Generalizing the Role of Determinization in Probabilistic Planning

Luis Pineda and **Shlomo Zilberstein**
Technical Report UM-CS-2017-006
College of Information and Computer Sciences
University of Massachusetts, Amherst, MA 01003
{lpineda, shlomo}@cs.umass.edu

Abstract

The stochastic shortest path problem (SSP) is a highly expressive model for probabilistic planning. The computational hardness of SSPs has sparked interest in determinization-based planners that can quickly solve large problems. However, existing methods employ a simplistic approach to determinization. In particular, they ignore the possibility of tailoring the determinization to the specific characteristics of the target domain. In this work we examine this question, by showing that learning a good determinization for a planning domain can be done efficiently and can improve performance. Moreover, we show how to directly incorporate probabilistic reasoning into the planning problem when a good determinization is not sufficient by itself. Based on these insights, we introduce a planner, FF-LAO*, that outperforms state-of-the-art probabilistic planners on several well-known competition benchmarks.

1 Introduction

One of the most popular models for probabilistic planning is the Stochastic Shortest Path SSP) [Bertsekas and Tsitsiklis, 1991]. A solution to an SSP is a sequence of actions that, starting from a given initial state, minimizes the expected cost of reaching a state from a given set of goal states. The uncertainty associated with the outcome of each action induces probabilistic transitions between states. This model has been used for a wide variety of applications, such as route planning in the presence of traffic delays [Lim *et al.*, 2013], quantifying the value of battery energy storage systems [Tan *et al.*, 2015], modeling wildfire propagation [Hajian *et al.*, 2016], and semi-autonomous driving [Wray *et al.*, 2016].

Despite its popularity, a major obstacle to the wide applicability of SSPs is the prohibitive computational cost of finding an optimal solution. This is due to the fact that optimal policies can, in the worst case, cover a number of states linear in the size of the state space, which in turn can be exponential in the number of variables describing the problem.

Given the difficulty of solving SSPs optimally, there has been much interest in developing methods that sacrifice optimality for the sake of computational efficiency. Among these methods, one of the most successful approaches has been the use of *determinization* complemented by replanning. These methods became popular thanks to the unexpected success of FF-Replan [Yoon *et al.*, 2007] in the International Probabilistic Planning Competition (IPPC) [Bryce and Buffet, 2008]. FF-Replan works by transforming the original problem into a deterministic one (e.g., by only considering most probable outcomes), and using the FF classical planner [Hoffmann and Nebel, 2001] to solve the resulting problems. This process is then repeated during execution whenever a state not covered by the current deterministic plan is observed.

Although FF-Replan is significantly faster than any optimal probabilistic planner, it performs poorly in many problem domains due to the fact that it ignores the uncertainty. This has led to the development of more robust determinization-based algorithms, that incorporate some form of probabilistic reasoning.

RFF [Teichteil-Königsbuch *et al.*, 2010]—the winner of IPPC'08—works by incrementally aggregating partial plans until the result covers a high probability envelop of states; each partial plan is computed using determinization and the FF planner. FF-Hindsight [Yoon *et al.*, 2008] works by sampling a set of deterministic "futures" of the original problem, solves each using FF, and combines their cost to estimate the costs in the original problem. HMDPP [Keyder and Geffner, 2008] introduces a self-loop determinization trick that nudges the deterministic planner into generating plans with low probability of deviation. SSiPP-FF [Trevizan and Veloso, 2014] works by creating short-sighted problems that consider only states up to a certain horizon from the current state. These smaller problems are solved optimally, and when a tip state is found during execution, the FF-Replan method is used.

Although these planners are capable of quickly solving large problems, they all employ a rather crude approach to determinization, by generally relying on either the most-likely-outcome (MLO) determinization, or the all-outcomes (AO) one. Using MLO can make planners ignore important sections of the state space (e.g., outcomes leading to dead-ends), resulting in poor policies. Using AO has other problems such as potentially wasting computation on irrelevant/unlikely sections of the state space, or treating all paths to the goal as equally important.

Recent work has shown that the choice of determinization can sometimes have a significant impact in the quality of a determinization-based planning approach [Pineda and Zilberstein, 2014]. In fact, even in some domains deemed "probabilistically interesting" [Little and Thiebaux, 2007] planning with a good determinization can actually result in optimal plans for the original SSP (e.g., the triangle tireworld domain). However, not much work has been done on generating methods for choosing a determinization that works well for a particular domain.

In this work, we address these issues with two main contributions. First, we present a new planner, FF-LAO*, that combines the LAO* optimal SSP solver [Hansen and Zilberstein, 2001] with the FF classical planner. FF-LAO* can leverage fast deterministic planning to estimate state values, but still partially reason about the complete probabilistic model if so-desired; it does this by relying on the *reduced models* framework introduced by Pineda and Zilberstein [2014].

