Take the First Right and Go Straight Forever: Novel Planning Algorithms in Stochastic Infinite Domains

Judah Schvimer*

Brown University
judah_schvimer@brown.edu

Abstract

We study new planning algorithms for stochastic, infinite state Markov Decision Processes (MDP). Our objective is to find the policy in the MDP with the greatest probability of reaching the goal. Given a reward function we would also like the algorithm to find the policy, among those tied for greatest probability of reaching the goal, with the greatest expected total reward. We first describe an algorithm that begins at the start state and incrementally expands more states as needed. It solves a sub-MDP to find a fringe state to expand, expands that state, and then checks if it should terminate. We show that the algorithm is only guaranteed to terminate on finite domains and select infinite ones. We show that if we only care about maximizing the probability of reaching the goal, some reward functions will cause the algorithm to expand fewer nodes than others. We finally consider a modified breadth-first search algorithm. This algorithm, while still not guaranteed to terminate for most domains, is guaranteed to find a finite optimal policy in finite time without knowing when to terminate. We additionally show that neither algorithm with any reward function will minimize the expected number of steps to the goal. These algorithms show two approaches to solving stochastic, infinite MDPs to optimize first for the probability of reaching the goal, and then the expected total reward.

I. Introduction

Stochastic Markov Decision Processes (MDPs) describe general decisions problems. Standard MDP planning algorithms—such as value iteration, policy iteration, or linear programming—do not work properly on infinite state spaces. These algorithms maintain a value for every state, choosing to take actions that lead to the greatest reward. As planning algorithms iteratively sweep through every state, they update their values and propagate reward values throughout the entire state space. This process is infeasible on large and infinite domains, so slower learning algorithms are needed. It is useful to develop effective planning algorithms that do not need to look at every state, making them applicable in infinite domains. Further, even for finite state spaces, traversing only a fraction of a large state space can have enormous runtime benefits and make planning more practical. Most MDP algorithms to date have focussed on finding the goal or on maximizing reward. Our algorithm will attempt to do both, in a lexicographic order. Our goal in this paper is to develop an algorithm for this problem. We will look at how the selection of reward function impacts performance and how we can minimize the number of iterations our algorithm requires to find an optimal policy.

II. Background

We begin with a general overview of reinforcement learning. Afterwards, we look at a result that proves that planning algorithms can find a path to the goal in finite deterministic MDPs in

*A special thank you to Professor Michael Littman for advising me through my thesis.

polynomial time. We then discuss evidence that different reward functions can lead to different runtimes in practice. We also consider one study that finds that certain reward functions behave identically when trying to maximize rewards. Finally, we conclude our background with an example of a heuristic search algorithm for MDPs.

I. Reinforcement Learning: The Basics

Sutton and Barto's *Reinforcement Learning: An Introduction* [6] does an excellent job detailing the problems tackled by reinforcement learning and standard methods of attacking them. Reinforcement learning attempts to map human reward-based learning onto a computer. An agent can be in multiple states, s, in a state-space, S. For each state, s, the agent can take various actions, a, from an action space, A. The agent receives rewards for taking actions and the agent's goal is to maximize the reward from the start state to a goal state. The configuration of states, actions, and rewards is called a domain. After it concludes learning, the agent determines which action to take from each state. This mapping from states to optimal actions is the policy, π. In our stochastic models, rather than an action deterministically leading to the same state each time, each action has multiple states to which it may transition with given probabilities. This probability of going from one state to another given some action is called a transition probability. To decide on a policy, we can simply follow the actions that provide the greatest expected reward at each step. If actions can transition to multiple states, we choose between them by taking a weighted average of their rewards.

Reinforcement learning uses algorithms to determine an action policy that yields the greatest expected reward. The algorithms consider both the reward for the immediate action taken, as well as rewards for future actions given the current action. Any future rewards can be discounted, with a discount factor of γ, if they want to be valued less than current rewards. This means that if a reward of 4 were received 5 actions in the future with a discount factor of $\gamma = 0.95$, the reward accrued would be $4 * (0.95)^5$. The sum of the current reward and all future rewards yields the value of taking a given action. This means that the algorithm will not necessarily maximize the expected total reward from the start to the goal, but rather will value immediate rewards more heavily [6].

There are two main types of reinforcement learning algorithms. The first type are learning algorithms, like Q-learning. Q-learning begins in a start state and takes an action, seeing where it transition and continuing from there. Q-learning learns the value of state-action pairs, $Q(s, a)$. When deciding on a policy, it chooses the action from the current state that has the greatest Q-value. After each step, Q-learning updates the value of the previous state-action pair with the reward it received plus a discounted value for the new state. This value is the maximum value among the actions available at the new state. Let s_{t+1} be the state that s_t transitions to when taking action a_t and receiving reward $r^a_{s_t s_{t+1}}$. Let α be the learning rate and γ be the discount factor:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r^a_{s_t s_{t+1}} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

The learning rate, α, is a parameter for controlling the effect of the new information from a new state on the previous Q-value. As it gradually wanders the state space, Q-learning updates the Q-values for each state-action pair until they converge.

The second kind of algorithm is a planning algorithm, such as Value Iteration (VI). Planning algorithms need a full description of the model, whereas learning algorithms wander the state space without a full view of the model. With more information, planning algorithms can save a value for each state, and not just each state-action pair. Planning works by sweeping through every state and updating their values relative to their neighbors. When updating a state's

value, VI maximizes the value over the possible actions it can take. Let s' be the state that s transitions to with probability $P^a_{ss'}$ when taking action a and receiving reward $r^a_{ss'}$:

$$V_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

Every state's value converges over time and then VI chooses the policy (π) based on which action has the maximum expected value over all states to which it can transition:

$$\pi(s) = \text{argmax}_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

These algorithms work both for deterministic MDPs and stochastic ones, meaning that an action may transition to various states with different probabilities [6].

One of the key decisions in any reinforcement learning algorithm, besides the algorithm itself, is the reward scheme. Historically reward schemes have modeled tasks inherent to the MDP being studied. One of our emphases, as well as one of the considerations of the various authors to come, is the effect that reward schemes have on algorithm performance. The two reward schemes we consider are Goal-Reward and Action-Penalty. Goal-Reward assigns a reward of 0 to any action that does not reach the goal and a reward of 1 for any action that does reach the goal. Action-Penalty assigns a reward of -1 for any action; the negative rewards only end when the agent finally arrives at the goal and terminates. Both of these reward schemes can have a different initial Q or V value for each state-action pair or state respectively [4]. Each can also be used in conjunction with standard discounting methods. For the sake of our analysis we considered discounting as part of the reward scheme, although they are separate choices. For the remainder of this paper we will refer to Goal-Reward with Discounting as GRWD, Goal-Reward with No Discounting as GRND, Action-Penalty with Discounting as APWD, and Action-Penalty with No Discounting as APND. Assume initial Q and V values to be 0 unless stated otherwise.

II. Complexity of Deterministic RL Algorithms

Koenig and Simmons (1992)[4] make one of the first attempts to quantify the complexity of reinforcement learning algorithms. They look both at Q-learning and at Value Iteration but restrict their analysis to deterministic domains. They hypothesize that since VI is able to see the model and has more information at its disposal, it should have a lower complexity than Q-learning. Koenig and Simmons look at both goal-reward and action-penalty reward schemes. They use discounting for the goal-reward representation, and do not care whether they use it for the action-penalty representation. Their first finding is that the goal-reward representation for Q-learning leads to exponential runtime because the agent does not learn anything until it hits the goal for the first time, forced to wander the state-space aimlessly. Action-penalty representation avoids this by preserving a sense of distance from the start state to traverse the state space in an organized pattern [4].

Koenig and Simmons then turn to the action-penalty reward representation. Let $n := |S|$ and $e := \sum_{s \in S} |A(s)|$. Koenig and Simmons find that the complexity of Q-learning in the general case is $O(en)$, with bounds of $O(n^3)$ if there are no duplicate actions allowed and $O(n^2)$ if there is a linear upper action bound. They define a linear upper action bound as: "a state space topology has a linear upper action bound $b \in N_0$ iff $e \leq bn$ for all $n \in N_0$". This means that the number of actions is linear with respect to the number of states. With a linear upper action bound, the worst-case complexity becomes $O(bn^2) = O(n^2)$. Value Iteration, on the other hand, always has a worst-case complexity of $O(n^2)$. This is in line with Koenig and Simmons' hypothesis that VI would have a lower worst-case complexity than Q-learning. One

further finding is that one-initialized Q-learning with the goal-reward representation behaves identically to Q-learning with the zero-initialized action-penalty representation. Koenig and Simmons justify their claims empirically on a grid world domain [4].

III. Evidence for the Effect of Reward Functions on Runtime

Singh et al. (2010)[5] look at "intrinsically motivated" artificial systems. This is an attempt to add rewards beyond the traditional action-penalty and goal-reward norm of a uniform reward for taking a step and a uniform reward for reaching a goal. They attempt to construct rewards for intermediary tasks that incentivize a Q-learning algorithm to make decisions that it ordinarily would not. By incentivizing these intermediary tasks, the agent is able to learn to reach the goal much quicker. Singh et al. conduct two experiments that show empirically that their intrinsically motivated reward scheme improves the time it takes to learn the optimal path to the goal. Even more interestingly, by tailoring the rewards just right they produce specific and unusual decision making behavior that still remains optimal [5].

In a similar line of study, Koenig and Liu (2002)[3] perform theoretical studies on the effect of reward functions on RL algorithm performance. Koenig and Liu study how reward scheme choices for planning algorithms can guarantee avoiding traps—policy choices that never reach the goal. They first make a few conclusions about deterministic domains. They conclude in their Theorem 1 that in deterministic domains, a plan that maximizes total reward for APND, APWD, or GRWD with the same discount factor as APWD, also maximizes total reward for the other two. They show in their Theorem 3 that APND guarantees reaching the goal if it is possible, regardless of the presence of traps. Their Theorem 4, however shows that GRWD does not guarantee reaching the goal if there are traps. In addition to these initial theorems, Koenig and Liu invent a new algorithm for selectively deleting traps from an MDP, that allows all reward functions to guarantee reaching the goal once the traps are removed [3].

Their most important finding for us, Theorem 9, shows that for any MDP, deterministic or stochastic, "A plan that maximizes the expected total reward for the action-penalty representation with discounting also maximizes the expected total reward for the goal-reward representation with the same discount factor, and vice versa." This means that whenever APWD or GRWD make a decision based on maximizing expected total reward given the same knowledge of the same MDP, they will make the same decision. This comes from the fact that their expected total rewards are linear transformations of one another so maximizing one requires the same parameters as maximizing the other [3].

