bers about it only on the day of the event. Getting to the concert venue requires a flight, either by hiring a business jet or by a regular airline with a layover. The first option is very expensive but almost guarantees making the concert on time. The second is much cheaper but, since the concert is so soon, missing the connection, a somewhat probable outcome, means missing the concert. Nonetheless, the cost of missing the concert is only the price of the ticket, so a rational agent would choose to travel with a regular airline. Accordingly, one of the MDP classes we propose, fSSPUDE, assumes that the agent can put a price (penalty) on ending up in a dead end state and wants to compute a policy with the least expected cost (including the possible penalty). While seemingly straightforward, this intuition is tricky to operationalize because of several subtleties. We overcome these subtleties and show how fSSPUDE can be solved with easy modifications to existing SSP algorithms.

In the third MDP class we introduce, iSSPUDE, not only are dead ends unavoidable, but the cost of hitting one is assumed to be infinitely large. Consider, for example, the task of planning an ascent to the top of Mount Everest for a group of human alpinists. Such an ascent is fraught with inherent lethal risks, and to any human, the price of their own life can be taken as infinite. Note the conceptual difficulty with this setting: since every policy reaches an infinite-cost state, the expected cost of any policy is also infinite. This makes SSP’s cost-minimization criterion uninformative. Instead, for an undertaking as above, a natural primary objective is to maximize the probability of getting to the goal (i.e., to minimize the chance of getting into a lethal accident, a dead-end state). However, of all policies maximizing this chance, we would prefer the least costly one (in expectation). This is exactly the multiobjective criterion we propose for this class of MDPs. Solving iSSPUDE is much more involved than handling the previous two classes, and we introduce two novel algorithms for it.

Intuitively, the objectives of fSSPUDE and iSSPUDE MDPs are related — as fSSPUDE’s dead-end penalty gets bigger, the optimal policies of the two classes coincide. We provide a theoretical and an empirical analysis of this insight, showing that solving fSSPUDE yields an optimal policy for iSSPUDE if the dead-end penalty is high enough.

Thus, the paper makes four contributions: (1) three new goal-oriented MDP models that admit the existence of dead-end states; (2) optimal VI and heuristic search algorithms for solving them; (3) theoretical results describing equivalences among problems in these classes; and (4) an empirical evaluation tentatively answering the question: which class should be used when modeling a given scenario involving unavoidable dead ends?

2 Background and Preliminaries

SSP MDPs. In this paper, we extend an MDP class known as the Stochastic Shortest Path (SSP) problems with an optional initial state, defined as tuples of the form \( \langle S, A, T, C, G, s_0 \rangle \), where \( S \) is a finite set of states, \( A \) is a finite set of actions, \( T \) is a transition function \( S \times A \times S \rightarrow [0, 1] \) that gives the probability of moving from \( s_i \) to \( s_j \) by executing \( a \), \( C \) is a map \( S \times A \rightarrow \mathbb{R} \) that specifies action costs, \( G \) is a set of (absorbing) goal states, and \( s_0 \) is an optional start state. For each \( g \in G \), \( T(g, a, g) = 1 \) and \( C(g, a) = 0 \) for all \( a \in A \), which forces the agent to stay in \( g \) forever while accumulating no reward.

An SSP must also satisfy two conditions: (1) It must have at least one complete proper policy, a rule prescribing an action for every state with which an agent can reach a goal state from any state with probability 1. (2) Every improper policy must incur the cost of \( \infty \) from all states from which it cannot reach the goal with probability 1.

When the initial state is unknown, solving an SSP MDP means finding a policy whose execution from any state allows an agent to reach a goal state while incurring the least expected cost. We call such a policy complete optimal, and denote any complete policy as \( \pi \). When the initial state is given, we are interested in computing an optimal (partial) policy rooted at \( s_0 \), i.e., one that reaches the goal in the least costly way from \( s_0 \) and is defined for every state it can reach from \( s_0 \) (though not necessarily for other states).

To make the notion of policy cost more concrete, we define a cost function as a mapping \( J : S \rightarrow \mathbb{R} \cup \{\infty\} \) and let random variables \( S_t \) and \( A_t \) denote, respectively, the state of the process after \( t \) time steps and the action selected in that state. Then, the cost function \( J^\pi \) of policy \( \pi \) is

\[
J^\pi(s) = \mathbb{E}_s^\pi \left[ \sum_{t=0}^{\infty} C(S_t, A_t) \right]
\]  

(1)

In other words, the cost of a policy \( \pi \) at a state \( s \) is the expectation of the total cost the policy incurs if the execution of \( \pi \) is started in \( s \). In turn, every cost function \( J \) has a policy \( \pi^J \) that is \( J \)-greedy, i.e., that satisfies

\[
\pi^J(s) = \arg\min_{a \in A} \left[ C(s, a) + \sum_{s' \in S} T(s, a, s') J(s') \right]
\]  

(2)

Optimally solving an SSP MDP means finding a policy that minimizes \( J^\pi \). Such policies are denoted \( \pi^* \), and their cost function \( J^* = J^{\pi^*} \), called the optimal cost function, is defined as \( J^* = \min_{\pi} \min_{s} J^\pi(s) \). \( J^* \) also satisfies the following condition, the Bellman equation, for all \( s \in S \):

\[
J(s) = \min_{a \in A} \left[ C(s, a) + \sum_{s' \in S} T(s, a, s') J(s') \right]
\]  

(3)

Value Iteration for SSP MDPs. The Bellman equation suggests a dynamic programming method of solving SSPs, known as Value Iteration (VI_{SSP}) [Bellman, 1957]. \( \text{VI}_{SSP} \) starts by initializing state costs with an arbitrary heuristic cost function \( \hat{J} \). Afterwards, it executes several sweeps of the state space and updates every state during every sweep by using the Bellman equation (3) as an assignment operator, the Bellman backup operator. Denoting the cost func-