Fast SSP Solvers Using Short-Sighted Labeling

Luis Pineda and **Kyle Hollins Wray** and **Shlomo Zilberstein**
College of Information and Computer Sciences
University of Massachusetts Amherst
{lpineda,wray,shlomo}@cs.umass.edu

Abstract

State-of-the-art planning methods for goal-oriented Markov Decision Processes often work by limiting planning to restricted regions of the state space. The resulting problems can then be solved quickly, and the process repeated during execution when states outside the restricted region are encountered. Typically, planning in these approaches is constrained to states that are within some distance measure of the start state (e.g., number of actions or probability of being reached). However, this short-sighted approach sometimes makes it difficult to propagate information from states that are closer to the goals than to the start state, and thus misses opportunities for improved planning. In this work, we show an alternative approach in which short-sightedness is used only for deciding whether a state should be labeled as solved or not, but otherwise the set of states that can be accounted for during planning is unrestricted. Based on this idea, we propose the FLARES algorithm and show that it outperforms other state-of-the-art methods in several benchmark domains.

Introduction

Markov Decision Processes (MDPs) are a highly-expressive model for probabilistic sequential decision making. One type of MDP that has received significant attention by the planning community is the Stochastic Shortest Path Problem model (SSP) (Bertsekas and Tsitsiklis [1991]), where the objective is to minimize the expected cost of reaching a goal state from a start state. The SSP model is more general than other MDP classes (finite horizon MDPs and infinite-horizon discounted-reward MDPs) (Bertsekas and Tsitsiklis [1995]), and can be used for decision-making in the presence of multiple goals.

Solving large MDPs and SSPs optimally is a computationally intensive task. Although they can be solved in polynomial time in the number of states, most interesting problems have a state-space whose size is exponential in the number of the variables that describe the problem (Littman [1997]). This has led to the development of a range of model reduction techniques (Dean *et al.* [1997]) as well as heuristic search algorithms (e.g. LAO* (Hansen and Zilberstein [2001]) and LRTDP (Bonet and Geffner [2003b])) that attempt to focus the computation on states that are relevant to an optimal policy. However, even restricting the attention to states reachable by optimal policies could lead to prohibitive computational costs.

Recent algorithms try to cope with this challenge by reducing the size of the reachable state-space. Examples of this are determinization-based replanning methods, such as FF-Replan (Yoon *et al.* [2007]), RFF (Teichteil-Königsbuch *et al.* [2010]), and model reduction approaches such as SSiPP (Trevizan and Veloso [2012b]) and the $\mathcal{M}_l^k$ reduced model framework (Pineda and Zilberstein [2014]). However, these methods still have some shortcomings: some are restricted to particular problem representations (e.g., FF-Replan and RFF use PPDDL (Younes and Littman [2004])), others require some form of pre-processing ($\mathcal{M}_l^k$ reduced model framework), or only result in moderate time reductions with respect to other planners (SSiPP). Moreover, planning in these approaches is typically constrained to states that are within some distance measure of the start state (e.g., number of actions, or probability); this makes it difficult to propagate information from states that are closer to the goal than to the start state, and thus misses opportunity for improved planning.

In this work we introduce a new algorithm for action selection in MDPs, FLARES (Fast Labeling from Residuals using Samples), that can find high-performing policies orders of magnitude faster than optimal algorithms. Unlike previous methods, FLARES doesn't require any specific model representations, and can efficiently propagate information from states closer to the goal. Moreover, with small modifications, it also can be used to find optimal policies. We show that FLARES is guaranteed to complete in a finite amount of time, and experimental results on several well-known benchmark domains show that it outperforms many state-of-the-art planning algorithms.

Related Work

Determinization-based approaches saw a surge in popularity after the success of FF-Replan on the IPPC'04 probabilistic competition (Younes *et al.* [2005]). FF-Replan works by creating a deterministic version of an MDP, solving this deterministic problem quickly using the FF planner, and then re-plan if a state outside the current plan is reached during execution. This algorithm is extremely fast but performance may be poor for certain classes of probabilistic planning problems (Little and Thiebaux [2007]).

More recent extensions of this idea offer performance improvements. For instance, RFF (Teichteil-Königsbuch *et* *al.* [2010]), the winner of the IPPC'08 planning competition (Bryce and Buffet [2008]), works by creating a high-probability envelope of states and finding a plan for each of these using determinization and FF. Another notable extension is FF-Hindsight (Yoon *et al.* [2008]), which works by sampling different deterministic realizations of the transition function, solve each of these using FF, and then aggregating the result. These methods work well in practice, but, unlike the method presented in this work, they are constrained to problems described in PPDDL format.

