Horizon-based Value Iteration

Peng Zang
Arya Irani
Charles Isbell

ABSTRACT

We present a horizon-based value iteration algorithm called Reverse Value Iteration (RVI). Empirical results on a variety of domains, both synthetic and real, show RVI often yields speedups of several orders of magnitude. RVI does this by ordering backups by horizons, with preference given to closer horizons, thereby avoiding many unnecessary and incorrect backups. We also compare to related work, including prioritized and partitioned value iteration approaches, and show that our technique performs favorably. The techniques presented in RVI are complementary and can be used in conjunction with previous techniques. We prove that RVI converges and often has better (but never worse) complexity than standard value iteration. To the authors' knowledge, this is the first comprehensive theoretical and empirical treatment of such an approach to value iteration.

1. INTRODUCTION

Reinforcement learning (RL) is a field that defines a particular framework for specifying how an agent can act to maximize its long-term reward in a stochastic (and perhaps only partially observable) environment. RL has broad applicability. While it has been traditionally used in the context of single agents for planning, learning, and control problems,there has been a tremendous amount of recent work in applying it in multi-agent scenarios as well [4], [13], [7]. A common approach to solving RL problems is to learn a *value function* that captures the true utility of being in a given state. In addition, value functions have many other uses. For example, Munos and Moore [9] use the value function to guide discretization decisions, Brafman uses them to guide exploration [2] and Kearns and Singh [6] use it to decide between exploration and exploitation.

Value iteration is a well established, dynamic programming approach to learning value functions. It is characterized by the use of *backups*[11] to propagate information about the true long-term utility of a state from potential future states. Unfortunately, a traditional implementation of this process can have several disadvantages.

One drawback of traditional value iteration is that its memory and computational requirements scale with the number of states, which grows exponentially with the number of state features. Various approaches such as function approximation [3], direct policy search[10], and re-representational techniques such as temporal and state abstraction [12] have been pursued to address this issue. However, value iteration remains one of the few techniques that can solve MDPs exactly with no domain knowledge. In addition, with hierarchical, modular, and re-representational techniques that subdivide, abstract or otherwise reduce the number of states, value iteration can be applied successfully on very large problems.

A naive implementation can also perform a lot of unnecessary work. A backup operation is peformed for every state in every iteration of the algorithm, until a global convergence criterion is met. This includes states which may already have converged to the correct values, as well as those for which no new information is available. In this paper, we introduce a new algorithm called Reverse Value Iteration (RVI) which reorders backups to mitigate this waste. We will show that RVI converges and typically has a lower complexity. Experimental evaluation shows that RVI often reduces the number of backups by several orders of magnitude when compared to standard value iteration. We also compare RVI to other speedup techniques, such as prioritization and partitioning, over which RVI still often sees a severalfold speedup. It is not a competition however; RVI is complementary to other techniques and can be used in conjunction with them.

The rest of this paper will be organized as follows: first, we will cover related work so that we can make appropriate comparisons throughout the paper. This is followed by a formal problem statement and introduction of our notation. We will then introduce our algorithm and discuss its properties such as convergence and complexity. Finally we will see how the algorithm bears out in practice through a thorough empirical evaluation, before some concluding remarks at the end.

Tech Report GIT-IC-07-07.

2. RELATED WORK

The main lines of research in improving value iteration have been in making backups asynchronous [1], prioritizing the order of backups [14], partitioning of the state space [14] and parallelization of the process as a whole [16].

Asynchronous value iteration is fundamental work demonstrating that backups can be performed in any order without breaking convergence guarantees, as long as all states are assured an infinite number of backups. This is the work that enables later refinements and improvements to value iteration.

Prioritization of backups in value iteration aims to bring the idea of prioritized sweeping, as seen in the model-free literature [8], to model-based value iteration. This is a method relying on heuristic(s) to guide the order of backups. In model-free literature, prioritized sweeping is cited as often leading to a five- to ten-fold speedup for some classes of domains such as mazes [11]. For value iteration, prioritization of backups has to be applied carefully, because the overhead of maintaining a priority queue can quickly overwhelm any savings in terms of number of backups.

Partitioning of backups was developed, in part, to overcome the overhead of prioritization methods. It also stems from the observation that value iteration performs best (*i.e.* with few wasted backups) when states are highly connected. Thus, it makes sense to partition the states such that edge cuts are minimized, yielding more strongly connected components. Prioritization can then be applied efficiently on the partitions as there are far fewer partitions than states. Normal value iteration can be used within the more strongly-connected partitions [15]. Experiments show partitioning can yield significant gains in conjunction with prioritization although some thought must go into the partitioning scheme. Partitioning of the states also leads naturally to parallelization: one can assign partitions to processors to gain significant speedup [16].

As we will describe in Section 4, the techniques introduced in this paper also seek to eliminate unproductive backups. However, rather than using a value-based heuristic for ordering backups as prioritization and partitions methods do, we order them based on a systematic (reverse) traversal of the state space. To the authors' knowledge however, this is the first comprehensive theoretical and empirical treatment of applying such a technique to value iteration for solving MDPs.

3. PROBLEM STATEMENT AND NOTATION

We define an (finite) MDP $M = (S, A, P^a_{ss'}, R^a_{ss'}, \gamma)$ by a finite set of states S, a finite set of actions A, a transition model $P^a_{ss'} = Pr(s'|s,a)$ specifying the probability of reaching state s' by taking action a in state s, a reward model $R^a_{ss'} = r(s,a,s')$ specifying the immediate reward received when taking action a in state s and reaching the new state s', and the discount factor $0 \leq \gamma < 1$.

