Belief-node Condensation for Online POMDP Algorithms

Gavin Rens*‡
Centre for Artificial Intelligence Research
*CSIR Meraka, South Africa
‡University of KwaZulu-Natal, South Africa
Email: grens@csir.co.za

Alexander Ferrein†‡
†Aachen University of Applied Sciences
Aachen, Germany
‡University of KwaZulu-Natal, South Africa
Email: ferrein@fh-aachen.de

***Abstract*—We consider online partially observable Markov decision processes (POMDPs) which compute policies by local look-ahead from the current belief-state. One problem is that belief-nodes deeper in the decision-tree increase in the number of states with non-zero probability they contain. Computation time of updating a belief-state is exponential in the number of states contained by the belief. Belief-update occurs for each node in a search tree. It would thus pay to reduce the size of the nodes while keeping the information they contain. In this paper, we compare four fast and frugal methods to reduce the size of belief-nodes in the search tree, hence improving the running-time of online POMDP algorithms.**

I. Introduction

Online partially observable Markov decision process (POMDP) algorithms typically perform forward search up to a fixed horizon in the belief space for selecting an agent's next action. Forward search in the belief space generates a *belief-decision-tree*, where the nodes in the tree represent belief-states (belief-nodes). One of the major problems with these belief-decision-trees is the exponential growth of the node-size with increasing depth of the tree. Assume node b is at tier h of a decision-tree, and a stochastic action with n probabilistic outcomes performed in b, with a subsequent observation. A successor node b' computed by a belief-update function (see Eq. 1 in Sec. II-A), contains $n \cdot |b|$ states (in the worst case). Then node b'' reached from b' could have $n \cdot |b'| = n^2 \cdot |b|$ states and so on. Computation of the belief-update is quadratic in the size of the belief-node. Hence, in general, there are $O(n^{2h}|b_0|^2) = O(n^{2h})$ computations for the belief-update at depth h, where b_0 is the root node (the agent's initial belief). The problem is compounded by belief-update occurring for *each* node in the search tree. Moreover, due to online algorithms usually being used in real-time, belief-update during search is required to be very fast.

Consider the following scenario: A robot is deployed in a courtyard after a party to pick up paper plates and empty beer and soft-drink cans. The wind may blow; the harder the wind blows, the more the plates and cans move around. Suppose the robot has its sites on a plate; the longer it takes to plan its route to the plate, the longer it will take to get there. And the longer it takes to reach the plate, the more likely it is that the plate will have blown away by the time the robot reaches the place where the plate was lying. If the robot is able to take some reasoning shortcuts, it could plan its route faster and reach the position of a plate or a can faster, perhaps before the littered item blows away. However, due to inaccuracies in its route planning, the robot may not end up in the desired location anyway. Nevertheless, if the items are blowing around at quite a rate (but not so much as to make the robot useless), it is reasonable to want the robot to be more reactive at the cost of accuracy. In a laboratory, for instance, actions must be accurate and mistakes are hardly tolerated. However, in many dynamic environments, like cleaning up after a party, inaccuracy is easily tolerated, as long as the job gets done.

In this work, we assume that the agents take into account the imperfections of their actuators and sensors, which causes the agents to have noisy data about them. The actions and sensors of such agents could be modeled and controlled by the POMDP framework. In particular, an online POMDP algorithms may be used. In this paper, we investigate four fast and frugal methods to condense (reduce) the size of belief-nodes in a belief-decision-tree and thus improve the running-time of online POMDP algorithms. We investigate the trade-off between rewards returned and running-time (i.e., reactivity) for different levels of dynamism in the environment. Through experiments, we show that some of the condensation methods make algorithms significantly more effective.

The paper is organized as follows. In the next section, we formally define partial observable Markov decision processes and sketch the idea behind online POMDP planning methods. Next, we mention the related work. Then we introduce the four different belief-state size reduction methods. Finally, our experiments are explained, the results discussed and conclusions drawn.

II. Preliminaries

A. Finite Horizon POMDPs

In a partially observable Markov decision process (POMDP), the actions the agent performs have non-deterministic effects in the sense that the agent can only predict with a likelihood in which state it will end up after performing an action. What is more, its perception is noisy. That is, when the agent uses its sensors to determine in which state it is, it will have a probability distribution over a set of possible states to reflect the conviction it has that it is in a state.

