Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Fast SSP Solvers Using Short-Sighted Labeling

Luis Pineda, Kyle Hollins Wray, Shlomo Zilberstein
College of Information and Computer Sciences
University of Massachusetts Amherst
Amherst, MA 01003, USA
{lpineda,wray,shlomo}@cs.umass.edu

Abstract

State-of-the-art methods for solving SSPs often work by limiting planning to restricted regions of the state space. The resulting problems can then be solved quickly, and the process is repeated during execution when states outside the restricted region are encountered. Typically, these approaches focus on states that are within some distance measure of the start state (e.g., number of actions or probability of being reached). However, these short-sighted approaches make it difficult to propagate information from states that are closer to a goal than to the start state, thus missing opportunities to improve planning. We present an alternative approach in which short-sightedness is used only to determine 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 performs consistently well on a wide range of benchmark problems.

Introduction

Markov decision processes (MDP) offer a highly-expressive model for probabilistic sequential decision making. One class of MDP problems that has received significant attention by the planning community is the Stochastic Shortest Path problem (SSP), where the objective is to minimize the expected cost of reaching a goal state from the start state (Bertsekas and Tsitsiklis 1991). In fact, it has been shown that the SSP model is more general than other MDP classes (finite-horizon and infinite-horizon discounted-reward MDPs) (Bertsekas and Tsitsiklis 1995).

Solving large MDPs and SSPs optimally is a computationally intensive task. Although they can be solved in polynomial time in the number of states, many problems of interest have a state-space whose size is exponential in the number of variables describing the problem (Littman 1997). This has led to the development of a range of model reduction techniques (Dean, Givan, and Leach 1997) as well as heuristic search algorithms, such as LAO* (Hansen and Zilberstein 2001) and LRTDP (Bonet and Geffner 2003b), that attempt to focus on states that are relevant to an optimal policy. Unfortunately, even considering just states reachable by an optimal policy could be computationally prohibitive.

Recent algorithms try to cope with this challenge by reducing the size of the reachable state-space. Examples include determinization-based methods such as FF-Replan (Yoon, Fern, and Givan 2007) and RFF (Teichteil-Königsbuch, Kuter, and Infantes 2010), and model reduction approaches such as SSiPP (Trevizan and Veloso 2014) and $\mathcal{M}_l^k$-reductions (Pineda and Zilberstein 2014). However, these methods have some shortcomings: some are restricted to particular problem representations (e.g., FF-Replan and RFF use PPDDL (Younes and Littman 2004)), while others require some form of pre-processing ($\mathcal{M}_l^k$-reductions) or substantial parameter tuning (SSiPP). Moreover, planning in these methods is typically constrained to states that are within some distance measure of the start state (e.g., number of actions, possible outcomes, or probability of being reached), making it hard to propagate information from states that are closer to a goal than to the start state, and potentially missing opportunities to improve planning.

Building on these earlier efforts, 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. FLARES does not require any specific model representations, and can efficiently propagate information from states closer to the goal. Moreover, it can be tuned to find approximate or optimal policies, as desired. We show that FLARES is guaranteed to terminate in a finite amount of time, and can outperform other short-sighted state-of-the-art planning algorithms, performing consistently well across a variety of benchmark domains.

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).

A variety of extensions of this idea offer performance improvements. For instance, RFF (Teichteil-Königsbuch, Kuter, and Infantes 2010), the winner of the IPPC'08 plan-

ning 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 2014) is a method that creates reduced problems containing states reachable with at most t 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. Another variant of SSiPP reduces the model by pruning states with low probability of being reached (Trevizan and Veloso 2014).

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* i (a measure of likelihood of reachability) 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. In contract, 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, which generalizes determinization (Pineda and Zilberstein 2014). In an $\mathcal{M}_l^k$-reduction some of the outcomes of each action schema are labeled as *exceptions*, and the transition function is modified so that exceptions are ignored once k of them occur; the parameter l represents the maximum number of the non-exception outcomes allowed per action. While this approach is more robust than simpler determinization, it requires some preprocessing to find the best reduction and finding good reductions is an open problem.

Problem Definition

We consider a special class of MDPs called Stochastic Shortest Path (SSP) problems (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 $\forall a \in A,\ T(s_g|s_g,a) = 1 \wedge C(s_g,a) = 0$.

