Heuristic Search for Generalized Stochastic Shortest Path MDPs

Andrey Kolobov Mausam Daniel S. Weld
{akolobov, mausam, weld}@cs.washington.edu
Dept of Computer Science and Engineering
University of Washington
Seattle, USA, WA-98195

Hector Geffner
hector.geffner@upf.edu
Departamento de TIC (DTIC)
ICREA & Universitat Pompeu Fabra
Barcelona, Spain, E-08018

Abstract

Research in efficient methods for solving infinite-horizon MDPs has so far concentrated primarily on discounted MDPs and the more general stochastic shortest path problems (SSPs). These are MDPs with 1) an optimal value function V^* that is the unique solution of Bellman equation and 2) optimal policies that are the greedy policies w.r.t. V^*.

This paper's main contribution is the description of a new class of MDPs, that have well-defined optimal solutions that do not comply with either 1 or 2 above. We call our new class Generalized Stochastic Shortest Path (GSSP) problems. GSSP allows more general reward structure than SSP and subsumes several established MDP types including SSP, positive-bounded, negative, and discounted-reward models. While existing efficient heuristic search algorithms like LAO* and LRTDP are not guaranteed to converge to the optimal value function for GSSPs, we present a new heuristic-search-based family of algorithms, FRET (Find, Revise, Eliminate Traps). A preliminary empirical evaluation shows that FRET solves GSSPs much more efficiently than Value Iteration.

Introduction

Research in efficient methods for solving infinite-horizon undiscounted MDPs has so far concentrated primarily on a particular subclass of these models, the stochastic shortest path (SSP) problems. According to their most general definition (Bertsekas 1995), in SSPs there must exist at least one *proper* policy, one that reaches the goal with probability 1, and all *improper* policies must have a reward of $-\infty$ in some state. Since the second condition is hard to verify, in practice it is replaced with a slightly stronger version that forbids 0- or positive-reward actions.

The baseline algorithm for solving SSP MDPs, Value Iteration (VI) (Bellman 1957), is a dynamic programming approach that starts with an initial estimate of the states' values, V_0, and iteratively applies the *Bellman backup* operator to them. The optimal value function V^* is the unique fixed point of Bellman backup on SSPs, and repeated application of this operator provably forces VI to converge to V^* irrespectively of the initializing value function V_0. Unfortunately, since VI stores values for the entire state space, it runs out of memory on all but the smallest MDPs.

State-of-the-art optimal SSP MDP algorithms have a much smaller memory consumption. Many of them, e.g. LRTDP (Bonet and Geffner 2003b) and LAO* (Hansen and Zilberstein 2001), fall under the heuristic search paradigm, conceptually described by the Find-and-Revise (F&R) framework (Bonet and Geffner 2003a). F&R algorithms use the knowledge of the initial state and an *admissible heuristic* (an initial estimate for the value function doesn't underestimate the values of any states under V^*) to compute the optimal policy for an SSP while avoiding visits to many of the states that are not part of that policy. Besides saving space, this makes them significantly faster than VI. Crucially, however, F&R algorithms use the Bellman backup operator, and their optimality, like that of VI, also hinges on this operator having a single fixed point on SSP MDPs.

Although SSP MDPs cover a lot of scenarios, many are left out. For instance, consider planning the process of powering down a nuclear reactor. In this scenario, one is interested in choosing the sequence of actions that maximizes the probability of a successful shutdown; accordingly, we set the costs of actions to 0 and assign a reward of 1 for attaining the goal state. Probability optimization is more advantageous than cost optimization in this case. Actual action costs (insertion of control rods, adjusting coolant level, *etc.*) and especially the penalty for unsuccessful shutdown, required for the latter approach, are difficult to estimate, and optimizing probability obviates the need for them. However, the resulting MDP contains "loops" of 0-cost actions (e.g., insert control rods, raise control rods) and is thus not an SSP.

Perhaps unexpectedly, known optimal algorithms that work on SSPs break down on the described problem and, more generally, when rewards are allowed to take on all real values. On MDPs with arbitrary-valued rewards, the Bellman backup operator may have *multiple* suboptimal fixed points. As a consequence, VI is not guaranteed to converge to the optimal one if initialized admissibly. More importantly, F&R algorithms are not guaranteed to converge to the optimal solution either, leaving us with no efficient methods to solve such MDPs.

To remedy the situation, we present the first heuristic search scheme that, when paired with an informative admissible heuristic, allows solving goal-oriented MDPs with a more general reward model than SSP admits optimally and efficiently. This paper makes the following contributions:

- We define a new class of MDPs, the generalized stochastic shortest path MDPs (GSSP), and the semantics of optimal solutions for it. GSSP allows a general action reward model, properly containing several notable classes of infinite-horizon problems, e.g. SSP, discounted, positive-bounded, and negative MDPs (Puterman 1994).

- We analyze the properties of GSSPs and show that Bellman backups fail to converge to the optimal value function on GSSPs when initialized admissibly, defeating existing heuristic search methods. The cause of this failure are multiple suboptimal fixed points for the Bellman backup operator on this MDP class.
- We introduce a new framework, FRET (**F**ind, **R**evise, **E**liminate **T**raps), capable of solving GSSP MDPs optimally and efficiently with the help of an admissible heuristic. It does so by not only improving the value function with Bellman backups but also by escaping this operator's suboptimal fixed points in a novel way that preserves the value function's admissibility. Besides describing the framework, we outline the proof of FRET's optimal convergence.
- We present an empirical evaluation on an interesting subclass of GSSP called MAXPROB to demonstrate that FRET significantly outperforms VI in both memory consumption and speed even when the latter is aided by reachability analysis.

Background

MDPs. In this paper, we focus on probabilistic planning problems that are modeled by undiscounted-reward infinite-horizon MDPs with a start state, defined as tuples of the form $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, s_0 \rangle$ where

- $\mathcal{S}$ is a finite set of states,
- $\mathcal{A}$ is a finite set of actions,
- $\mathcal{T}$ is a transition function $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0; 1]$ that gives the probability of moving from s_i to s_j by executing a,
- $\mathcal{R}$ is a map $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that specifies action rewards,
- s_0 is the start state.

