Adaptation for Changing Stochastic Environments through Online POMDP Policy Learning

Guy Shani, Ronen I. Brafman, and Solomon E. Shimony

Ben-Gurion University, Beer-Sheva, Israel
{shanigu,brafman,shimony}@cs.bgu.ac.il

Abstract. Computing optimal or approximate policies for partially observable Markov decision processes (POMDPs) is a difficult task. When in addition the characteristics of the environment change over time, the problem is further compounded. A policy that was computed offline may stop being useful after sufficient changes to the environment have occurred. We present an online algorithm for incrementally improving POMDP policies, that is highly motivated by the Heuristic Search Value Iteration (HSVI) approach — locally improving the current value function after every action execution. Our algorithm adapts naturally to slow changes in the environment, without the need to explicitly model the changes. In initial empirical evaluation our algorithm shows a marked improvement over other online POMDP algorithms.

1 Introduction

Consider an agent situated in a partially observable domain: It executes an action that may change the state of the world; this change is reflected, in turn, by the agent's sensors; the action may have some associated cost, and the new state may have some associated reward or penalty. Thus, the agent's interaction with this environment is characterized by a sequence of action-observation-reward steps. Our goal is to have the agent act optimally (in the sense of expected reward) given what it knows about the world. Our focus is thus on agents with imperfect and noisy sensors, in the well-known framework of partially observable Markov decision processes (see section 2 for an overview of POMDPs).

Finding the optimal policy for a POMDP is known to be computationally intractable in the worst case, proved to be PSPACE-hard. While there are numerous algorithms for solving POMDPs with various restrictions, the difficulty of the general problem has prompted the developement of numerous approximation algorithms. When the (stochastic) behavior of the environment does not vary over time, we may apply one of these approximate schemes. These techniques may take a long time to produce a policy that is good enough, but as this effort is executed offline, it does not effect the online policy execution.

However, when the above assumption of a static environment does not hold, a policy that was optimal may become very far from optimal, as changes in the environment parameters (changes in the reward function, the transition probabilities, or sensor accuracy) accumulate. The naive solution is a costly re-computation of the policy. There are two problems in the naive approach: a) Until the decision to re-compute is made, the agent is acting according to a sub-optimal policy, and b) complete re-computation

of a policy may have such great resource (time and memory) consumption that it can in practice interfere with agent actions. While presumably the first problem can be solved by re-computation every time the environment parameters change, the complexity of POMDP solution rule out such a scheme in practice.

Point-based algorithms [8, 10, 9] are among the new, emerging techniques for scaling up to larger environments. This family of algorithms computes a partial policy for a part of the belief space, in the hope that the computed policy will hold for other belief states as well.

Our method is to leverage the flexible nature of point-based approximation algorithms to adapt to changes in the environment parameters, as long as these changes are relatively slow. While over a significant period of time a policy may need to change drastically, over short intervals minor adjustments to the policy suffice in order to act nearly optimally. An online algorithm we present requires only a little computational effort after each action/observation, and generates an approximate policy that can track the changes in the parameters.

The underlying assumption, that the parameters change slowly, is reasonable in various applications. For example, in a sensor model changes occur due to sensor aging, which is a long-term process. Likewise, transition probabilities in many cases are due to mechanical wear in parts, also a long term process. In user models for recommender systems, transition probabilities measure trends in a population, and thus are also liable to change slowly. While admittedly environments may also exhibit drastic changes in parameters, such as paths becoming completely blocked in a robot navigation problem, we argue that such changes typically occur less frequently. Between such drastic changes requiring complete policy recomputation, the agent should still benefit from incremental policy updates.

Our algorithm, Simple Online Value Iteration (SOVI), hence updates the POMDP parameters and continuously computes an approximate policy, using an online algorithm heavily motivated by the Heuristic Search Value Iteration (HSVI) algorithm [9]. SOVI maintains both an upper bound and a lower bound on the learned policy, where the upper bound is used for directing exploration in the environment, while the lower bound maintains the learned policy.

This paper is structured as follows: Section 2 begins with an overview of MDPs, POMDPs and their respective policy optimization algorithms. We explain the point-based approach for solving POMDPs and present a number of previous online approaches. Our online learning algorithm is presented in Section 3, followed by an experimental evaluation of our work in Section 4.

2 Background and Related Work

2.1 MDPs and POMDPs

A Markov Decision Process (MDP) [4] is a model for sequential stochastic decision problems. An MDP is a four-tuple: $\langle S, A, R, tr \rangle$, where S is the set of the states of the world, A is a set of actions an agent can use, R is a reward function, and tr is the stochastic state-transition function. A solution to an MDP is a policy $\pi : S \rightarrow A$ that defines which action should be executed in each state.

