Reasoning, Planning, and Learning with POMDPs

Joelle Pineau
Nick Roy
Daniel Nikovski
CMU Robot Learning Lab Retreat
February 23, 2000

Outline

- Decision-theoretic vs. classical planning.
- Reasoning with POMDPs.
- Computing optimal policies for POMDPs:
  - Exact algorithms.
  - Approximate algorithms.
- Learning POMDP models from observations.
  - Learning POMDPs by parameter adjustment.
  - Learning POMDPs by state merging.

Decision-theoretic vs. classical planning for mobile robots

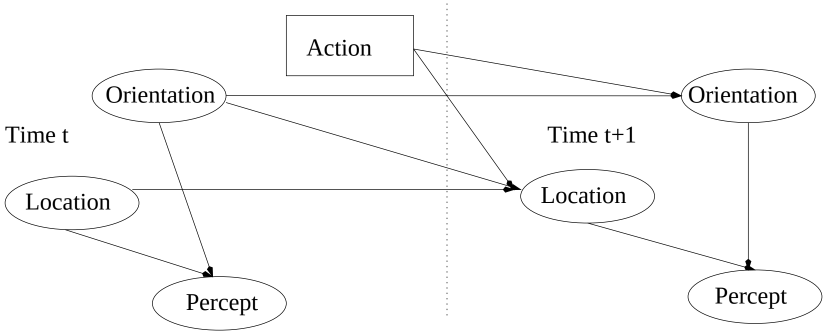
- Classical planning often fails because robot sensors are imperfect:
  - The same readings can be obtained at several locations (hidden state).
  - Different readings can be obtained at the same location (noisy percepts).
- DTP is a probabilistic extension of classical planning (Dean and Wellman 1991).
- DTP deals well with uncertainty in sensing and motion.
- DTP deals well with unobservable state.
- DTP has been demonstrated to work well on indoor mobile robots.
- Requires probabilistic models formulated as partially observable Markov decision processes.

Partially observable Markov decision processes (POMDPs)

- The world consists of a set of states $s_i \in \mathbf{S}$, which are not directly observable.
- In each state, a number of observations $o_j \in \mathbf{O}$ can be perceived, as specified by a probabilistic emission model $P(o_j|s_i)$.
- The agent can perform a number of actions $a_k \in \mathbf{A}$, which influence the state of the agent at the next time step via a transition model $P(s_i^{(t+1)}|s_j^{(t)}, a_k^{(t)})$.
- In each state, the agent receives some reward specified by a reward model $P(r|s_i)$.
- The objective of the agent is to choose actions so as to maximize the cumulative expected discounted reward $\sum_t r^{(t)}\gamma^t$, where $\gamma < 1$ is a discounting factor.
- For the purposes of mobile robot navigation, the reward for the goal (destination) state(s) can be 1, while the reward for all other states can be 0.

Distributed representations of POMDPs

- Temporal decision networks can represent efficiently PODMPs with factored state space (Dean and Wellman 1991, Boutilier et al 1995).
- Conditional independence exists between position and orientation for mobile robot navigation:



- Some actions affect only part of the state.
- Some part of the state might be observable.
- Conditional independence results in simpler models that might require less training samples.
- Learning might be faster in factorial POMDPs – generalized prioritized sweeping (Friedman et al. 1997).

Reasoning with POMDPs

- The agent doesn't know exactly where it is and has to maintain complete probability distribution $Bel(\mathbf{S})$ over all (possibly very many) states.
- Probability distributions over state space are maintained in a prediction-estimation cycle:

$$Bel(\mathbf{S}^{(t+1)}) = \alpha Bel(\mathbf{S}^{(t)}) P(\mathbf{S}^{(t+1)}|S^{(t)}, a^{(t)}) P(o^{(t+1)}|\mathbf{S}^{(t+1)}).$$

- For large state spaces, this might be very slow.
- Approximate reasoning techniques exist – Monte-Carlo methods (Thrun et al. 1998).