Solving an MDP means finding a policy whose execution from a given state results in accumulating the largest expected reward. A more precise definition is below.

The presence of a start state goes hand-in-hand with heuristic search methods because if one needs to find a solution for the entire $\mathcal{S}$ these methods' ability to save memory by avoiding visits to some of the states becomes irrelevant. Therefore, for all MDP classes discussed in this paper the knowledge of the start state will be assumed, even for those whose standard definition omits it.

A *value function* is a mapping $V : \mathcal{S} \to \mathbb{R}$. A *policy* is a rule π that prescribes an action to take for any given state.

Letting random variables S_t and A_t denote respectively the state of the process after t steps and A_t the action selected in S_t, the expected value V^π of policy π is

$$V^\pi(s) = \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \mathcal{R}(S_t, A_t) \right] \quad (1)$$

In other words, the value of policy π at a state s is the expectation of total reward the policy accumulates if the execution of π is started in s. In turn, every value function V has a policy π^V that is *greedy* w.r.t. V, i.e. that satisfies

$$\pi^V(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')V(s') \right] \quad (2)$$

Note, however, that V^{π^V} is not necessarily equal to V.

Optimally solving an MDP means finding a policy that maximizes V^π. Such policies are denoted π^*, and their value function $V^* = V^{\pi^*}$, called the *optimal value function*, is defined as $V^* = \max_\pi V^\pi$. For all MDPs we will discuss in this paper, V^* also satisfies the following condition, the *Bellman equation*, for all $s \in \mathcal{S}$:

$$V(s) = \max_{a \in \mathcal{A}} \left[\mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')V(s') \right] \quad (3)$$

In general, the expectation in (1) may diverge. This makes the definition of π^* not very meaningful, since it potentially has to choose among policies with infinitely high reward. To ensure that every policy's value is bounded from above, we need additional constraints on the problem specification. Next, we discuss a class of infinite-horizon MDPs that illustrate what these constraints may be.

Stochastic Shortest Path Problems. Perhaps the best-known class of undiscounted infinite-horizon MDPs are the stochastic shortest path (SSP) MDPs. They are defined as tuples $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{G}, s_0 \rangle$, where $\mathcal{G}$ is a set of (absorbing) goal states and other components are as above. For each $g \in G$, $\mathcal{T}(g,a,g) = 1$ and $\mathcal{R}(g,a) = 0$ for all $a \in A$, which forces the agent to stay in g forever while accumulating no reward.

The SSP definition also has the following restrictions:

- Each $s \in \mathcal{S}$ must have at least one *proper* policy, one that reaches a goal state from any state s with probability 1.
- Every *improper* policy must get the reward of $-\infty$.

The second requirement is hard to verify in the presence of arbitrary action rewards, so most optimal SSP solvers assume a stronger version that requires $\mathcal{R}(s,a) < 0$ for all states and actions.

This definition guarantees that $V^\pi < \infty$ for all π, and that for at least one policy $V^\pi > -\infty$. Thus, such MDPs always have at least one finite-valued policy.

The baseline method for solving SSPs, Value Iteration (VI), starts by initializing state values with an arbitrary V_0. Afterwards, it executes several sweeps of the state space and updates every state during every sweep by using the Bellman equation as an assignment, the *Bellman backup operator*. Viewing V_i as the value function operated upon in the i-th sweep of VI and denoting the Bellman backup operator as $\mathscr{B}$, the i-th update round can be expressed as

$$V_{i+1} = \mathscr{B}V_i \quad (4)$$

Another way to view this update is to define the *Q-value* of an action a in a state s w.r.t a value function V to be

$$Q^V(s,a) = \mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')V(s')$$

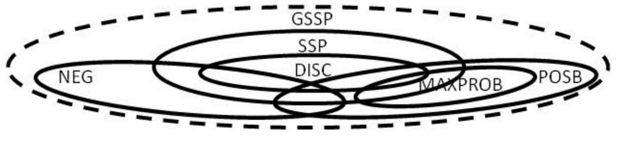

Figure 1: Hierarchy of infinite-horizon MDP classes.

and observe that for each s, $\mathscr{B}$ sets $V_{i+1}(s)$ to be the highest Q-value of any action in s w.r.t. V_i.

Find-and-Revise Framework. Because it stores and updates the value function for the entire $\mathcal{S}$, VI can be slow and lack memory for solving even relatively small SSPs. However, thanks to $\mathscr{B}$'s convergence to V^* on SSPs independently of initialization, VI has given rise to several much more efficient algorithms for this MDP class. An assumption they make is that the initial state s_0 is known in advance, and one is interested in a policy originating in s_0 only. With this small caveat, it was shown that if the initialization function V_0 (called a heuristic) is admissible, i.e. satisfies $V_0 \geq V^*$ under componentwise comparison, then one can compute V^* for the states relevant to reaching a goal from s_0 without updating or even memoizing values for many of the other states (Barto, Bradtke, and Singh 1995). The Find-and-Revise (F&R) framework (Bonet and Geffner 2003a) generalizes this idea by comprising the algorithms that start with an admissible V_0 and iteratively build the graph of the policy greedy w.r.t. the current V (i.e., the policy derived from V via (2)), finding those of the graph whose values haven't converged and updating them using $\mathscr{B}$. Since an admissible heuristic, computable on the fly, makes some states look bad a-priori and the policy is greedy, they may never end up in the policy graph, making F&R algorithms (e.g., (Bonet and Geffner 2003b)) both speedy and frugal in memory use. Nonetheless, the policies F&R yields under an admissible heuristic are provably optimal upon convergence.