A solution to an SSP is a *policy*, a mapping $\pi : S \to A$, indicating that action $\pi(s)$ should be taken at state s. A policy π induces a value function $V^\pi : S \to \mathbb{R}$ 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.

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

The *residual* error for state s under value function V is defined as $R(s) = |\text{BU}(s) - V(s)|$.

Finally, we define a *trial* of policy π as 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 FLARES Algorithm

Heuristic search algorithms offer some of the best methods for solving SSPs optimally, particularly LAO* (Hansen and Zilberstein 2001) and LRTDP (Bonet and Geffner 2003b). These algorithm are characterized by the use of an initial estimate for the optimal value function (a *heuristic* denoted 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 required to be *monotone*, satisfying:

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

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.

However, it is common to have an optimal policy in which many of the covered states can only be reached with very low probability. Thus, their costs have minimal impact on the expected cost of the optimal policy. This raises the question of how to better exploit this property to design faster approximate algorithms for SSPs.

We present the FLARES algorithm that leverages this property by combining short-sightedness and trial-based search in a novel way. Concretely, 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 property of FLARES, which distinguishes it from other short-sighted approaches,

is that it 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 pushing to a stack the states seen during a trial, and then calling CHECKSOLVED on each as they are taken out of the stack.

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.

While such a labeling approach could result in large computational savings, clearly CHECKSOLVED suffers from the same problem that affects optimal solvers—it may have to explore large low-probability sections of the state space because 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 the procedure DLCHECKSOLVED that implements this idea: a call with state s and horizon t visits 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 incorporates DLCHECKSOLVED into a trial based action selection mechanism (shown in Algorithm 2). Propositions 1 and 2 show the conditions under which FLARES, and more specifically DLCHECKSOLVED, maintains the labeling properties described above.

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 Sketch. Proof by contradiction. If a state x is labeled x.SOLV $=$ *true* incorrectly, then two things happen: i) *all* $=$ *true* at line 23, ii) there exists a descendant y in the greedy graph s.t. $R(y) > \epsilon$ and $y \notin$ *closed*. However, this implies some ancestor $u \neq x$ of y in the graph satisfies $\neg u$.SOLV $\wedge$ u.D-SOLV (line 18), which implies *all* $=$ *false* (line 20). □

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 ¬(s.SOLV ∨ s.D-SOLV) then
6          open.PUSH(⟨s, 0⟩)
7      while open ≠ EMPTYSTACK do
8          ⟨s, d⟩ = open.POP()
9          if d > 2t then
10             all = false
11             continue
12         closed.PUSH(⟨s, d⟩)
13         if s.RESIDUAL() > ε then
14             solved = false
15         a = GREEDYACTION(s)
16         for s' ∈ {s' ∈ S|P(s'|s, a) > 0} do
17             if ¬(s'.SOLV ∨ s'.D-SOLV) ∧ s' ∉ closed then
18                 open.PUSH(⟨s', d + 1⟩)
19             else if s'.D-SOLV ∧ ¬s'.SOLV then
20                 all = false
21     if solved then
22         for ⟨s', d⟩ ∈ closed do
23             if all then
24                 s'.SOLV = true
25                 s'.D-SOLV = true
26             else if d ⩽ t then
27                 s'.D-SOLV = true
28     else
29         while closed ≠ EMPTYSTACK do
30             ⟨s', d⟩ = closed.POP()
31             BELLMANUPDATE(s)
32     return solved
```

Proposition 2. *If 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 Sketch. Proof by induction. For the induction step, note that calling DLCHECKSOLVED on state x with all previous labels being correct, results in new labels set correctly in line 27; this is because the unlabeled descendants of x reachable within $2t$ steps will still be added to *closed*, but only those reachable within t steps are labeled. The base case, when no states have been previously labeled, is trivial, because in this case *all* descendants up to depth $2t$ are added to *open* (line 18). □

The assumption of Proposition 2 requires some explanation. State s can be labeled with s.D-SOLV while some of its low residual descendants within depth t are not (DLCHECKSOLVED only labels states up to depth t after checking the residual on all states up to depth $2t$). Since FLARES can perform Bellman backups of unlabeled states,

Algorithm 2: The FLARES algorithm.