Computing policies for POMDPs

- A decision-theoretic plan is always conditional – a policy over beliefs on state space $a = \pi(Bel(\mathbf{S}))$ (Dean and Wellman 1991).
- This is very similar to optimal control, but the policy is conditioned on the belief state rather than on the most likely state.
- Policies are derived by means of techniques from operations research (OR) and reinforcement learning (RL).

Exact vs. approximate planning in POMDPs

- Finding the optimal policy in POMDPs is PSPACE hard (Littman et al. 1996). The Witness algorithm can handle POMDPs with less than 100 states.
- Approximate planning techniques are much faster:
  - Assumptive planning (Nourbakhsh 1998) reduces the POMDP to a FSA and finds a plan for reaching the goal from the most likely state by means of iterative deepening.
  - The most-likely state MDP method is similar, but computes the optimal policy for the fully observable MDP:

$$Q(s,a) =< R(s) + \gamma \sum_{s'} P(s'|s,a)[\max_b Q(s',b)] >,$$

  - The voting method is also similar, but chooses the action which is optimal for most states, proportional to the belief in each of them.
  - The Q_{MDP} method takes the Q-values in consideration as well.
- These methods never choose an action solely to gain information.

Learning POMDPs – general approach

- Common formulation: given a sequence of observations, find a model that explains them well (has a high predictive accuracy).
- This can be achieved by maximizing the likelihood $L(M : D)$ of the model given the data D – maximum likelihood estimation. $L(M : D) = P(D|M)$.
- Sometimes it is advantageous to impose priors on the parameters and structure of the model – maximum *a posteriori* estimation. The posterior probability $P(M|D) = \alpha P(D|M)P(M)$ is maximized instead.
- After a model is learned, planning algorithms use it to compute policies.
- Some approaches to learning policies in POMDPs do not learn an intermediate model (McCallum 1996).

Learning POMDPs – the Baum-Welch algorithm

- The Baum-Welch learning algorithm can be adapted for learning POMDPs (Chrisman 1992, Koenig and Simmons 1996, Cassandra, Kaelbling and Kurien 1996).

$$\hat{w}_{ijk}^{T+1} = \frac{\sum_l \xi_{i,l}^T(j,k)}{\sum_l \gamma_{i,l}^T(j)},$$

$\hat{w}_{ijk}^{T+1}$ is the estimate of the transition probability $P[S_i^{(t+1)} = s_{ik} | \mathbf{U}_i^{(t)} = \mathbf{u}_{ij}]$ that node S_i will be in its j-th state s_{ij} given that its parents $\mathbf{U}_i$ are in their k-th configuration $\mathbf{u}_k$, after iteration $T+1$, and

$$\xi_{i,l}^T(j,k) = P[S_i^{(t+1)} = s_{ik}, \mathbf{U}_i^{(t)} = \mathbf{u}_{ij} | D_l, \hat{\mathbf{w}}^k]$$

$$\gamma_{i,l}^T(j) = P[\mathbf{U}_i^{(t)} = \mathbf{u}_{ij} | D_l, \hat{\mathbf{w}}^k].$$

Learning POMDPs by steepest gradient ascent

- Russell, Binder, and Koller proposed another learning rule that maximizes the likelihood of the training data:

$$\Delta w_{ijk} = \eta \sum_l \frac{\partial P(D_l|\mathbf{w})/\partial w_{ijk}}{P(D_l|\mathbf{w})} = \sum_l \frac{P(S_{ij}, \mathbf{U}_{ik}|D_l, \mathbf{w})}{w_{ijk}},$$

$$w_{ijk} = P(S_{ij}|\mathbf{U}_{ik}) = P(S_i = s_{ij}|\mathbf{U}_i = \mathbf{u}_{ik})$$

  where w_{ijk} is the entry of the CPT of state node S_i that designates the probability that this node S_i will be in its j-th state s_{ij} given that its parents $\mathbf{U}_i$ are in their k-th configuration $\mathbf{u}_k$. The summations index l runs over all data cases D_l in $\mathbf{D}$.