Various exact and approximate algorithms exist for computing an optimal policy, and the best known are policy-iteration [4] and value-iteration [1]. Solving MDPs is

known to be a polynomial problem in the number of states, and therefore exponential in the number of state variables. A value function assigns for each state a value $V(s)$ — the expected utility from acting optimally begining in s and on to infinity. Value iteration computes an optimal value function by iteratively solving the equation:

$$V_{n+1}(s) = \max_a \sum_{s'} tr(s, a, s') V_n(s') \tag{1}$$

A well known extension to the MDP model is the Partially Observable Markov Decision Process (POMDP) model [3]. A POMDP is a tuple $\langle S, A, R, tr, \Omega, O \rangle$, where S, A, R, tr define an MDP, Ω is a set of possible observations and $O(a, s, o)$ is the probability of executing action a, reaching state s and observing o. The agent is unable to identify the current state and is therefore forced to estimate the current state given the current observations (e.g. output of the robot sensors) and the agents' history. In many application domains POMDPs are a more precise and natural formalization than an MDP, but using POMDPs increases the difficulty of computing an optimal solution.

A compact approach to the representation of the history is in maintaining a belief state $b(s) = p(s|h)$ — the probability of being in state s after executing and observing history h. The next belief state $b^o_{a,b}$ resulting from executing action a and observing o in belief state a can be computed using:

$$b^o_{a,b}(s) = \frac{O(a, s, o) \sum_{s''} b(s'') tr(s'', a, s)}{pr(o|b, a)} \tag{2}$$

2.2 Approximate Solutions to POMDPs

Solving a POMDP is an extremely difficult computational problem, and various attempts have been made to compute approximate solutions that work reasonably well in practice.

An exact solution to a POMDP can be computed using the belief state MDP — an MDP over the belief space of the POMDP. A value function for a POMDP can be described using a set of $|S|$ dimensional vectors defining the expected utility, where each vector $\alpha_a \in V$ corresponds to an action a, that is, the value of executing the action a in every belief state b given a vector α that corresponds to action a, can be computed by $\alpha \cdot b = \sum_s \alpha(s) b(s)$. Using such a value function $V = \{\alpha^i\}$ we can define a policy π_V over the belief state:

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

We can compute the value function over the belief state MDP iteratively:

$$V_{n+1}(b) = \max_a b \cdot r_a + \gamma \sum_o pr(o|a, b) V_n(b^o_{a,b}) \tag{4}$$

where $r_a(s) = R(s, a)$. The computation of the next value function $V_{n+1}(b)$ out of the current one V_n (Equation 4) is known as a *backup* step. The backup step can be implemented more efficiently by splitting the backup step into:

$$backup(b) = \text{argmax}_{\{g^b_a\}a \in A} \, b \cdot g^b_a \tag{5}$$

$$g^b_a = r_a + \gamma \sum_o \text{argmax}_{\{g^i_{a,o}\}i} \, b \cdot g^i_{a,o} \tag{6}$$

$$g^i_{a,o}(s) = \sum_{s'} O(a, s', o) tr(s, a, s') \alpha^i(s') \tag{7}$$

Note that $g^i_{a,o}$ is independent of the belief state b and can therefore be cached.

A point-based algorithm [8] is an algorithm that computes a value function over a finite set of belief points (belief states). Point based algorithms compute an approximate solution as they do not iterate over the entire (infinite) belief space.

Spaan *et al.* explore the world randomly to gather a set B of belief points and then execute the Perseus algorithm (Algorithm 1)[10]. Spaan *et al.* also explain how backups can be computed efficiently. Perseus appears to provide good approximations with small sized value functions rapidly.

Algorithm 1 Perseus

Input: B — a set of belief points

repeat
  $\tilde{B} \leftarrow B$
  $V' \leftarrow \phi$
  while $\tilde{B}$ not empty **do**
    Sample $b \in \tilde{B}$
    $\alpha \leftarrow backup(b)$
    if $\alpha \cdot b > V(b)$ **then**
      $V' \leftarrow V' \cup \{\alpha\}$
    else
      $V' \leftarrow V' \cup \{\text{argmax}_{\beta \in V}\, \beta \cdot b\}$
    $\tilde{B} \leftarrow \{b \in \tilde{B} : V'(b) < V(b)\}$
  $V \leftarrow V'$
until V has converged

2.3 Online POMDP Algorithms

Agents that apply online (or real-time) algorithms roam the environment, learning a policy while acting. Such algorithms can be useful when the environment is too large to learn prior to acting, or as in our case, when the environment is slowly changing. There are many online algorithms for MDPs, the most famous of which is Q-learning, yet only a few attempts have been made at online learning for POMDPs.