Our second contribution is showing that it is possible to learn a good determinization on small instances of a planning domain, such that, when applied to larger instances, significant gains in efficiency and performance can be realized. This is the first work that selects a determinization based on the anticipated performance in the original probabilistic domain.

The rest of the paper is structured as follows: Section 2 gives background on SSPs and reduced models, Section 3 describes the FF-LAO* algorithm, Section 4 explains how to choose a good determinization for a particular planning domain, Section 5 presents experimental results, and Section 6 summarizes the conclusions and ideas for future work.

2 Background

A Stochastic Shortest Path (SSP) problem [Bertsekas and Tsitsiklis, 1991] is defined by a tuple $\langle S, A, T, C, s_0, G\rangle$, 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 G is a set of goal states satisfying $\forall s_g \in G, a \in A,\ T(s_g|s_g,a) = 1 \wedge C(s_g,a) = 0$. Moreover, we assume costs satisfy $\forall s \in S\setminus G, C(s,a) > 0$. Interestingly, SSPs are a variant of Markov Decision Process (MDPs) [Puterman, 1994] that has been shown to be more general than finite-horizon and infinite-horizon discounted MDPs [Bertsekas and Tsitsiklis, 1995].

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 $s_g \in G$ 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.

For an SSP to be well-defined, a policy must exist such that the goal is reachable from any state 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 then be found as the fixed point of the Bellman update operator (Eq. 1).

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

There is a variety of languages to compactly describe SSPs, of which PPDDL [Younes and Littman, 2004] has been most widely used within the AI community. As it turns out, it has been shown that finding whether a plan exists in a compactly described problem is EXPTIME-complete [Littman, 1997]. As mentioned earlier, this challenging complexity has led to the development of many approximate methods for solving SSPs. Particularly relevant to our work are determinization-based approaches, and, more generally, the reduced models framework.

2.1 Reduced Models

Given an SSP, the reduced models framework creates a simplified model characterized by two parameters: the number of outcomes per action that are fully accounted for (referred to as *primary* outcomes), and the maximum number of occurrences of the remaining outcomes that are planned for in advance (referred to as *exceptions*). This type of reduction generalizes single-outcome determinization, and introduces a spectrum of reductions with varying levels of probabilistic complexity.

The model assumes a factored representation of SSPs in which actions are probabilistic operators of the form:

$$a = \langle \textit{preconditions}, \textit{cost}, [p_1 : e_1, ..., p_m : e_m]\rangle,$$

where each effect e_i, for $i \in \{1, ..., m\}$, is associated with a probability p_i of occurring when a is executed, and there exists a successor function, τ, that maps effects to successor states, so that $s' = \tau(s, e_i)$ and $T(s'|s,a) = p_i$.

Using this formalism, a *reduced model* of an SSP $\langle S, A, T, C, s_0, G\rangle$ is defined as follows:

- The set of states is defined as $S' = S \times \{0, 1, .., k\}$, where k is a positive integer;
- The set of actions is the original set, A;
- The transition function is defined by Eq. (2);
- The cost function is defined as $C'((s,j),a) = C(s,a)$, for all $(s,j) \in S' \wedge a \in A$;
- The initial state is $(s_0, 0)$;
- The set of goals is defined as $G' = \{(s,j) \in S' | s \in G\}$.

The transition function defined below, while seemingly complicated, describes a simple process. The value j in state (s,j) keeps counts of how many exceptions have occurred up to that point in execution. For states where the count is less than the *exception bound*, k, the transition function operates as the original, except that the counter is increased by one for successors labeled as exceptions. On the other hand, for states with count $j = k$, the transition completely ignores exceptions, and only considers transitions to primary outcomes, redistributing the ignored probabilities so that they form a proper distribution (e.g., by normalizing). The notation $\mathcal{P}_a$ used below refers to the set of primary effects.

$$T'((s',j')|(s,j),a) = \begin{cases} p_i & \text{if } j < k \wedge j' = j \wedge e_i \in \mathcal{P}_a \\ p_i & \text{if } j < k \wedge j' = j+1 \wedge e_i \notin \mathcal{P}_a \\ p'_i & \text{if } j = j' = k \wedge e_i \in \mathcal{P}_a \\ 0 & \text{if } j = j' = k \wedge e_i \notin \mathcal{P}_a \end{cases} \quad (2)$$

where $s' = \tau(s, e_i)$ and the set $\{p'_1, ..., p'_m\}$ is any set of real numbers that satisfy

$$\forall i : e_i \in \mathcal{P}_a \;\; p'_i > 0 \quad \text{and} \quad \sum_{i: e_i \in \mathcal{P}_a} p'_i = 1 \quad (3)$$

Note that the reduced models framework encapsulates single-outcome determinization, which is simply a reduction where the set of primary outcomes has size 1 and the value of $k = 0$. In this work we are indeed concerned with reductions having a single primary outcome, but will include the possibility of using $k > 0$. We refer to these models as $\mathcal{M}_1^k$-reductions.