More recent methods have explored other forms of state-space reduction besides determinization. For instance, SSiPP (Trevizan and Veloso [2012b]) is a method that creates reduced problems containing states reachable with at most t number of actions, where t is an input parameter. The reduced problems can be solved quickly using optimal MDP solvers, providing a short-sighted method for action selection.

A more recent variant reduces the model by pruning states with low probability of being reached (Trevizan and Veloso [2012a]). This latter variant also has some similarities with the HDP(i,j) algorithm (Bonet and Geffner [2003a]). HDP incorporates Tarjan's connected component algorithm into a heuristic search probabilistic algorithm, by labeling states in the same strongly connected component as solved once some error criterion is met. HDP(i,j) is a variant that only considers states up to some *plausibility*[1] i away from the start state; the parameter j represents the plausibility value used for re-planning. A key difference between SSiPP/HDP and the approach presented here is that planning in these methods is constrained to states that are "close" to the start state, and can thus require large horizons to propagate information from states closer to the goal; on the other hand, the approach presented here does not restrict the search only to states close to the start.

Finally, another form of reduction is the $\mathcal{M}_l^k$ reduction (Pineda and Zilberstein [2014]), which is a generalization of determinization. In a $\mathcal{M}_l^k$ reduced model some of the outcomes of each action schema are labeled as *exceptions*, and the transition function is modified so that exceptions are ignored after k of them have occurred; the parameter l represents the maximum number of the non-exception outcomes for action that are allowed. This reduction is more robust than simpler determinization, but requires some preprocessing to find the best reduction. Finding good reductions is an open problem.

Background

In this work we consider a special type of MDP called a Stochastic Shortest Path problem (SSP) (Bertsekas and Tsitsiklis [1991]). An SSP is a tuple $\langle S, A, T, C, s_0, s_g \rangle$, where S is a finite set of states, A is a finite set of actions, $T(s'|s,a) \in [0,1]$ represents the probability of reaching state s' when action a is taken in state s, $C(s,a) \in (0,\infty)$ is the cost of applying action a in state s, s_0 is an initial state and s_g is a goal state satisfying $\forall a \in A,\ T(s_g|s_g,a) = 1 \wedge C(s_g,a) = 0$.

[1]Plausibilities are related to the likelihood of reaching a state from another one. A trajectory with plausibility 0 corresponds to a trajectory with the highest probability, and larger values correspond to trajectories with lower probability.

A solution to an SSP is a *policy*, a mapping $\pi : S \rightarrow A$, indicating that action $\pi(s)$ should be taken at state s. A policy π induces a value function $V^\pi : S \rightarrow \mathbb{R}$ that represents the expected cumulative cost of reaching s_g by following policy π from state s. An optimal policy π^* is one that minimizes this expected cumulative cost; similarly, we use the notation V^* to refer to the optimal value function.

We restrict our attention to problems in which a policy exists such that the goal is reachable from all states with probability 1. Under this assumption, an SSP is guaranteed to have an optimal solution, and the optimal value function is unique. This optimal value function can be found as the fixed point of the so-called Bellman update operator (Equation 1).

$$\text{BU}(s) := \min_{a \in A} \left\{ C(s,a) + \sum_{s' \in S} T(s'|s,a)V(s') \right\} \quad (1)$$

A *greedy policy* for value function V is the one that chooses actions according to Equation 2. Importantly, a greedy policy over V^* is guaranteed to be optimal.

$$\pi(s) = \arg\min_{a \in A} \left\{ C(s,a) + \sum_{s' \in S} T(s'|s,a)V(s') \right\} \quad (2)$$

Finally, the following additional definitions will be useful for the rest of the paper.

- A *trial* of policy π is the process of sampling a trajectory $(s_0, s_1, \ldots, s_N)$ s.t. $P(s_{i+1}|s_i, \pi(s_i)) > 0$ and $s_N = s_g$.
- The *residual* error for state s under value function V is defined as $R(s) = |\text{BU}(s) - V(s)|$.

The FLARES Algorithm

The fastest optimal solvers for SSPs are based on heuristic search (LAO* (Hansen and Zilberstein [2001]), LRTDP (Bonet and Geffner [2003b])). These algorithm are characterized by the use of an initial estimate for the optimal value function (referred to as a *heuristic* and denoted as h) to guide the search to the more relevant parts of the problem. Typically the heuristic is required to be *admissible*; i.e., a lower bound on the optimal value function. Moreover, often the heuristic is also required to be *monotone*, in which case it must satisfy:

$$h(s) \leqslant \min_{a \in A} \left\{ C(a,s) + \min_{s' \in S} T(s'|s,a)h(s') \right\} \quad (3)$$

Although heuristic search can result in significant computational savings over Value Iteration and Policy Iteration, their efficiency is highly correlated with the size of the resulting optimal policy. Concretely, in order to confirm that a policy is optimal, a solver needs to ensure that there is no better action for *any* of the states that can be reached by this policy. Typically, this involves performing one or more Bellman backups on all reachable states of the current policy, until some convergence criterion is met.

