uses a divide and conquer approach. It partitions the space into nested regions and considers only policies that transition between partitions through *portals*. A simple dynamic program over the partitioned structure then returns a near-optimal policy. Our technique is very general and is likely to give insight into related problems like Reinforcement Learning and Partially Observable MDPs. In fact, using our algorithm in conjunction with a result from [1], we show a strongly polynomial reinforcement learning algorithm for metric MDPs.

1 Introduction

The Markov Decision Process (MDP) is a popular formalism for modelling decision-making tasks in complex stochastic environments. The objective is to minimize the sum of accumulated costs incurred over a series of discrete time steps by probabilistic transitions within a state space. To solve an MDP problem means to compute a *policy* that defines an action for each state minimizing the total cost. Solutions of MDPs are both directly useful for decision making and fundamental for other tasks like Reinforcement Learning and Partially Observable MDP algorithms.

The classical algorithms (see [2]) for solving a finite-state infinite-horizon MDP are Value Iteration, Policy Iteration, and reduction to linear programming. The running time of Value Iteration is polynomial in the size of the state space $|S| = n$, the action space $|A|$, $\frac{1}{(1-\gamma)}$ where γ is the discount factor, and B the number of bits required to represent the transitions, rewards and discount factor. Consequently, value iteration is a pseudo-polynomial time algorithm, i.e. it is polynomial in the numerical size (rather than the combinatorial size), and only if γ is a fixed constant or represented in unary. This makes the solution somewhat unsatisfactory because its performance could be made significantly worse merely by increasing the precision of the numbers in the input or the discount factor. Policy Iteration and Linear programming solutions also have similar pseudo-polynomial time bounds.

It is an important open question whether a strongly polynomial time algorithm can be found for solving MDPs i.e. one which is polynomial in the combinatorial size of the input. Finding such an algorithm is of significant theoretical and practical interest. For problem domains where high-precision numbers routinely arise or γ is large, it would clearly be preferable to use such an algorithm. To date the best result on general MDPs is a randomized form of policy iteration with an expected running time of $O(2^{0.78n})$[3].

In this paper, we make progress towards answering this question by presenting an approximation scheme for MDP problems embedded in Euclidean metric spaces i.e. MDPs where the cost function is the distance between states in the metric space. Our algorithm takes time $\tilde{O}(n^{\frac{\log\log n}{\epsilon}})$ to return a policy whose value is within $1+\epsilon$-factor (ϵ-close) of the optimum. In particular, this bound is independent of the numerical size (the precision of the input) and γ (and hence the horizon time). It is also the first divide-and-conquer approach to solving MDPs that has a guaranteed approximation factor (unlike factored MDPs).

Metric MDPs occur naturally in many decision-making contexts. For example, most path-finding and robot navigation tasks require movement through a two or three-dimensional Euclidean space where costs are proportional to distance travelled. There are algorithms that can transform an arbitrary metric into a Euclidean metric in sufficiently high dimensions with at most $O(\log n)$ distortion [4]. *Multi-dimensional scaling* [5] is a statistical technique that fits a Euclidean metric to a general distance measure, but without any distortion guarantees.

Our method is radically different from the commonly used iterative methods for solving MDPs, and is of independent interest. Briefly, we recursively partition the metric space into rectangular regions and define points along the boundaries of the partitions called *portals* in the manner of [6]. The number of these portals, and hence the running time, will depend on the desired approximation ratio ϵ. We restrict our search to *portal respecting policies*, i.e. those that do not exit a region except through a portal on its boundary. It is shown that the value of the optimal policy-respecting policy is ϵ-close to the true optimal policy and moreover can be found by a dynamic program called the *Patching Procedure* that iterates over the recursive structure of the partitions. This technique is sufficiently general that we believe it could be extended to solving other problems in the same setting. For example, using our algorithm as the approximate planner in the Metric-E^3 algorithm of [1] yields a strongly polynomial reinforcement learning algorithm for Euclidean metric spaces.

For clarity, we present our results for the simpler case of MDPs with deterministic transition functions and then indicate how to modify the algorithm for stochastic MDPs where the transition probability from states s to s' is bounded by $\frac{1}{d(s,s')^M}$, where M is the number of dimensions.