3 FF-LAO*

Reducing an SSP can significantly accelerate planning times by pruning large sections of the state space. However, there are many domains in which solving a reduced model optimally is still prohibitively expensive. In fact, for complex domains like the ones used in IPPC [Bryce and Buffet, 2008], using determinization and $k = 0$ already results in problems too large to be solved optimally in a practical manner.

To address this issue, we present a planner that combines the flexibility of the reduced models framework with the efficiency of a classical planner. We call this planner FF-LAO*, as it is an extension of the LAO* algorithm [Hansen and Zilberstein, 2001] that leverages the FF classical planner [Hoffmann and Nebel, 2001] to accelerate computation.

FF-LAO* (Algorithms 1-4) receives as input an $\mathcal{M}_1^k$-reduction, $\mathbb{S}' = \langle S', A, T', C', (s_0, 0), G' \rangle$; i.e., one that becomes deterministic after the exception bound is reached (equivalently, one where $\forall a \in A, |\mathcal{P}_a| = 1$). The remaining inputs are the exception bound, k, and the error tolerance, ϵ. We use $\mathbb{S}$ to denote the original SSP from which $\mathbb{S}'$ is derived.

FF-LAO* works almost exactly as LAO*, except that FF is used to compute values and actions for states that have reached the exception bound (i.e., states of the form (s, k)). This occurs in lines 4 and 8 of Algorithm 1, where the state expansion and test convergence procedures are replaced with versions that use FF (Algorithms 2 and 3, respectively).

Readers familiar with LAO* may notice differences with respect to the usual expansion and convergence test procedures. In particular, note the inclusion of *if* statements in line 7 (both procedures), where the successors of the expanded state are only added to the stack if $j < k$. The reason is that states (s, k) will be solved by calling FF, so there is no need to expand their successors.

It is possible, of course, to remove these *if* statements and let FF-LAO* continue the search; in that case, FF will be used as an inadmissible heuristic. However, this does not improve the theoretical properties of the algorithm (neither version is optimal), and results in higher computation times, so we prefer the version shown in the pseudocode.

The actual call to FF is done in Algorithm 4 (FF-BELLMAN-UPDATE). This procedure performs a Bellman update (Eq. 1) for any state (s, j) with $j < k$, and stores the updated cost estimate and best action in global variables $V[(s, j)]$ and $\pi[(s, j)]$, respectively (lines 6-7). For simplicity of presentation, we use the following action-value function:

$$Q((s,j),a) \equiv C'((s,j),a) + \sum_{(s',j')} T'((s',j')|(s,j),a) V[(s',j')]$$

and assume, as is common for heuristic search algorithms, that the values $V[(s', j')]$ are initialized using an admissible heuristic for $\mathbb{S}'$.

For states (s, k), the FF-BELLMAN-UPDATE procedure creates a PDDL file[1], denoted as D, representing the deterministic problem induced by M when $j = k$, with initial state s (CREATE-PDDL in line 3). The procedure then calls FF with input D (line 4) and memoizes costs and actions for all the states visited in the plan computed by FF (lines 5-7). More concretely, for each state s_i visited by this plan, we set $V[(s_i, k)]$ to be the cost, according to C', of the plan computed by FF for that state (line 6), and set $\pi[(s_i, k)]$ to be the corresponding action (line 7). Additionally, note that the estimates $V[(s, k)]$ are not admissible, even with respect to the input $\mathcal{M}_1^k$-reduction, since FF is not an optimal planner for deterministic problems. Finally, in the case that FF returns failure, we set $V[(s, k)] = \infty$ and $\pi[(s, k)] = $ NOP.

FF-BELLMAN-UPDATE also returns the residual, defined as the absolute difference between the previous cost estimate, and the estimate after applying the Bellman equation. This residual is used by FF-TEST-CONVERGENCE to check the stopping criterion of the algorithm.

Handling plan deviations during execution While FF-LAO* solves $\mathcal{M}_1^k$-reductions, the ultimate goal is to solve the SSP from which the reduction is derived from. As mentioned before, we use $\mathbb{S}$ to denote this SSP. It is easy to see that a complete policy for $\mathbb{S}'$ is not necessarily complete for $\mathbb{S}$. Therefore, during execution we need to be able to handle deviations from the plan returned by FF-LAO*.

We use a replanning approach to address this issue, FF-LAO*-REPLAN, illustrated in Algorithm 5. The idea is simple: during execution, check if the current state has an action already computed with $j = 0$. If that's the case, this action is executed (line 7). Otherwise, FF-LAO* is called to solve the reduced model with initial state $(s, 0)$ (lines 5-6). FF-LAO*-REPLAN receives the choice of determinization as input (Δ), and creates an $\mathcal{M}_1^k$-reduction accordingly (line 1).