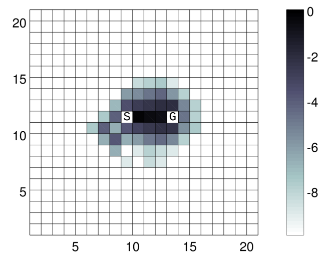

Figure 1: Example of a problem with large optimal policy but small high-probability envelope (scale shows log-probability). S: start state. G: goal state.

However, a common situation is having an optimal policy in which many of the covered states can only be reached with low probability. For instance, consider a grid like the one shown in Figure 1. Suppose every time the agent tries moves in one direction, it succeeds with probability 0.7, or moves in each of the other directions with probability 0.1; the goal is to move from position (9,11) to position (13,11). Due to the nature of the transition function, the optimal policy for this problem covers the entire state grid. Yet, as the color gradient shows, the (log)probability of visiting a state under the optimal policy quickly degrades with distance to the goal, resulting in a very small "envelope" of high-probability states, close to the most likely path between the start and the goal. This raises the question of how to better exploit this property to come up with faster approximate algorithms to solve SSPs.

One option is to create a depth-limited SSP (either by number of actions, or by highest probability paths) rooted at the current state and solve it to find a "short-sighted" action. However, this could make it hard to propagate information from states close the goal to the initial state, and can therefore require a large horizon to achieve good cost solutions—which can subsequently result in sub-problems covering a large number of states. On the other hand, an alternative is to use a trial-based search algorithm that samples high-probability paths, like RTDP, and then solve a short-sighted SSP rooted at each of the states visited during sampling. However, as it turns out, this approach doesn't result in major computational savings over a standard version of RTDP, and can even result in slower convergence[2]; furthermore, it lacks a crisp termination criterion relating the short-sighted approach with the solution of the complete problem.

Nevertheless, we can combine these two ideas to create an algorithm that can very quickly find good plans with near-optimal performance. The resulting algorithm, which we call FLARES, is an online approximate algorithm for action selection. FLARES works by performing a number of trials from the start to the goal, while trying to label states as solved according to a short-sighted labeling criterion. The key difference between FLARES and the other approaches mentioned above, is that FLARES can propagate information from the goal to the start state while simultaneously pruning the state-space, and do so without requiring a large search horizon. Intuitively, FLARES works by attempting to construct narrow corridors of states with low residual error from the start to the goal.

Readers familiar with heuristic search methods for solving MDPs will notice similarities between FLARES and the well-known LRTDP algorithm (Bonet and Geffner [2003b]). Indeed, FLARES is based on LRTDP with a particular change in the way states are labeled. For reference, LRTDP is an extension of RTDP that includes a procedure to label states as solved (CHECKSOLVED). In RTDP, trials are run repeatedly and Bellman backups are done on each of the states visited during a trial. This procedure can be stopped once the current greedy policy covers only states s s.t. $R(s) < \epsilon$, for some given tolerance ϵ. In LRTDP, this is improved by adding the states seen during a trial to a stack and calling CHECKSOLVED on each of them, in post-order traversal.

The CHECKSOLVED labeling procedure has the following property: it only labels a state s as solved if all states s' that can be reached from s following a greedy policy satisfy $R(s') < \epsilon$. The main advantage is that, once a state is labeled as solved, the stored values and actions can be used if this state is found during future trials or calls to CHECKSOLVED.

Typically, labeling states results in large savings in computation time with respects to methods than don't incorporate this approach. However, despite its advantages, CHECKSOLVED is subject to the same problems as other optimal solvers; that is, it can potentially explore large low-probability sections of the state space, since it must check *all* reachable states before labeling.

To address this problem, we introduce the following depth-limited labeling property as a way to accelerate heuristic search methods: *a state s is considered* ***depth-t-solved*** *only if all states s' that can be reached with t or less actions following the greedy policy satisfy $R(s') < \epsilon$.*

Algorithm 1 shows a procedure that implements this idea. A call to DLCHECKSOLVED from state s and horizon t works by visiting all states that can be reached from s by following at most $2t$ actions under the current greedy policy. If all states s' visited during this search satisfy $R(s') < \epsilon$, the method then proceeds to label as depth-t-solved only those states found up to horizon t. Note that doing the search up to horizon $2t$ allows DLCHECKSOLVED to label several states during a single call, instead of only the root state if the residuals were only checked up to depth t.