2 Metric MDPs and Portal Respecting Policies

We assume the reader is familiar with the standard terminologies from Markov Decision Processes and Reinforcement Learning (see [2]). An MDP $M = (S, A, T, \gamma, c)$ is embedded in a k-dimensional Euclidean metric space $\mathfrak{M}$, if there exists a function $f : S \mapsto \mathfrak{M}$ s.t. $c(s, a, s') = d_{\mathfrak{M}}(f(s), f(s'))$. Our goal in solving the MDP M is to find a policy π s.t. $V^\pi(s) \leq (1+\epsilon)V^*(s)$ for all $s \in S$. Note that c is a cost and not a reward function.

We make a few simplifications to the problem for the sake of clarity and brevity: We assume that all transitions are deterministic, that the graph is fully connected (i.e. there is an action from each state to every other), and that we are working in the $\mathbb{R}^2$ space. Only the first of these assumptions requires a non-trivial extension to our algorithm when it is relaxed. The modifications necessary for MDPs with probabilistic transitions are given in section 4.

Our exposition will proceed as follows: In this section we present the construction of randomized dissections and portals for a Metric MDP instance. Using this construction, we define a subset of candidate policies called *portal respecting*, and show that there are policies in this class that are ϵ-close to optimal (Theorem 1). Therefore it is enough to look for a good policy in this class. In section 3 we present the actual algorithm that finds this portal-respecting policy.

We sometimes use the term *node* instead of state and *edge* instead of action to make the relation to graph algorithms more explicit. We will abuse notation slightly by writing $V^\pi \leq B$ to mean $V^\pi(s) \leq B, \forall s \in S$, where B is a number.

2.1 Perturbation and Scaling

First we apply a simple perturbation to the MDP instance to ensure that the minimum distance between two nodes, $c_{min} \geq 1$ and the maximum distance, $c_{max} \leq \frac{n^2}{2}$.

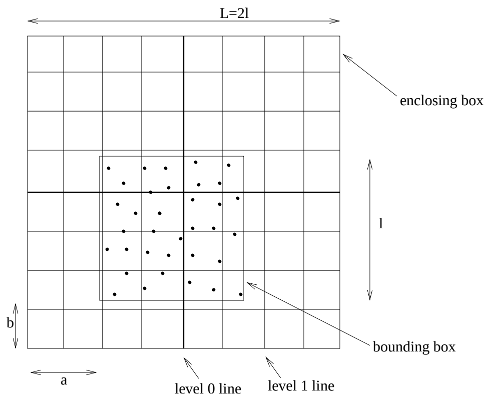

Figure 1: A randomized dissection. The nodes are all contained within the bounding box. Random vertical and horizontal shifts a and b are chosen for the enclosing box, which is then recursively partitioned.

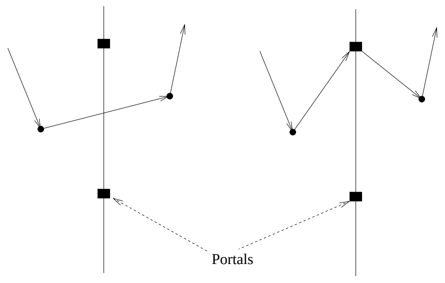

Figure 2: A portal respecting path (on the right) must take a detour through a portal every time it crosses a gridline. The extra cost of doing so (w.r.t. the original path on the left) is at most the inter-portal distance.

Let d be the maximum distance between any two nodes in the input. We lay down a grid in which the grid lines are separated by distance $\frac{\epsilon d}{n^{1.5}}$ and move each node to its nearest grid point. If this makes two or more points coincide, then they are treated as a single node for the purposes of our algorithm and the policy we return behaves identically on them. This procedure moves each node by at most $\frac{\epsilon d}{n^{1.5}}$, affecting the optimal value function by at most $\frac{\epsilon d}{(1-\gamma)n^{0.5}}$ which is negligible compared to ϵV^* when n is large. Then we rescale distances so that $c_{min} \geq 1$. Hence, $c_{max} \leq \frac{n^{1.5}}{\epsilon} \leq \frac{n^2}{2}$ as desired.