IV. Heuristic Search Algorithms for MDPs

Hansen and Zilberstein (1998)[2] find a heuristic search algorithm that allows them to more quickly search a stochastic, cyclic graph for an optimal path. The most famous heuristic search algorithm is A* search for finding a simple path in a standard deterministic graph. Hansen and Zilberstein create a novel algorithm, LAO*, that works on graphs as complex as those that we are studying. LAO* iterates through two steps, a forward search step and a dynamic programming step. The forward search step expands the current set of states by expanding some fringe state and adding its neighbors to the current set of states. The dynamic programming step then updates the values of each state with Policy Iteration or Value Iteration, two alternative planning algorithms [2]. This two step process forms the bedrock for the first of our algorithms below.

III. New Planning Algorithm

I. Objective

Our objective is to create a new planning algorithm that finds the optimal policy on both deterministic and stochastic domains of any—even infinite—size. We will define the best policy as the policy that in lexicographic order:

(1) Maximizes the probability of reaching the goal from the start state

(2) Maximizes the expected total reward

We will consider two similar problems. In the first problem, we are given a specific reward function for the second criterion. In the second problem, we are just asked to find that policy that maximizes the probability of reaching the goal, freeing us to use any reward function of our choice. This second problem is what Koenig and Liu (2002)[3] refer to as a "Goal Directed Markov Decision Process." The algorithm needs to find the goal, and only uses a reward function as a means of doing so [3]. Note that the second problem is just a comparison of the first problem across multiple reward functions.

Algorithm 1 should first and foremost guarantee returning the best policy as defined above. Secondly, it should expand as few states as possible. The algorithm's goal is to find the entire optimal policy, however a short circuiting step may be used to only find the optimal first action but terminate earlier. Whenever the algorithm makes an arbitrary choice, it should be consistent regardless of the reward function used. If not, some of our conclusions will not hold.

Algorithm 1 Probabilistic Heuristic Search

Input: Markov Decision Process
Output: Best Policy

1: **Set Policy:** Choose the policy with the greatest probability of reaching the goal, assuming optimistically that unexplored states are goal states
2: - If there is a tie, choose the policy with the greatest expected total reward
3: - If there is still a tie, choose the policy arbitrarily, though consistently
4: **Short Circuiting (Optional):** If the policy's pessimistic estimate for the probability of reaching the goal is better than the best optimistic estimate from a different first action, go to Step 9 and return only the optimal first action
5: **Termination:** If there are no more fringe states in the current policy, go to Step 9, otherwise return to Step 1
6: **Choose Expansion State:** Among all fringe states, choose the one reached with the greatest probability
7: - If there is a tie, choose one state arbitrarily, though consistently
8: Expand the chosen state and go to Step 1
9: **Policy Choice:** Return the last expanded policy

II. Algorithm Details

Terminology

This algorithm is similar to other envelope-based algorithms, such as those described in Dean et al. (1995)[1]. An envelope is the set of states that are visible to the algorithm at the current

moment. The fringe states of the algorithms are those states on the edge of the envelope. Envelope-based algorithms constrain an MDP to a smaller envelope and widen the envelope as they expand fringe states. The fringe states join the envelope, and the states they neighbor are added to the set of fringe states[1]. Our algorithm knows that the fringe states exist, but does not know anything about the fringe states—not even if they are the goal—until it expands them and adds them to the envelope.

Throughout the paper we will refer to optimistic and pessimistic estimates for both the probability of reaching the goal and the expected total reward. For the probability of reaching the goal, an optimistic estimate is an estimate calculated when treating any unexplored fringe state as if it is the goal. This means that exploring fringe states can only decrease the estimate, and thus the estimate can only decrease with further iterations of the algorithm. One consequence of this is that optimistic estimates are always greater than the actual probability of reaching the goal. Another result is that optimistic estimates remain at the actual probability of reaching the goal once the entire policy is expanded. A pessimistic estimate for the probability of reaching the goal is calculated by assuming unexplored fringe states never reach the goal. Opposite of the optimistic estimate, the pessimistic estimate can only increase with further state expansions and thus is always less than the actual probability of reaching the goal. Similarly to the optimistic estimate, once the policy is fully expanded this estimate will equal the actual probability of reaching the goal.

Estimates for the expected total reward are similar to those for the probability of reaching the goal. For our optimistic estimates, we are using an admissible reward-values for the reward at a fringe state. Admissible heuristics never overestimate the cost of reaching the goal. This means that the goal reward-value assigned to a fringe state is never lower than the reward that will actually be received for transitioning to the fringe state. An optimistic estimate assumes fringe states are goal states and thus, gives them the maximum reward possible. A pessimistic estimate assumes fringe states never reach the goal and gives them the minimum reward possible. These estimates have the same properties as the estimates for the probability of reaching the gaol regarding increasing and decreasing as the algorithm expands more states.

One last piece of notation is the "..." appearing in many of the diagrams. The "..." implies that the pattern seen so far continues *ad infinitum*. It is important to remember that this means there is no goal at the end of the path, though there may be goals along the way if that is part of the pattern.

Explanation of Step 1: the Set Policy Step

Steps 1-3 of Algorithm 1 choose the policy with the greatest optimistic estimate for the probability of reaching the goal. Ties are broken by first choosing the policy with the greatest expected total reward. If there remains a tie the algorithm chooses an arbitrary policy consistently. If you only care to find the policy with the greatest probability of reaching the goal, ties can be broken purely arbitrarily with no worry about a reward function. As we will show, however, this may increase the number of states that must be expanded.

Lemma 1 *The greatest expected total reward for an MDP with GRND is equal to the greatest probability of reaching the goal for the same MDP, and the policy that yields the greatest expected total reward is the same policy that has the greatest probability of reaching the goal.*

Proof: We will show that the greatest expected total reward for an MDP with GRND is equal to the greatest probability of reaching the goal with induction. Note that for GRND, the value of a state is equal to the maximum expected total reward for a policy starting at that

Algorithm 2 Set Policy

Input: Envelope of the Markov Decision Process
Input: Reward Function
Output: Policy with the highest probability of reaching the goal

1: Mark the fringe states of the envelope as goal states. Use GRND to assign each state its maximum expected reward. Interpret a state's maximum expected reward as its maximum probability of reaching the goal.
2: If there are multiple policies that reach the goal with the same maximal reward/probability, move on to Step 3 to break the tie, otherwise return the maximal policy.
3: Create an MDP of these tied policies, and use the input reward function to find the policy with the greatest expected reward.
4: If there are multiple policies with the same expected reward, choose a policy arbitrarily in a consistent manner.

state. Let $PG(s)$ be the probability that a policy starting at s reaches the goal. The base case is that we are at the goal, or a fringe state that we are optimistically considering a goal. The probability of reaching the goal, $PG(G) = 1$. GRND assigns a reward of 1 to the goal, so the base case is satisfied. Assume the inductive hypothesis, that for each state, k, between state s and the goal, the greatest expected total reward with GRND, $v(k)$ is equal to the maximum probability that a policy starting at the state reaches the goal, $PG(k)$. Let the action set be $A(s)$, the state space be S, and the transition function be $P(s'|s,a)$. State s is assigned a value of $v(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) * (r(s,a,s') + \gamma * v(s'))$. This simplifies to:

$$v(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) * v(s')$$

when accounting for the restraints of GRND. The maximum probability that a policy starting at state s reaches the goal is equal to:

$$PG(s) = \max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) * PG(s')$$

By our inductive hypothesis, $v(s') = PG(s')$. Thus, $v(s) = PG(s)$. We have shown by induction that the value GRND assigns to any given state is equal to the maximum probability of reaching the goal from that state. Thus, the value of the start state, the greatest expected total reward for the MDP, is equal to the greatest probability of reaching the goal from the start. Similarly, since the value of every state in the MDP is equal to the greatest probability of reaching the goal from that state, the policy that maximizes expected total reward is the same policy that maximizes the probability of reaching the goal. □

Theorem 2 *The Set Policy algorithm finds the policy on the current envelope with the greatest optimistic estimate for the probability of reaching the goal, followed by breaking ties as stated.*

Proof: The algorithm first needs to find a policy with the greatest probability of reaching the goal. Step 1 does this by solving the envelope using GRND, treating the fringe states as goal states. We are trying to find an optimistic estimate, so we assume any fringe state that hasn't been explored could reach the goal. If it has not been expanded then there is a chance it is the goal. If it has been expanded and isn't the goal, it obviously isn't the goal. The algorithm only marks that a fringe state is not the goal once it is confirmed, so it never underestimates the probability of reaching the goal.

From Lemma 1, the policy that maximizes expected total reward with GRND maximizes the probability of reaching the goal. Based on this lemma, planning with GRND finds all policies that are tied for the greatest probability of reaching the goal. Since no policy underestimates the probability of reaching the goal, it is impossible for any policy to increase its probability of reaching the goal and thus we have found the set of policies with the greatest possible probability of reaching the goal.

By following the policies that give us these maximal probabilities, we get a new MDP of the policies that all yield the maximum probability of reaching the goal. This MDP is just a subgraph of the original input envelope; it merely restricts the actions we can take from any given state but does not change any outcomes. Because removing actions only removes suboptimal policy choices, any policy chosen from this new, restricted MDP will have the same maximal probability of reaching the goal as the envelope. To break the tie, we need to find the policy in this new MDP that maximizes expected total reward. This is easy since we can just solve this MDP with the input reward function. We can then choose any policy arbitrarily from those that are tied for the greatest expected total reward. □

Explanation of Step 4: the Short Circuit Step

Step 4 short circuits the algorithm, allowing it to avoid expanding every state if it is unnecessary. Short circuiting comes at the price of discovering the entire optimal policy. If the algorithm short circuits it can only be confident in the optimality of the first action. We can short circuit whenever the current policy's pessimistic estimate for the probability of reaching the goal is greater than or equal to the optimistic estimate of any policy with a different first step. This means that the actual probability of reaching the goal for a policy starting with a different first action can never be as great as the actual probability of the current policy.

To see if short circuiting is possible, first find the current policy's pessimistic estimate for the probability of reaching the goal. To calculate it, follow the possible actions assuming fringe states have a 0 probability of reaching the goal. Now, remove the policy's first action from the envelope and solve it with GRND, assuming fringe states are goal states. This will find the greatest optimistic estimate of any policy with a different first action. If the pessimistic estimate is greater than the optimistic estimate, terminate and return the first action of the pessimistic estimate. As stated in the pseudocode, this step is optional. As a result we will not be assuming it in the proofs to follow, though it can improve the runtime and allow the algorithm to terminate on domains where it otherwise would not.

Explanation of Step 8: the State Expansion Step

Step 8 is the state expansion step. At this point we have chosen a state and wish to expand it. Planning requires full knowledge of the model. We use that full knowledge to see the possible actions and to what states those actions transition. Each action can transition to multiple states with various probabilities that are known to the decision maker. If these new states have not already been expanded, they now become fringe states. The expanded state now becomes part of the envelope. The algorithm discovers if a state is actually a goal state when the state is expanded, not when it first becomes a fringe state.