The FLARES algorithm, wneverhich incorporates DLCHECKSOLVED into a trial based action selection mechanism, is given by Algorithm 2. Propositions 1 and 2 show the conditions under which FLARES, and more specifically DLCHECKSOLVED, maintains both of the labeling properties described above. To simplify the presentation, we overload the notation so that the statement $s \in$ *closed* is equivalent to $\exists d$ *s.t.* $\langle s, d\rangle \in$ *closed*.

Proposition 1. DLCHECKSOLVED *labels a state s with s.*SOLV $=$ *true only if all states s' that can be reached from s following the greedy policy satisfy $R(s') < \epsilon$.*

Proof. Let x be the first state incorrectly labeled with

[2]We experimented with such an approach without much success. Results are not reported here due to space considerations.

Algorithm 1: A depth limited procedure to label states.

```
DLCheckSolved
    input : $s, t$
1   $solved = true$
2   $open$ = EMPTYSTACK
3   $closed$ = EMPTYSTACK
4   $all = true$
5   if $\neg(s.\text{SOLV} \vee s.\text{D-SOLV})$ then
6       $open$.PUSH($\langle s, 0\rangle$)
7   while $open \neq$ EMPTYSTACK do
8       $\langle s, d\rangle = open$.POP()
9       if $d > 2t$ then
10          $all = false$
11          continue
12      $closed$.PUSH($\langle s, d\rangle$)
13      if $s$.RESIDUAL() $> \epsilon$ then
14          $solved = false$
15      $a$ = GREEDYACTION($s$)
16      for $s' \in \{s' \in S | P(s'|s, a) > 0\}$ do
17          if $\neg(s.\text{SOLV} \vee s.\text{D-SOLV}) \wedge s' \notin closed$ then
18              $open$.PUSH($\langle s', d + 1\rangle$)
19          else if $s.\text{D-SOLV} \wedge \neg s.\text{SOLV}$ then
20              $all = false$
21  if $solved$ then
22      for $\langle s', d\rangle \in closed$ do
23          if $all$ then
24              $s'$.SOLV = $true$
25              $s'$.D-SOLV = $true$
26          else if $d \leqslant t$ then
27              $s'$.D-SOLV = $true$
28  else
29      while $closed \neq$ EMPTYSTACK do
30          $\langle s', d\rangle = closed$.POP()
31          BELLMANUPDATE($s$)
32  return $solved$
```

Algorithm 2: The FLARES algorithm.

```
FLARES
    input : $s_0, t$
    output: action to execute
1   while $\neg s_0.\text{SOLVED} \vee s_0.\text{D-SOLV}$ do
2       $s = s_0$
3       $visited$ = EMPTYSTACK
4       while $\neg(s.\text{SOLVED} \vee s.\text{D-SOLV})$ do
5           $visited$.PUSH($s$)
6           if GOAL($s$) then break
7           BELLMANUPDATE($s$)
8           $a$ = GREEDYACTION($s$)
9           $s$ = RANDOMSUCCESSOR($s, a$)
10      while $visited \neq$ EMPTYSTACK do
11          $s = visited$.POP()
12          if $\neg$DLCHECKSOLVED($s, t$) then
13              break
14      return GREEDYACTION($s$)
```

x.SOLV $=$ *true* by DLCHECKSOLVED. Then, during the call DLCHECKSOLVED(x), at line 23 we have $all = true$ while there exists a state y, reachable from x following the greedy policy, s.t. $R(y) > \epsilon$ and $y \notin closed$. The `if` block in lines 9-11 guarantees that this state was never added to *open*, otherwise $y \notin closed \implies \neg all$. Let u be the first ancestor of y in the greedy policy graph rooted at x that wasn't added to *open*. Thus, either u.SOLV or u.D-SOLV due to the `if` statement in line 17. Since x is the first state to be labeled SOLV incorrectly, the only possibility consistent with $R(y) > \epsilon$ is that $\neg u.\text{SOLV} \wedge u.\text{D-SOLV}$. However, since this implies $\neg all$ for u, the `else if` statement in lines 26 implies that u.SOLV, which is a contradiction. □

Proposition 2. *As long as no call to* BELLMANUPDATE(s') *with* $R(s') < \epsilon$ *results in* $R(s') \geqslant \epsilon$*, then* DLCHECKSOLVED *labels a state* s *with* s.D-SOLV *only if* s *is depth-t-solved.*