Note that there are other choices for the replanning criterion. For example, checking if there is any $j \in [0; k]$ such that $(s, j) \in \pi$. Another alternative is to keep track of exceptions

[1]In practice, we create the PDDL file representing M before calling FF-LAO* and store its name in memory. CREATE-PDDL is shown for simplicity of presentation.

Algorithm 1: FF-LAO*

```
   input: $\mathbb{S}' = \langle S', A, T', C', (s_0, 0), G' \rangle$, $k$, $\epsilon$
1  while true do
       // Node expansion step
2      while true do
3          visited $\leftarrow \emptyset$
4          cnt $\leftarrow$ FF-EXPAND$\big(\mathbb{S}', (s, j), k, visited\big)$
5          if cnt $= 0$ then
               // No tip nodes were expanded, so current policy
                 is closed
               break
       // Convergence test step
6      while true do
7          visited $\leftarrow \emptyset$
8          error $\leftarrow$
             FF-TEST-CONVERGENCE$\big(\mathbb{S}', (s, j), k, visited\big)$
9          if error $< \epsilon$ then
               return // solution found
10         if error $= \infty$ then
               break // change in partial policy, go back to
                 expansion step
```

Algorithm 2: FF-EXPAND

```
   input: $\mathbb{S}' = \langle S', A, T', C', (s_0, 0), G' \rangle$, $(s, j)$, $k$, visited
1  if $(s, j) \in$ visited then
       return 0
2  visited $\leftarrow$ visited $\cup \{(s, j)\}$
3  cnt $= 0$
4  if $\pi[(s, j)] = \emptyset$ then
       // Expand this state for the first time
5      FF-BELLMAN-UPDATE$\big(\mathbb{S}', (s, j), k\big)$
6      return 1
7  else if $j < k$ then
8      forall $(s', j')$ s.t. $T'((s', j')|(s, j), \pi[(s, j)]) > 0$ do
9          cnt += FF-EXPAND$\big(\mathbb{S}', (s, j), k, visited\big)$
10 FF-BELLMAN-UPDATE$\big(\mathbb{S}', (s, j), k\big)$
11 return cnt
```

during execution, and set the value of j accordingly; in this case, j should be set to 0 after re-planning. Other alternatives are possible. We choose the one used by FF-LAO*-REPLAN because it is, in principle, the more robust choice, given that we use the maximum "look-ahead" every time[2]. However, if computational efficiency is a concern, other alternatives might be better. We leave a more in depth analysis of these choices for future work.

Theoretical considerations We now show conditions under which FF-LAO* is guaranteed to succeed. The following definition will be useful: a *proper policy rooted at* s is one that reaches a goal state with probability 1 from every state it can reach from s.

[2]Note that this is not guaranteed to be better than using $j > 0$, since pathological scenarios can be created where increasing k leads to worse plans.

Algorithm 3: FF-TEST-CONVERGENCE

```
   input: $\mathbb{S}' = \langle S', A, T', C', (s_0, 0), G' \rangle$, $(s, j)$, $k$, visited
1  if $s \in$ visited then
       return 0
2  visited $\leftarrow$ visited $\cup \{(s, j)\}$
3  error $= 0$
4  $a \leftarrow \pi[(s, j)]$
5  if $a = \emptyset$ then
       the test reached a state that hasn't been expanded yet
6      return $\infty$
7  else if $j < k$ then
8      forall $(s', j')$ s.t. $T'((s', j')|(s, j), \pi[(s, j)]) > 0$ do
9          error = max $\big($error,
             FF-TEST-CONVERGENCE$\big(\mathbb{S}', (s, j), k, visited\big)\big)$
10 error=max$\big($error, FF-BELLMAN-UPDATE$\big(\mathbb{S}', (s, j), k\big)\big)$
11 if $\pi(s, j) \neq a$ then
12     return $\infty$ // the policy changed
13 return error
```

Algorithm 4: FF-BELLMAN-UPDATE

```
   input: $\mathbb{S}' = \langle S', A, T', C', (s_0, 0), G' \rangle$, $(s, j)$, $k$
   output: error
1  $V' \leftarrow V[(s, j)]$
2  if $j = k$ then
3      $D \leftarrow$ CREATE-PDDL$(\mathbb{S}', s)$
4      $\{s_1, a_1, s_2, a_2, ..., s_L, a_L\} \leftarrow$ CALL-FF$(D)$
5      for $1 \leq L$ do
6          $V[(s_i, k)] \leftarrow \sum_{i \leq x \leq L} C'((s_x, k), a_i)$
7          $\pi[(s_i, k)] \leftarrow a_i$
8  else
9      $V[(s, j)] \leftarrow \min_a Q((s, j), a)$
10     $\pi[(s, j)] \leftarrow \arg\min_a Q((s, j), a)$
11 return $|V[(s, j)] - V'|$
```