Perhaps the earliest attempt at online approaches for POMDPs is the BEL-RTDP algorithm of Bonet and Geffner [2]. If we choose to represent the policy not as a set of vectors but directly in belief space Q values, we can define the H operator that updates a value function V:

$$HV(b) = \max_a \sum_s b(s)R(s,a) + \gamma \sum_o pr(o|a,b)V(b^o_{a,b}) \tag{8}$$

Bonet and Geffner use the H operator for updating their value function at every observed belief state. They propose to discretize the belief space such that every belief state is mapped to the closest (discrete) belief state such that each entry in the discrete belief state has to be i/k, where k is a predefined constant (Bonet et al. use $k \in [10, 100]$). Thus, updating the value function for a belief state occurs also when similar belief states are updated. A limitation of this approach is that it attempts to

group together belief states in a crude manner, as the assumption that belief states with close values require the same action is often wrong.

Paquet et al. take this idea a step forward in their Real Time Belief State Search (RTBSS) algorithm [7]. They suggest to expand the H operator so that it will not look only into the next belief state $b^o_{a,b}$, but will expand the search farther into the future, tracking all possible trajectories until a predefined maximal depth k has been reached. Thus, an AND OR search tree is produced with depth k. Paquet et al. use a predefined optimistic heuristic function to prune the search tree, and to provide estimates to the value function at the leaves. They choose not to maintain the value function but rather to apply the search at every step. Their value function is therefore non-improving, and heavily depends on the heuristic function and k – the maximal search depth. While this approach provides total flexibility in the face of non-stationary environments, it is difficult in general to pre-compute a good value for k, and when the environment changes only slowly, much computational power is wasted when the search is restarted from scratch. RTBSS also searches a large number of unneeded belief states, and indeed, in our experiments explored a much larger portion of the belief space, without obtaining superior policies.

Smith and Simmons suggest a different approach. They propose the Heuristic Search Value Iteration algorithm, that was initially proposed as an offline algorithm, but can be easily expanded to an online version. They maintain both a lower bound and an upper bound on the value function. The lower bound is maintained as a set of vectors $\underline{V} = \{\alpha_i\}$ and the upper bound is maintained as a set of belief points $\bar{V} = \{b_i\}$. The upper bound is initialized using an optimistic heuristic function, such as the Q_{MDP} value function. Updating the lower bound is done using the backup operator, while updating the upper bound is done using the H operator. When a value for a belief point that is not in $\bar{V}$ during the H computation is needed, they suggest to use a linear program to find its value given the other points currently in $\bar{V}$. To optimize exploration, they choose to execute at each step the action with maximal H value, thus possibly lowering the upper bound. While their approach is highly attractive, we shall present a few improvements in the next section.

3 Simple Online Value Iteration

The HSVI algorithm above can be easily improved to produce faster convergence with lower computational overhead. In this section we present the Simple Online Value Iteration algorithm (SOVI) that improves upon the performance of HSVI.

First, the highest computational cost comes from the computation of linear programs with many parameters ($|\bar{V}|$). In our experiments, even on the smallest problems, HSVI failed to execute in reasonable time due to this computational cost.[1] We suggest a simpler approach — instead of computing values for new points using linear programming, we suggest using the heuristic value for these points.

Another possible improvement can be made to the lower bound updates using the backup operator. We use an approach motivated by the Prioritized Sweeping algorithm [6] for MDPs. Instead of allowing a single backup operation on each iteration,

[1] Smith et al. used in their reported experiments the commercial CPLEX solver whereas we used the non-commercial lp_solve solver.

we are willing to execute a predefined number k of backups. We maintain a priority queue of potential belief states, and update the k belief states with highest priorities. A backup is called *successful* if the resulting vector α improves the value function for the current belief state b by δ. After each successful backup we assign a priority δ to each observed predecessor of b in the belief space (assuming it previously had lower priority).

Instead of computing all possible previous belief states, we keep only the predecessors we have observed during the execution. This approach forces us to remember all the belief states we have passed through, but in our experiments, we did not hit any memory problems due to that approach.

