# Scaling Up: Solving POMDPs through Value Based Clustering

**Yan Virin** and **Guy Shani** and **Solomon Eyal Shimony** and **Ronen Brafman**[1]
Department of Computer Science, Ben-Gurion University
P. O. Box 653, 84015 Beer-Sheva, Israel
{virin,shanigu,shimony,brafman}@cs.bgu.ac.il

## Abstract

Partially Observable Markov Decision Processes (POMDPs) provide an appropriately rich model for agents operating under partial knowledge of the environment. Since finding an optimal POMDP policy is intractable, approximation techniques have been a main focus of research, among them point-based algorithms, which scale up relatively well - up to thousands of states. An important decision in a point-based algorithm is the order of backup operations over belief states. Prioritization techniques for ordering the sequence of backup operations reduce the number of needed backups considerably, but involve significant overhead. This paper suggests a new way to order backups, based on a soft clustering of the belief space. Our novel soft clustering method relies on the solution of the underlying MDP. Empirical evaluation verifies that our method rapidly computes a good order of backups, showing orders of magnitude improvement in runtime over a number of benchmarks.

## Introduction

Realistic environments require use of models that capture both uncertainty in the results of actions and partial observability. Partially Observable Markov Decision Process (POMDP) is a well-studied model that uses belief state to represent the state uncertainty. However, a major impediment to ubiquitous usage of POMDPs is that finding an optimal solution for these models is computationally intractable, and indeed solvers computing exact solutions do not in general scale up to domains with more than a handful of states.

Therefore, significant research effort is focused on computing *approximately* optimal policies. Approximation algorithms appear to supply good policies rapidly and some show the ability to solve problems with tens of thousands of states. This paper studies a novel approximation scheme that is shown empirically to outperform state of the art algorithms by orders of magnitude.

A well known approach to computing a policy is through a *value function*, specifying a value for each belief state. Such a value function can be represented by a set of $\alpha$-vectors (Smallwood & Sondik 1973). A recent promising approach for finding value function approximations is the point-based approach (Lovejoy 1991; Pineau, Gordon, & Thrun 2003), where only a small set of reachable belief-points is used for value function computation. A value function in a point-based algorithm is computed through a sequence of *backup* operations over single belief-states. The backup operation computes a new $\alpha$-vector which can be added to the value function, potentially improving not only the value of the belief-state itself, but also the values of other belief-states.

Point-based algorithms differ in two dimensions; The set of belief points the algorithm uses, and the order of backup operations over the belief point set. Methods for belief set selection include an expanding set attempting to cover the entire reachable belief space (Pineau, Gordon, & Thrun 2003), executing trials to gather sequences of belief states (Smith & Simmons 2004), and pre-computing through a heuristic walk over a fixed set of belief points. In this paper we focus on algorithms from the last family, that use a fixed set. Different algorithms also choose different ways to order backups over the selected belief states. It was previously shown that a good sequence of backups can converge to an optimal value function faster. However, finding good sequences typically incurs considerable computational overhead; hence even though the number of backups is reduced, in most such schemes the benefit does not fully manifest in total execution time.

In this paper we show how good sequences can be obtained cheaply by solving the underlying MDP, and aggregating states into clusters according to their optimal MDP value. The clusters are then projected onto the POMDP, resulting in a soft clustering over the belief space. A suitably defined POMDP cluster value is used, and our scheme iterates over clusters in order of decreasing value. We select belief states for backup based on their probability of belonging to the current cluster. Our experiments follow Shani et. al. (Shani, Brafman, & Shimony 2006), using the same evaluation system and thus allowing an accurate comparison to previous point-based algorithms results. The experiments confirm that our algorithm scales up better than state of the art algorithms on a set of well known benchmarks.

The rest of the paper is structured as follows; We begin with required background on MDPs and POMDPs, various point-based approaches, abstraction techniques in general, and clustering specifically. Then, we present our Soft Clus-



[1]Partially supported by the Lynn and William Frankel center for computer science, and by the Paul Ivanier center for robotics and production management.

tering Value Iteration approach, followed by a set of experiments demonstrating its strength in comparison to other well known point-based algorithms.

## Background and Related Work

### MDPs

A Markov Decision Process (MDP) is a tuple $\langle S, A, tr, R\rangle$ where $S$ is a set of world states, $A$ is a set of actions, $tr(s, a, s')$ is the probability of transitioning from state $s$ to state $s'$ using action $a$, and $R(s, a)$ defines the reward for executing action $a$ in state $s$. An MDP models an agent that can directly observe its state in the environment.