Generalized Stochastic Shortest Path MDPs

Although the SSP model is fairly general, it disallows interesting problem types, e.g. those concerned with maximizing the probability of reaching the goal. The key reason is the SSPs' reward model, which excludes MDPs with "cycles" of actions that the agent can execute forever without paying any cost and without reaching the goal. In this section, we define generalized stochastic shortest path MDPs (termed GSSPs for conciseness), a class of problems that relaxes the limitations both on action rewards. Exploring GSSPs mathematical properties will suggest the main idea behind efficient methods for handling this class of MDPs. It will also demonstrate that besides SSP, GSSP contains several other important MDP classes, including discounted-reward (DISC), positive-bounded (POSB), and negative (NEG) problems. See Figure 1 for a graphical representation of the hierarchy.

Definition A *generalized stochastic shortest path* (GSSP) MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{G}, s_0 \rangle$ with the same components as SSP under the following conditions:

1. There exists a policy π that is *proper w.r.t.* s_0, i.e. reaches a goal state from s_0 with probability 1.
2. $V_+^\pi(s) \equiv \mathbb{E}_s^\pi \left[\sum_{t=0}^{\infty} \max\{0, \mathcal{R}(S_t, A_t)\}\right] < \infty$ for all policies π for all states s reachable from s_0 under any policy. $V_+^\pi(s)$ is the expected sum of *nonnegative* rewards yielded by the given policy; in GSSPs this sum must always be bounded from above.

The objective is to find a reward-maximizing policy π^* that reaches the goal from s_0 with probability 1, i.e.

$$\pi^* = \underset{\pi \text{ proper w.r.t. } s_0}{\text{argmax}} V^\pi \qquad (5)$$

Accordingly, we define $V^*(s)$ to be

$$V^* = \sup_{\pi \text{ proper w.r.t. } s_0} V^\pi \qquad (6)$$

Note an important subtlety in Equation 5. When selecting the optimal policy, it considers only *proper* policies, whereas SSP's optimal policy is selected among *all* existing policies. Why do we need to make this distinction? The simple answer is that in SSP MDPs, the reward-maximizing policy is *always* proper, whereas in GSSP this may not be so. Intuitively, since SSPs disallow 0- and positive-reward cycles, the faster the agent reaches a goal state, the less cost it will incur, i.e. going for the goal is the best thing to do in an SSP. In GSSPs, 0-reward cycles *are* possible. As a consequence, if reaching the goal requires incurring a cost but a 0-reward cycle is available, the reward-optimal course of action for the agent is to stay in the 0-reward cycle. However, semantically we may want the agent to go for the goal. Considering only proper policies during optimization, as Equation 5 requires, enforces this semantics.

One may ask whether it is natural in a decision-theoretic framework to prefer a policy that reaches the goal over one maximizing reward, as the presented semantics may do. However, it is intuitively clear and can be shown formally that in any MDP if attaining the goal has a sufficiently high reward, the best goal-striving policy will also be reward-maximizing. In this case, both optimization criteria will yield the same solution. At the same time, determining the "equalizing" goal reward value can be difficult, and GSSP removes the need for doing this. To sum up, the GSSP and the traditional solution semantics are largely equivalent, but the former makes the modeling process simpler by needing fewer parameters to be estimated.

As another consequence of the optimal solution, for any state s from which no proper policy exists, $V^*(s) = -\infty$. This follows from Equation 6 and the fact that $\sup \emptyset = -\infty$. Moreover, no such state is reachable from s_0 by any policy proper w.r.t. s_0. Also, for any goal state g, $V^*(g) = 0$, since from the moment of reaching the goal onwards the system accumulates no reward.

Only conditions (1) and (2) of the GSSP definition prevent it from covering all MDPs with arbitrary action rewards. However, MDPs that violate (1) have $V^*(s_0) = -\infty$ and hence, by Equation 6, $V^\pi(s_0) = -\infty$ under any π. For such MDPs, the total undiscounted expected reward criterion fails to distinguish between the quality of different policies. The average reward (Puterman 1994) or the probability maximization criterion, mentioned previously and formalized later in this paper, is more preferable. Condition (2) is in place for technical reasons explained in later sections.

GSSPs have the following mathematical properties, illustrated by the MDP in Figure 2, some of which drastically change the behavior of known SSP solution techniques:

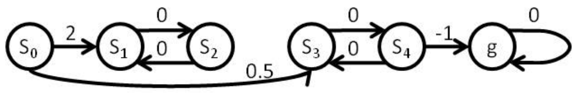

Figure 2: An example GSSP MDP presenting multiple challenges for computing the optimal value function and policy efficiently. State g is the goal.

- *V^* is a fixed point of Bellman Backup.* E.g., for the GSSP in Figure 2, $V^*(s_0) = -0.5$, $V^*(s_1) = V^*(s_2) = -\infty$, $V^*(s_3) = V^*(s_4) = -1$, $V^*(g) = 0$, which satisfies the Bellman equation.
- *Bellman backup has multiple suboptimal fixed points $V > V^*$.* This property invalidates optimality guarantees that Bellman backup-based algorithms give on SSPs. Consider an admissible $V(s_0) = 4$, $V(s_1) = V(s_2) = 2$, $V(s_3) = V(s_4) = 1$, $V(g) = 0$. All policies greedy w.r.t. V are clearly suboptimal, but V stays unchanged under $\mathscr{B}$. For s_1 and s_2 the value of V^* should be set to $-\infty$, since no policy can reach the goal from them, but $\mathscr{B}$ is unable to do it. The situation with s_3 and s_4 is even trickier. They are also part of a "vicious cycle", but reaching the goal from them *is* possible. However, staying in the loop forever accumulates more reward (0) than going to the goal (-1), frustrating $\mathscr{B}$ as well.
- *Not every policy greedy w.r.t. V^* is proper.* This fact further complicates finding the optimal policy for GSSPs, even if V^* is known. Consider the policy that loops from s_3 to s_4 in Figure 2 indefinitely. It is greedy w.r.t. V^* but never reaches the goal.