2.2 Randomized Dissections

A *dissection* is a recursive partitioning of a square region into squares such that the smallest squares in the dissection have sidelength 1; See the enclosing box of figure 1. We view this partitioning as a tree of squares (similar to a quadtree) with the original square as the root and each interior node having 4 children (corresponding to the four quadrants of the square). The *level* of a square in the dissection is its depth from the root; the root having depth 0. We also say that the horizontal and vertical lines of the dissection that divide a square of level i into 4 squares of level $i+1$ have *level* i.

The *bounding box* is the smallest axis-parallel square containing all the nodes of the problem instance. Let l be its sidelength and t be its lower left endpoint. An *enclosing box* is a square with sidelength $L = 2l$ positioned so that t is at a distance a from its left edge and b from its bottom edge where a and b (called the *horizontal* and *vertical shift* respectively) are integers such that $0 \leq a, b < l$. A *randomized dissection* is a dissection of the enclosing box with a and b chosen uniformly at random.

We assume without loss of generality that L is a power of 2 (if not, we increase the size of the bounding box) so the squares in the randomized dissection have integer endpoints (i.e. their sides are along the unit grid) and leaf squares have sides of length 1 and hence at most one node in them.

Proposition 1. *Let ℓ be a fixed line on the unit grid that intersects the bounding box. Then w.r.t. the randomized dissection of the enclosing box,*

$$\Pr_{a,b}[\ell \text{ is at level } i] = \frac{2^i}{l} = \frac{2^{i+1}}{L} \tag{1}$$

2.3 Portals and Portal Respecting Policies

We introduce new states called *portals* to our problem instance. The portals will serve as fixed points of exit and entry into each dissection square. A level i line will have $2^{i+1}m$ equally spaced portals on it, where m is the *portal parameter* which we set to be $\frac{8 \log n}{\epsilon}$. (The justification for this value will become apparent later). Thus, each square of the dissection (at any level) has at most $4m$ portals on its boundary. We also introduced edges from each portal to every state in the instance and vice versa, with costs equal to the distance between them (as usual).

Crucially, the portals are treated as *undiscounted*, which means that passing through a portal does not incur an extra time step. In the computation of the value function, the costs of succeeding states are not multiplied by an extra factor of γ after passing through a portal. In particular, the Bellman equation for a portal p would be

$$V^{\pi}(p) = d(p, \pi(p)) + V^{\pi}(\pi(p))$$

where the factor of γ is abent from the right-hand side. The intuition is that portals are only used as an interface between squares and are not states *per se*.

A *portal respecting path* is a path that only crosses a grid-line at a portal on that line (figure 2). A *portal respecting policy* is one in which all paths are portal respecting. Thus, for a portal respecting policy to move from state s_1 to s_2 it must visit one portal on each grid line between the two states.

The key idea of the algorithm is to restrict our search for an optimal policy to the set of portal respecting policies. In the next section we sketch a dynamic programming algorithm to find the optimal portal respecting policy. Now, we show that the value of this policy is ϵ-close to the true optimum with high probability, if we set $m \geq 8\frac{\log n}{\epsilon}$.

Let $V^*_{a,b,m}$ be the value function for the optimal portal respecting policy when the portal parameter is m and the random shifts are a, b respectively. For nodes s, s' let the *portal respecting cost*, $d_{a,b,m}(s, s')$ be the cost of the shortest portal respecting path from s to s'. We have the following lemma:

Lemma 1. *When the vertical and horinzontal shifts a and b are chosen uniformly at random from $0, 1, \ldots, l-1$, then*

$$E_{a,b}[d_{a,b,m}(s, s')] \leq (1 + \frac{2\log L}{m})d(s, s').$$

Proof. Please see supplementary material. □

Lemma 1 shows that the length of the shortest portal respecting path between two points cannot be too much larger than the straight-line distance. We use this to bound the value function of the optimal portal respecting policy in terms of the optimal value function V^*:

Theorem 1. *If L is the sidelength of the enclosing box and m is the portal parameter, then for any state s,*

$$E_{a,b}[V^*_{a,b,m}(s)] \leq (1 + 2\frac{\log L}{m})V^*(s)$$

Proof. Please see supplementary material. □

Consequently, the probability is at least $\frac{1}{2}$ that $V^*_{a,b,m} - V^* \leq 4\frac{\log L}{m}V^*$. Therefore, when $L \leq n^2$ (as ensured by the perturbation) and $m \geq 8\frac{\log n}{\epsilon}$, $V^*_{a,b,m} \leq (1+\epsilon)V^*$ with probability $\frac{1}{2}$. After repeated independent trials we get an ϵ-optimal policy with high probability.