Formally, a POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \mathcal{O}, b^0 \rangle$ with $\mathcal{S}$ a finite set of states of the world (that the agent can be in), $\mathcal{A}$ a finite set of actions (that the agent can choose to

execute), $\mathcal{T}(s, a, s')$ denoting the probability of being in s' after performing action a in state s, $\mathcal{R}(a, s)$ the immediate reward gained for executing action a while in state s, $\mathcal{Z}$ a finite set of observations the agent can perceive in its world, $\mathcal{O}(s', a, z)$ denoting the probability of observing z in state s' resulting from performing action a in some other state, and b^0 the initial probability distribution over all states in $\mathcal{S}$.

A belief-state b is a set of pairs (s, p) where each state s in b is associated with a probability p. All probabilities must sum up to one, hence, b forms a probability distribution over the set $\mathcal{S}$ of all states. To update the agent's beliefs about the world, a special function $\tau(z, a, b) = b_n$ is defined as

$$b_n(s') = \frac{\mathcal{O}(s', a, z) \sum_{s \in \mathcal{S}} \mathcal{T}(s, a, s') b(s)}{Pr(z|a, b)}, \qquad (1)$$

where $b_n(s')$ denotes the probability of the agent being in state s' in the 'new' belief-state b_n. Note that $Pr(z \mid a, b)$ is a normalizing constant.

Let the *planning horizon* h (also called the *look-ahead depth*) be the number of future steps the agent plans ahead each time it selects its next action. $V^*(b, h)$ is the *optimal* value of future courses of actions the agent can take with respect to a finite horizon h starting in belief-state b. This function assumes that at each step the action that will maximize the state's value will be selected.

$$V^*(b, h) = \max_{a \in \mathcal{A}} \Big[R_B(a, b) + \gamma \sum_{z \in \mathcal{Z}} Pr(z \mid a, b) V^*(\tau(z, a, b), h - 1) \Big], \quad (2)$$

where $R_B(a, b) \stackrel{def}{=} \sum_{s \in \mathcal{S}} \mathcal{R}(a, s) b(s)$, $0 < \gamma < 1$ is a factor to discount the value of future rewards and $Pr(z \mid a, b)$ denotes the probability of reaching belief-state $b_n = \tau(z, a, b)$. While V^* denotes the optimal state-value, function Q^* denotes the state-action value: $Q^*(a, b, h) = R_B(a, b) + \gamma \sum_{z \in \mathcal{Z}} Pr(z \mid a, b) V^*(\tau(z, a, b), h-1)$ is the value of executing a in the current belief-state, plus the total expected value of belief-states reached thereafter.

B. Online POMDP Algorithms

Online POMDP methods consist of two phases, a planning phase where a finite sequence of actions is computed, and an execution phase where the actions are executed in the real environment. After executing the actions, the agent switches back to the planning phase.

In the planning phase, a tree with belief-states as nodes is generated, with the current belief-state as the root node. It is expanded up to the given depth D. Arcs represent actions and their resulting observations. At each node (belief-state), certain action executions are considered, and a decision can be made about which action the agent would execute if it were in the projected belief-state. Such a tree for planning with belief-states is called *belief-decision-tree*. Figure 1 depicts a belief-decision-tree of depth 1. The actions that are considered are the action *left* and *right*, the observations the agent can make are o_1 to o_4, leading to four new different nodes. Planning continues until (i) the time for planning runs out, (ii) the value of the best action so far is satisfactory (ϵ-*optimal*) or (iii) the decision process has completed. If cases (i) or (ii) are not satisfied, a node from the fringe of the current tree is set as the root of a new tree which will be searched to depth D again, and so on. In case (iii), search will proceed to depth D, independent of time or whether the action considered for execution is satisfactory according to some prior knowledge. During planning with a belief-decision-tree, it is assumed that, at each node, the action that maximizes the expected rewards will be executed. That is, $a^* = \arg\max_{a \in \mathcal{A}} Q^*(a, b', h')$ is employed at the node representing b', where h' is the number of steps to go. Once the whole tree is created, node values are propagated backwards from the leaf nodes to ancestors, upto the root node, using Equation (2).