Explanation of Step 5: the Termination Step

We terminate if there are no more fringe states in the current policy. The current policy has both the greatest optimistic estimate for the probability of reaching the goal, and if tied the

greatest optimistic estimate for expected total reward. If there are no more fringe states then its estimate will never decrease for either quantity. As explained above, optimistic estimates can never increase with the expansion of more states, so no other policy's estimate can increase from its current suboptimal estimate. If the best estimate will never decrease and the other ones will never increase, then no other optimistic estimate can ever surpass the best one. Thus, at this point we may terminate.

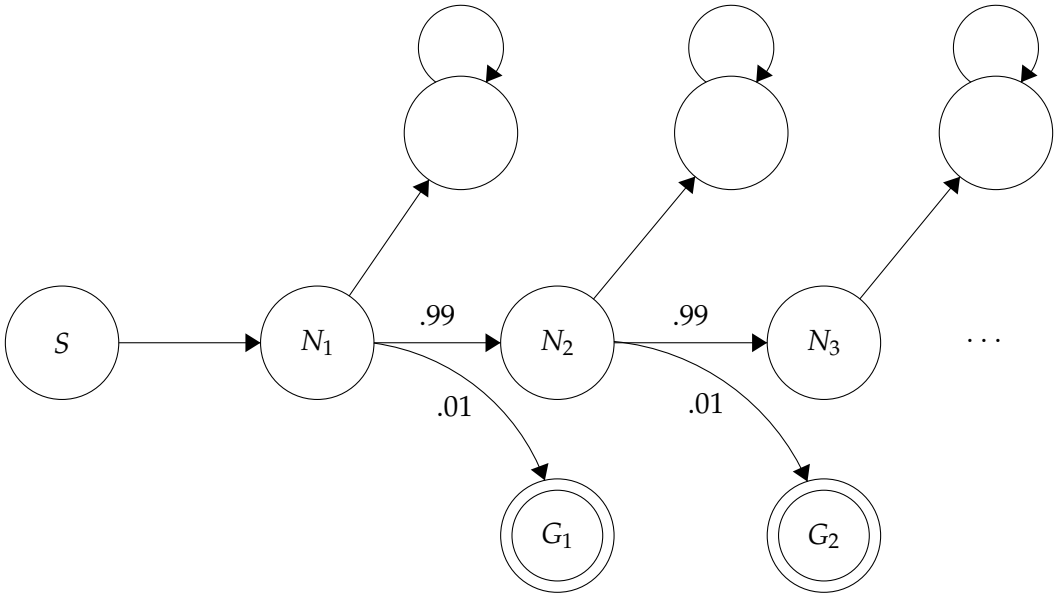
III. Termination

Theorem 3 *Algorithm 1 terminates on a finite MDP.*

Proof: The MDP has a finite number of states. With each iteration the algorithm increments the number of expanded states by 1. These two statements mean that eventually the algorithm will expand every single state. For the entire algorithm to terminate, the Set Policy Step just needs to terminate. VI is guaranteed to terminate, so if we use VI for the Set Policy Step then the algorithm is guaranteed to terminate. □

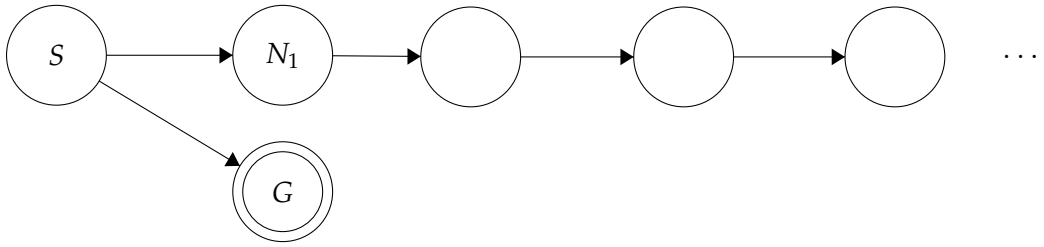
We will now use a series of counterexamples to show why certain types of infinite MDPs are not guaranteed to terminate. We will conclude with a proof that certain infinite MDPs are guaranteed to terminate. Note that we focus on problems with a finite set of actions. Searching for the best policy with an infinite set of actions is highly dependent on domain assumptions and out of the scope of this paper.

Counterexample 4 *If the optimal policy is infinite, Algorithm 1 is not guaranteed to terminate.*



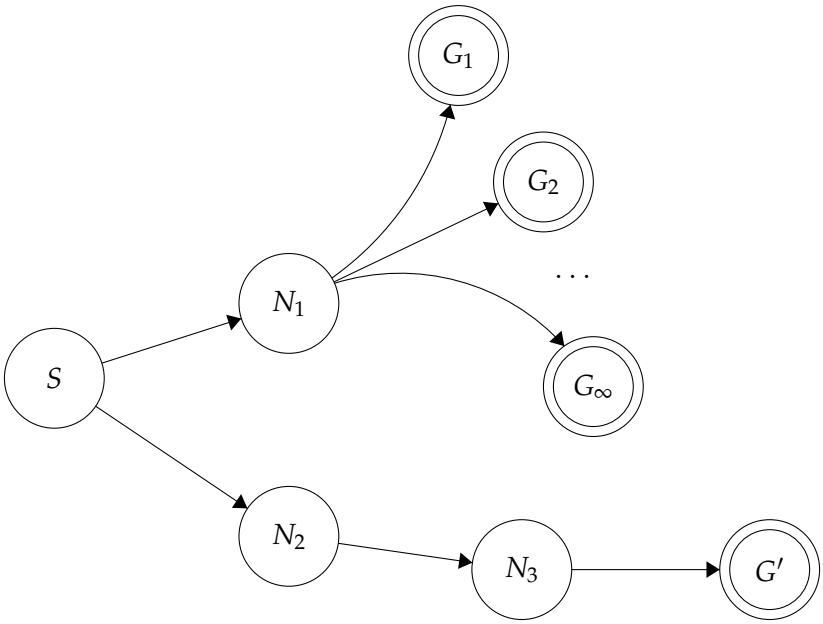
This counterexample shows that it is possible to construct an infinite MDP that does not terminate where each state has a finite number of transitions. Obviously none of the upward actions are optimal, but as long as there are still decisions to be made the algorithm will need to keep running.

Counterexample 5 *In any MDP with infinite states, Algorithm 1 is not guaranteed to terminate using GRND.*



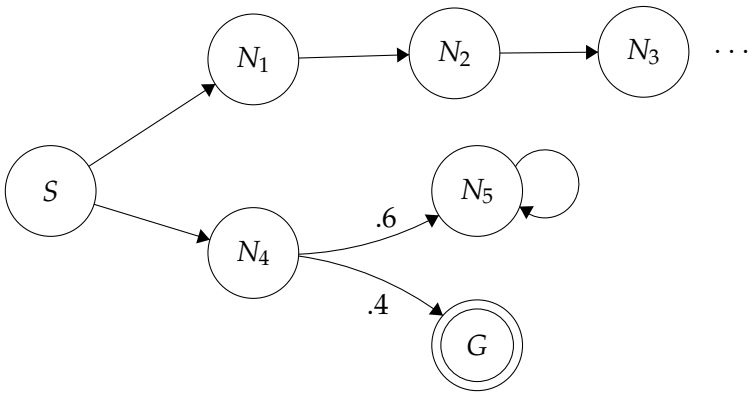
This an infinite MDP that is not guaranteed to terminate if the algorithm uses GRND to break ties among the policies with the greatest estimate for the probability of reaching the goal. Any reward scheme that captures a sense of distance from the start will find the optimal policy in finite time. GRND could traverse N_1's policy for infinite expansions without ever expanding G. Once it expands G, the algorithm will know its policy is optimal and terminate immediately.

Counterexample 6 *If any action can transition to an infinite number of states, Algorithm 1 is not guaranteed to terminate.*



This counterexample shows that it is possible to construct a domain with a finite optimal policy but an infinite number of states that an action can transition to that does not terminate. Let G_1 through G_∞ be an infinite number of goal states whose transition probabilities sum to 1. In the worst-case, the algorithm expands S, N_1, N_2, N_3, G_1, G_2, etc... For many reward schemes, G' has a lower expected total reward than G_1-G_∞ so the algorithm will not be expand it until they all are. Thus, it will continue to expand ∞ goal states and never terminate. N_1 is actually the best policy, so switching to expand G' would be incorrect anyway.

Counterexample 7 *If the domain is infinite and the greatest probability of reaching the goal is less than 1, Algorithm 1 does not necessarily terminate.*



This counterexample shows that if the greatest probability of reaching the goal is less than 1, the algorithm may not terminate. The top policy has an optimistic estimate for the probability of reaching the goal which is greater than the probability of reaching the goal of the optimal policy. The algorithm expands N_5 before G since N_5 is the more likely state to reach on the bottom policy. At that point the algorithm only expands states in the top policy going to N_1 until its optimistic estimate for the probability of reaching the goal falls below 0.4. Since the top policy is infinite, the top policy's optimistic estimate for the probability of reaching the goal will never fall below 1 and the algorithm will never expand G. It may seem like a contradiction that a suboptimal policy's estimate would be greater forever, however if the policy does not actually reach the goal then it is still suboptimal. Since the top policy is not actually optimal, and the top policy does not terminate, the algorithm will not terminate.

This counterexample shows that there is a very specific set of criteria that must be met for the algorithm to terminate. If the algorithm does not terminate, then it is not very useful. We will address this shortcoming later in the paper with an alternative algorithm. That algorithm will show that this termination guarantee is impossible because of this exact counterexample, but that it is possible to make a slightly looser guarantee that is almost as good as termination.

Theorem 8 *Algorithm 1 will terminate in finite time if:*

1. *the optimal policy is finite,*
2. *every state has a finite number of actions,*
3. *every action transitions to a finite number of states,*
4. *the greatest probability of reaching the goal is 1,*
5. *and the reward scheme causes states within a finite number of steps of the start to have greater values than states an infinite number of steps away from the start.*

Proof: Assume not. Then, there must be some series of an infinite number of state expansions that never meets either terminating condition, Step 4 or 5. It is sufficient to find a contradiction based on the premise that the algorithm does not terminate by the standard termination step. If the algorithm does not reach the standard termination step, Step 5, then there is always another fringe state whose probability of reaching the goal is at least as great and if tied, whose expected total reward is at least as great. Let the optimal policy be P_O and n

be the number of states in P_O, since the optimal policy is finite. The algorithm cannot have expanded n states from the optimal policy or else it would terminate. Thus, the algorithm must choose other policies to expand an infinite number of times before choosing the optimal policy n times.

The algorithm chooses policies first by the probability of reaching the goal. Since P_O is the optimal policy, it must eventually have the greatest probability of reaching the goal, or at least be tied for the greatest probability of reaching the goal and also have a greater than or equal expected total reward. Recall that every state has a finite number of actions and every action transitions to a finite number of states. Thus, every state can transition to only a finite number of states.