Due to multiple fixed points, Bellman backup initialized admissibly will generally converge to a suboptimal one. Since this operator is the basis for heuristic search algorithms, they will yield suboptimal solutions as well.

A potential optimal approach for tackling GSSP MDPs would first find the problematic sets of states like $\{s_1, s_2\}$ and $\{s_3, s_4\}$, replace each of them with a single state, and then consider the resulting (SSP) MDP. As we later show more formally, "collapsing" such regions of the state space is sound — all states within them *must* be connected with 0-reward actions, so the values of all states in a given region are equal under V^*. Unfortunately, discovering these regions generally involves visiting the entire state space, i.e. would take as much memory as the more straightforward approach, VI initialized with $V_0 \leq V^*$. Both approaches, though optimal, are much too expensive in practice.

Thus, the failure of Bellman backup leaves us with no optimal but space- and time- efficient techniques capable of solving GSSPs. In the next section, we present an algorithm that resolves this difficulty.

The FRET Framework

In this section, we introduce a framework called FRET (**F**ind, **R**evise, **E**liminate **T**raps) that encompasses algorithms capable of solving GSSPs with a known starting state when initialized with an admissible heuristic. At the highest level, FRET starts with an admissible $V_i = V_0$ and

- *In the Find-and-Revise step, finds the next largest V_i' that satisfies $V_i' = \mathscr{B}V_i'$.* This V_i' may have problematic regions described in the previous section (Figure 2).
- *In the Eliminate Traps step, changes V_i' to remove the problematic regions.* The result is a new admissible V_{i+1} s.t. $V_{i+1} < V_i'$.
- *Iterates the two steps above until convergence to V^*.* V^* has no problematic regions.
- *Extracts a proper optimal policy from V^*.*

To explain the technique in more detail, we need to introduce several additional definitions.

Definition A *policy graph of a policy π rooted at s_0* is a directed graph $G^\pi = \{S_\pi, A_\pi\}$, whose set of nodes S_π is the set of all states reachable from s_0 under π and $A_\pi = \{(s_i, s_j) | s_i, s_j \in S_\pi$ and $\exists a \in \mathcal{A}$ s.t. $\pi(s_i, a) > 0$ and $\mathcal{T}(s_i, a, s_j) > 0\}$.

Definition A *greedy graph of value function V rooted at s_0* is a directed graph $G^V = \cup_{\pi^V}(G^{\pi^V})$.

Put simply, G^V is the combined reachability graph of all policies greedy w.r.t V. G^V will play a critical role in FRET. It allows efficiently determining whether any greedy policy under the current V has a chance of reaching a goal from a given state s and adjusting V accordingly. Importantly, G^V is very easy to construct; its edges correspond precisely to the set of actions greedy w.r.t. V in any state.

Definition A *reachability graph rooted at s_0* is a directed graph $G = \cup_V G^V$.

Definition A *trap* is a maximal strongly connected component (SCC) $C = \{S_C, A_C\}$, of G_V with the following properties:

- For all $g \in G, g \notin S_C$.
- G_V has no edges (s_i, s_j) s.t. $s_i \in S_C$ but $s_j \notin S_C$. In particular, there is no path in G_V from any state $s \in S_C$ to a goal $g \in \mathcal{G}$.

Definition A *permanent trap* is a trap $C = \{S_C, A_C\}$ s.t. for every $s_i \in S_C$, every edge (s_i, s_j) of G is also in A_C.

Definition A *transient trap* is a trap $C = \{S_C, A_C\}$ s.t. for some $s_i \in S_C$, there is an edge (s_i, s_j) of G s.t. $s_j \notin S_C$.

Informally, a trap is just a strongly connected component of G_V with no outgoing edges in G_V and no goal states, a goal-free leaf in the DAG of maximal SCCs. A permanent trap consists of a set of states from that doesn't have any outgoing edges not only in G_V but also in G. Therefore, no policy can ever reach a goal from any of the states in a permanent trap, i.e. all states in permanent traps are *dead ends*. For states in transient traps, a proper policy may exist, but it can't be greedy w.r.t. the current V. In Figure 2, $\{s_1, s_2\}$ form a permanent trap, and $\{s_3, s_4\}$ a transient one.

We can now cast the operation of FRET in terms of these definitions. Throughout the explanation, we will be referring to the pseudocode in Algorithm 1. As already described, FRET iteratively applies two transformations (lines 12-16 of Algorithm 1) to an admissible V_0. The first of them, Find-and-Revise, behaves almost exactly as described in (Bonet and Geffner 2003a); it iteratively searches G^π of the current policy π for states whose values havent converged yet and updates them, possibly changing π in the process. In essence, it is performing asynchronous VI (Bertsekas 1995)