We denote a horizon h_k to be the set of states from which a terminal state can be reached in exactly k steps. We call absorbing states in which you cannot leave the state once you have arrived, terminal states.

Let $\texttt{parents}(s)$ be the set of states from which there is a non-zero probability of transitioning to state s: $\{s' : \exists a \in A(Pr(s'|s,a) > 0)\}$. Similarly, let $\texttt{children}(s) = \{s' : \exists a \in A(Pr(s|s',a) > 0)\}$.

A policy, $\pi(s)$, is a mapping that dictates what action an agent should perform in a particular state. The utility or *value* of a state $V^\pi(s)$ is the expected long-term discounted rewards an agent receives, when following policy π from state s. $V^*(s)$ is the value of state s when an agent follows an optimal policy that maximizes the long-term expected reward. Value iteration is an algorithm for calculating V^*.

To simplify discussion, we make the following assumptions, without loss of generality: (1) terminal states are represented as states with only self-transitions of zero reward, (2) all rewards are strictly positive (except the zero reward self-loops of terminal states), (3) all optimal paths end in a terminal state; if an optimal path is cyclic, we require either it be converted into a terminal state or all states in the cycle be added initially in to the queue in RVI. As a consequence, the value of any non-terminal state is strictly positive and the value of any terminal states is zero.

4. REVERSE VALUE ITERATION (RVI)

Consider a deterministic grid world. The actions are *left*, *right*, *up*, and *down*. The values of each state is initialized to zero, except for the terminal state in the bottom right, which is set to 1.

Note how the terminal states in an MDP are the only ones with values $V^*(s)$ that are known from the onset. All other values are ultimately a function of the utilities of terminal states and the transition and reward models. In our discussion, we will call a state and its value "informative" if and only if its value reflects, at least in part, the utility of a terminal state. Observe that uninformative values are eventually overwritten and do not contribute to computing V^*. We will borrow the term "information frontier" to refer to

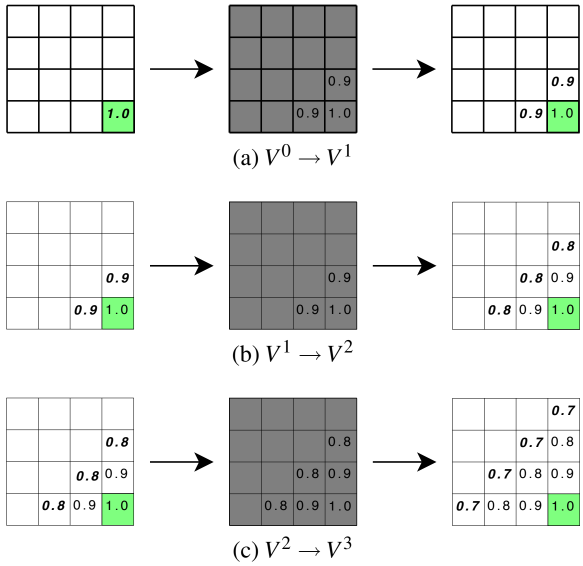


Figure 1: Progress of Value Iteration in a simple gridworld. The information frontier is italicized. Empty cells indicate values of zero. For a given row, the left column shows the value function at the start of a given iteration; the center column shows, in grey, which states are backed up, and the right column shows the resulting value function.

the boundary between states with informative values and those with uninformative values[1].

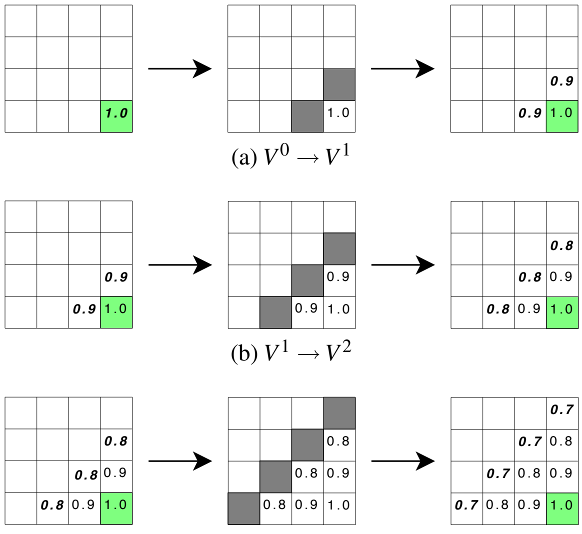
Consider how value iteration would solve this MDP, assuming a typical left-right, top-bottom state enumeration scheme. In Figure 1 note that the information frontier grows by just one step with each iteration, despite all states receiving backups. This is due to the unfortunate mismatch between the direction the backups are performed, and the direction the information frontier travels.

The natural observation then is that if we can order the backups along the direction of information flow, we can avoid many wasted backups and achieve significant speedup. The key is that information propagates along the set of optimal paths of the MDP, but in the reverse direction *i*.e. beginning with the terminal state(s) and flowing outward to all possible starting states. The set of optimal paths is generally unknown to us, but we do know that they must end in a terminal state; the set of optimal paths is a subset of the set of paths that end in a terminal state. RVI works by simply ordering the backups along the reverse direction of paths that end in a terminal state, as shown in Figure 2.

The basic RVI algorithm is given in Algorithm 1. RVI works by first performing backups on the states in horizon h_1 (the parents of terminal nodes), and then the states in each successive horizon. The queue Q dictates which states remain to be backed up. Each queue element is a tuple (s,h) where s is the state and h is the horizon associated with s. A given state can appear in multiple horizons, but only receives one backup per horizon (Line 14). Furthermore, observe that the queue guarantees that all backups for a given horizon completes before backups for subsequent horizons begin.