Since we are expanding a infinite number of states, the algorithm must expand states that are infinitely far from the start state. The optimal policy is finite and its probability of reaching the goal is 1, so every state on the optimal policy will be a finite number of steps from the start. Our reward scheme causes states within a finite number of steps of the start to have greater values than states an infinite number of steps away from the start, so all states on the optimal policy will have a greater value than states infinitely far away. Thus, since the algorithm never chooses to expand the states on the optimal policy, the states not on the optimal policy must have a strictly greater probability of reaching the goal. However, we said that the greatest probability of reaching the goal, and thus that of the optimal policy, is 1. This is a contradiction since it is impossible to have a probability strictly greater than 1, so the algorithm must terminate. □

IV. Correctness

Theorem 9 *If it terminates and the reward scheme has an admissible goal state reward, Algorithm 1 finds the policy that lexicographically maximizes the probability of getting to the goal from the start state, and then maximizes the expected total reward for the given policy.*

Proof: To find the best policy, two criteria must be satisfied in lexicographic order. First and foremost the policy must maximize the probability of reaching the goal from the start. We will start by considering termination via the standard termination step, Step 5, not using short circuiting. Consider the iteration of the algorithm when it expands the final fringe state and terminates. The algorithm chooses a state from the policy with the greatest probability of reaching the goal, assuming all unexplored states are goal states. This is an optimistic choice, and thus there is no way any policy could have a higher probability of reaching the goal, no matter how many other states we expand. Thus, it is safe to conclude that this policy, and those tied with it, are the policies with the greatest probability of reaching the goal.

We now need to prove that among those policies with the maximum probability of reaching the goal state, we choose the policy that maximizes the expected total reward for the policy. In the Set Policy Step, we break ties by choosing the policy with the greatest expected total reward. Our expectation for total reward is also an optimistic estimate since we use reward schemes with admissible goal state rewards. Thus, no other policy's expected total reward can ever surpass the expected total reward of the chosen policy. The algorithm terminates if there are no more fringe states in the current policy. This means that the entire policy has been expanded and the current policy has both the greatest probability of reaching the goal and the greatest expected total reward. This is no longer just an optimistic estimate, but the actual expectation for the policy since the entire policy has been expanded. Thus, since no other policy can improve their expected total reward due to their optimistic nature, no policy's

expected total reward can ever surpass the current policy.

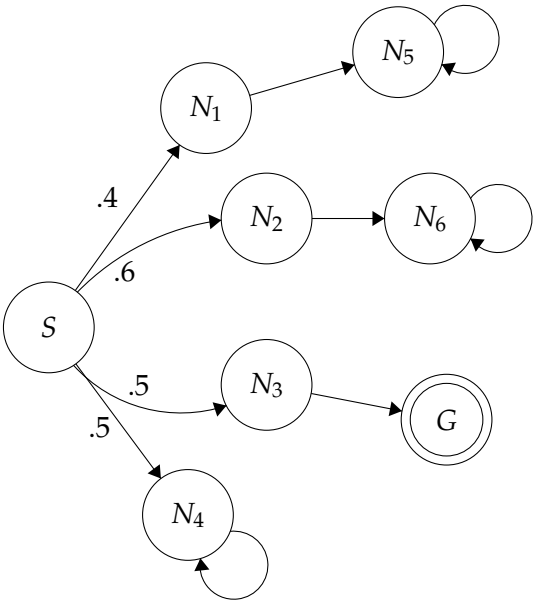
If the short circuiting termination step is taken, then we know that there is only one policy with the greatest probability of reaching the goal. Thus, this is certainly the optimal first step and the algorithm makes the correct decision. □

IV. Complexity Guarantees

We will now look at what guarantees, if any, we can make regarding which states Algorithm 1 expands. This section will discuss guarantees we can make regardless of what reward function we are told or we choose to use. Note that we will not consider short circuiting performance. Short circuiting will often lead to better performance, but that is not the focus of our study. Many of the proofs below will refer to the algorithm's "worst-case performance." Algorithm 1 makes arbitrary choices in Steps 2 and 7. Worst-case means the algorithm makes the choice among these arbitrary choices that leads to the greatest number of state expansions.

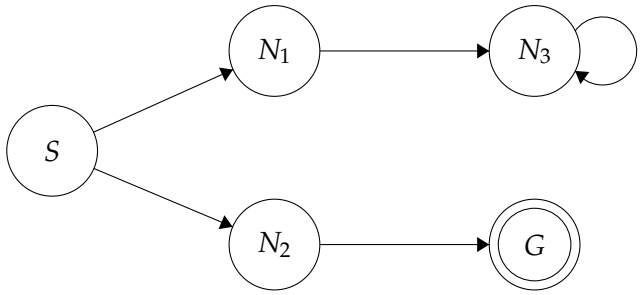
Conducting this study, we made multiple attempts to quantify or even qualify how many states the algorithm expands in its worst-case. We quickly realized that quantifying the number would not work because it fluctuates wildly depending on the domain at hand. Rather, we tried to tag each state with some quantity related to its probability and compare each tag to a universal threshold. The algorithm may expand states whose tags are better than the threshold and will not expand states whose tags are worse than the threshold. As we will see, some reward functions lead to better performance than others in their worst-case, so we are looking for a maximum number of states expanded. If a tag says the algorithm might expand a state it is okay for the algorithm to not actually expand it. But if a tag says the algorithm will not expand a state then the algorithm should definitely not expand it. A tagging scheme does not work if a state's tag says that algorithm will not expand the state, but the algorithm does it expand it in a valid execution.

The first tag to try is: "tag states with the maximum over all policies that reach the state of the probability of reaching the state from the start." We compare each tag to the lowest tag of a goal state. If it is higher the algorithm might expand it and if the tag is lower then the algorithm does not.



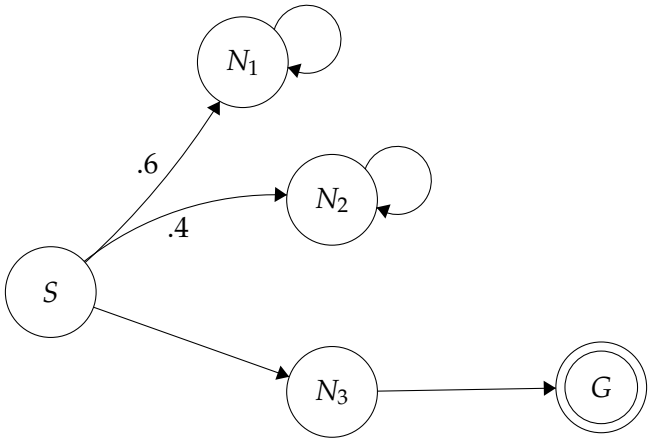
In this counterexample, the algorithm first expands S, N_1, N_2, N_3, and N_4. After N_4 is expanded, the probability of reaching the goal for G's policy reduces to 0.5, compared to the probability of reaching the goal for the top policy which is still estimated to be 1. Only after N_6 is expanded, will the estimate for the probability of reaching the goal for the top policy fall below 0.5 and will the algorithm expand G . N_1's tag is 0.4, while G's tag is 0.5. Thus, the algorithm expands N_1 but the tagging scheme doesn't think it should.

The second tag to try is: "tag states with the maximum over all policies that reach the state of the probability that the policy reaches the goal." We compare the tags to the greatest probability of reaching the goal from the start.



This counterexample shows the tag does not work. The greatest probability of reaching the goal is 1 and the tag for N_1 is 0, so N_1 should not be expanded. In the worst-case, however, Algorithm 1 will expand N_1 because its estimated probability of reaching the goal is 1 until N_3 is expanded.

The third tag to try is: "tag states with the maximum probability over all policies that reach the state that the policy hits the state or a goal." We also compare these tags to the greatest probability of reaching the goal from the start.



The algorithm first expands S, N_3 and either N_1 or N_2. Once it expands N_1 or N_2 it won't expand the other since that policy will then have a lower probability of reaching the goal than N_3's policy. N_1 and N_2's tags are 0.6 and 0.4 respectively, but the highest probability of reaching the goal is 1. Thus, even though the tag is lower, the algorithm still expands state N_1 or N_2.

I. The Algorithm Only Expands Useful States

We will now show that our algorithm is not wasteful and only expands states that could be part of an optimal policy.

Theorem 10 *If the reward scheme has an admissible reward for reaching the goal, then at the moment of every expansion, the best policy could contain the state that Algorithm 1 expands.*

Let s be the current state being expanded and let P be the policy from which s was chosen. We know P has the greatest estimated probability of reaching the goal because it was chosen by the algorithm.

Consider the case where there is no tie and every other policy has a probability of reaching the goal strictly less than P. No other policy's probability of reaching the goal can ever surpass P's current estimation because their optimistic estimates are lower. If each of P's fringe states that we are optimistically considering a goal is actually a goal state, then P's probability will remain the same and P will remain the best policy. Thus, there is a chance P is the best policy and we must expand s.

The other case is where there is a tie and multiple policies have the same probability of reaching the goal. In that case, P's expected total reward is greater than or equal to every other tied policy's. Since the reward scheme is admissible, no policy's total expected reward will ever surpass P's current expected total reward. If every one of P's fringe states are actually goal states, then P's expected total reward will remain at its optimistic estimate, and no other policy could ever surpass it. Thus, there is a chance P is the best policy (or tied for the best policy), and we must expand s. □

V. Comparison of Reward Functions

We have already shown that our choice of reward scheme can determine whether or not Algorithm 1 terminates, and even if it's correct. If we are given a reward function to optimize, we obviously must choose that one. However, if we are given a Goal Directed Markov Decision Process with the choice of which reward function to use, we will see below that not all reward functions are created equal, and some may improve the algorithm's performance. We will only consider reward schemes with admissible goal-state rewards, so correctness is guaranteed. We will, however, consider GRND which does not cause states finitely close to the start to have lesser values than states infinitely far from the start, so termination for GRND is not guaranteed, as shown in Counterexample 5. GRND, however, is a commonly used reward scheme and it is worth showing how its performance compares to the others.

We will now show how our choice of reward scheme affects the runtime of the algorithm, specifically in terms of how many states it causes the algorithm to expand. We prove that under certain conditions GRWD and APWD are equivalent, and that they, as well as APND, perform better than GRND. GRWD and APWD are equivalent because of an important result discovered by Koenig and Liu (2002).