A solution to an MDP can be computed through a value function — a function that assigns value $V(s)$ to each state $s$. We can define so called $Q$-values — assigning a value $Q(s, a)$ to each state and action pair. The value for the state is therefore $V(s) = max_a Q(s, a)$. Algorithm 1 computes an optimal value function for an MDP.

**Algorithm 1** Value Iteration

```
Initialize ∀s ∈ S, a ∈ A, Q(s, a) = 0, V(S) = 0
repeat
    for each s ∈ S do
        for each a ∈ A do
            Q(s, a) = R(s, a) + γ Σ_{s'∈S} tr(s, a, s')V(s')
        V(s) = max_a Q(s, a)
until convergence
```

We denote by $Q*$ and $V*$ — the optimal $Q$-function and value functions, respectively.

### POMDPs

A Partially Observable MDP (POMDP) is a tuple $\langle S, A, tr, R, \Omega, O, b_0\rangle$ where $S, A, tr$ and $R$ define an MDP, known as the *underlying-MDP*, $\Omega$ is a set of observations and $O(a, s, o)$ is the probability of observing $o$ after executing $a$ and reaching state $s$. POMDPs are more suitable than MDPs for realistic agents, such as robots, that do not have direct access to the current state of world, but rather observe the world through a set of noisy sensors. The agent hence must maintain a *belief* over its current state — a vector $b$ of probabilities such that $b(s)$ is the probability that the agent is at state $s$. Such a vector is known as a belief state or *belief point*. $b_0$ defines the initial belief state before the agent has executed an action or received an observation.

The transition from belief state $b$ to belief state $b'$ using action $a$ is deterministic given an observation $o$ and defines the $\tau$ transition function. That is, we denote $b' = \tau(b, a, o)$ where:

$$b'(s') = \frac{O(a, s', o)\sum_s b(s)tr(s, a, s')}{pr(o|b, a)} \quad (1)$$

$$pr(o|b, a) = \sum_s b(s)\sum_{s'} tr(s, a, s')O(a, s', o) \quad (2)$$

### Value Functions for POMDPs

It is well known that the value function $V$ for the belief-space MDP can be represented as a finite collection of $|S|$-dimensional vectors known as $\alpha$ vectors. Thus, $V$ is both piecewise linear and convex (Smallwood & Sondik 1973). A policy over the belief space is defined by associating an action $a$ to each vector $\alpha$, so that $\alpha \cdot b = \sum_s \alpha(s)b(s)$ represents the value of taking $a$ in belief state $b$ and following the policy afterwards. It is therefore standard practice to compute a value function — a set $V$ of $\alpha$ vectors. The policy $\pi_V$ is immediately derivable using:

$$\pi_V(b) = \text{argmax}_{a:\alpha_a \in V}\, \alpha_a \cdot b \quad (3)$$

The value function can be iteratively computed

$$V_{n+1}(b) = \max_a [b \cdot r_a + \gamma \sum_o pr(o|a, b)V_n(\tau(b, a, o))] \quad (4)$$

where $r_a(s) = R(s, a)$ is a vector representation of the reward function. The computation of the next value function $V_{n+1}(b)$ out of the current $V_n$ (Equation 4) is known as a *backup* step, and can be efficiently implemented (Cassandra, Littman, & Zhang 1997; Pineau, Gordon, & Thrun 2003) by:

$$g^\alpha_{a,o}(s) = \sum_{s'} O(a, s', o)tr(s, a, s')\alpha^i(s') \quad (5)$$

$$g^b_a = r_a + \gamma \sum_o \text{argmax}_{g^\alpha_{a,o}:\alpha \in V}\, b \cdot g^\alpha_{a,o} \quad (6)$$

$$backup(b) = \text{argmax}_{g^b_a : a \in A}\, b \cdot g^b_a \quad (7)$$

### Point Based Algorithms

Computing an optimal value function over the entire belief space does not seem a feasible approach. A possible approximation is to compute an optimal value function over a subset of the belief space (Lovejoy 1991). Note that an optimal value function for a subset of the belief space is no more than an approximation of a full solution. These schemes assume that the computed value function will generalize well for unobserved belief states.