The absence of terminal states does not invalidate the algorithm. In general, our goal is to traverse all optimal paths in reverse so

[1]In the original usage of this term [15], it was associated with the rate of change of the value of a state. Here we will not make such an association because change in the value of state is not necessarily due to accurate information.



(a) $V^0 \to V^1$

(b) $V^1 \to V^2$

(c) $V^2 \to V^3$

Figure 2: Progress of Reverse Value Iteration in a simple gridworld.

that the ordering of backups can be maximally aligned with the direction information flow. Terminal states simply serve as a way to narrow the set of paths we have to consider. If there are no terminal states, then every state could potentially be a "terminal state": the final state where an optimal path terminates. So if a MDP has no predefined terminal state(s), all states are initially added to the queue.

RVI has two additional minor optimizations, omitted from Algorithm 1 for the sake of readability. First, when performing backups, children with values of zero are ignored. The reasoning is that a state can only have a value of zero if it has never been backed up before. In that case, the child's value has no relevant information to add when the expectation is being calculated for the backup. We ignore the zero valued state by pretending that it is unreachable. Any probability mass originally associated with that state is redistributed to the rest. Note that this requires at least one next state be non zero. When there are terminal states, this is guaranteed. When there are no terminal states, this optimization is not performed.

The second optimization we perform is detecting and solving self loops. If a state s has a self loop with reward r then its value should be $r/(1-\gamma)$. We detect his case, and instead of performing a backup, we set its value directly.

Algorithm 1 Reverse Value Iteration (RVI)

Require: MDP $M = (S, A, P^a_{ss'}, R^a_{ss'}, \gamma)$,
Discount factor γ,
Precision ε

1. Initialize value table $V(s) = 0$ for all states $s \in S$.
2. $T \leftarrow \{s \in S : s \text{ is a terminal state }\}$
3. **if** $T \neq \emptyset$ **then**
4. Initialize queue $Q = \{(s,1) : s \in \texttt{parents}(t), \forall t \in T\}$.
5. **else**
6. Initialize queue $Q = \{(s,0) : s \in S\}$.
7. **end if**
8. **while** Q not empty **do**
9. $(s,h) \leftarrow \texttt{pop}(Q)$
10. Backup state s:

$$V(s) \leftarrow \max_{a \in A} \left(\sum_{s' \in \texttt{children}(s)} P^a_{ss'} \left[R^a_{ss'} + \gamma V(s') \right] \right)$$

11. **if** backup resulted in change greater than ε **then**
12. **for all** $p \in \texttt{parents}(s)$ **do**
13. **if** $(p, h+1) \notin Q$ **then**
14. $\texttt{push}(Q, (p, h+1))$
15. **end if**
16. **end for**
17. **end if**
18. **end while**

5. CONVERGENCE PROPERTIES OF RVI

In this section we will show that RVI converges and analyze the speed of that convergence. We will perform this analysis in two parts. In the first subsection we will focus on proving convergence. We leave discussion of the speed of that convergence to the following subsection.

5.1 Convergence Guarantee

For the purposes of discussion, let RVI$'$ be an algorithm identical to RVI, with the exception that when a state s is backed up, its parents are added to the backup queue regardless of whether the value of s has changed; the check at line 13 is ignored.

LEMMA 1. *The first backup of a state s always results in a change.*

PROOF. $V(s)$ is set to zero initially for all states. When a backup is performed on a state s, its new value will be strictly greater than zero since all rewards are strictly positive. Thus a change is guaranteed. □

LEMMA 2. *All non-terminal states will receive at least one backup.*

PROOF. If there are no terminal states in the MDP then all states are initially added into the queue and thus each (non-terminal) state will receive at least one backup. The rest of the proof will focus on the case of MDPs with at least one terminal state.

First consider this for RVI$'$. For any state s, let us then denote the states along the optimal path from s to a terminal state as $s_0, s_1, s_2, \ldots, s_n$ where s_n is the start state and s_0 is the terminal state. We know s_1 will receive at least one backup, because all parents of terminal states are added onto the queue in the first iteration of the algorithm. We also know that if state s_k receives a backup, s_{k+1} will receive a backup as well since the parents of a state being backed-up are added on to the queue. By induction, s_n must eventually receive a backup as well.

Now consider RVI. RVI differs from RVI$'$ in that parents of a backed-up state s are only queued in RVI if the value of s changes. For this difference to cause s_n not to receive a backup, there must be some state along the path $s_0 \ldots s_n$ which is never changed during its backup. However, we know by Lemma 1 that at least one (the first) backup for each state will cause a change. Thus RVI is the same as RVI$'$ in that s_n will eventually receive a backup. □

LEMMA 3. *In RVI$'$, a backup on state s in horizon $h \geq 1$ that does not change the value of s can be omitted without changing any computed values of any state.*

PROOF. Omitting a backup in horizon h obviously will not have an effect on the previous calculation of earlier horizons. Thus we will focus on proving that the omission will not have any impact on values of states in horizon h and later.

Base case: Suppose a state s is backed up in horizon h with no change in value. Clearly, the omission of this backup won't affect the value of s itself. Since unchanging values have no affect on backups, no other backups in h will be affected either.

Inductive case: We've seen that an omission at horizon h has no effect on the values calculated in horizon $h = h+0$. Let us assume that an omission performed at horizon h has no effect on the values calculated in horizon $h+k$ for some $k \geq 0$. We would like to show that it will also not effect horizon $h+k+1$.