Proof. Following Proposition 1, we ignore the label set in line 25 because this label is always correct. We use induction to prove that the label set in line 27 is correct. Suppose that when a call to DLCHECKSOLVED all labels have previously set correctly. Then, if a tuple $\langle x, d_x\rangle$ is taken from from the stack in line 8, all the descendants y of x that are reachable within $2t - d_x$ actions and satisfy $R(y) \geqslant \epsilon$ will also be added to *closed*. Thus, any state s labeled in line 27 with depth $d \leqslant t$ are in fact depth-t-solved. Note that the assumption on the Bellman updates guarantees that once a state is correctly labeled, further backups of any unlabeled descendant, s', will still keep the label correct, because $R(s')$ will never increase above ϵ. To complete the proof, it only remains to prove that the base case (when no labels have been previously set) is also correct. It is easy to see that this is true, because in this case all states up to depth $2t$ will be added to the stack, and, if their residuals are below ϵ, only states up to depth t are labeled. □

The assumption at the start of proposition 2 requires further explanation. It is possible for a state s to be labeled with s.D-SOLV $=$ *true* while some of its low residual descendants within depth t are not; this is because DLCHECKSOLVED only labels states up to depth t after checking the residual on all states up to depth $2t$. Therefore, since FLARES can still perform Bellman backups of states that have not yet been labeled, and because residuals are not guaranteed to be monotonically decreasing for all states, it is possible for the residual of an unlabeled state to increment above ϵ during a trial, and therefore break the depth-limited labeling guarantee of its ancestors.

Unfortunately, there is no simple way to get around this issue without resorting to some cumbersome backtracking, and no way to predict whether such an increment will happen on a given run of FLARES. However, our experiments suggest that this event is uncommon in practice (it was never encountered during our experiments). Moreover, we can ob-

Algorithm 3: An optimal version of FLARES.

```
OPT-FLARES
    input: $s_0$
1   $t = t_0$
2   while $\neg s_0$.SOLVED do
3       for $s \in S$ do
4           $s$.D-SOLV = false
5       FLARES$(s_0, t)$
6       $t = \rho(V, t)$
```

tain a revised labeling error guarantee during planning, by keeping track of all states for which a Bellman backup increased the residual above ϵ, and use the maximum of those residuals as the revised error.

Next we prove that FLARES is guaranteed to terminate in a finite number of iterations.

Theorem 1. *If the heuristic is admissible and monotone, FLARES terminates after at most $\epsilon^{-1} \sum_{s \in S} V^*(s) - V(s)$ trials.*

Proof. It is a well-known fact that Bellman backups preserve admissibility and monotonicity of the value function, regardless of the order in which states are backed up. Thus, if the initial heuristic is admissible and monotone, all state values computed $V(s)$ by FLARES are bounded above by $V^*(s)$. At the same time, each trial of FLARES either labels all visited states as solved or increases the value of at least one them by more than ϵ. Therefore, in the worst case, each trial will increase the value of a single state by ϵ until all values reach $V^*(s)$, which results in the proposed bound. ☐

Even though this is the same bound as LRTDP's, in practice convergence will happen much faster because the final values computed by FLARES are only lower bounds of the optimal values. Unfortunately, like other methods that choose actions based on lower bounds, it is possible to construct examples where the final policy returned by FLARES can be arbitrarily bad. On the other hand, it is easy to see that FLARES is asymptotically optimal as $t \to \infty$ because it simply turns into the LRTDP algorithm.

In fact, as the following theorem shows, there exists a finite value of t for which FLARES returns the optimal policy. We can use this fact and the labels SOLV computed by FLARES to produce an following optimal SSP solver, which is shown in Algorithm 3. Here ρ represents a function that takes the current value function V and the current t and produces a new value for t. Typically, $\rho(V, t) = t + 1$, but it's possible that there are more clever ways to increment t based on an analysis of the current values and previous calls to FLARES. Theorem 2 shows the conditions under which OPT-FLARES is guaranteed to be optimal.

Theorem 2. *If the initial heuristic is admissible and monotone, and ρ satisfies $\forall V, t,\ \rho(V, t) > t$, with $t_0 \geqslant 0$, then OPT-FLARES computes an optimal policy.*

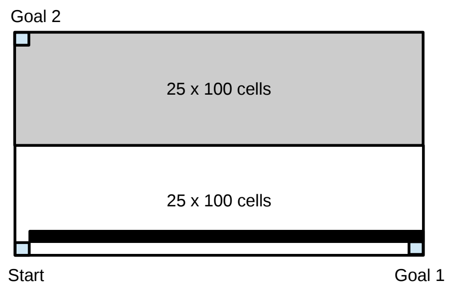

Figure 2: Grid world used for illustrating advantages of FLARES.