Proposition 1. *Given an admissible heuristic for the reduced model* $\mathbb{S}'$*, if* $\mathbb{S}'$ *has at least one proper policy rooted at* $(s_0, 0)$*, then* FF-LAO* *is guaranteed to find one in finite time.*

Proof. Whenever FF-LAO* expands a state (s, k) and calls FF on this state, if the call succeeds, the states s_i, for $i \in [1, ..., L]$, that are part of the plan computed by FF essentially become terminal states of the problem, with costs set as in line 6. Since FF is a sub-optimal planner for deterministic problems, we have that $\sum_{i \leq x \leq L} C'((s_x, k), a_i) \geq V[(s_i, k)]$, and thus the values of all other states (s, j), with $j < k$, are guaranteed to be admissible with respect to the new updated value of the added terminal states. Therefore, after every successful call to FF, the resulting set of values and terminal states form a well-defined SSP, which LAO* is able to solve.

Moreover, in the case that a call to FF fails for some state $\hat{s}$, this state will be assigned an infinite cost, and thus the improved version of LAO* will avoid $\hat{s}$ as long there is some other path to the goal. Because FF is complete, any state belonging to a proper policy will be assigned a positive

Algorithm 5: FF-LAO*-REPLAN

```
  input: $\mathbb{S} = \langle S, A, T, C, (s_0, 0), G \rangle, \Delta, k, \epsilon$
1 $\mathbb{S}' \leftarrow$ CREATE-REDUCTION($\mathbb{S}, \Delta$)
2 $s \leftarrow s_0$
3 while $s \notin G$ do
4     if $(s, 0) \notin \pi$ then
5         REPLACE-INITIAL-STATE($\mathbb{S}', (s, 0)$)
6         FF-LAO*($\mathbb{S}', k, \epsilon$)
7     $s \leftarrow$ EXECUTE-ACTION($s, \pi[(s, 0)]$)
```

cost, so $\hat{s}$ couldn't have been part of a proper policy for M. Thus, under the conditions of the theorem, every call to FF transforms the problem becomes into an MDP with avoidable dead-ends [Kolobov *et al.*, 2012], which LAO* is able to solve[3]. □

Unfortunately, as is the case for virtually all replanning algorithms, not much can be guaranteed about the quality of plans found by FF-LAO*-REPLAN for $\mathbb{S}$. However, as we show in our experiments, by carefully choosing the input determinization, Δ and the bound k, FF-LAO*-REPLAN can find successful policies extremely quickly, even in domains well-known for their computational hardness and the presence of dead-end states.

4 Choosing a Good Determinization

Many stochastic domains have an inherent structure that make some of their determinizations significantly more effective than others. Consider, for instance, the triangle-tireworld domain [Little and Thiebaux, 2007]. The agent has to reach one of the vertices in a planar graph of triangular shape, but after every move there is the possibility of getting a flat tire (see Figure 1). If this happens, it must get a spare tire before being able to move again. However, spares are only available in certain locations, and there is only a single path from the start to the goal such that all locations in the path have spares. This domain has two possible determinizations, depending on whether a flat tire happens after moving or not. As it turns out, it is possible to get the optimal policy for this problem by planning as if a flat tire will always occur. The interesting part is that this is true for *all* instances of this problem, regardless of size.

Triangle-tireworld is a great example of a domain where all domain instances share a probabilistic structure that can be captured by determinization. In practical terms, this means that it is possible to learn a determinization on the smaller problems, and then use it for solving larger ones.

We adopt this approach for choosing the input determinization to FF-LAO*-REPLAN, Δ. We assume a set of problems of varying sizes are available and that it is easy to identify the smaller ones. This is the case for the IPPC benchmarks that we consider in our experiments, as they are typically ordered

[3]While this is not true for the original version of LAO*, this is true of the so-called *improved* version of LAO* that we use in this work, which performs Bellman backups of states in depth-first fashion, in post order traversal.

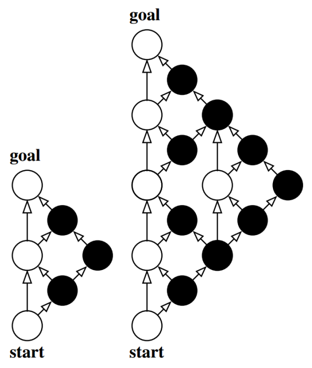

Figure 1: Two instances of the TRIANGLE-TIREWORLD domain. Locations with spare tires are marked in black.