Let us consider the parent state p which would have been added to horizon $h+k+1$ if child state s had received a backup in horizon h despite not changing in value. Either (1) no other children of p had a value change, in which case p's value will remain unchanged in $h+k+1$, or (2) another child $c \neq s$ of p had a value change in horizon $h+k$, in which case p is added to horizon $h+k+1$ for backup independently of s's backup in horizon $h+k$, and so horizon $h+k+1$ will turn out the same for this case as well. □

LEMMA 4. *For all horizons $h \geq 1$, if the backup to a state s leads to an incorrect value, it will get backed up at least once more in a later horizon.*

PROOF FOR RVI′. We argue by induction. Suppose we have a state s which is backed up incorrectly in horizon $h_1 = 1$. This means that the backed-up value of s relies upon the incorrect value of one of its children, c. We know that when c receives its first update in horizon $h_c \geq h_1$, s will be queued for backup again in horizon $h_c + 1 \geq h_1 + 1$.

We have just shown that if a state s is backed up incorrectly in horizon h_1, s will be backed up in horizon $h_s \geq h_1 + 1$. Suppose it's true for some $k \leq 1$ that if a state is backed up incorrectly in horizon $i \leq k$, that state will be backed up again horizon $h_s \geq i+1$. This is our inductive hypothesis. Then what about a state s which is backed up incorrectly in horizon $k+1$?

Again we know that the value of s depends on the incorrect value of some child of s, c. If c's last backup was in horizon $h_c \leq k$, then by the inductive hypothesis, c will receive, or has received, another backup in horizon $h_{c'} \geq h_c + 1$. By repeated application of the inductive hypothesis, we further know that c will receive a backup in horizon $h_{c*} \geq k+1$, at which time s will be queued to receive a backup in horizon $h_s \geq k+2$. If c has not received a backup in by the end of horizon k we know by Lemma 2 that it will eventually, and must in horizon $h_c \geq k+1$. When that occurs, state s will be added in to the queue for backup one horizon afterwards.

We have shown that if the inductive hypothesis is true for all horizons $1 \leq i \leq k$, then it is true for horizon $k+1$ and consequently for all horizons $i \geq 1$. □

PROOF FOR RVI. In RVI, a parent state p is queued for backup only when the values of one or more of its children change. If the values of the children do not change, the value of p would not change even if it were backed up. By Lemma 3, we can omit such an update to p without affecting the values of any other states. Thus if an incorrect backup would later be corrected in RVI′ (as we have shown), it will be corrected in RVI as well. □

THEOREM 1. *After RVI terminates all reachable states will have the correct values.*

PROOF. We know by Lemma 2 that all reachable states receive at least one backup. We know by Lemma 4 that if a backup is incorrect, the state will receive another backup before RVI finishes. Thus, RVI will not terminate while any states have incorrect values. □

THEOREM 2. *RVI will converge to V^**

PROOF. RVI can be considered a form of asynchronous value iteration. Convergence is guaranteed for such algorithms provided that every state is backed up infinitely often [1], [5]. We note however that once the computed value of a state has reached $V^*(s)$ it can be omitted from further backups without harm. Thus we can consider the convergence to hold as long as incorrect states (states whose computed values are not equal to $V^*(s)$) are guaranteed to get future updates for any and all points in time. Since, Lemma 4 guarantees us exactly this, RVI must converge to V^*. □

5.2 Convergence Speed

Let us denote the initial value function in which all states are initialized to zero as V_0. RVI creates a series of value functions, one after each backup. Let us consider the value functions generated after all states of any one horizon has been backed up and before the next horizon has begun. We will denote the the value function at the end of horizon h_k by V_k. If RVI runs for n iterations, V_n is the final value function returned. We are guaranteed by the previous theorems that V_n converges to V^*.

Recall the Bellman (optimality) equation governing the value of a state:

$$V^*(s) = max_a E_{s'}(R(s,a,s') + \gamma V^*(s'))$$

Let us consider these equations in terms of horizon.

$$\begin{aligned} V^0 &= max_a E_{s'} R(s,a,s') \\ V^1 &= max_a E_{s'}(R(s,a,s') + \gamma V^0(s')) \\ V^k &= max_a E_{s'}(R(s,a,s') + \gamma V^{k-1}(s')) \\ V^{\inf} &= V^*(s) \end{aligned}$$

Note that for a terminal or absorbing state s, $V^0(s) = V^*(s) = 0$. Absorbing states are states with a single self-loop of zero reward. Thus, no matter how far one looks into the future (horizon) the result is still zero.

In some cases, for a state s, $V^k(s) = V^*(s)$. Consider a state that is adjacent to a terminal state s_t and whose optimal action is the one which takes it to s_t with certainty. Due to the absorbing nature of the terminal state, a horizon of one is sufficient to calculate its value, $V^*(s)$. More formally:

$$\begin{aligned} V^1(s) &= max_a(E_{s'}(R(s,a,s') + \gamma V^0(s')) \\ &= R(s,a,s_t) + \gamma V^0(s_t) \\ &= R(s,a,s_t) + \gamma V^1(s_t) \\ &= R(s,a,s_t) + \gamma V^*(s_t) \\ &= V^*(s) \end{aligned}$$

We will call states with this property *h-complete*, meaning $V^h(s) = V^*(s)$.

LEMMA 5. *If horizon h is the largest horizon that a state s appears in, then s is h-complete and its value will be correct after horizon h. Further, state s will never receive a changing backup after horizon h so its value will stay correct for all subsequent horizons.*

PROOF. We use an induction proof. When we refer to states "appearing" we mean appearing in an optimal path of the MDP.

Base case ($k = 0$): Suppose horizon 0 is the largest horizon that a state s appears in. Since optimal paths must end in a terminal state, we know s must be a terminal state. The value of terminal states is always zero for all horizons. That is, $V^0(s) = V^*(s)$. RVI initializes the value of all terminal states to zero and does not backup terminal states. Thus its value after horizon 0 is correct and will stay correct for all horizons greater than 0.