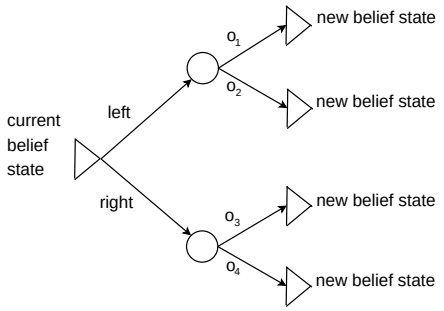

Fig. 1. A belief-decision-tree, with two actions, *left* and *right*, and two corresponding possible observations each, o_1, o_2 and respectively o_3, o_4. Triangles represent belief-states, and circles represent points when the agent processes its perception.

In this work, we use two online POMDP algorithms: the Monte Carlo (MC) Sampling approach of McAllester and Singh [1], and a version of Real-Time Belief Space Search (RTBSS), which was first proposed by Paquet, Tobin and Chaib-draa [2].

III. Related Work

Two sources for the intractability of solving POMDPs optimally are usually cited in the literature [3]. First is the *curse of dimensionality*, which refers to (in the case of a model with discrete states) a belief space having a dimension equal to the number of states. For instance, a domain modeled with 1000 states has a 1000-dimensional belief space! Poupart and Boutilier [4] show how to compress the state space such that the value function finds (almost) the same policies when referring to the compressed and original state spaces. Compression is steered by finding the Krylov subspace for the reward function. Roy, Gordon and Thrun [5] show how to compress a state space by applying a variant of Principal Components Analysis to a set of samples from beliefs that the agent can expect to experience in the higher-dimensional space. Planning is then done in the lower-dimensional space.

Second is the *curse of history*, which refers to the number of possible belief-states that must be considered during planning increases exponentially with the planning horizon. Kurniawati *et al.* [6] reduce the effective horizon in robot motion planning by using a particular (offline) point-based POMDP solver: a compact representation of the state space is constructed by sampling "milestones" from the state space, and then uses this representation to guide sampling in the belief space. He, Brunskill and Roy [7] tackle the horizon problem for online planning for large systems ("many state variables,

where each variable may take on a large or infinite number of potential values") that need predictions for actions many steps into the future. Their work uses *macro-actions*, which are pre-defined or learned meaningful sequences of action. Single actions are selected by doing forward search in a restricted policy space defined by a set of macro-actions.

A strategy for policy generation in dynamic environments that deals with the two 'curses' mentioned is *continuous planning* or *agent-centered search* [8]. Agents employing this strategy compute future actions with only local look-ahead. Online POMDP algorithms, as discussed in Section II-B use this strategy [9]. But the *curse of outcomes*—exponential growth of belief-state size in the number of steps—can be considered as a third source of potential intractability in POMDP algorithms. While much work is focused on overcoming the curse of dimensionality and some work focuses on overcoming the curse of history, we focus on overcoming the curse of outcomes. Much work has been done to represent the POMDP more compactly, either the dynamics of the process are factored into independent structures [10], [1], [11] or the states are specified/identified in some logical formalism [12], [13], [14]. These approached usually reduce the *effective* number of outcomes.

All these approaches want to reduce the computation by reducing the problem representation size directly or by identifying the informational structure of the problem. Our approach does not require any special representation and it does not require pre-computations on the belief space. Our approach does local online optimization. That is, to condense a belief-node, the nature of each node is considered as it is 'encountered' during planning. This has the advantage that no matter how dynamic the domain, that is, no matter how often the domain model changes, nothing extra needs to be done for planning to continue. Nevertheless, just as methods to overcome the first two curses do not necessarily compete against each other, methods to overcome the third curse does not necessarily compete with the other two.

IV. The Four Condensation Methods

In this paper, we assume that the small local policy sought by an online algorithm is simply one action. Hence, an agent will deal with its environment as follows: using an online POMDP algorithm with the agent's current belief-state as input, a single action is selected and immediately executed. When the current observation is obtained, the belief-state is updated and the process repeats. The basic idea is to reduce the size of a belief-state by retaining only a small number of representative states. As the number of states in a belief reduces, performing belief-update on the 'condensed' belief will be significantly faster.

Four fast and frugal methods will be investigated: (1) *Mean-as-threshold* (MT): retain all states with probability greater than or equal to the mean of the probabilities of all the states. (2) *Most-expected-medoid* (MEM): retain only the single state that is closest to the center, weighted by its probability. (3) *Centroids-of-dense-regions* (CDR): retain states that are the centroids of dense regions, according to some measure of density. (4) *Random-states* (R(n)): randomly select n states from each belief-node during search.