```
FLARES
    input : s_0, t
    output: action to execute
1   while ¬s_0.SOLVED ∨ s_0.D-SOLV do
2       s = s_0
3       visited = EMPTYSTACK
4       while ¬(s.SOLVED ∨ s.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 ≠ EMPTYSTACK do
11          s = visited.POP()
12          if ¬DLCHECKSOLVED(s, t) then
13              break
14      return GREEDYACTION(s)
```

and because residuals are not guaranteed to be monotonically decreasing, it is possible for the residual of an unlabeled state to increment above ϵ during a trial, breaking 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 observed). Moreover, we can obtain 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. *With admissible&monotone heuristic, FLARES terminates after at most* $1/\epsilon \sum_{s \in S}[V^*(s) - V(s)]$ *trials.*

Proof Sketch. The proof follows from a similar argument to the proof of LRTDP's termination. □

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 on 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. It is then easy to construct an optimal SSP solver using FLARES, by running FLARES with increasing values of t until s_0.SOLV = *true*, and clearing all D-SOLV labels before each run.

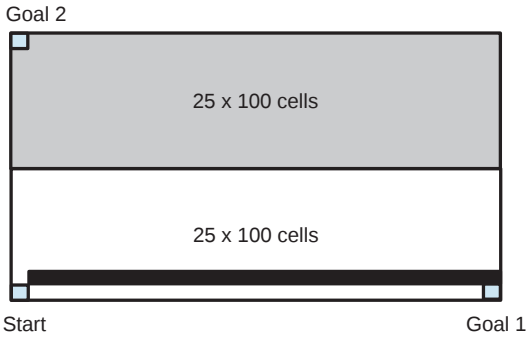

Figure 1: Grid world illustrating the advantages of FLARES.

Theorem 2. *With an admissible and monotone heuristic, there exists a finite* t *for which FLARES converges to the* ϵ*-optimal value function.*

Proof Sketch. Since the state space is finite, there exists a finite value of t for which all calls to DLCHECKSOLVED cover the same set of states as CHECKSOLVED (a trivial solution is $t \geqslant |S|$). Under these conditions, the algorithm becomes equivalent to LRTDP, and is thus optimal. □

Experiments

In this section we compare FLARES to an optimal algorithm, LRTDP, and two other short-sighted solvers, HDP(i,j) and SSiPP. We start by illustrating some of the advantages of FLARES over these approaches by applying them to a simple grid world problem that is easy to analyze. The remaining experiments, on the racetrack domain (Barto, Bradtke, and Singh 1995), the sailing domain (Kocsis and Szepesvári 2006), and the triangle-tireworld domain (Little and Thiebaux 2007), aim to show that FLARES can perform consistently well across a variety of planning domains, in terms of solution quality and computation time. Unless otherwise specified, we used the following settings for our experiments:

- All algorithms were implemented by us and tested on a Intel Xeon 3.10 GHz computer with 16GB of RAM.
- We used a value of $\epsilon = 10^{-3}$.
- We used the $h_{\min}$ heuristic, computed as needed using a labeled version of LRTA* (Bonet and Geffner 2003b).
- All results are averaged over 100 runs of complete planning and execution simulations.
- Estimated values are stored for re-planning within the same run, but they are reset after each run.
- Average times include the time spent on re-planning.

A simple grid world problem

Consider the grid world shown in Figure 1. The agent can move in any of the four grid directions (up, down, right, left). After moving, there is a 0.7 probability of succeeding or a 0.3 probability of moving in another direction (chosen uniformly at random). The cost of moving is 1, except for some

algorithm	cost	time
LRTDP	135	34.02
FLARES(0)	134.43 ± 0.88	1.28
HDP(3,0)	135.28 ± 0.82	0.62
SSiPP(64)	136.85 ± 0.96	10.53

Table 1: Results on the grid world shown in Figure 1.

"dangerous" cells (highlighted in gray) with cost 20; additionally, some cells have obstacles that cannot be crossed (shown in black). 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 goal state to the right, so that the agent avoids the dangerous states in the top part of the map.

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 estimated by the algorithm. Notably, FLARES with $t = 0$ already returns essentially the optimal policy, while being on average two orders of magnitude faster than LRTDP. Although HDP(i,j) is even faster on this problem, it required some parameter tuning to find appropriate values for i and j. The parameter settings shown are the lowest value of i for which results comparable to FLARES(0) are obtained.

On the other hand, SSiPP is slower than the other approximate methods, and substantial parameter tuning was also required. Table 1 shows only results obtained with $t = 64$, which is the first value of t (in powers of 2) that results in comparable expected costs to FLARES(0). Note the large horizons required to find a good policy, resulting in a very large running time, which is close to 8 times slower than FLARES(0).