Proof. Note that the value function will remain admissible and monotone after every call to FLARES. Therefore, every call to FLARES in line 5 is done using an admissible heuristic and a larger value of t. Since the state space is finite, there exists a finite value of t for which all calls to DLCHECKSOLVED covers the same set of states as CHECKSOLVED (a trivial solution is $t \geqslant |S|$). Because all D-SOLV labels are cleared in the loop in lines 3-4, FLARES at that point becomes equivalent to LRTDP and is thus optimal. ☐

Experiments

In this section we compare FLARES to three other algorithms: LRTDP, HDP(i,j) and SSiPP. We start by illustrating some of the advantages of FLARES over optimal algorithms and other short-sighted approaches, by means of a simple grid world problem that is easy to analyze. We then show results on the racetrack domain (Barto *et al.* [1995]) and the sailing domain (Kocsis and Szepesvári [2006]), two common benchmarks for probabilistic planning algorithms. All algorithms were implemented by us and tested on a Intel Xeon 3.10 GHz computer with 16GB of RAM. For all experiments we used a value of $\epsilon = 10^{-3}$.

A Simple Gridworld Problem

Consider the grid world shown in Figure 2. In this problem the agent can move in any of the four grid directions (up, down, right, left). Every time it moves, there is a 0.7 probability of succeeding or a 0.3 probability of moving to one of the other 3 directions (chosen uniformly at random). The cost of moving is 1, except for some "dangerous" cells (highlighted in gray) where the cost of moving is 20; additionally, some cells have obstacles that cannot be crossed (shown in black color near the bottom edge). The grid has width 100 and height 51, for a total of 5100 states. The start state is at the bottom left corner, and there are two goals, one at the top-left corner and one at the bottom-right. The optimal policy attempts to reach the state at the goal state at the right, so that the agent avoids the dangerous states at the top.

Table 1 shows the expected cost (mean and standard error) and average planning time for each of the algorithms; the cost shown for LRTDP is the optimal cost calculated by the algorithm. The heuristic used is the Manhattan distance to the closest goal. All results are averaged over 100 runs

algorithm	cost	time
LRTDP	135	34.02
FLARES(0)	134.07 ± 0.84	0.586
FLARES(1)	135.63 ± 0.99	0.589
HDP(0,0)	208.9 ± 10.92	0.195
HDP(4,0)	135.22 ± 1.09	0.610
HDP(4,4)	133.91 ± 0.83	0.593
SSiPP(16)	441.12 ± 4.87	10.87
SSiPP(32)	400.87 ± 1.85	51.49
SSiPP(64)	136.49 ± 0.76	9.49

Table 1: Results on the gridworld shown in Figure 2.

of complete planning and execution simulations. The estimated values are reused within the same run if re-planning is needed, but they are reset to the heuristic after each run. The average time includes time spent on re-planning.

Notably, FLARES with $t = 0$ already returns a policy that is essentially optimal, while being on average two orders of magnitude faster than the optimal algorithm, LRTDP. HDP(i,j) is also quite fast on this problem, but it required some parameter tuning to find the best values for i; results comparable to FLARES(0) are obtained using HDP(4,4).

On the other hand, SSiPP is slower than the other approximate methods in this problem, and substantial parameter tuning was also required; the table shows only results obtained with $t = 16$, $t = 32$ and $t = 64$. Note the large horizons required to find a good policy; a near-optimal policy was obtained only with $t = 64$, and required close to 18 times more time than FLARES with $t = 0$. The larger time required for $t = 32$ (compared to the time using $t = 64$) was due to many more re-planning episodes needed during execution.

First, this simple problem highlights several qualities of FLARES. Although an optimal policy for this problem must cover the entire state space, every state outside the narrow corridor at the bottom is only reached with low probability. This is an example of a problem where an optimal solver would be unnecessarily slow. On the other hand, FLARES only needs to realize that the policy going up leads to a high cost, which happens during the first few trials. Then, once the algorithm switches to the policy that moves to the right, it quickly stops when all states in the corridor reach a low residual error.

Second, since the algorithm is only short-sighted during labeling, but is otherwise not restricted during the trials phase, it is able to quickly account for the dangerous states that are far away from the start state. This is the reason why a low t is enough to generate very good policies. On the other hand, limiting the search to states close to the start, requires much larger horizons to achieve comparable results.

Racetrack domain

We experimented with the racetrack domain described by Barto *et al.* [1995]. We modify the problem so that, in addition to a 0.2 probability of slipping, there is a 0.1 probability of randomly changing the intended acceleration by one unit; similar modifications have been used before to increase the density of the transition function and the difficulty of the problem (McMahan *et al.* [2005]). We used the h_{min} heuristic for this problem, which was pre-computed for all states before the runs.