I. GRWD is equivalent to APWD

Theorem 11 *Algorithm 1 expands the same states using GRWD as it does using APWD.*

Proof: Theorem 9 of Koenig and Liu (2002), states that: "A plan that maximizes the expected total reward for the action-penalty representation with discounting also maximizes the expected total reward for the goal-reward representation with the same discount factor, and vice versa" [3]. The algorithm chooses what policy it expands first based on the probability of reaching the goal. The reward scheme chosen does not affect this decisions so both reward schemes will pick the same state. If there is a tie, the algorithm breaks it based on which policy has the greatest expected total reward. Theorem 9 of Koenig and Liu (2002) says that

the same plan will maximize the expected total reward for both reward schemes, so they will both break the tie in the same way. If there is still a tie the policy to expand is chosen arbitrarily. As long as this is not dependent on reward and done consistently—as the algorithm instructs—they will make the same choice. The state chosen in the given policy is chosen first based on likelihood of being reached and then arbitrarily, neither of which is reward based. In either case, the algorithm makes the same choice with both reward schemes. Thus, APWD and GRWD always choose the same states to expand. □

II. GRWD, APND, and APWD perform equal to or better than GRND

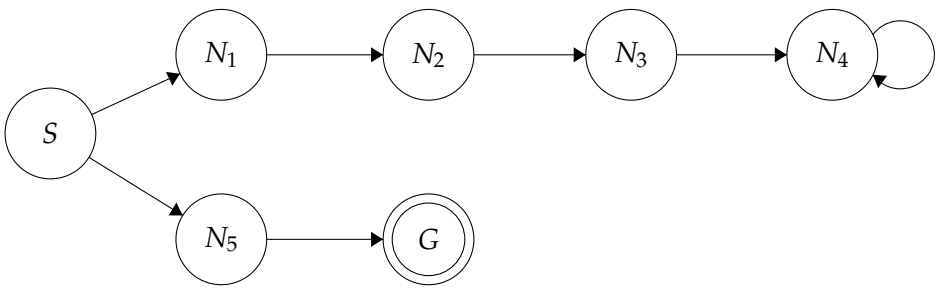
To show this, we will prove the following even more general theorem:

Theorem 12 *The worst-case performance for Algorithm 1 with any reward scheme is equal to or better than the worst-case performance of GRND.*

Proof: Let R be an arbitrary reward scheme. Assume not. Then, there must be a domain, D, where GRND's worst-case performance is better than R's worst-case performance. Let the list of expanded states in R be n_1, n_2, n_3, ... n_n and the list of expanded states in the GRND be n'_1, n'_2, n'_3, ... n'_m, for $m < n$. These lists are identical up to some state $n_j = n'_j$ when the two algorithms expand their first pair of different states. When both algorithms decide to expand different states they have the same probability distribution over each state for reaching the goal but still choose different states. They cannot choose the same policy because the state chosen within a given policy is independent of the reward scheme. Both reward schemes would then choose the same state to expand.

The only other option is that they choose a different policy to expand. They first must choose a policy that maximizes the probability of reaching the goal. The reward scheme does not effect the probability of reaching the goal, so the tie breaker must choose a different policy for each. The first tie breaker is to choose the policy with the greatest expected total reward. R will choose some policy to expand based on its reward scheme, and if there is a tie it will choose arbitrarily among them. The tie breaker for GRND, however, does not reduce the number of possible best policies. From Lemma 1 we know that maximizing the expected total reward for an MDP with GRND is equal to finding the maximum probability of reaching the goal. Each policy has the same probability of reaching the goal, so they will have the same expected total reward. Expected total reward is the tie breaker though, so this tie breaker will still leave all of the policies in a tie. GRND will then go to the next tie breaker, which is arbitrary. But GRND could arbitrarily choose the policy R chooses. GRND could then choose the same sequence of states to expand as R and perform identically. We assumed R performed worse than GRND, and GRND could perform equally to R, so this is not GRND's worst-case performance, which is a contradiction. Thus, GRND's worst-case performance can never be better than R's worst-case performance.

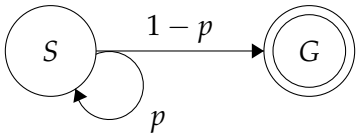
Finally, we must show that R can perform better in the worst-case than GRND. Consider the following domain:



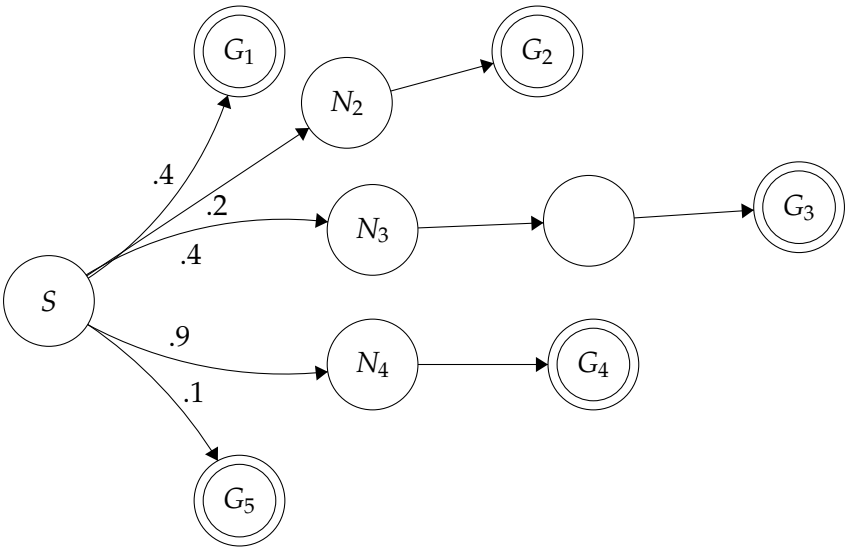
GRND at its worst-case performance will expand S, N_1, N_2, N_3, N_4, and then finally N_5 and G, which is 7 states in total. This is because the expected total reward is always 0 until you hit a goal state. R will at worst expand every state, performing equally to GRND, but may not have to depending on what reward scheme is used. For example, APND will only expand S, N_1, N_2, N_5, and G. Thus, R can perform better in its worst-case than GRND, and GRND will never perform better in its worst-case. Alternatively, R will always perform equal to or better than GRND in the worst-case. □

III. GRWD compared to APND

Comparing GRWD and APWD was easy because they were linear transformations of each other. The comparison between GRWD and APND is complicated by the fact that GRWD has a discount factor and APND does not. Theorem 9 of Koenig and Liu (2002) is proven by showing that the expected total reward for GRWD is γ^i and the expected total reward for APWD is $-\sum_{j=0}^{i-1}\gamma^i = \frac{\gamma^i-1}{1-\gamma}$ [3]. These two quantities are linear transformations of each other. Looking at a simple example will elucidate why GRWD and APND are not linear transformations in the same way:

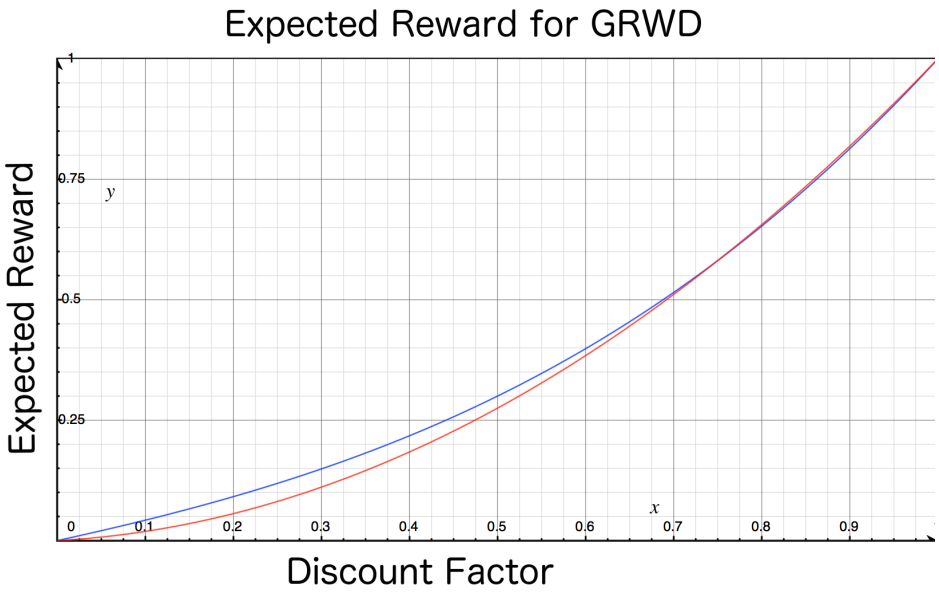


Here the expected total reward for GRWD is $\frac{1-p}{1-p\gamma}$ and the expected total reward for APND is $\frac{-1}{1-p}$. These are clearly not linear transformations of each other. To make it more concrete, consider the following domain:



Let P_1 be the top policy going to G_1, N_2, or N_3; and P_2 be the bottom policy leading to N_4 or G_5. Policy P_1 terminates in 1 step with probability 0.4, 2 steps with probability 0.2, and 3 steps with probability 0.4. Policy P_2 terminates in 1 step with probability 0.1, 2 steps with probability 0.9, and 3 steps with probability 0. Step 2 of Algorithm 1 chooses to expand the policy with the greatest expected total reward among policies tied for the greatest probability of reaching the goal. Using APND yields an expected total reward of -2 for P_1 and -1.9 for P_2. Thus, APND always chooses to expand P_2. GRWD yields a different expected reward for every discount factor, γ. When $\gamma < 0.75$, GRWD chooses P_1; and when $\gamma > 0.75$, GRWD chooses

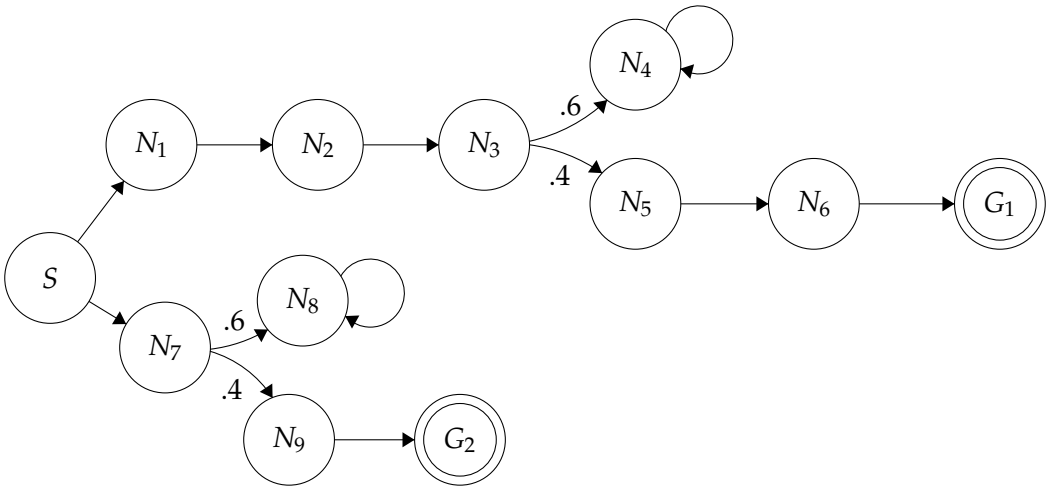
P_2. Below is the graph of the rewards for the policies in the above example. The algorithm chooses to expand whichever policy's reward line is higher:



GRWD's reward: Blue is P_1, Red is P_2

This graph confirms that there is a shift at 0.75 from choosing P_1 to P_2. Since APND always chooses P_2, when $\gamma < 0.75$ each reward schemes chooses to expand a state from a different policy. Making a different decision on an early iteration could lead to a radically different number of expansions later in the algorithm execution. Even with the same reward function, different values of γ could require the algorithm to expand a different number of states. Note this is different than when we compared GRND to APND, because in that case when the two schemes made different choices, GRND's choice was completely arbitrary, here it is not.