- Both rules are similar, but Baum-Welch seems to converge to better local maxima.

- **Local maxima turn out to be major problem for iterative algorithms for parameter adjustment.**

Learning HMMs by model merging

- Proposed by Stolcke and Omohundro for speech recognition.
- Basic idea: *iteratively merge those states that will decrease the log-likelihood the least*.
- Start with a trivial model, where each of N observations is assumed to come with a different state. This model badly overfits the data.
- At each step t, $(N - t + 1)(N - t)/2$ mergers are possible – instead of trying all, choose K candidate pairs randomly.
- Accept the best merger and continue greedily.
- Stop when further mergers make the model very improbably in view of the data.

Bayesian criterion for stopping

- ML criteria would not work, because the initial model has the highest log-likelihood of zero.
- Instead, maximize the posterior probability of the model given the data:

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)}$$

- That is, optimize $log P(D|M) + log P(M)$. As long as decreases in $P(D|M)$ are compensated by increases in $P(M)$, keep merging.
- The model prior can be further decomposed into parameter and structural priors:

$$P(M) = P(M_p|M_s)P(M_s)$$

- An appropriate bias is given by a MDL structural prior $P(M_s) = ke^{-|M|}$, where k is a constant and $|M|$ is the number of states and transitions in the HMM.

Implementation issues

- When searching for mergers, consider first those states that have emitted the same symbol.
- If implemented directly, the algorithm has complexity $O(KN^4)$, where N is the number of observations and K is the number of candidate pairs. (There is a state for each observation initially.)
- Baum-Welch takes $O(INK_s^2)$, where I is the number of iterations, and K_s is the number of states ($K_s << N$).
- However, there are at most N transitions in the transition matrix with N^2 entries. This brings down complexity to $O(KN^3)$, but is still slow.
- Do on-line updating: instead of generating a full HMM with N states, start with a shorter one and add a new sample in the place of a merged one, which results in complexity $O(KN^2)$.

Implementation issues continued

- Another solution: compute the log-likelihood of the data only on the Viterbi path, and assume that mergers do not change it, so it doesn't have to be recomputed:

$$P(D|M) = \sum_{q_1,q_2,\ldots,q_l \in Q_l} P(x_1|q_1)P(q_2|q_1)\ldots P(x_{1-1}|q_{1-1})P(q_l|q_{l-1})P(x_l|q_l)$$

$$P(D|M) \approx \max_{q_1,q_2,\ldots,q_l \in Q_l} P(x_1|q_1)P(q_2|q_1)\ldots P(x_{1-1}|q_{1-1})P(q_l|q_{l-1})P(x_l|q_l)$$

$$P(D|M) \approx \prod_{q \in Q_l} \Big(\prod_{q' \in Q_l} P(q'|q)^{c(q'|q)} \prod_{\sigma \in \Sigma} P(\sigma|q)^{c(\sigma|q)} \Big)$$

- Only the effect of mergers can be computed in this expression, resulting in linear complexity $O(KN)$.