Algorithm 6: LEARNING-DET

```
  input: $\mathcal{D}, \mathbb{S}_l, k$
  output: $\Delta$
1 $\{\Delta_1, ..., \Delta_M\} \leftarrow$ Create all possible determinizations of $\mathcal{D}$
2 forall $i \in \{1, ..., M\}$ do
3     $P_i, \mathbb{C}_i \leftarrow$ Estimate probability of successs and expected
          cost of executing FF-LAO*-REPLAN($\mathbb{S}_l, \Delta_i, k$)
4 $P^* \leftarrow \max_i P_i$
5 $\Delta \leftarrow \Delta_{\min_i \mathbb{C}_i \text{ s.t. } P_i = P^*}$
```

by problem size/difficulty. However, in general, some analysis of the problem might be required, for example, by counting the number of possible grounded atoms that the problem description induces. Another possibility is to generate small problems automatically from the domain description. This has the advantage that it doesn't require additional problems for learning the determinization to use (besides the actual problem to solve). Although this is certainly an interesting possibility, it requires some finesse and it is outside the scope of this work.

Algorithm 6 illustrates LEARNING-DET, a brute-force approach to learn a determinization Δ for domain $\mathcal{D}$; $\mathbb{S}_l$ represents the problem used for learning. This procedure does a comprehensive search over the space of all determinizations. For each, we estimate the probability of success (P_i) and the expected execution cost ($\mathbb{C}_i$) of running FF-LAO*-REPLAN on $\mathbb{S}_l$; the costs are estimated using Monte-Carlo simulations. Finally, we pick the determinization with the lowest expected cost, among the ones with the highest probability of success.

There are some subtleties involved in this process. Note that FF-LAO* is only guaranteed to terminate if the input reduction has a proper policy from its initial state. This will most likely not be the case for many of the determinizations explored by LEARNING-DET; in fact, under some determinizations the goals might be completely unreachable from any state.

We address this issue using a well-known technique for planning with problems involving dead-ends. In particular, we use a cap M on state costs, including the costs assigned when FF fails, and modify the Bellman backup operator used by FF-LAO* as

$$V(s) = \min \left\{ M, \min_{a \in A} \left\{ C(s,a) + \sum_{s' \in S} T(s'|s,a)V(s') \right\} \right\}$$

which guarantees the convergence of heuristic search algorithms [Kolobov *et al.*, 2012]. While this introduces a new parameter impacting the planner's decisions, and hides the true impact of dead-end states. Note that LEARNING-DET still attempts to maximize the multi-objective evaluation criterion typically used when unavoidable dead-ends exist [Kolobov *et al.*, 2012; Steinmetz *et al.*, 2016].

5 Experiments

Domains and methodology

We evaluated FF-LAO* and LEARNING-DET on a set of problems taken from IPPC'08 [Bryce and Buffet, 2008]. Specifically, we used the first 10 problem instances of the following four domains: TRIANGLE-TIREWORLD, BLOCKSWORLD, EX-BLOCKSWORLD, and ZENOTRAVEL. Unfortunately, the rest of the IPPC'08 domains are not supported by our PPDDL parser [Bonet and Geffner, 2005]. Additionally, we modified the EX-BLOCKSWORLD domain to avoid the possibility of blocks to be put on top of themselves [Trevizan and Veloso, 2014].

The evaluation methodology was similar to the one used in past planning competitions: we give each planner 20 minutes to solve 50 rounds of each problem (i.e., reach a goal state starting from the initial state). Then we measure its performance in terms of the number of rounds that the planner was able to solve during that time. All experiments were conducted on an Intel Core i7-6820HQ machine running at 2.70GHz with a 4GB memory cutoff.

We evaluated the planners using the MDPSIM [Younes *et al.*, 2005] client/server program for simulating SSPs, by having planners repeatedly perform the following three steps: 1) connect to the MDPSIM server to receive a state, 2) compute an action for the received state and send the action to the MDPSIM server, and 3) wait for the server to simulate the result of applying this action and send a new state. A simulation ends when a goal state is reached, when an invalid action is sent by the client, or after 2500 actions have been sent by the planner.

We compared the performance of FF-LAO* with our own implementations of FF-REPLAN and RFF, as well as the original author's implementation of SSIPP [Trevizan and Veloso, 2014]. We evaluated two variants of FF-REPLAN, one using MLO (FF_s) and another one using AO (FF_a). For RFF we used MLO and the *Random Goals* variant, in which before every call to FF, a random subset (size 100) of the previously solved states are added as goal states. Additionally, we used a probability threshold $\rho = 0.2$. The choice of these parameters was informed by analysis in the original work [Teichteil-Königsbuch *et al.*, 2010]. For SSIPP we used $t = 3$ and the h_{add} heuristic, parameters also informed by the original work [Trevizan and Veloso, 2014].

For FF-LAO*, we learned a good determinization to use by applying LEARNING-DET on the first problem of each domain (p01), with $k = 0$. This choice of k was motivated both by time considerations, and by the rationale that $k = 0$ should better reflect the impact of each determinization (since FF-LAO* becomes a fully determinization-based planner). We used a dead-end cap $\mathcal{D} = 500$ throughout our experiments. We initialized values with the non-admissible FF heuristic [Bonet and Geffner, 2005].