Point-based algorithms (Pineau, Gordon, & Thrun 2003; Spaan & Vlassis 2005; Shani, Brafman, & Shimony 2006) choose a subset of the belief points reachable from the initial belief state and compute a value function only over these belief points. Such algorithms differ by the way they choose the set of belief points, and by the way they order backup operations over these points.

Spaan and Vlassis (Spaan & Vlassis 2005) explore the world randomly, gathering a set $B$ of belief points, and then execute the Perseus algorithm. Perseus iterates over the set $B$, each time selecting a random belief point to update. Once a belief point has been updated, all other belief points whose value has changed are removed from the set for the current iteration. Once the belief point set has emptied, all the original belief points are restored and another iteration is performed. Perseus appears to provide good approximations with small sized value functions rapidly. However, it is very stochastic due to the random selection of belief points and the random selection of backup operations. These random selections cause high variation in performance and in more complicated problems may cause the algorithm to fail to converge at all. Also, for larger domains, random selections seem to provide less useful backups. Therefore, as the domain size grows, Perseus is no longer efficient.

HSVI (Smith & Simmons 2004) uses a different strategy to find belief states to backup. HSVI is a trial-based algorithm that finds traversals through belief space. Once a traversal has reached its end, belief states within the traversal are updated in reversed order. HSVI finds good trajectories and updating them from end to start provides good sequences of backups. However, HSVI uses an upper bound over the value function as a heuristic for selecting the coming belief point and determining the end of the traversal. Maintaining this upper bound requires considerable overhead, thus the reduced number of backups does not fully manifest in the runtime.

### The underlying MDP optimal solution

Using the underlying MDP optimal value function as an aid in POMDP solution is a well known idea, various aspects of which were explored in the past. Littman et al. (Littman, Cassandra, & Kaelbling 1995) suggest to use the optimal $Q$-values of the underlying MDP to create the $Q_{MDP}$ value function for a POMDP:

$$Q_{MDP}(b) = \max_a Q(s,a)b(s) \quad (8)$$

Many grid-based techniques (e.g. (Zhou & Hansen 2001)) initialize the upper bound over the value function using the underlying MDP. Bonet et al. (Bonet & H.Gefner 1998) initialize a $Q$ function for the POMDP using the optimal $Q$ function for the MDP. Algorithms that use forward search over the belief space to determine the best action online (e.g. (Paquet, Tobin, & Chaib-draa 2005)) use the MDP $Q$ function to order the search tree expansion.

To find the optimal value function for the underlying MDP we must solve it, but this additional computation time is typically negligible compared to POMDP solution time, both theoretically and as shown empirically in our experiments.

## Prioritizing MDP Solvers

MDP value iteration (Algorithm 1) computes a value function by iterating over the state space, updating the value of each state. The order of state updates, however, has a significant impact on the speed of convergence of the value function $V$. For example, updating the values of the successors $s' : tr(s,a,s') \neq 0$ of state $s$ prior to updating $s$ itself will reduce the overall number of updates. Also, in many cases it is not necessary to update all the states in each iteration. In fact, some states need to be updated more often and some states should be updated later than others. It is possible to formalize such an approach through priorities, where each state is assigned a dynamic priority, and states are updated in order of decreasing priorities. Algorithm 2 presents a general prioritized value iteration.

Wingate and Seppi (Wingate & Seppi 2005) investigated the implementations of the prioritized value iteration algorithm for MDPs. They use a number of prioritization techniques based on the well known Bellman error:

$$e(s) = \max_a [R(s,a) + \gamma \sum_{s' \in S} tr(s,a,s')V(s')] - V(s) \quad (9)$$

measuring the possible change for the value of $s$ from executing the next backup.

**Algorithm 2** Prioritized Value Iteration

1: Initialize $\forall s \in S, V(S) = 0$
2: Initialize state priorities
3: **repeat**
4: $s \leftarrow$ state with maximal priority
5: $V(s) = \max_a R(s,a) + \gamma \sum_{s' \in S} tr(s,a,s')V(s')$
6: Update the priorities of the predecessors of $s$
7: **until** convergence

In an MDP context, the Bellman error is useful because it is easily computed and updated through the value iteration process. Each time a state value is updated we only need to recompute the Bellman error for its predecessors.

Nevertheless, Wingate and Seppi conclude that prioritization itself is insufficient in scaling up to solve larger MDPs. They therefore suggest to split the state space into clusters of states. Each cluster has a priority, and states within a cluster have priorities. They select a cluster and update the values of states inside the cluster until they converge, only then moving to a different cluster. They introduce the General Prioritized Solver algorithm (GPS — Algorithm 3). To cluster the states Wingate and Seppi use the transition function or some geometrical knowledge.