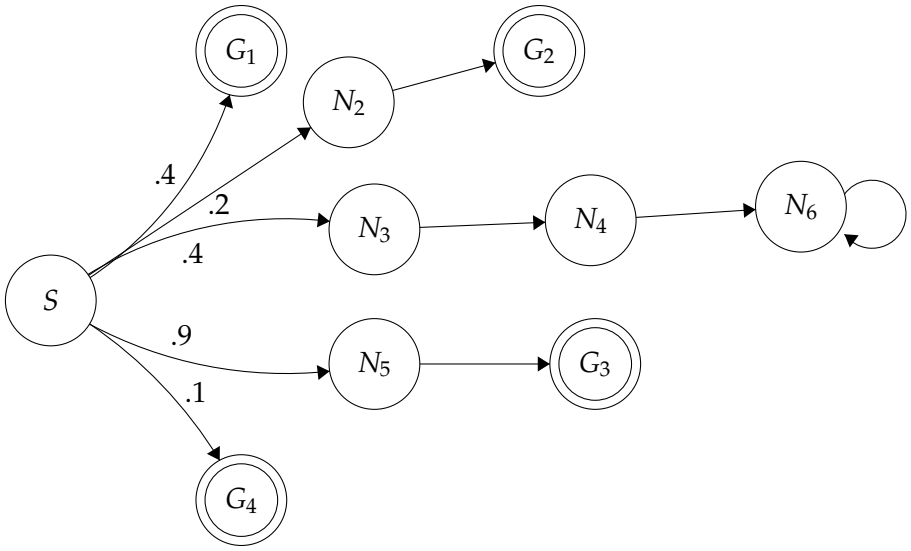
We will now show an example when GRWD performs better than APND in the worst-case and an example of where APND performs better in the worst-case. First is an example of when GRWD performs better:



Both reward functions start by expanding S, N_1, N_7, N_2, N_8, N_3, N_4 since they all have the same probability of reaching the goal. This order chooses the greatest expected reward each time, breaking ties arbitrarily. N_4 and N_8 must be chosen over N_5 and N_9 since they

are more probable nodes. Then the algorithm must decide which to expand first: N_5 or N_9. APND in the worst-case chooses the top policy with N_5; GRWD in the worst-case chooses the bottom policy with N_9. With APND, both policies have a reward of $-\infty$, so in the worst-case it arbitrarily chooses to expand every node except for G_2. GRWD, with a discount factor of 0.9, gives the top policy a reward of 0.3645 and the bottom one a reward of 0.45. Since the bottom policy has a greater reward, GRWD will choose the expand the bottom policy and expand N_9. It will then expand G_2 since it has a reward of 0.405 which is still greater than 0.3645. Thus, GRWD does not expand N_5, N_6, or G_1, overall expanding 2 fewer states than APND.

Below is an example of when APND outperforms GRWD, similar to that from before:



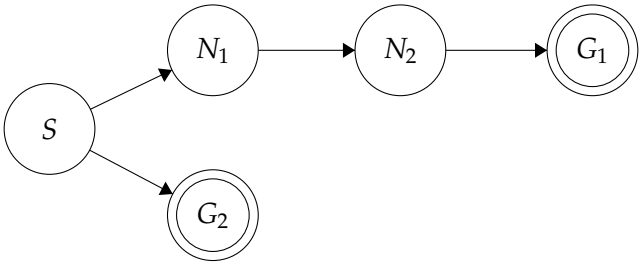
Consider GRWD with a discount factor of $\gamma = 0.2$. Both reward functions will tell the algorithm to first expand S, G_1, N_2, N_3, N_5, G_4. At that point the decision becomes which of G_2, N_4, and G_3 to expand. APND calculates an expected reward of -1.6 for the top policy and an expected reward of -1.9 for the bottom one. Due to its greater reward, APND will expand N_4, decreasing the top policy's expected reward to -2. APND will finally expand G_3 and terminate. GRWD, however, calculates an expected reward of 0.104 for the top policy and an expected reward of 0.056 for the bottom policy. It then expands N_4, reducing the top policy's expected reward 0.0912. It still wants to expand the top policy so it expands N_6 (because that's the most probable node) before noticing it must expand G_3 to find the optimal policy. Thus, GRWD expands one more state than APND.

This shows that different domains lead different reward schemes to perform best in the worst-case.

VI. Minimizing Expected Number of Steps

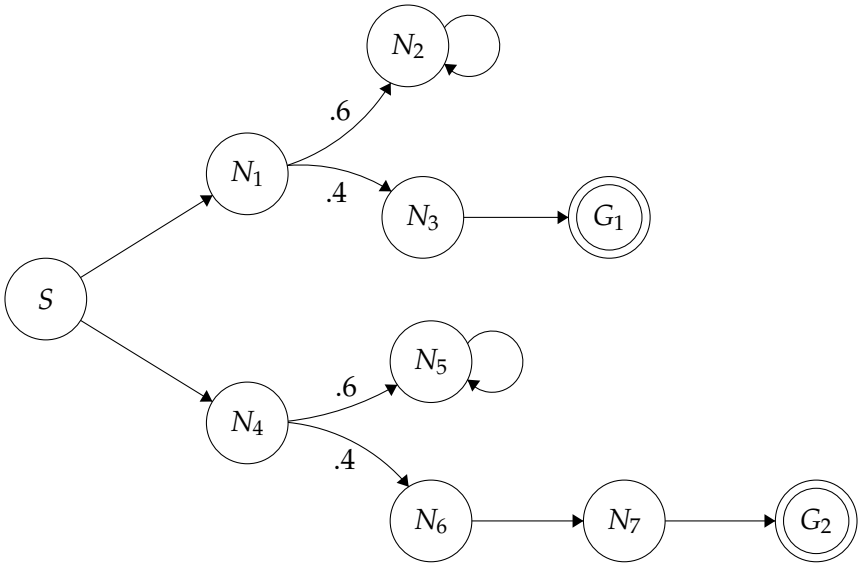
One logical criteria for optimality of a policy that we have not yet considered is the expected number of steps to reach the goal. While an algorithm may exist that optimizes for this exact criteria, this algorithm does not—at least not with any of the reward functions considered so far.

First consider solving the following domain with GRND:



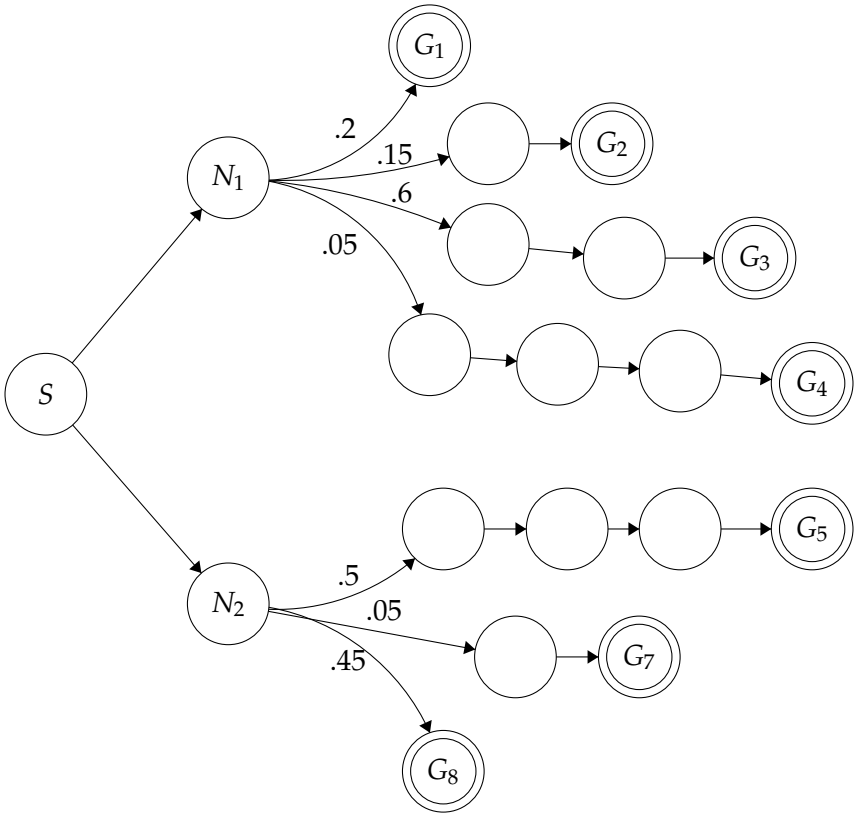
GRND would in the worst-case expand S, N_1, N_2, and then finally G_1 before terminating, and return the top policy as the optimal policy. This is clearly not the minimum expected number of steps.

Next consider the following domain with APND:



APND in the worst-case will expand N_1, N_4, N_2, N_5. At this point the expected total reward for both policies is the same: $-\infty$. It will then in the worst-case choose to expand N_6, N_7, G_2 and terminate. This policy has a greater number of expected steps, so it is not optimal by that criterion.

Finally we must consider APWD and GRWD, which we already showed make the same decisions and expand the same nodes.



This domain shows an example of when APWD and GRWD choose a policy that does not minimize expected number of steps. Let the discount factor, $\gamma = 0.9$. Since we know they will perform identically, we will use GRWD to make calculations easier. The expected reward for the top policy that goes to N_1 first is $\gamma * 0.2 * \gamma + \gamma * 0.15 * \gamma^2 + \gamma * 0.6 * \gamma^3 + \gamma * 0.05 * \gamma^4 = 0.694535$. The expected reward for the bottom policy that goes to N_2 first is $\gamma * 0.5 * \gamma^4 + \gamma * 0.05 * \gamma^2 + \gamma * 0.45 * \gamma = 0.696195$. Both always reach the goal, so the algorithm will choose the bottom policy because it has a greater expected total reward. The top policy has an expected $1 + 0.2 * 1 + 0.15 * 2 + 0.6 * 3 + 0.05 * 4 = 3.5$ steps, whereas the bottom policy has an expected $1 + 0.5 * 4 + 0.05 * 2 + 0.45 * 1 = 3.55$ steps. Thus, the algorithm chooses the policy with a greater number of expected steps.