We ran LEARNING-DET offline, prior to the MDPSIM evaluation. Note, however, that the time taken by the brute force search plus the time used to solve problem p01 with the chosen determinization was, in all cases, well below the 20 minutes limit (approx. 2 minutes in the worst case). The remaining parameter for FF-LAO* is the value of k. We report the best performing configuration in the range $k \in [0, 3]$, which was $k = 0$ for most domains, with the exception of EX-BLOCKSWORLD, which required $k = 3$. Note that FF-LAO* with $k = 0$ is essentially equivalent to FF-REPLAN, so any advantage obtained over FF_s and FF_a is completely derived from the choice of determinization.

Results and Discussion

Figure 2 shows the number of successful rounds obtained by each planner in the benchmarks. In general, FF-LAO* either tied for the best, or outperformed the baselines. All planners had a 100% success rate in BLOCKSWORLD, so there is not much room for comparison.

In the TRIANGLE-TIREWORLD domain, FF-LAO* and FF_s had 100% success rate, while RFF ran out of time in the last 3 problems. On the other hand, the performance of SSIPP and FF_a deteriorated quickly as the problem instance increased. It is worth pointing out that the performances of FF_s and RFF in this domain are quite sensitive to tie-breaking—there are only two outcomes to choose from, each occurring with 0.5 probability. As the results of FF_a suggest, a different choice would have resulted in a much worse success rate. On the other hand, the use of LEARNING-DET gets around this issue by automatically choosing the best determinization to use, a process that took seconds. While we do note that the *best goals* parameterization of RFF gets around this issue, its computational cost is much harder, so it's not obvious that it would actually improve performance in this case [Teichteil-Königsbuch *et al.*, 2010].

In the EX-BLOCKSWORLD domain FF-LAO* (with $k = 3$) and SSIPP significantly outperform the other two planners, solving 252 and 250 rounds, respectively, against 187 for both FF_s and RFF, and 200 for FF_a. Interestingly, in this domain the determinization found by LEARNING-DET is not sufficient to obtain good performance; in fact, only 3 problems had a non-zero success rate with $k = 0$. This highlights the utility of doing probabilistic reasoning with FF-LAO*. Although not shown here for space considerations, the performance with $k = 1$ (214 successful rounds) was already better than all the baselines, except for SSIPP.

In ZENOTRAVEL, FF-LAO* and FF_a were remarkably better than the other two planners: they achieved 100% success rate in all domain instances, while the other baselines failed almost all of the rounds. In the case of the determinization-based planners, this is due to the goal becoming unreachable under MLO, so the choice of determinization