In the case when no condensation method is applied to the planning algorithm, we call it the "baseline" method (BL). Except for the 'random' method, which needs no further explanation, we motivate and describe the other four methods in the following subsections. Methods MEM and CDR require a measure of distance between states. We define the distance between two states as the sum of 'differences' in state variables between two states s_1 and s_2. Formally,

$$dist(s_1, s_2) \stackrel{def}{=} \sum_{i=1}^{|s|} diff(v_i(s_1), v_i(s_2)), \tag{3}$$

where $|s|$ is the number of features describing a state (i.e., the number of variables), $v_i(s)$ is the value of variable i for state s and $diff(v, v')$ is the user-defined difference between variable values v and v'. Of course, other distance measures can also be applied.

A. Mean as Threshold

For each belief-node generated, a *subset* of states with probabilities above a certain threshold are retained. Using the mean of the probabilities μ_b of the states in b as a threshold seems to be a reasonable heuristic for selecting states with probabilities *relatively* high compared to all the states in the node. We define the set of *most probable* states of a set b as $mp(S) \stackrel{def}{=} \{s \in S \,|\, s \geq \mu_b\}$. The computational complexity of MT is in $O(2|b|)$.

B. Most Expected Medoid

Given a set of data points, the set's *medoid* is the point in the set that is on average closest to all points in the set [15]. Here we define the most representative state of a belief as the *most expected medoid* (MEM). Let $m = mem(b)$, where b is a belief-state. Then m is the most expected medoid of b: $mem(b) \stackrel{def}{=} \arg\max_s \left(\frac{b(s)}{D(s,b)}\right)$, where $D(s,b) = \frac{1}{|b|}\sum_{i=1}^{|b|} dist(s, s_i)$. $D(s,b)$ is the average distance of s to every state in b. Note that if the probability distribution over b is uniform, $m = \arg\min_s(D(s,b))$, that is, when the likelihoods of states are not considered, then m is the 'regular' medoid. Hence, with the MEM method, the condensed belief-state $b' = \{(mem(b), 1)\}$. The computational complexity of MEM is in $O(|b|^2)$.

C. Centroids of Dense Regions

This method retains the states at the centers of groups of states, where such groups have a particular property. The center state is called the *centroid*, and the group is all the states that are within a radius r around the centroid (i.e., the r-*region*). The value of r is chosen such that, on average for the states s in belief b, the *density* of the r-region around s is maximal. Let $neibs(s, r, b)$ be the neighboring states of centroid s within the r-region, in belief-state b: $neibs(s, r, b) \stackrel{def}{=} \{s' \in b \,|\, dist(s', s) \leq r\}$. Then, $density(s, r, b) \stackrel{def}{=} \frac{1}{r}\sum_{s \in neibs(s,r,b)} b(s)$. The property mentioned above—that a region must have for its centroid s to be retained—is that the density of s's r-region must be greater than or equal to the average density of r-regions in b: $densityAvg(r, b) \stackrel{def}{=} \frac{1}{|b|}\sum_{s \in b} density(s, r, b)$. Determining

which radius maximizes $densityAvg(r, b)$ is estimated by choosing $r \in Radii$ that maximizes $densityAvg(r, b)$, where $Radii$ is a set of radii. $Radii$ is determined as follows. Let d_{min} be the distance (using $dist(\cdot)$, Eq. 3) of the nearest neighbor, on average for states in b. Let n be a user-supplied parameter from the natural numbers. Then, $Radii = \{d_{min} \times k \in \mathbb{R} \mid k = 1, 2, \ldots, n\}$. Our experimental result suggests a number of $n = 3$. We define the *estimated density factor* of belief-state b, given a set of $Radii$ as $edf(b, Radii) \stackrel{def}{=} \max_{r \in Radii} densityAvg(r, b)$. The computational complexity of CDR is quadratic in the size of the belief-state being reduced. In the worst case, an r-region will contain all states in b. Determining d_{min} is in $O(|b|^2)$; calculating the estimated density factor is in $O(|Radii||b|^2)$ in the worst case and deciding which states to retain lies in $O(|b|^2)$ in the worst case. Normalizing b' is linear in $|b'|$. Therefore, CDR is in $O((2 + |Radii|)|b|^2 + 2|b'|)$ in the worst case.