Note that this procedure can be de-randomized by trying all choices of the shifts a and b to get a deterministic algorithm (with some loss of efficiency). Finally observe that we can recover an ϵ-optimal policy for the original problem by shortcutting the portals on each portal respecting path, which only reduces the total cost.

3 The Patching Procedure

In the previous section, we showed that the optimal portal respecting policy is ϵ-close in value to the true optimum. It suffices then to show an algorithm for finding an optimal (or near-optimal) portal respecting policy, which makes the problem easier. The key insight is that for a portal respecting policy, the nodes inside a square S interact with the outside only through the portals of S. This

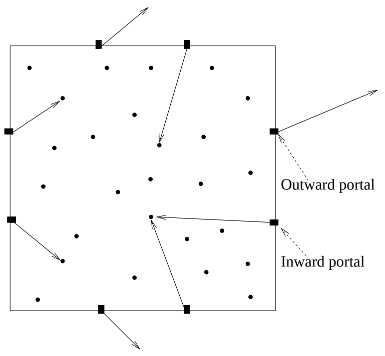

Figure 3: The portals on the boundary of a dissection square point either inward or outward. Those that point outward have their values fixed and taken from $\mathbb{V}$. This defines an *interface* to the square.

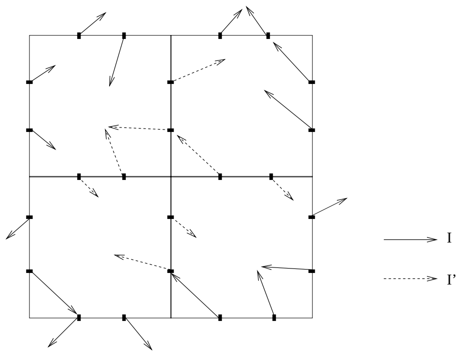

Figure 4: The patching procedure. The interface I for the portals on the outer boundary is given. When an interface I' is chosen for the portals along the bisecting lines, the interfaces for the child squares are completed and its policy can be looked up in the interface table.

suggests the following dynamic programming solution: First solve the subproblem for the 4 child squares of S assuming the values at the portals are known, then use these to solve S by "patching" (Figure 4).

For each portal p on the boundary of S, we choose whether it *points inwards* or *outwards*, i.e. whether the policy for that portal will lead to a node inside or outside S (see figure 3). If p points inward, it is treated as a regular node as far as solving the sub-problem for S is concerned. If p points outwards, its optimal action cannot be determined by solving S but we still need to know its value since other nodes in S could lead to it. So for each outward-pointing portal p we guess a value for $V(p)$ from the following set:

$$\mathbb{V} = \{V_{min},\ (1+\delta)V_{min},\ (1+\delta)^2 V_{min}, \ldots, V_{max}\}$$

where, $V_{min} = c_{min}/(1-\gamma)$, $V_{max} = c_{max}/(1-\gamma)$ and $\delta = O(\frac{\epsilon}{n^2})$. The intuition is that the optimal value function must be within a factor of $(1+\delta)$ of one of these values and is thus well-approximated by it. The number of values in $\mathbb{V}$ is

$$\log_{1+\delta} \frac{V_{max}}{V_{min}} = \log_{1+\delta} \frac{c_{max}}{c_{min}} = 2 \log_{1+\delta} n$$

An *interface* for a square S is an assignment of values from $\mathbb{V}$ to each outward-pointing portal on the boundary of S. The *value function for interface I and policy π*, V_I^π, is the value function (on the nodes in S) achieved by the portal respecting policy π when the values of all outward pointing portals of S are given by I. The *optimal value function for interface I*, V_I^*, is the lowest such function.