Algorithm 1 FRET

```
1:  **Input**: GSSP MDP $M$, admissible value function $V_0$.
2:  **Output**: Optimal policy $\pi^*$.
3:
4:  declare $G = \{S_G, A_G\} \leftarrow M$s reachability graph
5:
6:
7:  **function FRET**($M$, $V_0$)
8:  declare $V_i \leftarrow V_0$
9:  declare $V_i' \leftarrow$ Find-and-Revise$(M, V_i)$
10: declare $V_{i+1} \leftarrow$ Eliminate-Traps$(M, V_i')$
11:
12: **while** $V_{i+1} \neq V_i'$ **do**
13:     $V_i \leftarrow V_{i+1}$
14:     $V_i' \leftarrow$ Find-and-Revise$(M, V_i)$
15:     $V_{i+1} \leftarrow$ Eliminate-Traps$(M, V_i')$
16: **end while**
17:
18: declare $V^* \leftarrow V_{i+1}$
19: declare $\pi^* \leftarrow$ the optimal policy
20: declare $Processed \leftarrow \mathcal{G}$
21: declare $G^{V^*} = \{S_{G^{V^*}}, A_{G^{V^*}}\} \leftarrow$ greedy graph of $V^*$
22:
23: **while** $Processed \neq S_{G^{V^*}}$ **do**
24:     choose $s \in S_{G^{V^*}} \setminus Processed$
25:     choose $a \in \mathcal{A}$ s.t. $\mathcal{T}(s, a, s') > 0$ for some $s' \in Processed$
26:     $Processed \leftarrow Processed \cup \{s\}$
27:     $\pi^*(s) \leftarrow a$
28: **end while**
29:
30: **return** $\pi^*$
31:
32:
33: **function Eliminate-Traps**($M$, $V$)
34: declare $V_{next} \leftarrow V$
35: declare $G_V = \{S_V, A_V\} \leftarrow V$s greedy graph
36: declare $\mathcal{SCC} \leftarrow Tarjan(G^V)$
37: declare $C \leftarrow \emptyset$
38:
39: **for all** SCC $C = \{S_C, A_C\} \in \mathcal{SCC}$ **do**
40:     **if** $(\nexists (s_i, s_j) \in A_G : s_i \in S_V, s_j \notin S_V)$ and $(\nexists g \in \mathcal{G} : g \in S_C)$ **then**
41:         $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$
42:     **end if**
43: **end for**
44:
45: **for all** Trap $C = \{S_C, A_C\} \in \mathcal{C}$ **do**
46:     **if** $\nexists a \in \mathcal{A}, s \in S_C, s' \notin S_C : \mathcal{T}(s, a, s') > 0$ **then**
47:         **for all** $s \in S_C$ **do**
48:             $V_{next}(s) \leftarrow -\infty$
49:         **end for**
50:     **else**
51:         declare $A_e \leftarrow \{a \in \mathcal{A} \mid \exists s \in S_C; s' \notin S_C : \mathcal{T}(s, a, s') > 0\}$
52:         **for all** $s \in S_C$ **do**
53:             $V_{next}(s) \leftarrow \max_{s \in S_C, a \in A_e} Q^V(s, a)$
54:         **end for**
55:     **end if**
56: **end for**
57:
58: **return** $V_{next}$
```

until convergence — this is the view of F&R algorithms we are going to adopt in this paper. Properties of F&R guarantee that if F&R's initialization function for the $(i+1)$-th iteration, V_i, is admissible, so is its output function in that iteration, V_i'. However, the examples in the previous section have demonstrated that in the presence of 0-reward actions F&R may not reach V^*, and that $G^{V_i'}$ may contain permanent and transient traps.

The second step, Eliminate Traps (ET) (lines 33-56), searches $G^{V_i'}$ for traps and changes $V_i'(s)$ for states s in them to $V_{i+1}(s) < V_i'(s)$ in such a way that $V_{i+1}(s)$ is still admissible. For states not in any trap, it sets $V_{i+1}(s) = V_i'(s)$ (line 34). To find traps, ET employs Tarjan's algorithm (Tarjan 1972) (its pseudocode is omitted) to identify all SCCs of $G^{V_i'}$ and considers only those that have no goal states and outgoing edges in this graph (lines 39-43), i.e. satisfy the definition of a trap. For each such SCC C:

- If no $s_i \in S_C$ has actions that may transition to a state $s_j \notin S_C$, C is a permanent trap. (Note that this check is equivalent to verifying the definition of a permanent trap but avoids building the full reachability graph G.) Therefore, ET sets $V_{i+1}(s) = -\infty$ for all $s \in S_C$ (lines 46-49).
- Else, C is a transient trap. The intuition behind our analysis of this case is as follows. Currently, we can't tell with certainty whether some policy can actually reach a goal state from the states of C. However, *if such a policy existed*, it would inevitably have to exit C via some action. Therefore, we can set the next estimate for these states' values to be the Q-value of the best such "exit" action under the current value function estimate, in the view of the latter's admissibility. More formally, since V_i' is admissible but none of its greedy policies can reach a goal from any $s \in S_C$, two options are possible. All $s \in S_C$ may in fact be dead ends, but the current V_i' doesn't show it. Alternatively, there does exist a policy π that reaches a goal from all $s \in S_C$ (recall that C is a strongly connected component, so any state in it can be reached from any other with probability 1) but for each $s \in S_C$, $V^\pi(s) < V_i'(s)$. To preserve the admissibility of the value function through the iterations, FRET assumes the latter and produces $V_{i+1}(s)$ s.t. $V^* \leq V_{i+1}(s) < V_i'(s)$ for these states (lines 51-54). Namely, it sets $V_{i+1}(s)$ for all the states in a trap to the highest Q-value of any action in any of C's states that has a chance of transitioning to some $s' \notin S_C$.

To understand the reason behind this step, we need the following proposition (proved in the next section):

Proposition (Lemma 1 in the Convergence Proof section). *All actions that transition from a state of a transient trap only to states of the same trap have the reward of 0.*

Now, suppose a goal-reaching policy π from one of the states $s \in S_C$ exists. Then, in some $s_e \in S_C$ it must use an exit action a_e whose edges aren't part of $G^{V_i'}$ and that transitions to a state outside of S_C. If A_e is the set of all actions that have a chance of transitioning to the outside of S_C from a state in S_C (line 46), then since V_i' is admissible, we have

$$Q^{V^*}(s_e, a_e) \leq Q^{V_i'}(s_e, a_e) \leq \max_{s \in S_C, a \in A_e} Q^{V_i'}(s, a)$$

Since for every a in A_e at least one of a's edges does not belong to G^{V_i}, we must have $\max_{s \in S_C, a \in A_e} Q^{V'_i}(s,a) < V'_i(s')$ for all $s' \in S_C$. Thus, $\max_{s \in S_C, a \in A_e} Q^{V'_i}(s,a)$ provides both an admissible $V_{i+1}(s_e)$ and an improvement over $V'_i(s_e)$. But what about other states in C? According to the above proposition, traveling among states within C is "free". Therefore, π can get from any $s \in S_C$ to s_e without accumulating additional reward, and we can set $V_{i+1}(s) = V_{i+1}(s_e)$ for all $s \in S_C$.