Inductive case: Consider a state s in which horizon h is the largest horizon it appears in. We will denote the set of optimal paths in which this occurs as P. Let s' be any next state along a path $p \in P$. The largest horizon s' appears in must be horizon $h-1$ or earlier. (If s' were to appear in a larger horizon we could have constructed a longer optimal path for s ending in a later horizon.) By the inductive hypothesis, $V(s') = V^*(s')$ and further must remain at $V^*(s)$. Thus after state s is backed up it must have value $V^*(s)$ and also remain that way.

At the time state s receives its backup in horizon h, all of its children (which must have received their last backup in horizon $h-1$ or earlier) are correct by the inductive hypothesis. Thus the backup state s receives in h will set is value to $V^*(s)$. Since the inductive hypothesis also guarantees us that the children will stay correct, the value of state s must stay correct as well. □

THEOREM 3. *Let R_{max} denote the largest reward. Then for any precision $\varepsilon > 0$, there exists a longest maximum path whose length is $L \leq log_{\gamma}(\varepsilon/R_{max})$. Further, RVI will converge after horizon $L+1$.*

PROOF. A consequence of γ being less than 1 and a fixed precision ε is the existence of a upper bound on the length to any optimal path. More formally, because $||V^k(s) - V^{k-1}(s)|| \leq \gamma^k R_{max}$, as the horizon $k \to \infty$ the difference between $V^k(s)$ and $V^{k-1}(s)$ becomes arbitrarily small. Thus for any given ε, we can solve for the horizon $k = log_{\gamma}(\varepsilon/R_{max})$ in which any changes from value updates must be below the precision level. Any optimal paths longer than k then, do not have to be considered as they will have negligible impact. Thus k or as we will name it, L, can be considered an upper bound on the maximal length of the MDP.

The largest horizon h that a state s appears in (on an optimal path) must be less than or equal to L. By Lemma 5, after horizon $h \leq L$, the value of state s will be correct and stay correct. Thus by horizon $L+1$, we know no backups can generate value changing updates as all values are already correct and must stay correct. Since the parents of non-changing backups are not added to the queue, the queue must be empty after horizon $L+1$ (if not sooner) and RVI must terminate. □

6. COMPLEXITY

RVI works much like value iteration. All backups with the same priority corresponds to a horizon and can be compared to an iteration in value iteration. By Theorem 3 RVI converges after roughly $log_{\gamma}(\varepsilon/R_{max})$ horizons (iterations) giving it the same convergence speed as value iteration.

Any difference must then lie with the iterations. For value iteration, each iteration requires N backups where N is the number of states. Since each backup is $O(MN)$ where M is the number of actions, the complexity of each iteration is $O(MN^2)$.

RVI has an initial overhead associated with building the parent table of $O(N^2)$. Since each iteration of value iteration is already $O(MN^2)$ we will ignore this overhead. The complexity of RVI then depends on the size of its iterations (horizons). In general, this is problem dependent but we note that it can be no worse than value iteration in which each iteration contains all states. To give the reader some intuition of possible horizon sizes, we provide an illustrative example. Consider a deterministic, circular gridworld. The largest horizon size can be characterized by the circumference and the number of states in the world by the area. When compared, this yields a maximum horizon size that is roughly r times smaller than the number of states (where r is the radius). This means RVI will be roughly r time faster on such domains. This holds similarly for circles of higher dimensions (*e.g.* spheres).

7. ANALYSIS

In the previous section, we saw that while RVI runs for roughly as many iterations as value iteration, each iteration (horizon) is often considerably smaller. In this section, we will pursue where the gains RVI might yield come from. We hypothesize three areas from which RVI might extract savings:

Improved ordering which results in fewer wasted backups. If we knew the set of optimal paths *apriori*, we could compute the optimal ordering and simply perform backups according to that ordering. However, this is rarely the case. RVI thus must follow the ordering of all paths in general. This maintains the property that children will be backed up before their parents so we expect some gains. However, as suboptimal paths often yield suboptimal orderings, wasted backups will be inevitable. When there are terminal states, RVI can cut down on the number of suboptimal paths traversed by considering only paths that end in a terminal state. We can compare this to prioritization methods for value iteration, which also aim to improve the ordering. Prioritization methods use heuristics to guess the optimal ordering. Depending on the heuristics used, this can lead to arbitrary speed-ups and slow-downs. RVI follows the ordering of a set of paths known to contain the optimal paths and thus the optimal ordering. From a prioritization point of view it can be considered a well bounded heuristic.

Propagation of changes in value stop as soon as updates stop affecting the values of other depended-upon states. Value iteration updates every state in each iteration, regardless of whether or not it might actually benefit from an update. In RVI, if an update fails to change the value of a state, then subsequent backups to its ancestors are not performed. This can yield large savings as it enables traversal of suboptimal paths to stop early.

Updates are never performed on unreachable states. Since ordering is based on traversal starting from the end of paths, we are guaranteed to never update states which cannot reach termination states. This means RVI is sensitive to the discount factor γ as that implicitly encodes a maximum length. In particular, RVI will not update states beyond the maximum length meaning if γ is small, RVI will only need to visit the (relatively) small number of states around the termination points.

Understanding where RVI saves on backups allows us to characterize the classes of MDPs it should work best for and those in which its gains are less pronounced. RVI should work best on MDPs with the following characteristics.