This simple problem highlights several qualities of FLARES. First, 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 its trials are unrestricted, it can quickly account for the dangerous states far away from the start. This is the reason why $t = 0$ can already generate 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, Bradtke, and Singh 1995). We modify the problem so that, in addition to a 0.35 probability of slipping, there is a 0.20 probability of randomly changing the intended acceleration by one unit; similar modifications have been used before to increase the difficulty of the problem (McMahan, Likhachev, and Gordon 2005).

algorithm	square-4	ring-5
LRTDP	13.83	36.48
FLARES(0)	14.18 ± 0.40	43.56 ± 1.21
FLARES(1)	13.84 ± 0.33	37.81 ± 0.99
HDP(0)	13.85 ± 0.28	37.18 ± 1.05
HDP(0,0)	13.68 ± 0.30	36.86 ± 0.95
SSiPP(2)	15.30 ± 0.50	40.17 ± 1.03
SSiPP(4)	14.16 ± 0.36	37.15 ± 1.29

Table 2: Average cost on several racetrack domain problems.

algorithm	square-4	ring-5
LRTDP	69.16	16.96
FLARES(0)	0.43	2.38
FLARES(1)	1.37	4.51
HDP(0)	27.00	7.74
HDP(0,0)	24.52	8.99
SSiPP(2)	0.29	0.16
SSiPP(4)	1.33	1.29

Table 3: Average planning time (seconds) on several racetrack domain problems.

Tables 2 and 3 show average costs and times, respectively, obtained with the different algorithms in two racetracks. The table also shows the optimal cost obtained with LRTDP for reference. In terms of expected cost, FLARES(1), HDP, and SSiPP(4) all have near-optimal performance, with FLARES being considerably faster than HDP (up to 20 times in problem square-4), but SSiPP being faster than FLARES in problem ring-5. In general, FLARES was able to find near-optimal policies significantly faster than an optimal algorithm, with times competitive to state-of-the-art SSP solvers.

Sailing domain

We next present results on four instances of the sailing domain, described by (Kocsis and Szepesvári 2006). The problems 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). Tables 4 and 5 show average costs and times, respectively.

In this domain, FLARES(1) and HDP have similar results in terms of average cost, although HDP is around 5-10% better in all cases. However, FLARES(1) was significantly faster, with running times between 30% to 100% shorter. On the other hand, SSiPP required larger horizons ($t = 4$) to get reasonable results (still significantly worse than optimal) and in most instances had running times longer than FLARES.

Triangle-tireworld domain

We experimented on the triangle-tireworld, a so-called probabilistically interesting problem (Little and Thiebaux 2007), in order to study FLARES' ability to handle very-large problems. We varied our experimental settings to follow an approach similar to the one used during IPPC'08 (Bryce and

algorithm	size=20 goal=corner	size=40 goal=corner	size=20 goal=middle	size=40 goal=middle
LRTDP	89.47	178.21	47.23	93.27
FLARES(0)	125.13 ± 6.88	228.45 ± 6.55	74.39 ± 47.13	140.4 ± 6.66
FLARES(1)	94.31 ± 2.98	181.45 ± 3.56	49.68 ± 2.05	94.57 ± 2.94
HDP(0)	88.13 ± 2.62	178.03 ± 4.20	46.21 ± 1.87	93.78 ± 2.25
HDP(0,0)	88.99 ± 3.00	177.57 ± 4.29	46.13 ± 1.73	94.38 ± 2.75
SSiPP(2)	226.9 ± 12.45	410.98 ± 10.26	125.08 ± 8.34	259.96 ± 14.31
SSiPP(4)	123.9 ± 5.92	282.53 ± 11.32	64.95 ± 3.54	156.03 ± 8.34

Table 4: Average cost on several sailing domain problems.

algorithm	s=20 g=corner	s=40 g=corner	s=20 g=middle	s=40 g=middle
LRTDP	2.20	15.11	1.32	11.10
FLARES(0)	0.33	2.96	0.16	1.22
FLARES(1)	0.99	7.47	0.42	2.97
HDP(0)	1.25	10.58	0.79	6.13
HDP(0,0)	1.25	10.56	0.79	5.83
SSiPP(2)	0.41	1.34	0.20	0.62
SSiPP(4)	2.33	7.00	1.19	3.89

Table 5: Average planning time (seconds) on several sailing domain problems. Problems are labeled as s=(size) and g=(goal location).

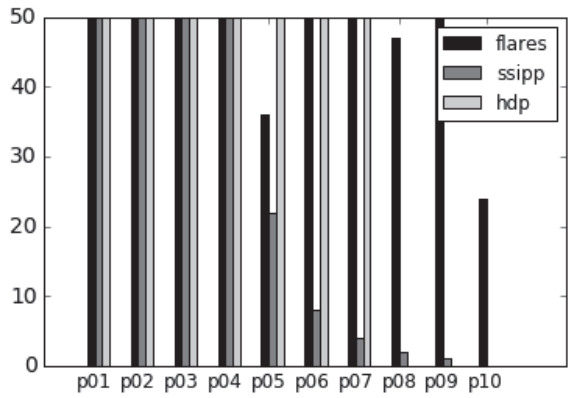

Figure 2: Number of successes in the first ten instances of the triangle-tireworld domain.

Buffet 2008), where the problems were obtained from. We give planners 20 minutes to solve each problem 50 times, and report the number of successes that were obtained within that time. Note that due to the large size of the problems, we used the inadmissible FF heuristic provided by the mGPT planner (Bonet and Geffner 2005). Although this invalidates some of the theoretical properties of FLARES, our goal is to empirically evaluate its performance on a large problem for which an admissible heuristic doesn't scale well.