Figure 3 shows boxplots of the costs obtained from 100 simulations on four instances of the racetrack domain of varying sizes and shapes. In general, LRTDP, FLARES(0) and various instances of HDP obtained similar performance. In fact, we ran t-tests comparing the results of each algorithm with those of LRTDP, and FLARES(1) was never significantly different from LRTP (p-values higher than 0.27), except for the ring-5 track where the expected cost obtained with FLARES(1) was 10% higher; similar results were obtained with most versions of HDP. On the other hand, SSiPP had expected costs considerably higher than LRTDP.

Table 2 show average planning times of all the algorithms in these problems. In all cases, FLARES(0) and FLARES(1) were two orders of magnitude faster than the other algorithms. HDP(i,j) was in general about twice faster than LRTDP, but still two orders of magnitude slower than FLARES. The reason for this large computation time is unclear, but it appears there is a lot of thrashing between different policies due to the short-sightedness of the algorithm. On the other hand, SSiPP required high horizons (results shown up to $t = 8$) which resulted in considerably high computation times, even though the expected costs were still significantly higher than optimal.

	square-4	square-5	ring-5	ring-6
LRTDP	49.89	262.84	11.64	65.02
FLARES(0)	**0.276**	**1.637**	**0.052**	**0.341**
FLARES(1)	**0.260**	**1.645**	**0.058**	**0.362**
HDP(0,0)	23.71	151.15	5.50	37.15
HDP(0,1)	26.84	145.12	5.84	37.08
HDP(1,0)	27.50	145.83	6.09	36.09
HDP(1,1)	28.41	142.02	6.10	38.27
SSiPP(4)	16.24	76.11	4.78	25.11
SSiPP(8)	48.61	178.98	20.13	89.72

Table 2: Average planning time (seconds) on several racetrack domain problems. Fastest times shown in bold.

Sailing Domain

We next present results on a four instances of the sailing domain, as described by Kocsis and Szepesvári [2006]. The instances vary in terms of grid size (all grids are squares) and where the goal is located in the grid (opposite corner or middle of the grid). Table 3 shows average costs and times over 100 simulations.

In this domain, FLARES(1) and HDP have similar results in terms of average cost; Figure 4 shows boxplots of the costs obtained. Out of all instances of FLARES and HDP, FLARES(0) was the worse in terms of average cost, with average costs more than 40% higher than the other algorithms; however, its running time was 2-4 times faster (see Table 3). FLARES(1) had slightly worse average performance than HDP, but with times that are between 30% to

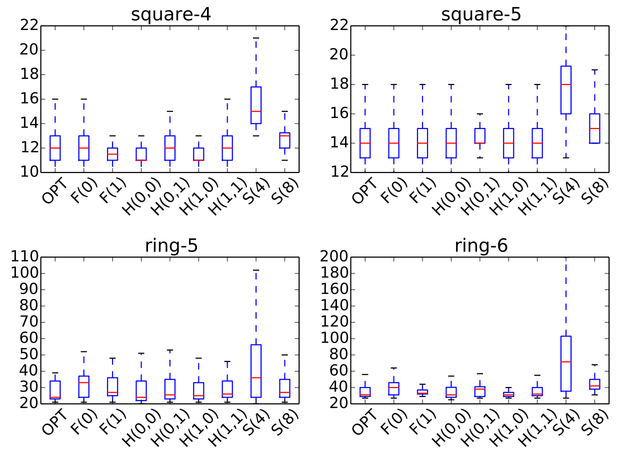

Figure 3: Boxplots of costs obtained by the algorithms on four racetrack domain instances (100 simulations). Labels: OPT (optimal, LRTDP), F(i) - FLARES with $t = i$, H(i,j) - HDP(i,j), S(i) - SSiPP with $t = i$.

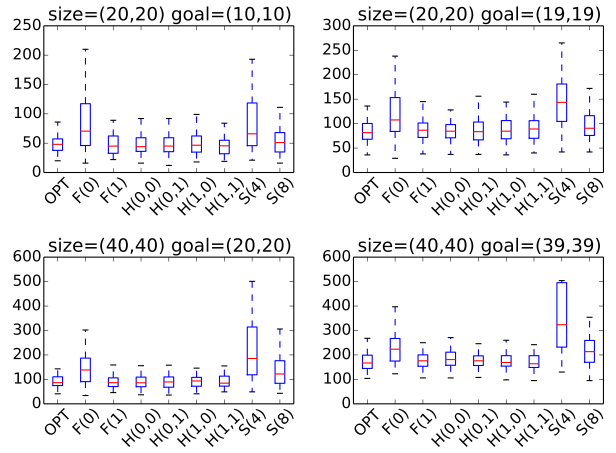

Figure 4: Boxplots of costs obtained by the algorithms on four sailing domain instances (100 simulations). Labels: OPT (optimal, LRTDP), F(i) - FLARES with $t = i$, H(i,j) - HDP(i,j), S(i) - SSiPP with $t = i$.