HSVI prunes the lower bound value function $\underline{V}$ only when pointwise dominated vectors have been found. We say that α is pointwise dominated by α' if for each s, $\alpha(s) < \alpha'(s)$. While such vectors should surely be pruned, this approach does not prune many other dominated vectors, such as vectors that are dominated by a conjunction of two vectors. We use a more sophisticated pruning technique. For every vector α we maintain a witness b (denoted $witness(\alpha)$) s.t. α was a result of a backup operation on b. When a new vector that improves the value of b is computed using $\alpha = backup(b)$, we scan all other vectors $\alpha' \in \underline{V}$ and check whether $witness(\alpha')$ is also improved by the newly computed α. If so, α' is removed from $\underline{V}$.

Algorithm 2 SOVI

Input: b — a single belief point

UpdateV(b)
for $i \leftarrow 0$ to k **do**
  $b' \leftarrow$ belief state with maximal priority
  UpdateV(b')
  $priority(b') \leftarrow 0$
$\bar{V} \leftarrow \bar{V} \cup H\bar{V}(b)$

Algorithm 3 UpdateV

Input: b — a single belief point

$\alpha \leftarrow backup(b)$
$witness(\alpha) \leftarrow b$
if $\alpha \cdot b > \underline{V}(b)$ **then**
  $\delta \leftarrow \alpha \cdot b - \underline{V}(b)$
  for each $b' \in predecessors(b)$ **do**
    if $priority(b') < \delta$ **then**
      $priority(b') \leftarrow \delta$
  for each $\alpha' \in \underline{V}$ **do**
    if $witness(\alpha') \cdot \alpha > witness(\alpha') \cdot \alpha'$ **then**
      remove α' from $\underline{V}$
  $\underline{V} \leftarrow \underline{V} \cup \{\alpha\}$

3.1 Adapting SOVI for Non-Stationary Environments

When the environment slowly changes, the computed policy of SOVI may need adjustments. Such adjustments may raise the value of an action, an operation that is automatically computed by SOVI, but may also cause the decrease of the value of an action, and hence, the decrease of a vector.

Perseus, HSVI and SOVI begin with an underestimate of the value function and slowly raise it to meet the optimal value function. These algorithms add a new vector α to V only if there exists a belief state b for which $\alpha \cdot b$ is greater than $V(b)$. The value function hence never decreases.

Such a decrease may be needed, for example, when the straight road to a reward becomes blocked and the agent needs to take a longer detor. SOVI can detect this problem in the value function, where an over-optimistic vector resides within V by checking whether the vector can really obtain the value it promises. This is done by comparing $\alpha \cdot witness(\alpha)$ and $HV(witness(\alpha))$. If the H operator returns a value that is less than the value of $witness(\alpha)$ according to α then α is overoptimistic and should be removed from V.

We execute this check every time α has been selected as the best vector for any belief state b. If α was found to be over-optimistic, it is removed from V and a new $UpdateV$ operation is executed for $witness(\alpha)$.

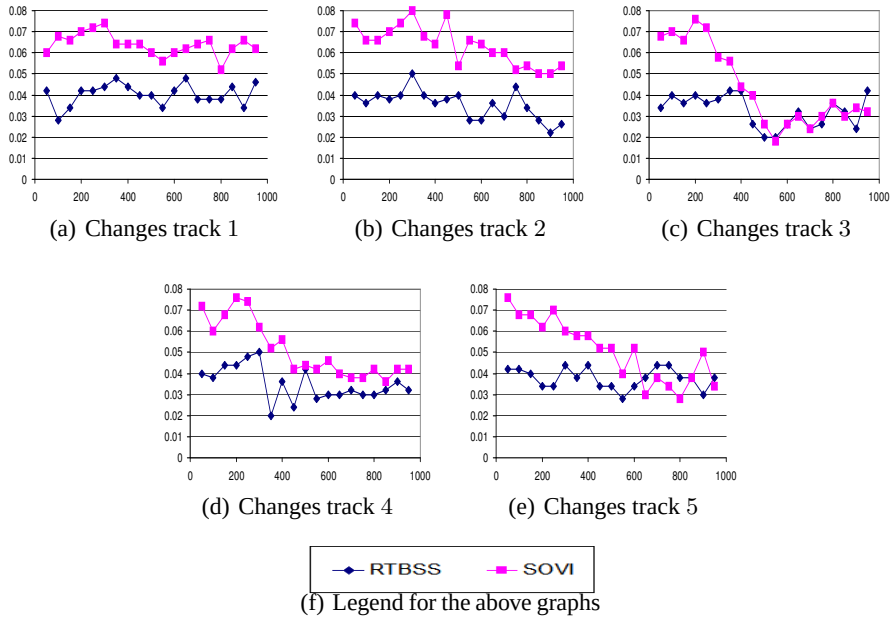
Thus, over-optimistic vectors are pruned from V, so that the value function remains a lower bound on the optimal value function, and hence the update step of SOVI remains valid.