### Prioritizing POMDP Solvers

Shani et al. (Shani, Brafman, & Shimony 2006) adopt priorities to POMDP point-based solvers. They use a fixed set of belief points and compute priorities for these belief points through the Bellman error. Unfortunately, computing the Bellman error for POMDPs cannot be easily done due to two major difficulties: First, the set of predecessor belief points cannot be computed and is potentially infinite,

because there can exist a belief-point reachable from all the others. It is therefore impossible to update the Bellman error only for the predecessors of a belief point. Second, the backup process,

updating the value of a belief point, computes a new $\alpha$-vector, that may change the values of many belief points. It is therefore difficult to even find which belief points were updated following a single update (backup) operation.

Shani et al. overcome these difficulties by recomputing the Bellman error for a sampled subset of the belief points after each backup. Still, their algorithm does not scale up to handle large POMDPs. They show, however, that prioritization techniques can potentially considerably reduce the number of backups needed for value function convergence.

## SCVI - Soft Clustering Value Iteration

To overcome the difficulties in computing good sequences of backups, mostly the excessive overhead, we leverage off clustering. Our approach builds upon the scheme used in the GPS algorithm, adapting it to work for POMDPs. Such an adaptation is non-trivial, as the Bellman error is a poor candidate for a prioritization mechanism on POMDPs, and belief state transitions are difficult to compute and therefore cannot be used for efficient clustering. However, in a POMDP context we can use the underlying MDP to compute belief space clustering and order the sequence of backups.

**Algorithm 3** GPS

**Function Initialization**
1: Partition the problem
2: Order variables within each partition
3: Compute initial partition priorities

**Function GPS**
4: Initialization
5: **repeat**
6: Select a partition $p$
7: Compute current optimal policy and value function of states in $p$, keeping all other partitions constant
8: Recompute priorities of partitions depending on $p$
9: **until** convergence

Like Perseus and PVI we also use a predefined fixed set of belief points, and create a value function for these belief points only. We cluster these belief points and execute the GPS algorithm. We use a smart clustering algorithm, based on the underlying MDP optimal solution, that is both very fast and creates very good clusters. We define a soft clustering over the belief space and iterate over the belief states in order of decreasing relevance to the current cluster.

## Clustering the Belief Points

The $Q$-values of MDP states typically "trickle down" from states with high rewards to their neighbors. The final value of a state usually depends on its neighbors that have higher values. Therefore, in a MDP context, if optimal $Q$-values were known prior to value function computation, a good state update order would be by decreasing $Q$-value. To obtain $Q$-values one must first compute the value function, so this approach is clearly not feasible for MDPs. However, the idea of sorting MDP states according to their $Q$-values can be used for accelerating POMDP solvers.

We first solve the MDP, and then execute a clustering algorithm, where distance between states is the difference between their optimal $Q$-values. In our implementation we used the $K$-means algorithm, yet replacing the clustering algorithm should not be considered a significant change. The important aspect of the clustering process is the distance metric, not the specific algorithm. Figure 1 shows the clustered state space for the Hallway and Hallway2 domains. Given any hard clustering of the MDP states we can define a soft clustering of the POMDP belief states. As a belief state $b$ is a vector of state probabilities, we define the probability of $b$ belonging to cluster $c$ to be:

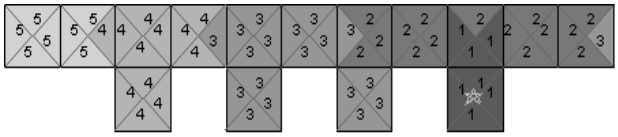
$$pr(b \in c) = \sum_{s \in c} b(s) \tag{10}$$

We collect a set $B$ of belief states using a random walk in belief space (as done by Perseus (Spaan & Vlassis 2005)), and define the soft clustering over these belief states.