- Mostly deterministic domains. RVI extracts savings from stopping propagation when values of states stop changing. This keeps the number of states per horizon down. Nondeterminism makes this difficult as it increases the number of children supporting the correct value of a state. If any one of the children change in value the parent must be re-added to the queue for backup.

- Low connectivity. The parents of states whose values have changed are added into the next horizon. With low connectivity, this in-degree will be low as well meaning horizon sizes will tend to grow slower if at all.

- Low γ. A low γ, with respect to a fixed ε, significantly decreases the number of reachable states, which RVI can take great advantage of.

- MDPs with terminal states. Terminal states allow RVI to narrow the set of paths it must traverse to only those that end in a terminal state. This improves ordering and helps to keep horizon sizes low.

The reverse then is also true. RVI's performance gains will be least significant in highly non-deterministic, heavily connected, high gamma MDPs with no terminal states. We note however that RVI, unlike heuristic based prioritization techniques, is never worse than value iteration. As a simple example, consider a worst case scenario of a fully connected MDP with random rewards. In such a case RVI will perform (almost) identically to value iteration. The first horizon will contain all states as there are no terminal states, and subsequent horizons will contain all states as the changing value of any one state will guarantee the addition of all states into the next horizon.

Notice that many of the characteristics mentioned above are ones that are general indicators of MDP difficulty. One way to think of RVI and its performance characteristics is that RVI is an algorithm that takes advantage of MDPs that are actually easier than they appear. That is, RVI solves easy problems fast and hard problems slow whereas value iteration solves easy problems slowly and hard problems even more slowly.

8. EXPERIMENTS

We performed a thorough empirical evaluation of RVI over a series of experiments designed to explore its behavior with respect to changing world size (number of states), differing amounts of non-determinism, presence of terminal states, changing discount factor and other value iteration improvement techniques. Each of these experiments are covered in detail in the following subsections.

Note that our evaluation is done in terms of number of backups as timing is dependent on implementation language, evaluation machine characteristics and timing tool used. In cases in which those factors are constant, for example, when comparing our implementation of RVI and VI, we note that running times have been consistent with the number of backups. As an example, on a 1000 by 1000 grid world, value iteration ran a billion backups, taking over a thousand seconds. In comparison, RVI ran about two million backups, taking roughly three seconds. Speedup both in terms of number of backups (approximately 500x) as well as running time (approximately 350x) is about three orders of magnitude.

8.1 Scaling over world size

In this experiment, we examine how RVI scales over increasing number of states. Value iteration will serve as the control for comparison. We use a plain, two dimensional deterministic gridworld. There are no obstacles. There is one terminal state. It is placed in the cell in the center of the grid. All rewards are set to a uniform -1. The discount factor γ is set to 0.999, the precision ε is set to 0.1.

Results are shown in Figure 3. Note the log-log scale. We see a large speedup with RVI. Further RVI has a shallower slope. In fact, RVI scales linearly with the number of states. Since value iteration does not, the resulting difference only grows. Initially RVI is only about two orders of magnitude faster, but by a million states, the difference has increased to three orders of magnitude.

8.2 Effects of non-determinism

We argued in section 7 that RVI should have the most prominent gains in deterministic domains. This experiment seeks to empirically measure this effect. We use the same gridworld setup as in the previous experiment, with two differences. First, the grid size is set to 100 by 100. Second, cells are no longer plain ones of deterministic transitions. Instead, a percentage of cells are turned into random cells, in which any action leads to any of its adjacent neighbors, with uniform probability. The percentage of random cells will simulate differing amounts of non-determinism. The number of backups that RVI and value iteration take are the response variables.

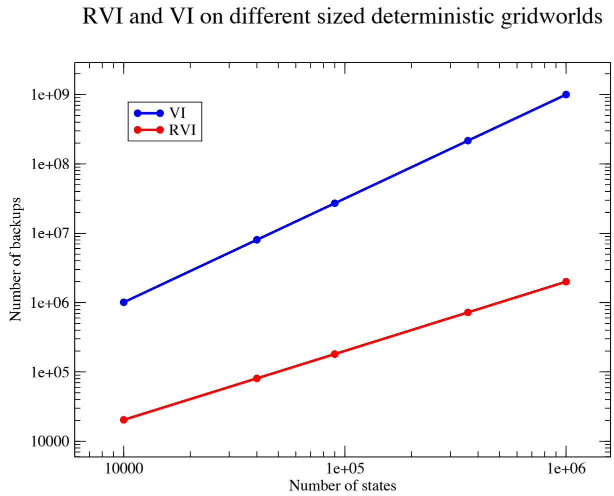


Figure 3

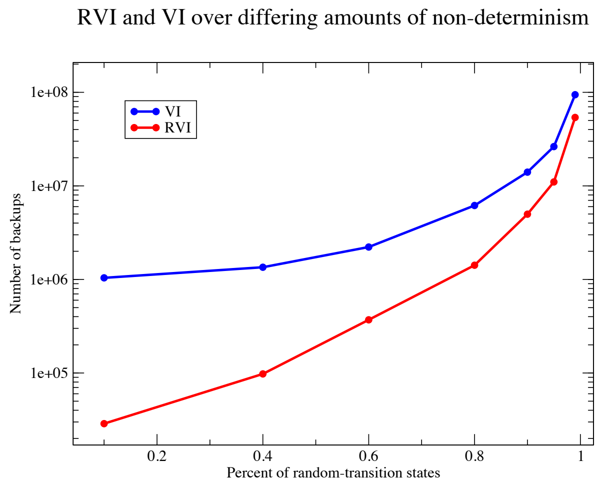


Figure 4

Results are shown in Figure 4. As we can see, the amount of non-determinism has a strong influence on the efficacy of RVI. While initially, when the world is deterministic, RVI yields a two order of magnitude speedup, by the time half the world states are random, the speed up is a single order of magnitude. In the most contrived case, when (almost) all states are random, RVI is only about two times faster than value iteration.