Our dynamic program maintains a look-up table $\mathbb{T}$ that stores for each square and each interface, a near-optimal portal respecting policy. The look-up table is filled in bottom-up. After the table has been filled in for the four child nodes of S, we do the same for S and each possible interface I of S by "patching" together the subsolutions:

Procedure *Patch*$(S, I, \mathbb{T})$
Input: Square S at level i, Interface I for S, Look-up Table $\mathbb{T}$ for descendants of S

```
1. For every possible interface $I'$ for the portals on the level $i$ lines bisecting $S$ (See fig. 4.)
   (a) Use $I$ and $I'$ to define interfaces $I_1, \ldots, I_4$ for the child squares $S_1, \ldots, S_4$ of $S$.
   (b) $\pi_{I'} := \mathbb{T}(S_1, I_1) \cup \mathbb{T}(S_2, I_2) \cup \mathbb{T}(S_3, I_3) \cup \mathbb{T}(S_4, I_4)$
   (c) Compute $V_{I'}^{\pi_{I'}}$ by Gaussian elimination on the following system of linear equations:
       $V_{I'}^{\pi_{I'}}(s) = I(s)$   ($\forall s$, $s$ is an outward-pointing portal at level $i-1$ )
       $V_{I'}^{\pi_{I'}}(s) = d(s, \pi_{I'}(s)) + V_{I'}^{\pi_{I'}}(\pi_{I'}(s))$   ($\forall s$, $s$ is an inward portal at level $i-1$ or a portal at level $i$)
       $V_{I'}^{\pi_{I'}}(s) = d(s, \pi_{I'}(s)) + \gamma V_{I'}^{\pi_{I'}}(\pi_{I'}(s))$   (otherwise)
```

2. $I^o := \text{argmin}_{I'} V_{I'}^{\pi_{I'}}$; $\mathbb{T}(S, I) := \pi_{I^o}$
3. Return $\mathbb{T}$

This is repeated all the way up to the root square, where the maximization is done without choice of an interface to get a portal respecting policy π for all the nodes. Now we prove that π is ϵ-close to the optimal portal respecting policy, which we have already seen is ϵ-close to the true optimum.

Theorem 2. *The Dynamic Program outlined above when run on a problem instance with random shifts a and b and portal parameter m, returns a portal respecting policy π s.t. $V^\pi \leq (1+\delta)^h V^*_{a,b,m}$, where $V^*_{a,b,m}$ is the optimal portal respecting value for a, b and m, and h is the depth of the dissection tree ($h = O(n^2)$).*

Proof. Please see supplementary material. □

From the above analysis, it follows that setting $\delta = O(\frac{\epsilon}{n^2})$ gives an approximation ratio to the optimal value of $(1 + \frac{\epsilon}{n^2})^{n^2} \approx 1 + n^2 \cdot \frac{\epsilon}{n^2} = 1 + \epsilon$, as desired.

3.1 Complexity Analysis

The interface table for a dissection square stores a policy for each possible interface. Therefore, its size is at most:

$$\begin{aligned}(2\log_{1+\delta} n)^{4m} \cdot n &= (2\log_{1+\delta} n)^{4\frac{\log n}{\epsilon}} \cdot n \\ &= \tilde{O}(n^{\frac{\log\log n}{\epsilon}})\end{aligned}$$

There are $O(n^2)$ such dissection squares. To fill in each entry, the corresponding entries for the sub-squares are looked up, the policies are patched together, and Gaussian elimination is used to compute V_I. This takes subquadratic time per entry, and therefore the running time of the overall algorithm is bounded by the size of the table : $\tilde{O}(n^{\frac{\log\log n}{\epsilon}})$

It might be possible to keep the interface table polynomial-size by using the fact that not every entry will be looked up by the parent. For example, a smart patching procedure would try lower values first for the interface and if it finds a policy that achieves this value, it would stop searching.

4 Stochastic MDPs

In sections 2 and 3 we presented our divide-and-conquer algorithm for solving deterministic MDP instances. This algorithm generalizes naturally to the case where the transitions are probabilistic, with a few assumptions. First, we need the probability of making a transition to a state far away from the current one to be small:

$$T(s, a, s') = O(\frac{1}{d(s, s')^M}) \tag{2}$$

where M is the number of dimensions. This is needed to bound the probability that an action moves to a state outside a given square. Also, the number of possible actions per state cannot be too large, i.e. $|A| = O(n^\alpha)$ for some constant α, so that the size of the interface table is bounded. These assumptions are reasonable since most MDPs with states embedded in a metric space are likely to have few actions and generally allow movement only in short jumps.