200% faster. Moreover, running t-tests to compare the costs obtained with those of LRTDP, results in no significant difference with respect to FLARES(1) (the lowest p-value was 0.15, obtained on the largest problem). Finally, SSiPP required large horizons ($t = 8$) to get reasonable results and its running time was more than twice higher than the running times of FLARES and HDP; moreover, the average costs obtained were still substantially worse than optimal.

	s=20 g=corner	s=40 g=corner	s=20 g=middle	s=40 g=middle
LRTDP	1.81	14.65	1.37	12.09
FLARES(0)	**0.33**	**3.15**	**0.138**	**1.142**
FLARES(1)	1.01	7.79	0.417	3.065
FLARES(2)	1.47	9.51	0.731	4.094
HDP(0,0)	1.33	11.93	0.854	7.034
HDP(0,1)	1.33	12.04	0.854	7.245
HDP(1,0)	1.35	11.85	0.853	7.133
HDP(1,1)	1.33	11.88	0.853	7.159
SSiPP(4)	3.05	8.83	1.60	5.69
SSiPP(8)	7.14	52.47	3.93	19.77

Table 3: Average planning time (seconds) on several sailing domain problems. Problems are labeled as s=(size) and g=(goal location). Fastest times shown in bold.

Conclusions

In this work we present a new approach to short-sightedness for MDPs that applies the reduction only for labeling states as solved. In contrast to previous approaches, which focus on states that are close to the initial state, applying the reduction only during labeling allows for larger sections of the state space to be explored, and quickly propagate information of states close the goal. Moreover, this results in needing lower horizon values for labeling, which significantly accelerate running times.

Based on this idea, we introduced the FLARES algorithm, a modified version of LRTDP that incorporates a bounded labeling procedure to produce a very fast approximate action selection mechanism. We prove that FLARES is guaranteed to terminate with a policy in a finite amount of time, and that it can be easily extended to produce an optimal algorithm. Experimental results in three different planning domains show that FLARES can produce near-optimal policy orders of magnitude faster than state-of-the-art algorithms.

Although we implemented our reduced labeling approach as an extension of LRTDP, we note that the key ideas can be used in conjunction with other search-based methods. Reduced labeling can be combined with other action or outcome selection mechanism for planning, using a framework like THTS (Keller and Helmert [2013]). We are working on new algorithms based on this idea.

Finally, in this work we focused on a reachability reduction based on number of actions, but a version that uses trajectory probabilities, similar to (Trevizan and Veloso [2012a]), is a straightforward extension that we plan to explore in the future.

References

Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995.

Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

Dimitri P. Bertsekas and John N. Tsitsiklis. Neuro-dynamic programming: An overview. In *Proceedings of the 34th*

IEEE Conference on Decision and Control, pages 560–564, 1995.

Blai Bonet and Hector Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 1233–1238, 2003.

Blai Bonet and Hector Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, pages 12–21, 2003.

Daniel Bryce and Olivier Buffet. 6th international planning competition: Uncertainty part. In *Proceedings of the 6th International Planning Competition*, 2008.

Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 124–131, 1997.

Eric A. Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, 2013.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, pages 282–293, 2006.

Iain Little and Sylvie Thiebaux. Probabilistic planning vs. replanning. In *Proceedings of the ICAPS'07 Workshop on the International Planning Competition: Past, Present and Future*, 2007.

Michael L. Littman. Probabilistic propositional planning: Representations and complexity. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 748–754, 1997.

H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 569–576. ACM, 2005.

Luis Pineda and Shlomo Zilberstein. Planning under uncertainty using reduced models: Revisiting determinization. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling*, pages 217–225, 2014.

Florent Teichteil-Königsbuch, Ugur Kuter, and Guillaume Infantes. Incremental plan aggregation for generating policies in MDPs. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pages 1231–1238, 2010.

Felipe Trevizan and Manuela Veloso. Trajectory-based short-sighted probabilistic planning. In *Advances in Neural Information Processing Systems*, pages 3248–3256, 2012.

Felipe W. Trevizan and Manuela M. Veloso. Short-sighted stochastic shortest path problems. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, pages 288–296, 2012.

Sung Wook Yoon, Alan Fern, and Robert Givan. FF-Replan: A baseline for probabilistic planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, pages 352–359, 2007.

Sungwook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. Probabilistic planning via determinization in hindsight. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, pages 1010–1016, 2008.

Håkan L. S. Younes and Michael L. Littman. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. *Technical Report CMU-CS-04-162*, 2004.

Håkan L. S. Younes, Michael L. Littman, David Weissman, and John Asmuth. The first probabilistic track of the International Planning Competition. *Journal of Artificial Intelligence Research*, 24(1):851–887, 2005.