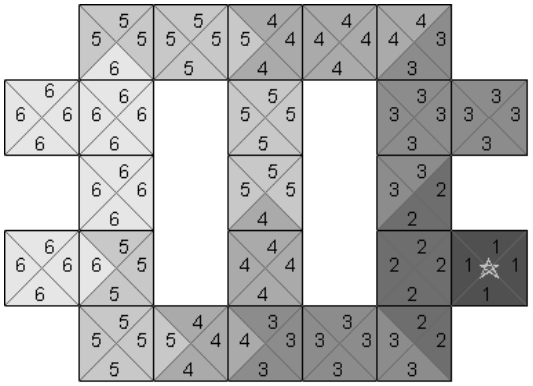
## Cluster and Belief Point Iteration

We define the value of a cluster to be the average of the $Q$-values of the states that belong to that cluster:

$$V(c) = \frac{\sum_{s \in c} max_a Q(s,a)}{|c|} \tag{11}$$



(a) Hallway



(b) Hallway2

Figure 1: Clustered state space for the Hallway and Hallway2 domains. Each state consists of being in a square and having a direction. The star indicates the goal state and the numbers indicate the order of the clusters in the SCVI algorithm.

We iterate over the clusters by order of decreasing cluster values. As we use soft clustering, each belief state belongs to all clusters with some probability. Alternatively, each cluster defines a distribution over belief space. Therefore, after choosing a cluster we iterate over the belief states in $B$ in decreasing order of probability of belonging to the current cluster, and update them, until some probability threshold has been reached. As clusters and soft clustering method are fixed, the order of belief points per cluster is static.

## SCVI

The Soft Clustering Value Iteration (SCVI) algorithm (concisely presented as Algorithm 4) works as follows:

- To initialize, compute the MDP optimal value function, to be used to determine the clustering, and a cluster value.
- Compute a hard clustering of the MDP state space using some clustering algorithm over the optimal $Q$-values. The partitioning process causes states with similar $Q$-values to be grouped together.
- Sample a set $B$ of belief states using a random walk or some heuristic selection over the belief space. We compute the value function only for the set $B$.
- Define a soft clustering for the belief states in $B$ using Equation 10.
- Sort the belief states in every cluster $c$ by $pr(b \in c)$ - decreasing probability of belonging to $c$. This order will remain fixed throughout the execution of the algorithm.
- Iterate over clusters in decreasing order of cluster value (Equation 11).

- For each cluster $c$ iterate over the belief states in $B$ in decreasing order of $pr(b \in c)$ and update their value using a point-based backup.

**Algorithm 4** Soft Clustering Value Iteration

Solve the underlying MDP to compute a $Q^*$ function
$P \leftarrow$ partition $S$ using the $Q^*$ function
Sample $B$ using a random walk
**for each** $c \in P$ **do**
  $B_c \leftarrow$ all belief states in $B$ sorted by $pr(b \in c)$
Sort clusters in $P$ by decreasing value
Initialize value function $V$ for the POMDP
**while** $V$ has not converged **do**
  **for each** $c \in P$ in decreasing value order **do**
    **for each** $b \in B_c$ in decreasing $pr(b \in c)$ order **do**
      $\alpha \leftarrow backup(b)$
      $add(V, \alpha)$

## Discussion

Shani et al. (Shani, Brafman, & Shimony 2006) show how good sequences of backups are considerably shorter, but spend much time in computing these sequences. The overhead of priority computation is a major component of the total runtime. Our clustering method is very fast, and the iteration over $B$ involves almost no effort, as the sequence is fixed and pre-computed.

SCVI can be considered as an implementation of the GPS algorithm for POMDPs. The idea of iterating over the clusters in decreasing cluster priorities is borrowed from Wingate and Seppi method, where it was applied to MDP. The ideas presented here for defining the clusters, as well as the adaptation of an MDP algorithm to POMDPs are novel. While GPS uses a hard clustering we use a soft clustering and belief state probabilities are determined by cluster relevance, rather than by the Bellman error prioritizing scheme. Hence, inner cluster ordering of belief states is static and does not involve additional sorting, unlike GPS where state priorities are dynamically updated.

Another very fast way to generate backup sequences is by decreasing $Q_{MDP}$ value of belief states. It is important to understand that there is a significant difference between this and our approaches. In SCVI a belief-point with weak relevance to a high-valued cluster is backed up before a belief-point with strong relevance to a low valued-cluster, while the $Q_{MDP}$ approach is not even capable of distinguishing between these two cases.

## Empirical Evaluation

To evaluate our SCVI algorithm, we compare it to three other state of the art point-based algorithms — Perseus (Spaan & Vlassis 2005), HSVI (Smith & Simmons 2004) and PVI (Shani, Brafman, & Shimony 2006), over a set of benchmark domains from POMDP literature. As SCVI attempts to compute good sequences of backups rapidly, we focus on the speed of convergence to an optimal solution, showing that SCVI converges faster than other algorithms.