VII. Breadth-First Search

The major drawback of the previous algorithm is its very limited conditions under which it terminates on infinite domains. One of the main goals for pursuing Algorithm 1 was to solve exactly this problem—a planning algorithm for infinite domains. Once that algorithm did not have the guarantees we were looking for, we searched for a new algorithm. Our intuition was that a modified Breadth-First Search (BFS) algorithm could solve this issue, and, as we will see, it does to some extent. This algorithm, like a standard BFS algorithm, maintains a queue of states to expand. It picks a state off the queue, expands it, and then adds the new fringe states to the queue. This algorithm will primarily find the optimal first action to take, and not necessarily the entire policy. This algorithm does not use a reward function, so it only works for Goal Directed Markov Decision Processes. If we desired, we could modify the algorithm to find the entire policy by using an alternative termination step.

Algorithm 3 Breadth-First Search

Input: Markov Decision Process
Output: Best Policy
1: Enqueue starting state in a queue
2: Dequeue a state from the queue
3: Expand state
4: Enqueue new fringe states to the queue
5: **Termination:** If the algorithm has terminated, return the optimal policy, otherwise go to Step 2

Explanation of Termination Step

Recall that our goal is simply to find the best first action to take. The short circuit step in Algorithm 1 found the best first action given an intermediary MDP, so we can model this termination step closely off of that. We will terminate if the greatest pessimistic estimate for a policy's probability of reaching the goal is greater than or equal to the optimistic estimate for all other policies whose first step differs from the best policy's. We will also terminate if the greatest pessimistic estimate for a policy's probability of reaching the goal is 1.

First we must find the policy with the greatest pessimistic estimate. In the previous algorithm, the chosen policy at each step was the best policy so far, so we could just use that policy's pessimistic estimate. This time we must find the greatest pessimistic estimate. We will use GRND assuming fringe states have a 0 probability of reaching the goal. To find the best optimistic estimate among the unchosen policies, we run GRND on a simplified version of the MDP, removing the first action of the optimal policy. We can then compare the greatest pessimistic estimate to the other optimistic estimates to see if we can terminate.

If we want to find the entire policy and not only the first step, we need a tighter condition. Rather than comparing the pessimistic estimate to the optimistic estimates of policy's with different first steps, we must compare the pessimistic estimate to *any* other policy, even those with the same first step. We will not go into how to do this here, as this is not as straightforward as the above. It is not obvious how to find the second best policy in an MDP, so we cannot just plan with GRND anymore. An alternative termination condition would be similar to the previous algorithm, terminating if the policy with the greatest pessimistic estimate for the probability of reaching the goal has no fringe states. While being simpler, it will take much longer to meet this termination condition and it is unclear if it makes any gains over Algorithm 1. As such we will only consider the case where we return the first action.

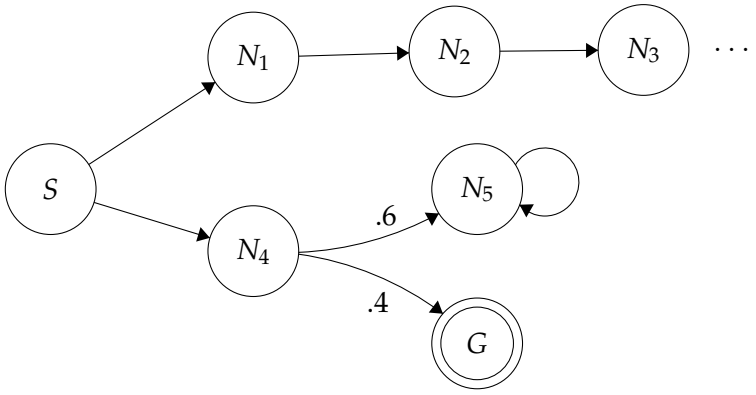
I. Correctness

Theorem 13 *If Algorithm 3 terminates, it returns the first action of the policy with the greatest probability of reaching the goal.*

Proof: The algorithm terminates when it finds that a policy's pessimistic estimate for the probability of reaching the goal is greater than the optimistic estimate of any policy that starts with a different first action. Since we assume it terminates, the algorithm will return a policy whose pessimistic estimate is greater than the optimistic estimate of any policy that starts with a different first action. If the policy returned has a pessimistic estimate greater than any other optimistic estimate, then it is impossible for any other first action to yield a better policy. □

II. Termination

For the same example as with Algorithm 1, this algorithm is not guaranteed to terminate on an infinite domain. Similarly to before, we can only guarantee that if the optimal policy has a probability of 1 of reaching the goal then it will terminate:



The algorithm expands S, N_1, N_4, N_2, N_5, G, N_3, and then any states coming after N_3. The best pessimistic estimate for the probability of reaching the goal will be the bottom policy, with an estimate of 0.4. The top policy's optimistic estimate, however, will never go below 1, so it will never terminate. This is actually more general than a shortcoming in these two algorithms.

Theorem 14 *If an algorithm maximizes the probability of reaching the goal, then it will not terminate on the above domain.*

Proof: Assume not. Then, there exists some algorithm, A, that terminates on this domain. The MDP only allows us to look at states and the actions that those states take. No algorithm can both look at an infinite number of states and terminate. Let the finite list of states that the algorithm observes be $S, \ldots, G$. Since the top policy, P_1, of this domain is infinitely long, the algorithm cannot have fully expanded it. Let N_i be the furthest state from S that A expands on P_1. Since A expands a finite number of states, N_i must be a finite distance from S. Additionally, the optimistic estimate for P_1 must still be 1 at the termination of the algorithm. Consider a similar domain that is exactly the same, except N_{i+1} is a goal state. A sees the exact same domain for every expansion, and thus must terminate before expanding N_{i+1}. However, then the probability that P_1 in this new domain reaches the goal would be 1, and P_1 would be the optimal policy. This statement contradicts that the algorithm finds the policy with the greatest probability of reaching the goal. Thus, our assumption must be false and no algorithm can terminate on this domain. □

Theorem 14 shows that no algorithm can possibly guarantee to terminate without further restrictions beyond finiteness of the optimal policy. This theorem shows that actually, one of our greatest reservations about Algorithm 1—the weakness of its termination condition—was unfounded. Note however, that after expanding G, no other policy will ever have a greater pessimistic estimate than the optimal policy. Thus, we can claim, and will now prove, that if Algorithm 3 is externally terminated after expanding the entirety of the optimal policy, it will always return the optimal policy. Furthermore, the algorithm will do so in finite time. It is, however, impossible to know when this moment comes, which is why we cannot prove that the algorithm terminates.

Lemma 15 *If the optimal policy is finite and if every action transitions to a finite number of states, Algorithm 3 expands every state in the optimal policy in finite time.*

Proof: Assume not. Let n be the number of states in the optimal policy. The furthest state from the start on a policy with n states can be at most n states away from the start state. Since the algorithm does not terminate, it must expand an infinite number of states before expanding a state n states away from the start. Since every state has a finite number of actions and every action transitions to a finite number of states, every state can transition to a finite number of states. From these two facts, the algorithm must expand states that are infinitely far from the start state. The algorithm removes fringe states from the queue in a first-in-first-out order, so it cannot expand a state that is farther away before a nearer one. But that is a contradiction because a state infinitely far from the start is farther than one that is n from the start. Thus, the algorithm will expand every state in the optimal policy in finite time. □

Theorem 16 *If the optimal policy is finite and if every action transitions to a finite number of states, then in finite time Algorithm 3 will have found the optimal policy and the optimal policy will never change.*

Proof: Lemma 15, guarantees that the algorithm will expand every state in the optimal policy in finite time. Once every state in a policy is expanded, the policy's estimate for the probability of reaching the goal will equal its actual probability of reaching the goal and will never change. A pessimistic estimate is made by assuming every fringe state has a 0 probability of reaching the goal. The only states assumed to reach the goal are actual goal states, and there are other fringe states that could also be goal states. Expanding more states can then only change the estimate by finding a goal state, which will only increase the estimate. With every expansion, the estimate increases and after expanding the entire policy, it must equal the actual probability of reaching the goal. Thus, a policy's pessimistic estimate can never be greater than its actual probability of reaching the goal.

Since the optimal policy has the greatest probability of reaching the goal, no policy's pessimistic estimate can exceed the optimal policy's actual probability of reaching the goal. If it did, then its actual probability of reaching the goal would be greater than the optimal policy's and it would be the real optimal policy. Once the optimal policy is fully expanded, since the optimal policy's pessimistic estimate is equal to its actual probability, no other policy will ever have a greater pessimistic estimate and the algorithm can never return any other policy. □

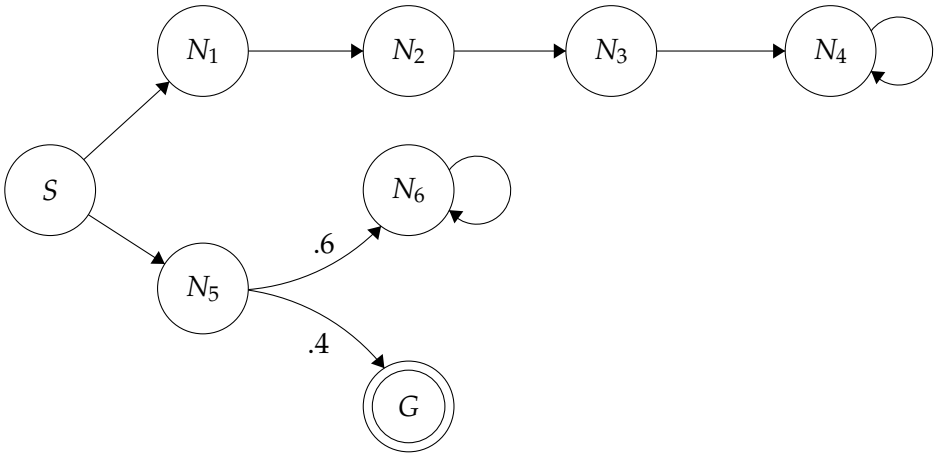
Theorem 17 *Algorithm 3 will terminate in finite time if:*

1. *the optimal policy is finite,*
2. *every state has a finite number of actions,*
3. *every action transitions to a finite number of states,*
4. *and the greatest probability of reaching the goal is 1.*

Proof: Theorem 16, shows that in finite time the algorithm will expand the optimal policy, and that policy will never change. If the greatest probability of reaching the goal is 1, then after expanding every state in the optimal policy the algorithm will have found a policy with a pessimistic estimate of 1. After expanding the entire optimal policy, the algorithm can terminate because it will never find a better policy. □

III. Runtime

We cannot say much about the runtime for this algorithm. The algorithm will not end immediately when the optimal policy is found. Consider the following domain:



The algorithm will expand N_1, N_5, N_2, N_6, G. At this point the optimistic estimate for the probability that the top policy reaches the goal is 1 and the estimate for the bottom policy is 0.4. The algorithm must expand N_3 and N_4 before it can finally establish that the bottom policy is optimal.

As shown by this example, there is no rule as to how many states the algorithm will explore. Furthermore, if the MDP is infinite then the algorithm will not terminate. Terminating the algorithm early, however, will give an approximation of the best first step. The algorithm at any time can be terminated and return the first action that yields the policy with the greatest pessimistic estimate for the probability of reaching the goal. While it might not give the optimal first step if it is still too early, the algorithm will return the first action whose best policy is guaranteed to perform best in the worst-case.

