Prioritized Sweeping for Stochastic Planning

Basic idea: Only do Bellman backups in states where they will be useful.

- Maintain a priority queue of triples (s, a, δ), sorted in descending order by δ. δ estimates how much $Q(s, a)$ will be changed if we do a Bellman backup on it.
- At each step, pop (s, a, δ) off the queue, and perform a Q backup:

$$Q^{new}(s, a) = \sum_{s'} P(s'|s, a)[R(s'|s, a) + \gamma V(s')]$$

  Let $\Delta = |\max_a Q^{new}(s, a) - \max_a Q^{old}(s, a)|$. If Δ is large, then this update has made a big change to the value of state s, so we should update our predecessors.

- For each state and action (s^-, a^-) such that $P(s|s^-, a^-) > 0$, we push the triple $(s^-, a^-, P(s|s^-, a^-) \cdot \Delta)$ onto the priority queue

Initialization

- For an episodic task, we can initialize prioritized sweeping by pushing all triples $(s, a, \delta \cdot R(s'|s,a))$ onto the priority queue where $P(s'|s,a) > 0$ and s' is a terminal state.
- For a continuing task, we need to initialize prioritized sweeping by visiting all states and actions once.

Value Iteration with Prioritized Sweeping

PSweepValueIteration(threshold ϵ, steps n)

  priority queue PQ

  Q function, initially zero

  for each state s **do**

    let $V_{old} = \max_a Q(s,a)$

      for each action a **do**

        $Q(s,a) := \sum_{s'} P(s'|s,a)[R(s'|s,a) + \gamma \max_{a'} Q(s',a')]$

      end for a

    let $V_{new} = \max_a Q(s,a)$

    PushPredecessors($s, |V_{new} - V_{old}|, \epsilon$)

    PSweep(n, ϵ)

  end for s

  PSweep(∞, ϵ)

end PSweepValueIteration

PushPredecessors and PSweep

```
PushPredecessors(state s, change Δ, tolerance ϵ)
    for each s⁻, a⁻ such that P(s|s⁻, a⁻) > 0 do
        if P(s|s⁻, a⁻) · Δ > ϵ then
            PQ.push(s⁻, a⁻, P(s|s⁻, a⁻) · Δ)
        end for
end PushPredecessors

PSweep(steps n, tolerance ϵ)
    repeat n times
        pop (s, a, δ) off PQ
        let V_old = max_a Q(s, a)
        Q(s, a) := Σ_s' P(s'|s, a)[R(s'|s, a) + γ max_a' Q(s', a')]
        let V_new = max_a Q(s, a)
        PushPredecessors(s, |V_new − V_old|, ϵ)
    end repeat
end PSweep
```

Refinements and Alternatives

- **Full Backups**. The algorithm can also be organized around states rather than (s, a) pairs. In the PSweep inner loop, we perform a full Bellman backup, instead of just a backup for one action.

- **Avoid duplicates in the priority queue**. When (s, a, δ) is pushed onto the queue, if there is already an entry for s, a, keep the one with the larger value of δ. Designing a data structure to do this requires a bit of thought, because we need to be able to find an element in the queue (e.g., by maintaining a hash table of pointers into the queue), and we need to be able to delete elements from the middle of the queue. We can use a heap if we ensure that each cell in the heap knows its current position in the heap. The hash table can then point to these cells. You can delete an element from a heap in log time.