4 Experimental Results

We conducted experiments over a number of well known POMDP problems: Hallway, Hallway2, Tiger Grid [5], Tag Avoid [8], Rock Sample 5,5 and Rock Sample 5,7 [9]. The observed results were very similar, so we focus more extensive experiments on the Hallway and Hallway2 problems. For each of these problems we defined 5 tracks of changes. Each track of changes constituted of a random number of changes, starting approximately 10 steps from each other, and each lasting 50 steps. Each such change randomly picked a non-zero entry from the transition or observation matrix, defined a new value for it, and split the difference between all other non-zero entries uniformly. Transition to the new value is done gradually, when on each action execution, the value is slightly modified (increased or decreased towards the target value) and other values adjusted so that probabilities always sum up to 1. Over each track of changes we executed 5 executions of SOVI and RTBSS[2] for a 1000 steps. After each 50 steps, learning was stopped and the current policy was evaluated for 500 steps. Reported results are averaged over these 5 executions. Figure 1 and Figure 2 present the changed average reward per action execution over time.

Most researchers use the time it took for the algorithm to execute to estimate its performance. While execution time comparison is indeed important, it has some major drawbacks. Time can vary greatly due to implementation; For example, choosing to use Java or C++ or MATLAB, the CPU power and available memory — all effect the

[2] We also tried to execute HSVI, but as stated before, the algorithm did not terminate within reasonable time.



(a) Changes track 1 (b) Changes track 2 (c) Changes track 3

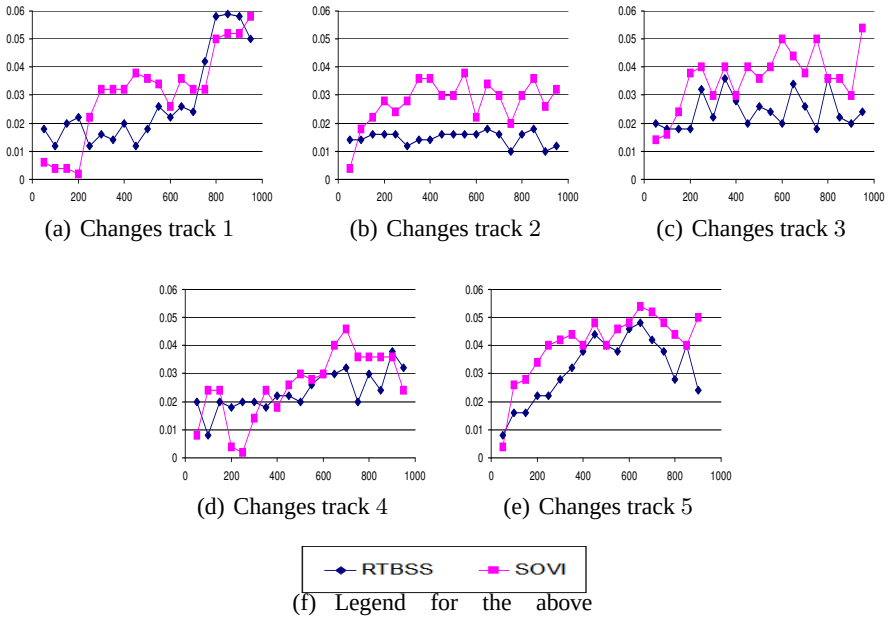
(d) Changes track 4 (e) Changes track 5

(f) Legend for the above graphs

Fig. 1. Hallway problem: Comparing average reward over time of RTBSS and SOVI over 5 different tracks of changes. Average reward per action (step) is plotted vs. the step number.

execution time greatly. We suggest a number of metrics that can be used to obtain a more precise estimate. A few basic operations stand at the basis of the above algorithms. The first, most basic, is the belief update computation —- computing $b^o_{a,b}$. The most basic implementation is using Equation 2, however, structured representations of the transition and observation functions can speed computation considerably. Counting the number of belief updates the algorithm has executed is hence a good measure. Likewise, the g operation (Equation 7) is also a very basic operation that stands at the base of the backup procedure. We thus include in our experiments comparisons using the number of explored belief states, and the size of the value function, as well as the actual runtime for algorithm execution.