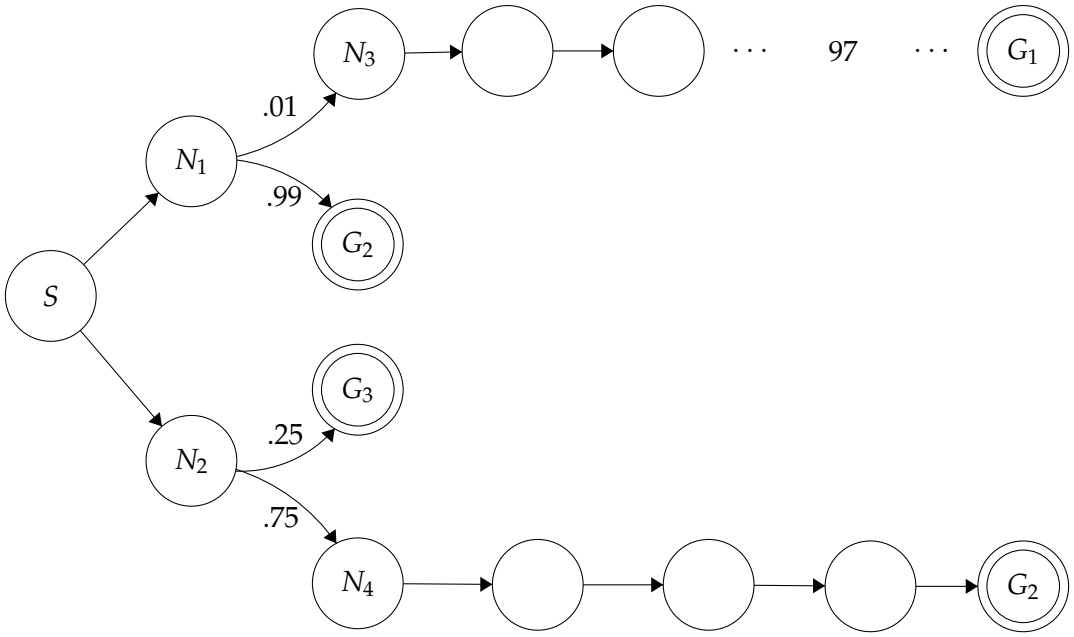
We will now consider changes to the algorithm that may yield better performance. Theorem 10 for Algorithm 1 proved that we only expand states that could possibly be part of an optimal policy. So far we have not shown a comparable result for this algorithm. If we only expand states that can possibly be part of an optimal policy, then the algorithm may expand fewer states. This requires us to keep track of which first actions lead to any given state.

Before adding a state's neighbors to the queue, look to see if the state can possibly be part of an optimal policy. To check if any policies that reach it can be optimal, look at every first action that leads to the current state. Given each action, find the policy with the greatest optimistic estimate for the probability of reaching the goal. This policy may not actually go through the state, but otherwise we would need to keep track of every policy that reaches the state which would incur an infeasible memory requirement. If this estimate is lower than the best pessimistic estimate for a policy with a different first action, then it is impossible for that action to be useful. We can then skip any states emanating from that original action.

If every first action that can possibly take us to the current state is suboptimal, this state is useless and there is no need to pursue it or its neighbors. If another state later borders one of those skipped neighbors, it is still possible we will have to expand them, but we just do not have to yet. We can also cache which first actions are guaranteed to be suboptimal, and do these calculations far less often.

IV. Expected Number of Steps

Since this algorithm gradually increases the radius of expanded states, it seems plausible that this algorithm could naturally find the policy with the fewest number of expected steps. Unfortunately, the following example demonstrates that this is not the case. Consider the following domain:



The expected number of steps to the goal for the top policy going to N_1 is $0.01 * 100 + 1 * 0.99 = 1.99$. The expected number of steps to the goal for the bottom policy going to N_2 is $0.75 * 4 + 1 * 0.25 = 3.25$. Both policies have a probability of 1 of reaching the goal, so whichever is fully explored first will be declared the optimal policy. The bottom policy will finish being explored first because it's furthest state is only 6 steps from the start, so it will be considered the optimal policy. The top policy, however, has the fewest expected number of steps, so the algorithm does not return the policy with the fewest expected number of steps.

VIII. Conclusion

We have shown here two novel algorithms for planning on large and infinite stochastic domains. Many domains have an exponential, or even infinite, number of states. By expanding states as needed we are able to ignore entire swaths of our state space that have no chance of being useful or necessary. Further, most decision-theoretic problems are reduced to either reaching a goal state or maximizing reward. This research, on the other hand, looks at first finding the goal, and then maximizing reward. This allows for more expressive descriptions of tasks.

Our Probabilistic Heuristic Search algorithm iteratively chooses a policy and state to expand, expands the state, and then checks if the algorithm has terminated. To choose a policy, we run a planning algorithm on the states seen so far, assuming unexpanded fringe states are goal states. The algorithm is proven to work, finding the policy that first has the greatest probability of reaching the goal, and among those tied has the greatest expected total reward. Theorem 3 shows that it terminates for finite graphs and Theorem 8 shows that it sometimes even terminates for infinite graphs.

The greatest shortcoming of this algorithm is the weakness of its termination condition. The domain must be very specific for it to guarantee termination on an infinite domain, which

is the entire goal of the research. While we do prove with Theorem 14 that no algorithm could terminate on some of these domains, we cannot guarantee finding the optimal policy in finite time regardless of termination like our Breadth-First Search algorithm can. One way to improve upon this property would be to add randomness to the algorithm. The algorithm does not always expand the entire optimal policy because it gets stuck in infinite progressions of useless, optimistic states. We could break free of these traps by occasionally randomly expanding other states than the one specified by the algorithm. A good next step to the research could be to consider this solution more carefully and search for any guarantees that would result.

We can make some limited guarantees about the runtime. The first result is Theorem 10, which proves that we only expand states that could be in the optimal policy. If we're allowed to use any reward function we want, we show that some will need to expand more states than others. They will not necessarily find the same optimal policy since optimality is dependent on the reward scheme, however they will all find a policy that maximizes the probability of reaching the goal. Theorem 11 proves that the algorithm expands the same states using GRWD as it does using APWD. Then, Theorem 12 proves that GRWD, APND, and APWD perform equal to or better than GRND in the worst case. Unfortunately, we are unable to compare GRWD and APND since each one may perform better than the other in different domains, or for different discount factors. Our last conclusion regarding Algorithm 1 is that, with any of the above reward functions, it does not find the policy with the fewest expected number of steps.

To hopefully improve our termination guarantee or create an algorithm that does find the policy with the fewest expected number of steps, we explored a second, breadth-first search based, algorithm. We proved that this algorithm similarly does not terminate on many domains. One consolation, however, is that we proved that it is impossible for *any* algorithm to terminate on some of the most problematic domains for both of our algorithms. This BFS-based algorithm, however, has the interesting property, from Theorem 16, that it often will find the optimal policy in finite time but not know that it has done so. External termination of the algorithm can give a good approximation, if not the actual optimal policy. One way to modify and possibly improve the algorithm would be to keep track of the pessimistic estimate for the probability of reaching the goal and terminate the algorithm if it hasn't improved in a given amount of time. We also briefly looked at the runtime of the algorithm, which does not have the same guarantees as Algorithm 1. Finally, we showed that this algorithm also does not find the policy with the fewest expected number of steps.

One next step to this research direction is to further characterize the runtime in different scenarios. If possible, it would be useful to characterize more concretely which states are expanded under any given reward function. We showed here a handful of tagging schemes that fail, but an as-yet unfound, successful tagging scheme could still exist. Further, it would be interesting to explore other reward functions and how they improve or hamper performance. Singh et al. (2010)[5] looked at intrinsically motivated reward functions, which certainly could have interesting effects. In certain scenarios, it is likely that intrinsic rewards could be used to ignore huge sections of the state space, especially when the probabilities of reaching the goal are similar across the board.

We could also look at different ways speed up either of our algorithms. Koenig and Liu (2002)[3] wrote an algorithm that removes traps from the state space to allow the algorithm to find policies that always have a probability of 1 of reaching the goal. An algorithm like theirs, which takes a completely different approach than ours to finding traps, could find policies that have a probability below 1 of reaching the goal. With that information, the algorithm could more directly find the policy with the highest probability of reaching the goal. Another

incredibly useful step would be to program these algorithms and empirically compare their performance across multiple reward functions and against Q-learning on various size domains.

Our question really at its core is, given a characterization of a "best" policy, find an algorithm that efficiently finds it on a domain of any type and any size. We, however, chose our designation of the "best" policy fairly arbitrarily. We easily could have made the maximum expected reward more important than the probability of reaching the goal, or considered the expected number of steps to the goal to be the most important criterion. One obvious way of characterizing the "best" policy, differently from how we did it here, is to set a threshold for how high the greatest probability of reaching the goal must be, and then treat any policy whose probability of reaching the goal is higher than that as tied. The algorithm here will likely be very similar to our first one, but the Set Policy Step would be different. Our Set Policy Step uses a planning algorithm with GRND to find the probability of reaching the goal. To consider any policy with a reward above a given threshold as tied, it would only need to change the way it returned the best policy at the end. The rewards could all be generated as usual, but when generating the tie breaking MDP, the algorithm could use more than just the best policy.

This research is only a stepping stone. There are many different criteria for the "best" policy which could vary from use case to use case, adding greater expressiveness to various tasksx. As shown, much of what we discovered were counterexamples against characteristics that are important for any useful algorithm. An algorithm, likely radically different from these two, that has tighter termination guarantees and finds the shortest policy in the quickest time would be even better and may still be out there. Being able to use planning on a large or infinite domain opens up many possibilities and could significantly improve runtimes of algorithms on large state spaces.

IX. Acknowledgments

I would like to thank Professor Michael Littman for advising me through my research. I learned everything I know about reinforcement learning with his guidance, and he helped me navigate the complex waters of the academic process. Professor Littman taught me so much throughout this process and I would not have been able to complete any of this without him.

References

[1] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1):35–74, 1995.

[2] Eric A Hansen and Shlomo Zilberstein. Heuristic search in cyclic and/or graphs. *AAAI/IAAI*, pages pp. 412–418, 1998.

[3] Sven Koenig and Yaxin Liu. The interaction of representations and planning objectives for decision-theoretic planning tasks. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(4):303–326, 2002.

[4] Sven Koenig and Reid G. Simmons. Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains. Technical report, Pittsburgh, PA, USA, 1992.

[5] Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. Intrinsically

motivated reinforcement learning: An evolutionary perspective. *Autonomous Mental Development, IEEE Transactions on*, 2(2):70–82, 2010.

[6] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. A Bradford book. MIT Press, 1998. ISBN 9780262193986. URL `http://books.google.com/books?id=CAFR6IBF4xYC`.