In our implementation of the SCVI algorithm we used $K$-Means as the clustering algorithm.

| Method | ADR | $\lvert V \rvert$ | Time (secs) | # Backups |
|---|---|---|---|---|
| **Hallway** $\lvert S \rvert$**=60** | | | | |
| PVI | 0.518 | 251 | 468 | 945 |
| Perseus | 0.518 | 545 | 146 | 1591 |
| HSVI | 0.518 | 186 | 402 | 741 |
| SCVI (5) | 0.518 | 370 | 79 | 960 |
| **Hallway2** $\lvert S \rvert$**=92** | | | | |
| PVI | 0.347 | 430 | 492 | 447 |
| Perseus | 0.347 | 1166 | 537 | 1305 |
| HSVI | 0.347 | 469 | 386 | 637 |
| SCVI (6) | 0.347 | 618 | 183 | 750 |
| **Pentagon** $\lvert S \rvert$**=212** | | | | |
| PVI | 0.368 | 846 | 1660 | 850 |
| Perseus | 0.368 | 2279 | 3691 | 3170 |
| HSVI | 0.368 | 659 | 737 | 1025 |
| SCVI (6) | 0.368 | 506 | 165 | 510 |
| **MIT** $\lvert S \rvert$**=204** | | | | |
| PVI | 0.877 | 793 | 1935 | 795 |
| Perseus | 0.877 | 3200 | 8991 | 3263 |
| HSVI | 0.877 | 1280 | 1708 | 1945 |
| SCVI (6) | 0.877 | 1011 | 603 | 1020 |
| **Rock Sample 5x5** $\lvert S \rvert$**=800** | | | | |
| PVI | 19.2 | 381 | 36 | 400 |
| Perseus | 19.2 | 396 | 303 | 15352 |
| HSVI | 19.2 | 473 | 104 | 671 |
| SCVI (6) | 19.2 | 341 | 15 | 750 |
| **TagAvoid** $\lvert S \rvert$**=870** | | | | |
| PVI | -6.3 | 233 | 137 | 500 |
| Perseus | -6.3 | 458 | 390 | 22417 |
| HSVI | -6.3 | 294 | 150 | 719 |
| SCVI (8) | -6.3 | 359 | 127 | 3662 |

Table 1: Performance measurements. $\lvert S \rvert$ indicates the number of states in each domain and the numbers in the brackets indicate the number of clusters used by SCVI.

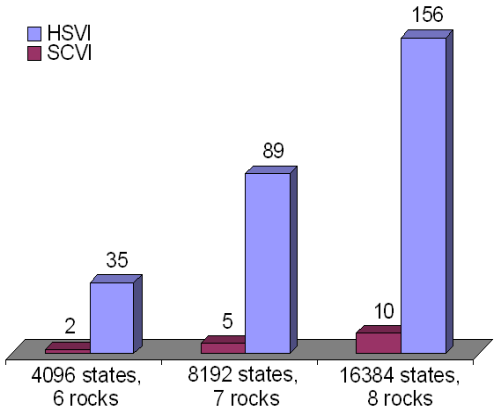



Figure 2: HSVI vs. SCVI convergence time(sec) in the RockSample domain with increasing number of rocks and fixed board size of 8x8.

Following previous research (Shani, Brafman, & Shimony 2006) we executed all algorithms within the same framework, allowing us to count execution time and the number of backups accurately. We tested all algorithms over several well known benchmarks from the point-based literature — Hallway and Hallway2 (Cassandra, Kaelbling,

& Littman 1994), RockSample (Smith & Simmons 2004), TagAvoid (Pineau, Gordon, & Thrun 2003) , and also over larger navigation problems from (Cassandra, Kaelbling, & Kurien 1996) (Pentagon and MIT). As we are interested in the speed of convergence, each algorithm was executed until its value function produced a pre-defined ADR. For each algorithm and problem we report $V$ — the number of vectors in the final value function, CPU time till convergence, and the number of backups performed.

Like SCVI, Perseus and PVI both use a pre-computed set of belief points and differ in the way they order the sequence of backups over these belief points. Perseus selects the next belief point to update rapidly, but must execute many backups till convergence, while PVI executes a small number of backups, but does considerable computation to select the next point to backup. Comparing SCVI with these two algorithms allows us to estimate the balance between the number of backups and the effort required to select the next point to backup. We executed all algorithms using the same belief point set of size $500$ gathered using a random exploration of the belief space.