The algorithm proceeds as follows: The perturbation and the randomized dissection remain unchanged from the deterministic case. Portals are placed on the dissection lines just as before, however the actions defined on them are different: For each action that transitions with a certain probability from one state to another outside its dissection square, another action is introduced that redistributes the probability mass for that transition non-uniformly over bordering portals. The transition probabilities for these actions can be pre-computed in $O(n^{\frac{1}{\epsilon}})$ time. The patching procedure remains the same, however we can no longer talk about portals pointing inward or outward. Instead, the value chosen for each portal in the interface is used by the squares on both sides of the dissection line. For lack of space we omit the details, but will provide them in an extended version of this paper.

1. Use random moves until encountering a state s with at least one known action a (i.e. with at least α-close previous experiences to (s, a)).
2. Execute Plan twice, once using the generative model for $\hat{M}_{\text{exploit}}$ and once using the generative model for $\hat{M}_{\text{explore}}$. Let the resulting policies be π_{exploit} and π_{explore}.
3. If $V_{\hat{M}_{\text{explore}}}(\pi_{\text{explore}}, s) > \beta$, execute π_{explore} for the next T steps, then go to step 1.
4. Else HALT and output π_{exploit}.

Figure 5: The Metric-E^3 Algorithm from [1].

5 Metric Reinforcement Learning

In [1], an algorithm called Metric-E^3 was developed for optimal exploitation versus exploration in metric MDPs assuming the existence of an approximate planning algorithm and a local modeling assumption (described below). As the authors explain, the idea is not to trivialize the planning component of reinforcement learning but to abstract it away from the exploration vs. exploitation problem. Now, we demonstrate that using our metric MDP solver as the planner in conjunction with Metric-E^3 gives a complete, strongly polynomial algorithm for near-optimal reinforcement learning in a metric space. First, let us recapitulate the assumptions of [1]:

Local Modeling Assumption. There exists an algorithm *Model* for the MDP M such that, for any $(s, a) \in (S, A)$, if *Model* is given k transitions $(s', a') \to s''$ and costs distributed in accordance with M and in which all $d(s, s') \leq \alpha$, then *Model* outputs a state $\hat{s} \sim \hat{P}(\hat{s}|s, a)$ and a cost $\hat{c}$, where $\sum_{\hat{s}} |\hat{P}(\hat{s}|s, a) - P(\hat{s}|s, a)| \leq \alpha$, and $|c(\hat{s}) - c(s)| \leq \alpha$. Let t_{model} be the maximum running time of *Model*.

This is easy to satisfy in our case, where the distance measure d is truly a metric (which is not necessary in [1]) and therefore satisfies the triangle inequality. Given m sample transitions $(s', a') \to s''$, we simply select the mean of the s'''s, $\bar{s}$ as the state returned by *Model* with a cost given by $d(s, s'')$. This immediately ensures that the second condition (on the costs) holds. Using the bound on the transition probabilities (Eq. 2) and a Chernoff bound it can be shown that the first condition is satisfied when $k = \Omega(\log m)$.

Approximate Planning Assumption. There exists an algorithm *Plan*, which given a generative model for an unknown MDP M, and a state s, returns a policy π whose value $V_M(\pi, s) < V_M(\pi^*, s) + \beta$, where π^* is the optimal policy for M. Let t_{plan} upper bound the running time of *Plan* and c_{gen} upper bound the calls to the generative model.

It immediately follows that our metric MDP algorithm satisfies the approximate planning assumption with $\beta = \epsilon V_M(\pi^*, s)$.

The Metric E^3 algorithm of [1] is shown in figure 5. $\hat{M}_{\text{exploit}}$ is the MDP obtained by restricting the state space of the input MDP M to the currently known states and a single absorbing state for all the unknown states of M. $\hat{M}_{\text{explore}}$ is similiarily defined, except that rewards are given only for escaping known states and reaching unkown states.