Thus, ET always makes an improvement to a strictly admissible value function and produces a new admissible V_{i+1} for use in the next iteration of FRET.

Upon convergence, FRET constructs a proper policy as in lines 18-28 of the pseudocode.

The pseudocode in Algorithm 1 is only meant as a conceptual description of FRET and omits many optimizations. For instance, as Lemma 1 implies, all states in a given transient trap are connected with 0-reward actions. Hence, all of them must have the same value under V^*. Therefore, once a trap has been identified, F&R can treat all its states as one in Bellman backups. This is equivalent to "collapsing" a trap into one state, which makes both the subsequent F&R and ET steps faster than in a straightforward implementation of the pseudocode.

Example. Consider the GSSP in Figure 2 and suppose FRET starts with $V_0(s_0) = 4, V_0(s_1) = V_0(s_2) = 2, V_0(s_3) = V_0(s_4) = 1, V_0(g) = 0$. This function already satisfies $V_0 = \mathscr{B}V_0$, so the F&R step in the first iteration finishes immediately with $V'_0 = V_0$. ET then builds $G^{V'_0}$, which includes only states s_0, s_1, and s_2. The only (permanent) trap is formed by s_1, s_2. Thus, ET produces $V_1(s_0) = 4, V_1(s_1) = V_1(s_2) = -\infty, V_1(s_3) = V_1(s_4) = 1, V_1(g) = 0$. In the second round, F&R starts with V_1 and converges to $V'_1(s_0) = 1.5, V'_1(s_1) = V'_1(s_2) = -\infty, V'_1(s_3) = V'_1(s_4) = 1, V'_1(g) = 0$. $G^{V'_1}$, consisting of s_0, s_3, s_4, and g, again contains a trap, this time a transient one formed by s_3, s_4. The best exit out of it (and the only one in this case) is the action that goes from s_4 to g with $Q^{V'_1} = -\infty$. The ET step in the second round realizes this and produces $V_2(s_0) = 1.5, V_2(s_1) = V_2(s_2) = -\infty, V_2(s_3) = V_2(s_4) = -1, V_2(g) = 0$. Finally, in the third round, F&R converges to $V'_3(s_0) = -0.5, V'_3(s_1) = V'_3(s_2) = -\infty, V'_3(s_3) = V'_3(s_4) = -1, V'_3(g) = 0$. $G^{V'_3}$ contains no traps, so FRET has converged: $V'_3 = V^*$.

FRET Convergence Proof

Due to the lack of space, this section presents only a proof outline. The main idea of the proof is to observe that in every iteration, both the F&R and the Eliminate Traps step either lowers the value function or leaves it unchanged, and both steps preserve its admissibility. Thus, the sequence of value functions at the ends of FRET's iterations, $\mathcal{V} = \{V_i\}_{i=1}^{\infty}$ is monotonically decreasing and bounded from below by V^*. Therefore, this sequence converges to some value function $V_L \geq V^*$. V_L satisfies $V_L = \mathscr{B}V_L$ over the states in G^{V_L} and has a greedy proper policy. These facts allow us to complete the proof by demonstrating that $V_L = V^*$.

The key step in the proof is showing that the elimination of transient traps preserves admissibility. Conceptually, once FRET detects a transient trap, it collapses all the states in the trap into one "superstate" and performs a Bellman backup on it. The following lemma, referred to in the previous section, shows that, by itself, collapsing a trap doesn't affect admissibility of a value function:

Lemma 1. *Suppose V is admissible and satisfies $V = \mathscr{B}V$. For every state s in a transient trap $C = \{S_C, A_C\}$ of G^V, for every action $a \in \mathcal{A}$, if for every s' s.t. $\mathcal{T}(s,a,s') \neq 0$ s' is also in C then $\mathcal{R}(s,a)$ must be 0.*

Proof Sketch. The lemma postulates that all intra-trap actions must carry 0 reward. To prove it, consider a policy π^V_u that on every visit to a given state s of C chooses an action a uniformly at random from the set of all greedy actions in s. Since C is a trap, an agent that ended up in C and uses a greedy policy (such as π^V_u) will remain in C forever. Moreover, if the agent uses π^V_u it will visit every $s \in S_C$ an infinite number of times in expectation.

Therefore, if some s, a had $\mathcal{R}(s,a) > 0$ then π^V_u would pick a an expected infinite number of times and we would have $V_+^{\pi^V_u}(s) = \infty$. This violates condition (2) in the GSSP definition. In the light of this, $\mathcal{R}(s,a) \leq 0$ for all s, a of C. However, if some s, a had $\mathcal{R}(s,a) < 0$ then π^V_u would accumulate a reward of $-\infty$. It can be shown that in this case trap C would not be part of G^V, leading to another contradiction. Thus, $\mathcal{R}(s,a) = 0$. $\square$

Since transitions within a transient trap incur no cost, under V^* all states of the trap have the same value. Therefore, they can be merged and updated as one from the moment the trap is discovered. Thus, FRET's execution can be viewed as a sequence of Bellman backups intertwined with merging of states, both of which preserve admissibility. The sequence starts from an admissible V_0, so for all $i > 0$, V_i is also admissible.

With some additional work, we can establish the main result regarding convergence of FRET on GSSPs:

Theorem 2. FRET *converges to V^* on GSSP MDPs when initialized with an admissible V_0.*

GSSP and Other MDP Classes

Despite GSSP relaxing the conditions on action rewards and thus covering a very diverse class of problems, one might view the requirements of goal and proper policy existence as too restrictive. In this section, we try to allay these concerns by showing that, besides SSP, at least two major established classes of MDPs free of these requirements can be viewed as subclasses of GSSP, with the benefit of FRET being applicable to them as well. Figure 1 displays the hierarchy of MDP classes derived in this section.