8.3 Presence of terminal states

One source of speedup RVI enjoys is the presence of terminal states which enables us to narrow the set of paths we must traverse backwards. In this experiment we seek to gain a feel for how important this condition is. Does RVI still yield significant benefits when there are no terminal states? For this experiment we used the same setup as in section 8.1, with one exception: γ has been raised to 0.9995 to ensure that all paths are considered and not prematurely cut off by the discount factor.

Figure 5 shows that similar to the terminal state case, RVI yields large speedups. However, the amount of speedup is stable across

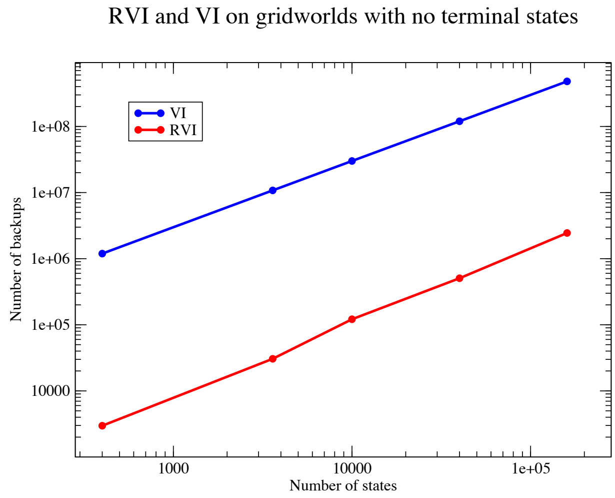


Figure 5

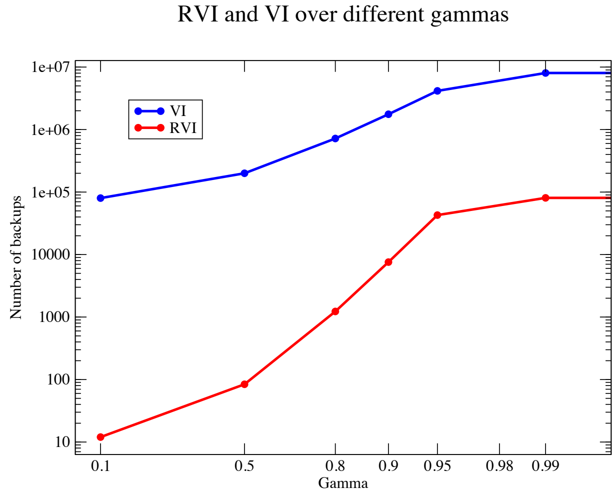


Figure 6

different number of states. RVI has the same slope as VI. It is consistently about two orders of magnitude faster. This result implies that the pruning of paths that RVI performs using the terminal states has increasing yields with the number of states. This is consistent with our understanding. As the number of states increase, the number of non-optimal paths increases much faster. Thus, so must the savings that is gained from pruning paths that do not end in a terminal state. As the number of terminal states increases such that the amount pruned is reduced, we expect this source of speedup to be less significant.

8.4 Response to the discount factor

Here we seek to empirically measure how RVI responds to differing discount factors. If our argument in Section 7 is correct, RVI should yield better gains when γ is low. We use the same gridworld setup as in Section 8.1, with two differences. First, the world is set to 200 by 200. Second, γ will vary.

Figure 6 reflects our general sentiment that RVI works best with a low discount factor. When $\gamma = 0.1$, RVI is faster by four orders of magnitude. As γ is increased, both RVI and VI take longer, as the problem becomes inherently more difficult. A larger γ means longer and longer paths must be taken into account when computing the value of a state. However, RVI is able to extract less gains with increasing γ. By $\gamma = 0.95$, RVI yields a two order of magnitude speedup. Shortly after this point though, the curve for RVI and VI level off and further increases to γ have little effect on the number of backups. This is because the MDP used has an intrinsic maximum path length. In a 200 by 200 grid world, the longest optimal path is roughly 200 steps. Thus, we expect the curve to level off when the maximum path length that the γ supports approaches 200 steps. With a γ of 0.99, the 200^{th} step is discounted at 0.13, which is above our precision level, $\varepsilon = 0.1$. Thus we expect raising γ beyond 0.99 to have little impact. Sure enough, the curve is virtually flat past $\gamma = 0.99$.

8.5 Comparison to other methods

While we have explored different aspects of RVI's performance on various synthetic gridworld domains, we have not compared it with respect to modern value iteration techniques. That will be our focus in this section. The most effective value iteration speedup techniques to date that the authors are aware of is Wingate's work in partitioned and prioritized value iteration [15], [14], [17]. Thus, comparisons here will be against his GPS (General Prioritized Solvers) family of algorithms. Again, comparison will be in terms of number of backups to control for machine, language, and timing differences.

To ensure comparisons are performed under the same conditions and specifically, the same world dynamics we us three different MDPs from Wingate's work: mountain car (MCAR), single-armed pendulum (SAP) and double-armed pendulum (DAP). We provide a brief description of each world below. Details can be found in [17]. We used the same parameters, $\gamma = 0.9$ and $\varepsilon = 0.0001$ as was reported in [14].

Mountain car is the well known two-dimensional, continuous, minimum-time optimal control problem. A car must rock back and forth until it gains enough momentum to carry itself atop a hill. All rewards are zero except for the final reward for succeeding at the task, which is one. The state space is a combination of position and velocity. There are three possible actions: full throttle forward, full throttle reverse, and zero throttle. A 90,000-state discretization of this continuous domain was used.