V. Experiments

We performed experiments in a POMDP domains called CleanUp (presented for the first time in this paper). CleanUp$[M, N]$ is a $M \times M$ grid-world with N scattered *items* to collect. The aim in this domain is for the agent to collect as many items as possible. But depending on the *dynamism* of the domain, items may move. In CleanUp, states are quadruples (x, y, d, t), with $x, y \in \{1, \cdots, M\}$ being the coordinates of the agent's position in the world, $d \in \{North, East, West, South\}$ the direction it is facing, and $t \in \{0, 1\}$, $t = 1$ if an item is present in the cell with the agent, else $t = 0$. The agent can perform five actions $\{left, right, forward, see, collect\}$, meaning, turn left, turn right, move one cell forward, see whether an item is present and collect an item. The only observation possible when executing one of the ontic actions is $obsNil$, the null observation, and see has possible observations from the set $\{0, 1\}$ for whether the agent sees the presence of an item (1) or not (0). Next, we define the possible outcomes for each action: When the agent turns left or right, it can get stuck in the same direction, turn $90°$ or overshoots by $90°$. When the agent moves forward, it can get stuck or move one cell too far, the agent can see an item or see nothing (no item in the cell) and collecting is deterministic (if there is an item present, it will be collected with certainty, if the agent executes $collect$).

So that the agent does not get lost too quickly, we have included an automatic localization action, that is, a sensing action returns information about the agent's approximate location. The action is automatic because the agent cannot choose to perform it or not to perform it; the agent localizes itself after every regular/chosen action is executed. However, just as with regular actions, the localization sensor is noisy, and it correctly reports the agent's location with probability 0.95, else the sensor reports a location adjacent to the agent with probability uniformly distributed over 0.05. To avoid the agent getting stuck (making stupid decisions), the agent would get punished for visiting cells, in proportion to the number of times the cell has been visited in the present trial. Rewards are given in proportion to the Manhattan distance from the closest item and a big (2200 units) reward for collecting an item. We found that the agent performs better when it gets ten units for seeing, else it tends not to want to perform see. Actions *cost* nothing. The measure used in MEM and CDR for the distance between two states is the Manhattan distance between the positions represented in the respective states, plus $Angle/90°$, where $Angle$ is the lesser angle between two directions (e.g., difference between $North$ and $East$ is one, and between $West$ and $East$ is two). Tables I and II show the results for CleanUp[6,12] (288 states, 5 actions, 3 observations). The agent's initial position and direction per trial are random.

We shall define three levels of dynamism: low, medium and high. Dynamism is *low* when no items move. When dynamism is *medium*, an item moves every 4.83 seconds (for MC) or every 1.08 seconds (for BB). When dynamism is *high*, an item moves every 2.42 seconds (for MC) or every 0.54 seconds (for BB). Parameters were set as follows: discount factor of $\gamma = 0.9$, number of samples $c = 20$ (for MC) and horizon $h = 5$. Results are provided for the average number of items collected per hour per trial (ic/h), the average number of rewards per action executed ($\mathcal{R}/a$), the average time (in seconds) it takes to select an action (sec/a), the average rewards gained per second ($\mathcal{R}/\mathrm{sec}$) and the average number of states per node ($\#sn$). For each of the four condensation methods, for both of the POMDP algorithms, ten trials were run for each of the three levels of dynamism. Due to the extreme running times of the baseline method, only five trials could be run (for each level of dynamism). The agent was allowed to execute thirty-six actions per trial. All experiments were conducted on an Intel CORE i7 CPU, 2.0 GHz (boosting to 2.8 GHz), with 4 GB RAM.

VI. Discussion of Results

In each of the six tables, results were ranked according to items collected per hour (ic/h)—a method and its results are placed higher in a table if it causes the agent to collect more items per hour on average per trial. It is important to note, however, that ic/h is only one of many measures of effectiveness. Using ic/h assigns more or less equal weight to accuracy ($\mathcal{R}/a$) and reactivity (sec/a). In every case, applying either Mean-as-threshold or randomly selecting one state (R(1)) yields the best performance according to our simple ic/h metric. We also recorded standard deviation of $\mathcal{R}/a$ (not shown). When dynamism is medium (e.g.), with the MC algorithm, the percentage standard deviation for MT was 24 and for R(1) was 73, and with the BB algorithm, the percentage standard deviation for MT was 64 and for R(1) was 62. Applying no condensation (the baseline method) or the MEM method always results in the worst performance according to our metric. Employing the Monte Carlo algorithm, the CDR method yields average performance for all three levels of dynamism. However, when employing the Branch-and-Bound algorithm, the method fares poorly. Ignoring BL, actions are executed (planned for) at the same speed on average for all three levels of dynamism (for both algorithms). Again, ignoring BL, items collected per hour increase on average, with the increase in dynamism—slightly for MC and noticeably for BB. However, the rank orders remain more or less the same for all the experiments.