Our proofs of class containment involve constructing an MDP M' from the original MDP M. To distinguish the components of these MDPs, we will denote the former as $\mathcal{S}_{M'}, \mathcal{R}_{M'}$, *etc.*, and $\mathcal{S}_M, \mathcal{R}_M$, *etc.*, respectively.

Positive-Bounded MDPs. Positive-bounded (POSB) MDPs (Puterman 1994) are defined as having no goal states and imposing the following model restrictions:

- Actions are allowed to have arbitrary rewards, but for every state s there must exist $a \in \mathcal{A}_s$ s.t. $\mathcal{R}(s, a) \geq 0$.
- $V^{\pi}_{+}(s) = \mathbb{E}^{\pi}_{s} \sum_{t=0}^{\infty} \max\{0, R(S_t, A_t)\} < \infty$ for all policies π for all states s (GSSP has an identical requirement).

POSBs are interesting to us in this context because they are not only non-goal-oriented but also don't explicitly require the existence of a proper policy (every policy in them is proper, see Proposition 7.2.1b in (Puterman 1994)) while nonetheless allowing a mix of positive-, 0-, and negative-reward actions.

Theorem 3. *$POSB \subset GSSP$, i.e. for every MDP $M \in POSB$ there exists an MDP $M' \in GSSP$ s.t. optimal policy sets of M and M' are equal.*

Proof. As in the previous section, we provide only the main proof idea. Construct M' by letting $\mathcal{S}_{M'}, \mathcal{A}_{M'}, \mathcal{T}_{M'}, \mathcal{R}_{M'}$, and s_0 be the same as for M. Suppose G_M is the full reachability graph of M rooted at s_0. Build a DAG of SCCs of G_M and identify SCCs with no outgoing edges whose internal edges correspond to 0-reward actions in $\mathcal{A}_M$ (at least one such SCC must exist because of the $V^{\pi}_{+} < \infty$ requirement). Let $\mathcal{G}_{M'}$ be the set of states in these SCCs.

M' is a GSSP whose goals are states in 0-reward regions of M. Both in M' and in M, the optimal policy will try to reach these states while getting maximum reward, i.e. the sets of optimal solutions for these MDPs are identical. □

Negative MDPs. Negative MDPs (NEG) (Puterman 1994) are non-goal-oriented and have the following model restrictions:

- For all $s \in \mathcal{S}, a \in \mathcal{A}$ the model must have $\mathcal{R}(s, a) \leq 0$.
- At least one proper policy must exist.

While requiring the existence of a proper policy, similarly to POSBs NEGs have no goal and allow 0-reward actions in addition to negative-reward ones. A result similar to the above holds for them too, with an analogous proof.

Theorem 4. *$NEG \subset GSSP$.*

MAXPROB MDPs. Last but not least, we turn to another problem that FRET can solve, finding the "best" policy in MDPs that do not have a proper policy but otherwise fall under the definition of SSP. In these MDPs, all policies accumulate a reward of $-\infty$. One way to distinguish between them is to use the average reward optimality criterion (Puterman 1994) instead of the undiscounted total expected reward discussed in this paper. However, to our knowledge, there are no efficient methods (e.g., akin to heuristic search) to optimize according to the former.

An alternative evaluation measure is used at the International Probabilistic Planning Competition (IPPC) (Bryce and Buffet 2008) to judge the performance of various MDP solvers. It judges policy quality by its probability of eventually reaching the goal as opposed to hitting a dead end, preferring policies with the highest probability of doing the former. Observe that this is also the optimization criterion of the nuclear reactor shutdown mentioned in the Introduction section. We'll call this optimization criterion *MAXPROB*.

To find the best policy of an SSP M according to this criterion, we need to solve a version M', which we call the MAXPROB MDP, of the original SSP MDP M's. M' is derived from M by assigning 0 reward for all of M's actions in all states and giving a reward of 1 for visiting M's goal. (Formally, M' adds a special action a_g that carries a reward of 1 in all of M's goal states and 0 elsewhere, and leads to a new state g that has a 0-reward self-loop.) In other words, in M' the agent gets a reward only for visiting a former goal of M and executing a_g in it.

Crucially, MAXPROB is *not* a subclass of SSP, since the former, unlike the latter, admits the existence of traps. In fact, the following two results are easily verified:

Theorem 5. *$MAXPROB \subset POSB \subset GSSP$.*

Theorem 6. *$V^*(s_0)$ of MAXPROB MDP M' derived from an SSP MDP M is the probability with which the optimal policy of M eventually reaches a goal state of M from s_0.*

To give a flavor of FRET's effectiveness, we concentrate further on MAXPROB MDPs derived from hard problems used at IPPC that lack a proper policy. We show that in terms of the MAXPROB criterion, FRET can solve them optimally and much more efficiently than the only other optimal algorithm available for solving them, VI.

Heuristics for GSSP

Unsurprisingly, due to the complete prior lack of heuristic search methods for solving MDPs with traps (such MDPs are abound even in previously established classes like POSB and NEG), research in admissible heuristics for them has also been nonexistent. To prompt it, we suggest a technique that can both augment an admissible heuristic and serve as such on its own in factored GSSP MDPs, where states are represented as vectors of variable values.

SixthSense (Kolobov, Mausam, and Weld 2010) has been proposed recently as an algorithm for identifying dead ends in factored SSPs of the kind used in IPPC. It operates by learning sound but incomplete rules that identify states as dead ends by the values of state variables in them while the host MDP solver is running.

The following observation leads us to suggest SixthSense for GSSPs as well. In GSSPs, many of the dead ends are located in permanent traps. As demonstrated, permanent traps are the regions of the state space difficult for Bellman backups to improve. Consequently, an admissibly initialized F&R algorithm is likely to visit them and enter them in the state value table before Eliminte Traps corrects their values. Employing SixthSense during the Find-and-Revise step should change values of many dead-end states early, thereby helping F&R avoid actions that may lead to them. These actions typically have other states as their potential outcomes as well, so in the long run identifying dead ends helps prevent unnecessary visits to large parts of the state space. Moreover, since SixthSense's rules efficiently identify a dead end on-the-fly, these states's values don't need to be memorized. The empirical evaluation presented next supports the validity of this reasoning.