The single-arm pendulum (SAP) is also a two dimensional, continuous, minimum time optimal control problem. The objective is to learn to swing up the pendulum and balance it. The agent has two actions available, a positive and a negative torques. Similar to MCAR, torques available are underpowered. Reward is set to zero except for the balanced region which has reward one. The state space is composed of the angle of the link and its angular velocity. A discretization of 160,000 states was used.

The double-arm pendulum (DAP) is a two-link (and thus four dimensional) variant of SAP. Reward is setup similarly. The state space is composed of the two link angles and their respective angular velocities. A discretization of 810,000 states was used.

GPS is a family of algorithms. We compare against the two variants, one that uses the *H1* heuristic, and one that uses the *H2* heuristic with voting. See [14] and [17] for details. These two variants were chosen such that at least one is the best (or near best) for all experiments reported in the paper.

On the first two domains RVI performs the best. It is three orders of magnitude faster than value iteration and an order of magnitude faster than the best GPS technique. This demonstrates the strength of our approach.

In the last domain, while still faster than value iteration, RVI

Table 1: Comparison of RVI, Value Iteration, and two GPS variants. The unit is millions of backups

Test World	*RVI*	*Value Iteration*	*H1*	*H2 + voting*
MCAR	0.2	30	6	2
SAP	0.4	30	15	2
DAP	29	40	18	26

is slower than *H1* and also slightly slower than *H2 + voting*. A detailed examination of the DAP domain explains why. In DAP, due to the coarseness of the discretization, the world is highly non-deterministic and connected. Some states can be reached by as many as 19 different states. This leads to quickly-growing horizon sizes. States are added on to subsequent horizons quickly due to the large branching factor and they are not removed because changes in later horizons tend to affect states in previous horizons (due to the non-determinism) requiring them to be re-added. On DAP, by horizon 30, the number of states had reached 800,000 – roughly the total number of states. Since RVI runs for roughly as many iterations as value iteration and saves primarily from smaller horizon (iteration) sizes it is not difficult to see why RVI was only 25 percent faster.

Where RVI fails however, is a perfect place for integration with other speedup techniques. RVI behaves very similar to value iteration. It simply changes which states (and how many) are in which iteration. Since gains are made by small horizons, large horizons provide a natural place to use prioritized and partitioned backup schemes. While integration with other techniques and associated empirical evaluation is beyond the scope of this paper, we are optimistic that it will bring the best of both worlds together to yield further speedups.

9. CONCLUSION

Traditional value iteration is often slow due to the many useless, non-informative backups which it performs. In this paper we developed a new algorithm, RVI, that reorders backups to avoid as many of these as possible. RVI reorders based on systematically traversing paths in reverse order, in an effort to back up children before their parents.

We proved that RVI converges to the optimal value function, and with a similar speed in terms of number of iterations as value iteration. However, due to the much smaller iteration sizes RVI is able to use, it typically has a lower running time complexity.

In empirical evaluations, this bore out as RVI often improved performance by several orders of magnitude when compared to value iteration. On some of the harder cases, RVI was only able to improve by 25%. To explore why this might be the case, we covered where RVI extracts its savings and provide characterizations of the types of MDPs RVI works best on. We note that RVI can never do worse than value iteration.

When compared to other techniques for speeding up value iteration, our experiments showed mix results. In some domains, RVI out performed prioritized and partitioned approaches by an order of magnitude. In others it was twice as slow. A fruitful outcome of this comparison beyond the numbers is an understanding that our approach is complimentary to previous ones and we point out a way that other techniques can be used in conjunction with RVI. While no experiments were performed on such a merging, it will be the focus of our future work.

10. REFERENCES

[1] D. P. Bertsekas and J. N. Tsitsiklis. Convergence rate and termination of asynchronous iterative algorithms. In *ICS '89: Proceedings of the 3rd international conference on Supercomputing*, pages 461–470, New York, NY, USA, 1989. ACM.

[2] R. I. Brafman and M. Tennenholtz. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. In *IJCAI*, pages 953–958, 2001.

[3] D. Chapman and L. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *IJCAI*, pages 726–731, 1991.

[4] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, pages 746–752, 1998.

[5] V. Gullapalli and A. G. Barto. Convergence of indirect adaptive asynchronous value iteration algorithms. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 695–702. Morgan Kaufmann Publishers, Inc., 1994.

[6] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Proc. 15th International Conf. on Machine Learning*, pages 260–268. Morgan Kaufmann, San Francisco, CA, 1998.

[7] T. Makino and K. Aihara. Multi-agent reinforcement learning algorithm to handle beliefs of other agents' policies and embedded beliefs. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 789–791, New York, NY, USA, 2006. ACM.

[8] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[9] R. Munos and A. W. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *IJCAI*, pages 1348–1355, 1999.

[10] M. T. Rosenstein and A. G. Barto. Robot weightlifting by direct policy search. In *IJCAI*, pages 839–846, 2001.

[11] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[12] R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[13] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative learning. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages 487–494. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[14] D. Wingate. Solving large mdp quickly with partitioned value iteration. Master's thesis, 2004.

[15] D. Wingate and K. D. Seppi. Efficient value iteration using partitioned models. In *ICMLA*, pages 53–59, 2003.

[16] D. Wingate and K. D. Seppi. P3vi: a partitioned, prioritized, parallel value iterator. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 109, New York, NY, USA, 2004. ACM.

[17] D. Wingate and K. D. Seppi. Prioritization methods for accelerating mdp solvers. *Journal of Machine Learning Research*, 6:851–881, 2005.