TABLE I. EXPERIMENT RESULTS FOR THE MONTE CARLO ALGORITHM. (A) LOW DYNAMISM. (B) MEDIUM DYNAMISM (C) HIGH DYNAMISM.

(a)

Method	ic/h	$\mathcal{R}/a$	$\sec/a$	$\#sb$	$\#sa$
MT	16.7	274	26.3	2.52	1.32
R(1)	11.1	112	15.3	2.00	1.00
CDR	10.7	262	39.3	4.18	2.41
R(3)	5.26	111	34.2	5.19	2.96
R(5)	3.14	76.7	47.8	8.13	4.86
MEM	2.98	108	16.8	1.98	1.00
BL	0.490	217	737	55.7	55.7

(b)

Method	ic/h	$\mathcal{R}/a$	$\sec/a$	$\#sb$	$\#sa$
MT	19.0	263	20.5	2.60	1.36
R(1)	11.4	127	16.6	1.99	1.00
CDR	7.12	197	43.5	4.29	2.47
R(3)	6.36	165	36.1	5.19	2.95
R(5)	6.02	231	54.8	8.18	4.86
MEM	0.947	64.4	10.6	1.97	1.00
BL	0.520	180	538	53.9	53.9

(c)

Method	ic/h	$\mathcal{R}/a$	$\sec/a$	$\#sb$	$\#sa$
R(1)	29.0	229	12.3	2.01	1.00
MT	15.2	210	20.3	2.49	1.30
CDR	13.0	268	30.1	4.06	2.33
R(3)	8.82	233	37.4	5.21	2.95
R(5)	6.15	235	58.5	8.26	4.87
MEM	4.94	81.0	12.1	1.99	1.00
BL	0.511	153	430	56.8	56.8

TABLE II. EXPERIMENT RESULTS FOR THE BRANCH-AND-BOUND ALGORITHM. (A) LOW DYNAMISM. (B) MEDIUM DYNAMISM (C) HIGH DYNAMISM.

(a)

Method	ic/h	$\mathcal{R}/a$	$\sec/a$	$\#sb$	$\#sa$
MT	114	163	2.60	1.99	1.88
R(1)	77.8	76.5	1.67	1.72	1.00
R(3)	70.3	79.7	1.85	4.95	2.94
R(5)	48.6	88.0	2.68	7.82	4.82
CDR	35.2	132	6.81	3.72	2.25
BL	6.30	98.8	27.0	37.0	37.0
MEM	4.55	92.7	4.40	2.12	1.00

(b)

Method	ic/h	$\mathcal{R}/a$	$\sec/a$	$\#sb$	$\#sa$
R(1)	121	97.2	1.24	1.75	1.00
R(5)	82.9	120	2.53	7.25	4.78
MT	82.1	193	3.17	2.82	1.46
R(3)	73.4	151	3.13	3.85	2.94
CDR	54.2	188	5.72	3.20	1.87
MEM	8.59	94.4	5.82	1.97	1.00
BL	6.52	176	41.4	46.3	46.3

(c)

Method	ic/h	$\mathcal{R}/a$	$\sec/a$	$\#sb$	$\#sa$
MT	190	157	1.21	1.93	1.30
R(1)	126	191	2.23	1.66	1.00
R(3)	86.6	206	3.46	4.89	2.96
R(5)	74.6	140	2.28	8.16	4.85
CDR	52.0	195	5.58	3.75	2.21
BL	6.81	210	47.0	34.4	34.4
MEM	5.00	71.2	4.01	1.91	1.00