Experimental Results

In this section, we provide a preliminary experimental evaluation of FRET by comparing it to VI. VI and policy iteration (PI) are the only two other major algorithms able to solve GSSPs optimally. However, since they do not use heuristics

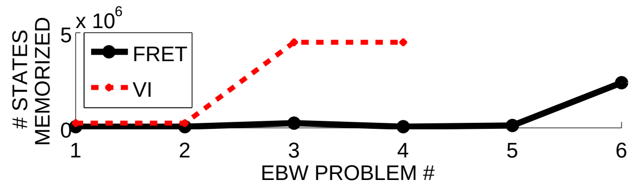

Figure 3: Under the guidance of an admissible heuristic, FRET uses much less memory than VI.

to exclude states from consideration, they necessarily memorize the value of every state in $\mathcal{S}$. Therefore, their space efficiency, the main limiting factor of MDP solvers' scalability, is the same, and for this criterion the results of the empirical comparison of FRET and VI apply to PI as well.

In the experiments, we use problems from the Exploding Blocks World (EBW) set of IPPC-2008 (Bryce and Buffet 2008). MDPs in this set are interesting because although they have strictly negative rewards they do not necessarily have a proper policy. Under the IPPC rules, competition participants were given some time to find a policy for each of the 15 problems of the EBW set and made to execute the policy 100 times on each problem. The winner on the EBW set was deemed to be the solver whose policy achieved the goal in the largest percentage of these trials on average.

Note that the participants's policies were effectively judged by the MAXPROB criterion — the planner whose policy maximized MAXPROB would win in the expectation. As FRET provides a technique to optimize for this criterion exactly, the objective of our experiments was comparing FRET's performance on MAXPROB with that of VI.

In the experiments, we use LRTDP in the Find-and-Revise step of FRET, aided by the following heuristic V_0. Recall that SixthSense is a mechanism for quickly identifying dead ends in MDPs. In the MAXPROB MDP M' derived from an SSP MDP M, the former dead ends of M are states from which getting to the former goals of M, and hence getting any reward other than 0, is impossible. Therefore, their value under V^* in M' is 0. Since SixthSense can point out a large fraction of M's dead ends, we can assign V_0 for these states to be 0. Similarly, V_0 will be 0 at the special state g of M', where the action a_g leads. For all other states, V_0 will assign 1, the highest reward any policy possibly can obtain in M'. Since SixthSense is provably sound, V_0 is guaranteed to be admissible.

All MDPs in EBW have a specified initial state s_0. FRET can use this information to its advantage but VI by default will still try to compute the value function for all states, even those not reachable from s_0. To make the comparison fair, we modify VI to operate only on states reachable from s_0 as well. In the experiments, VI uses an inadmissible heuristic that assigns value 0 to all the states.

The experiments were run with 2GB of RAM. The results are presented in Figures 3 and 4. EBW problems increase in state space size from the lowest- to highest-numbered one. Out of the 15 available, FRET solved the first 6 before it ran out of memory, whereas VI managed to solve only 4. (As a side note, some IPPC participants managed to solve more problems; however, this is not surprising because these algorithms are suboptimal). Also, FRET solved problem 6 faster than VI solved problem 4. These data indicate that even with a crude heuristic, FRET significantly outperforms VI in time and memory consumption. Moreover, no heuristic can help VI avoid memorizing the entire reachable state space and thus allow it to solve bigger problems, whereas with the advent of more powerful admissible GSSP heuristics the scalability of FRET will only increase.

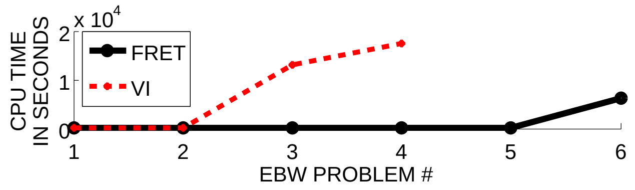

Figure 4: VI lags far behind FRET in speed.

Conclusion

We presented GSSP, a new class of goal-oriented infinite-horizon MDPs whose main advantage is the lack of restrictions on action reward values. GSSP contains several other established infinite-horizon MDP classes, including stochastic shortest path, positive-bounded, negative, and discounted-reward problems. We described a heuristic search framework for solving GSSPs, FRET. It starts with an admissible value function and repeatedly modifies it with a novel operator. Unlike pure Bellman backup, this operator is guaranteed to converge to the optimal solution on GSSPs when initialized admissibly. Finally, we suggested using an algorithm called SixthSense as a starting point for constructing admissible GSSP heuristics. The preliminary empirical evaluation clearly shows FRET's advantage in efficiency over non-heuristic-search methods like VI.

Acknowledgments. We would like to thank Martine De Cock, Jesse Davis, and the anonymous reviewers for insightful comments and discussions. This work was supported by WRF/TJ Cable Professorship and the following grants: ONR N000140910051, NSF IIS-1016465, TIN2009-10232 (MICINN, Spain).

References

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Bertsekas, D. 1995. *Dynamic Programming and Optimal Control*. Athena Scientific.

Bonet, B., and Geffner, H. 2003a. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI*, 1233–1238.

Bonet, B., and Geffner, H. 2003b. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS'03*, 12–21.

Bryce, D., and Buffet, O. 2008. International planning competition, uncertainty part: Benchmarks and results. In *http://ippc-2008.loria.fr/wiki/images/0/03/Results.pdf*.

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

Kolobov, A.; Mausam; and Weld, D. 2010. SixthSense: Fast and reliable recognition of dead ends in MDPs. In *AAAI'10*.

Puterman, M. 1994. *Markov Decition Processes*. John Wiley & Sons.

Tarjan, R. 1972. Depth-first search and linear graph algorithms. In *SIAM Journal on Computing*, 1(2):146–160.