As Table 1 shows, SCVI is faster than PVI and Perseus over all domains. In many cases PVI executes fewer backups, but the overhead for computing priorities in PVI makes it slower than SCVI that does not need to do any computation when selecting the next belief state to backup. Perseus, on the other hand, also requires no effort to select the next belief state, but frequently makes useless backups.

SCVI must cluster the MDP states and afterwards sort the belief states given their probability of belonging to a cluster. In all the experiments we conducted, the clustering and sorting took no longer than 1 second. The reported CPU time spans the entire SCVI algorithm, including the clustering and sorting process.

As HSVI, another point-based algorithm, has shown good results in scaling up to larger domains, we also compare SCVI to HSVI on a number of RockSample problems. HSVI does not use a predefined set, but executes trajectories through belief space and then backs up the belief states in these trajectories. Thus, such experiments compare not just the order by which backups are executed but the entire scope of the algorithm. Figure 2 shows how SCVI scales compared to HSVI over increasing problem size. The advantage of SCVI over HSVI becomes more apparent as the domain size grows.

As expected, over small problems, a partitioning of the state space into 5-6 clusters provided reasonable results. For larger problems a partition of 10 - 25 clusters gives better results. As the domain gets more complex, increasing the number of clusters makes the convergence faster. Otherwise, when a small domain is considered, the best performance can be achieved with a small number of clusters. Our algorithm showed little sensitivity to minor changes in the number of clusters, so we report here only results with the optimal number of clusters.

## Conclusion

We present the SCVI (Soft Clustering Value Iteration) algorithm — a point-based POMDP solver. SCVI better balances between the number of point-based backups executed by the algorithm and the effort required to prioritize backups. SCVI is a non-trivial adaptation of the GPS algorithm (Wingate & Seppi 2005) from MDPs to POMDPs.

We examine a new way to cluster the POMDP belief space through a clustering over the underlying MDP state space. Our approach to cluster the MDP uses a pre-computed optimal MDP value function and thus clusters together states with similar optimal values. We then relate each belief state to every cluster with different probabilities (soft clustering).

A set of experiments suggest that SCVI outperforms state of the art point-based solvers over a large set of benchmarks from POMDP and point-based literature.

We believe this set to be large enough to show the ability of our approach to generalize to other domains as well.

Moreover, as the problem increases in size, the advantages of SCVI over other algorithms become more apparent.

Our clustering mechanism can be viewed as an abstraction method. Future research should look into creating an abstract POMDP model through a clustering over the underlying MDP. Our prioritization technique for both belief states and clusters is static. Considering a dynamic way to update these priorities may result in reduced number of backups, and thus more rapid convergence.

## References


Bonet, B., and H.Gefner. 1998. Solving large POMDPs using real time dynamic programming. In *Fall AAAI Symposium on POMDPs*, 61–68.

Cassandra, A.; Kaelbling, L.; and Kurien, J. 1996. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *IROS*, 963–972.

Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *AAAI'94*, 1023–1028.

Cassandra, A.; Littman, M. L.; and Zhang, N. L. 1997. Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In *UAI–97*, 54–61.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1995. Efficient dynamic-programming updates in partially observable Markov decision processes. Technical Report CS-95-19, Brown University.

Lovejoy, W. S. 1991. Computationally feasible bounds for partially observable markov decison processes. *OR* 39:175–192.

Paquet, S.; Tobin, L.; and Chaib-draa, B. 2005. Real-time decision making for large POMDPs. In *AI'2005*.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI-2003*.

Shani, G.; Brafman, I.; and Shimony, E. 2006. Prioritizing point-based POMDP solvers. In *ECML-2006*, 389–400.

Smallwood, R. D., and Sondik, E. J. 1973. The optimal control of partially observable processes over a finite horizon. *OR* 21:1071–1088.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *UAI-2004*, 520 – 527.

Spaan, M. T. J., and Vlassis, N. 2005. Persus: Randomized point-based value iteration for POMDPs. In *JAIR*, volume 24, 195–220.

Wingate, D., and Seppi, K. D. 2005. Prioritization methods for accelerating mdp solvers. *JMLR* 6:851–881.

Zhou, R., and Hansen, E. A. 2001. An improved grid-based approximation algorithm for POMDPs. In *IJCAI*, 707–716.