Table 1 and Table 2 compare the performance of the RTBSS and SOVI algorithms on the Hallway and Hallway2 problems, testing 5 change tracks for each problem. All results are averaged over 5 executions for each track of changes. In the tables, time is the average time per step in milliseconds, belief updates denotes the average number of belief updates per step, belief points is the number of the computed belief points during the execution, G computations is the number of times Equation 7 was executed (not relevant for RTBSS), and $|\mathbb{V}|$ is the size (number of vectors) of the computed value function (not relevant for RTBSS).



(a) Changes track 1 (b) Changes track 2 (c) Changes track 3

(d) Changes track 4 (e) Changes track 5

(f) Legend for the above graphs.

Fig. 2. Hallway2 problem: Comparing average reward over time of RTBSS and SOVI over 5 different tracks of changes. Average reward per action (step) is plotted vs. the step number.

Method	Time	Belief updates	Belief points	G computations	$\|\underline{V}\|$
SOVI change 1	186.289	70.085	53,047	9846.795	34.42
RTBSS change 1	364.51	5660.394	193,693	N/A	N/A
SOVI change 2	170.65	66.087	49,855	7082.56	30.31
RTBSS change 2	302.343	5798.659	188,725	N/A	N/A
SOVI change 3	207.4	65.675	48,552	10110.24	34.89
RTBSS change 3	365.82	6583.431	178,805	N/A	N/A
SOVI change 4	252.06	64.49	48,121	10875.06	38.68
RTBSS change 4	398.54	6683.095	188,440	N/A	N/A
SOVI change 5	263.96	70.54	51,445	16648.48	46.42
RTBSS change 5	514.804	5756.356	196,062	N/A	N/A

Table 1. Hallway problem: Comparing the performance of RTBSS and SOVI over 5 different tracks of changes.

5 Conclusions and Future Work

In this paper we have presented the Simple Online Value Iteration (SOVI) algorithm, that is heavily motivated by Smith and Simmons's HSVI algorithm. Our approach differs from HSVI in a simplified upper and lower bound maintenance, and by allowing additional policy updates based on priorities. HSVI is also originally presented as an offline POMDP policy computation when we suggest adjustments to online learning.

Method	Time	Belief updates	Belief points	G computations	$\|\Upsilon\|$
SOVI change 1	271.71	186.34	37,167	3120.77	25.26
RTBSS change 1	1231.82	820.72	152,729	N/A	N/A
SOVI change 2	259.08	167.07	44,202	2849.88	30.1
RTBSS change 2	1124.36	724.22	157,901	N/A	N/A
SOVI change 3	943.3	177.39	42,350	3058.04	31.21
RTBSS change 3	795.71	535.51	126,962	N/A	N/A
SOVI change 4	380.15	183.33	51,316	6037.72	40.36
RTBSS change 4	1003.78	848.06	118,164	N/A	N/A
SOVI change 5	344.39	204.02	58,830	5679.87	41.63
RTBSS change 5	1781.85	797.46	162,044	N/A	N/A

Table 2. Hallway2 problem: Comparing the performance of RTBSS and SOVI over 5 different tracks of changes.

We show SOVI to preform well on slowly changing, non-stationary, environments, and to outperform RTBSS, which computes a full search on each step.

We also suggest a number of metrics for comparing online solvers, as well as the execution time, which is implementation dependant.

Future research should fully evaluate the performance of SOVI on non-stationary environments, in both online and offline execution and compare it versus other point-based algorithms such as HSVI, Perseus and PBVI [8].

6 Acknowledgements

Partially supported by the Lynn and William Frankel center for computer sciences, and by the Paul Ivanier center for Robotics and Production Management.

References

1. R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1962.
2. B. Bonet and H.Gefner. Solving large POMDPs using real time dynamic programming. In *AAAI Fall Symposium on POMDPs*, 1998.
3. A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *AAAI'94*, pages 1023–1028, 1994.
4. R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
5. M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *ICML'95*.
6. Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Journal of Machine Learning*, 13:103–130, 1993.
7. S. Paquet, L. Tobin, and B. Chaib-draa. Real-time decision making for large pomdps. In *AI'2005*, Victoria, Canada.
8. J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, August 2003.
9. T. Smith and R. Simmons. Heuristic search value iteration for pomdps. In *UAI 2004*, Banff, Alberta, 2004.
10. M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. Technical Report IAS-UVA-04-02, University of Amsterdam, 2004.