VII. CONCLUSION

Four methods to reduce the size of belief-nodes in POMDP decision-trees were investigated. The motivation for the methods being frugal is that processes applied online should be fast when the agent being controlled must be reactive. In general, planning can be up to almost 26 times faster with relatively little reduction in task quality, in the CleanUp domain. The performance metric we used will not suit all domains. Nevertheless, there are surely domains for which at least a variant of the metric is applicable. And in these domains, the agent designer may consider employing the MT or R(1) condensation method, especially if the agent will be controlled with an online POMDP algorithm. An interesting/unexpected result is that on avg, more rewards are gained as the system's dynamism increases. This could be explained by the possibility that the agent has to travel more when the items are stationary; the items tend to be closer to the agent when they move more

For future work, it would be interesting to see the influence of other distance measures (where applicable) than the one we have used (Eq. 3). Also, could condensation methods be applied in offline value or policy iteration algorithms, and what would the effects be? Another question we want to address in the future is how such methods could be used beneficially to determine heuristics in POMDP algorithms which use heuristics as discussed, for example, in research by Smith and Simmons [16] and Paquet, Tobin and Chaib-draa [2]. Intuition says that condensation by the MT method leaves a belief-state with states which are more relevant than if the R(1) method were employed. Yet, this observation is not born out by our results. We should find out the reasons for this unintuitive result.

REFERENCES

[1] D. McAllester and S. Singh, "Approximate planning for factored POMDPs using belief state simplification," in *Proc. of 15th Conf. on Uncertainty in Artificial Intelligence (UAI-99)*. San Francisco, CA: Morgan Kaufmann, 1999, pp. 409–416.

[2] S. Paquet, L. Tobin, and B. Chaib-draa, "Real-time decision making for large POMDPs," in *Adv. in Artif. Intell.: Proc. of 18th Conf. of the Canadian Society for Compl. Studies of Intell.*, ser. LNCS, vol. 3501. Springer, 2005, pp. 450–455.

[3] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proc. IJCAI*, 2003, pp. 1025–1032.

[4] P. Poupart and C. Boutilier, "Value-directed compression of POMDPs," in *Advances in Neural Information Processing Systems (NIPS 2003)*. Massachusetts/England: MIT Press, 2003, pp. 1547–1554.

[5] N. Roy, G. Gordon, and S. Thrun, "Finding approximate POMDP solutions through belief compressions," *J. Artif. Intell. Res. (JAIR)*, vol. 23, pp. 1–40, 2005.

[6] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *Intl. J. Robotics. Res.*, vol. 30, no. 3, pp. 308–323, 2011. [Online]. Available: http://dx.doi.org/10.1177/0278364910386986

[7] R. He, E. Brunskill, and N. Roy, "Efficient planning under uncertainty with macro-actions," *J. Artif. Intell. Res. (JAIR)*, vol. 40, pp. 523–570, 2011.

[8] S. Koenig, "Agent-centered search," *AI Mag.*, vol. 22, pp. 109–131, 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=567363.567371

[9] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa, "Online planning algorithms for POMDPs," *J. Artif. Intell. Res. (JAIR)*, vol. 32, pp. 663–704, 2008.

[10] C. Boutilier and D. Poole, "Computing optimal policies for partially observable decision processes using compact representations," in *Proc. of 13th Natl. Conf. on Artificial Intelligence*, 1996, pp. 1168–1175.

[11] E. Hansen and Z. Feng, "Dynamic programming for POMDPs using a factored state representation," in *Proc. of 5th Intl. Conf. on Artificial Intelligence, Planning and Scheduling (AIPS-00)*, 2000.

[12] D. Poole, "Decision theory, the situation calculus and conditional plans," *Linköping Electronic Articles in Computer and Information Science*, vol. 8, no. 3, 1998.

[13] C. Wang and J. Schmolze, "Planning with POMDPs using a compact, logic-based representation," in *Proc. of 17th IEEE Intl. Conf. on Tools with Artif. Intell. (ICTAI'05)*. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 523–530.

[14] S. Sanner and K. Kersting, "Symbolic dynamic programming for first-order POMDPs," in *Proc. of 24th Natl. Conf. on Artificial Intelligence (AAAI-10)*. AAAI Press, 2010, pp. 1140–1146.

[15] F. Gullo, G. Ponti, and A. Tagarelli, "Clustering uncertain data via k-medoids," in *Scalable Uncertainty Management*, ser. LNCS, S. Greco and T. Lukasiewicz, Eds. Springer, 2008, vol. 5291, pp. 229–242.

[16] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, ser. UAI '04. AUAI Press, 2004, pp. 520–527. [Online]. Available: http://portal.acm.org/citation.cfm?id=1036843.1036906