Figure 2 shows the number of successes obtained by the planners in the first ten problems of this domain. Note that, with few exceptions, FLARES with $t = 4$ is able to achieve a high success rate in all problems, with 100% success rate in 7 out of 10 problems. This performance is comparable with RFF (Teichteil-Königsbuch, Kuter, and Infantes 2010), the best planner in the IPPC'08 competition, but, remarkably, it doesn't rely on the FF classical planner (Hoffmann and Nebel 2001) to speed up computation time [1].

On the other hand, the rest of the planners had significantly worse performance. HDP(0) achieves 100% success rate in problems up to p07, but fails completely on the remaining problems. Note that, in this domain, HDP(0) becomes equivalent to the optimal HDP algorithm; thus, it either successfully completes all runs, or it spends all the time during the initial planning without moving on to execution.

Finally, SSiPP can successfully handle instances up to p04, with a horizon of $t = 8$, but its performance quickly degrades with larger instances. The reason is the large horizon needed, increasing planning time considerably. Note that SSiPP can be coupled with the FF planner to obtain better performance (Trevizan and Veloso 2014), but that relies on the PPDDL representation of the domain, while FLARES is not tied to any particular problem representation.

[1] While FLARES is indeed using the FF heuristic, the speed-up obtained by RFF through direct use of the FF *planner* on much simpler determinized problems is significantly larger.

Conclusions

We present a novel approach to short-sightedness for SSPs, employing a new mechanism for labeling states as solved. In contrast to previous approaches, which focus computation on states that are close to the initial state, our approach allows larger sections of the state space to be explored and enables states close to the goal to quickly propagate information. As a consequence, this strategy requires smaller short-sighted horizons to be effective, and can thus significantly accelerate running times.

Following this idea, we introduce the FLARES algorithm, a modified version of LRTDP that incorporates a short-sighted labeling procedure to produce a fast approximate mechanism for action selection. 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 four different planning domains show that FLARES can produce a near-optimal policy faster than state-of-the-art algorithms, and performs

consistently well across a variety of domains.

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 short-sightedness based on the number of actions, but a version that uses trajectory probabilities, similar to work by Trevizan and Veloso (2014) is a straightforward extension.

Acknowledgement

We thank Sandhya Saisubramanian for fruitful discussions of this work. Support for this work was provided in part by the National Science Foundation under grant IIS-1524797.

References

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

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

Bertsekas, D. P., and Tsitsiklis, J. N. 1995. Neuro-dynamic programming: An overview. In *Proceedings of the 34th IEEE Conference on Decision and Control*, 560–564.

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

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

Bonet, B., and Geffner, H. 2005. mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research* 24:933–944.

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

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

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14(1):253–302.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, 135–143.

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

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

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

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

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

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

Trevizan, F. W., and Veloso, M. M. 2014. Depth-based short-sighted stochastic shortest path problems. *Artificial Intelligence* 216:179–205.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, 1010–1016.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, 352–359.

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

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the International Planning Competition. *Journal of Artificial Intelligence Research* 24(1):851–887.