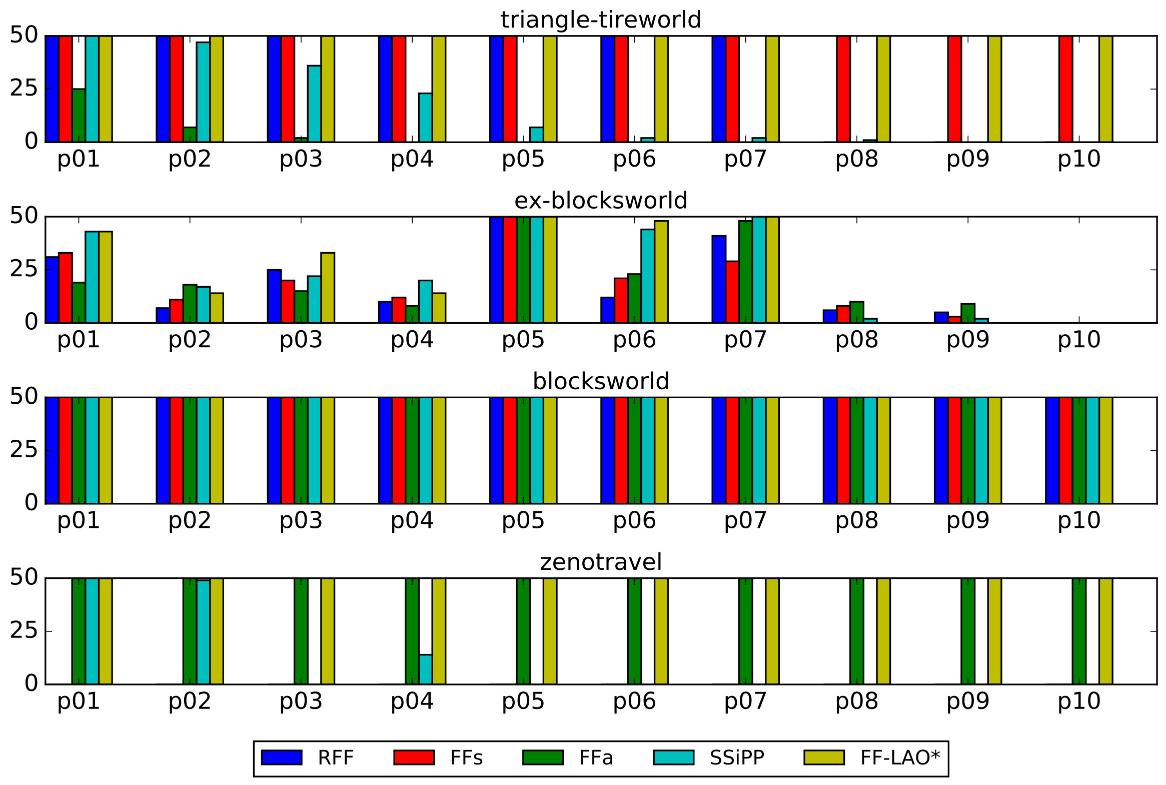

Figure 2: Number of solved rounds by 5 different planners in IPPC'08 benchmarks.

has a significant impact on performance.

6 Conclusion

In this work we presented a novel perspective on the use of determinization for probabilistic planning, by showing that a careful choice of determinization can outperform state-of-the-art planners. We also introduced a new planner, FF-LAO*, that, given a choice of determinization, leverages the power of classical planning algorithms for computational efficiency, but can also reason probabilistically if desired.

We proposed a strategy for selecting a determinization that takes advantage of the inherent structure of a given stochastic domain. We show that the choice of determinization can generalize across problems of varying size, in particular, in terms of its impact on planning performance (probability of success and expected cost).

We compared our approach to state-of-the-art planners for goal-oriented probabilistic problems, using a set of benchmarks taken from the International Probabilistic Planning Competition. Our results strongly support the claim that the choice of determinization can lead to very substantial gains in performance, and position FF-LAO*—using our determinization learning approach—as a competitive planner for large stochastic domains.

In future work, we plan to explore ways to vary the choice of determinization according to state-space features, and to develop automated methods for generating problems useful for learning a good determinization of a given domain.

References

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

[Bertsekas and Tsitsiklis, 1995] 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.

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

[Bryce and Buffet, 2008] Daniel Bryce and Olivier Buffet. Sixth international planning competition: Uncertainty part. In *Proceedings of the Sixth International Planning Competition (IPC'08)*, 2008.

[Hajian *et al.*, 2016] Mohammad Hajian, Emanuel Melachrinoudis, and Peter Kubat. Modeling wildfire propagation with the stochastic shortest path: A fast simulation approach. *Environmental Modelling & Software*, 82:73–88, 2016.

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

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

[Keyder and Geffner, 2008] Emil Keyder and Hector Geffner. The HMDPP planner for planning with probabilities. In D. Bryce and O. Buffet, editors, *ICAPS Third International Probabilistic Planning Competition*. IPPC'08, 2008.

[Kolobov *et al.*, 2012] Andrey Kolobov, Mausam, and Daniel S. Weld. A theory of goal-oriented MDPs with dead ends. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 438–447, Catalina Island, California, 2012.

[Lim *et al.*, 2013] Sejoon Lim, Christian Sommer, Evdokia Nikolova, and Daniela Rus. Practical route planning under delay uncertainty: Stochastic shortest path queries. In *Proceedings of Robotics: Science and Systems*, volume 8, pages 249–256, 2013.

[Little and Thiebaux, 2007] 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.

[Littman, 1997] Michael L. Littman. Probabilistic propositional planning: Representations and complexity. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 748–754, Providence, Rhode Island, 1997.

[Pineda and Zilberstein, 2014] 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, Portsmouth, New Hampshire, 2014.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[Steinmetz *et al.*, 2016] Marcel Steinmetz, Joerg Hoffmann, and Olivier Buffet. Revisiting goal probability analysis in probabilistic planning. In *In Proceedings of the 26th International Conference on Automated Planning and Scheduling*, 2016.

[Tan *et al.*, 2015] Xiaoqi Tan, Yuan Wu, and Danny H.K. Tsang. A stochastic shortest path framework for quantifying the value and lifetime of battery energy storage under dynamic pricing. *IEEE Transactions on Smart Grid*, pages 769–778, 2015.

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

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

[Wray *et al.*, 2016] Kyle Hollins Wray, Luis Pineda, and Shlomo Zilberstein. Hierarchical approach to transfer of control in semi-autonomous systems. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1285–1286, 2016.

[Yoon *et al.*, 2007] Sung Wook Yoon, Alan Fern, and Robert Givan. FF-Replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 352–359, Providence, Rhode Island, 2007.

[Yoon *et al.*, 2008] Sungwook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. Probabilistic planning via determinization in hindsight. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*, pages 1010–1016, Chicago, Illinois, 2008.

[Younes and Littman, 2004] 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.

[Younes *et al.*, 2005] 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.