Theorem 3. *Suppose $\beta > (T+1)$. With probability $1 - \delta$, after at most $k = \frac{Tm}{\epsilon - (T+1)} \ln(\frac{1}{\delta}) + O(\frac{\log n}{\epsilon})$ actions in M, Metric-E^3 with the metric planning algorithm of section 3 halts at a state s, and outputs a policy π such that $V_M(\pi, s) \leq V_M(\pi^*, s) + 3\epsilon V_M(\pi, s) + 2(T+1)$. The running time of Metric-E^3 is at most $O(k(k-1)/2 + 2k(t_{plan} + t_{model}))$.*

Proof. Follows by combining Theorems 4.1 and 4.2 of [1] with the above discussion. $\square$

6 Related Work

The current best bound on the running time of general MDPs in terms of combinatorial size alone, independent of discount factor is by [3] who showed that random policy iteration, where the set of states at which the policy update is performed is randomly chosen at each iteration, takes time $O(2^{0.78n})$. [7] showed an analogous result for value iteration under the average-cost criterion by transforming to the problem of finding highest-mean cycles. [8] studied policy iteration as a Newton iteration method and showed strong polynomial bounds for MDPs with almost acyclic graph structure.

There has been very little attention given to the study of MDP problems in metric spaces. [9] gave a linear programming approximation for a Markov Control Process in a metric space, which is a different framework from MDPs (e.g. they have infinite state-spaces) and hence cannot be compared to our work. Moreover, they do not give a time bound for their algorithm. [10] gave algorithms for learning feasible trajectories to goal regions in high-dimensional metric spaces.

There is a superficial similarity between our result and work on factored MDPs and hierarchical reinforcement learning. For example, the *Airport algorithm* of [11] builds a hierarchy of landmarks that are used as sub-goals for navigating from state to state. Current work on *Factored MDPs* [12] use approximate Linear Programming to exploit modular structure when the Value function is a linear combination of basis functions. Note however that there the hierarchies of the state-space or the basis decompositions are usually pre-defined by the programmer and there are almost no results on the *global* optimality of the solution.

7 Conclusions

We presented a near-polynomial time approximation scheme for solving Markov Decision Processes embedded in metric spaces. The running time for this method is independent of the size of the representation of the parameters and the discount factor, the best such result so far (to our knowledge) on a non-trivial class of MDP problems. It is also the first algorithm that uses problem structure to obtain a global bound on the optimality of the solution. Most importantly, the technique used is fundamentally different from previous methods and we believe it is likely to lead to better insights on related problems.

References

[1] S. Kakade, M. Kearns, and J. Langford, “Exploration in metric state spaces”, in *Proc. of the 20th International Conference on Machine Learning*, 2003.

[2] Michael Littman, *Algorithms for Sequential Decision Making*, PhD thesis, Brown University, 1996.

[3] Yishay Mansour and Satinder Singh, “On the complexity of policy iteration”, in *Proc. Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI ’99)*, 1999.

[4] Jean Bourgain, “On Lipschitz embeddings of finite metric spaces in Hilbert space”, *Israel J. Math.*, vol. 52, no. 1–2, pp. 46–52, 1985.

[5] J.B. Kruskal and M. Wish, *Multidimensional Scaling*, Sage Publications, Beverly Hills, CA, 1977.

[6] Sanjeev Arora, “Polynomial-time approximation schemes for euclidean TSP and other geometric problems”, *Journal of the ACM*, vol. 45, no. 5, pp. 753–782, 1998.

[7] Omid Madani, “Polynomial value iteration algorithms for deterministic MDPs”, in *Proc. Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI ’02)*, 2002.

[8] Omid Madani, “On policy iteration as a Newton’s method and polynomial policy iteration algorithms”, in *Proc. National Conference on Artificial Intelligence (AAAI ’02)*, Menlo Park, CA, USA, 2002, pp. 273–278, American Association for Artificial Intelligence.

[9] O. Hernandez-Lerma and J.B. Lasserre, “Linear programming approximations for Markov Control Processes in metric spaces”, *Acta Appl. Math.*, vol. 51, pp. 123–139, 1997.

[10] Andrew W. Moore, “The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces”, in *Advances in Neural Information Processing Systems*, Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, Eds. 1994, vol. 6, pp. 711–718, Morgan Kaufmann Publishers, Inc.

[11] Andrew Moore, Leemon Baird, and Leslie Pack Kaelbling, “Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs”, in *Proc. Sixteenth International Joint Conference on Artificial Intelligence (IJCAI ’99)*, 1999, pp. 1316–1323.

[12] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman, “Efficient solution algorithms for factored MDPs”, *Accepted in Journal of Artificial Intelligence Research (JAIR)*, 2002.