Learning factorial POMDPs by model merging

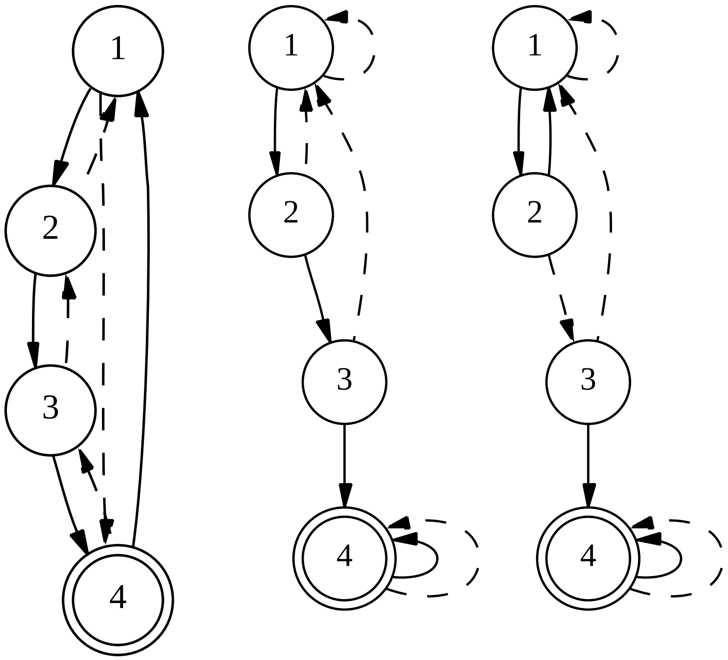
- Adding actions is straightforward – use a different transition matrix for each action.
- An unfortunate effect is that states downstream have more incoming edges than outgoing, and vice versa. Upstream states are harder to reach, which is not good for mobile robot navigation.
- Solution – use reversible actions, whenever possible. A side effect is that the initial model is not the one with highest likelihood – a merger might improve it.
- If part of the state is known, merge only those states for which that part matches.
- Continuous Gaussian emission models could be merged just as easily as discrete emission tables – there is no need to commit to a fixed number of percepts.

Discussion of best-first model merging

- Bayesian model-merging is more flexible than Baum-Welch as regards learning biases.
- It is reported to give better results than Baum-Welch on speech recognition problems.
- The resulting POMDPs tend to have low entropy of transition tables, which might speed up planning.
- Continuous observation models can be learned as well.
- The algorithm is particularly suitable for learning factored POMDPs, if part of the state is known.
- Problem-domain heuristics for merging can be exploited – for example, odometry for mobile robot navigation.

Comparison between Baum-Welch and BFMM

- Experimental comparison on fourteen test worlds
- Follows closely the methodology of Cassandra, Kaelbling, and Kurien, 1996.



Comparison between Baum-Welch and BFMM

- Cumulative discounted reward was computed with discount factor $\gamma = 0.9$.
- MDP-based policies were found by means of Q-learning with learning rate $\eta = 0.1$ and discounting factor $\gamma = 0.9$ were used, for a total of 1000 sweeps.
- The effect of aliasing, world size, and world complexity were explored.
- Four planners were compared against random action selection.
- Statistically significant differences at the 0.05 level with respect to random control are shown in bold.
- Framed rewards for a learning method are better than the ones for the other method with statistical significance at the 0.05 level.

State merging by trajectory clustering (SMTC)

- A major shortcoming of BFMM – state merging proceeds greedily and never reconsiders suboptimal merges of pairs of states.
- A different approach – cluster states based on the possibility that they correspond to the same underlying state.
- Compute a similarity between the trajectories that lead to two states:
  - Length of matching sequences prior to the two states – identical to the measure used by (McCallum 1995).
  - Length of matching sequence after the two states.
  - The sum of the two previous measures.
  - Decrease of the log-likelihood of the observation sequence when the two states are merged. This is the same measure that was used in the BFMM algorithm, and can be computed efficiently by making the Viterbi path assumption.
- Similarities between all possible pairs of states can be computed and placed in a similarity matrix.

- Clustering is performed based on the similarity matrix. The K-means algorithm cannot be applied, because there is no metric space for code vectors.
- Similarity-based clustering can be performed by means of minimum spanning trees (MST) (Duda and Hart 1973):
  1. Find the MST of the graph whose adjacency structure is defined by the similarity matrix.
  2. Sever the edges corresponding to the the least similar pairings. This results in several cliques, which define clusters of states that are likely to correspond to the same state in the true POMDP that generated the observations.
  3. Label the cliques and merge the states from the same clique.
  4. Label the hidden states in the observed sequence and compute the transition and emission matrices of the now fully-observable MDP.
- This approach considers all possible merges before actually carrying out any of them.
- This method is guaranteed to recover completely a fully-observable POMDP, while this is not necessarily true for EM and BFMM.

Computational issues in SMTC

- Direct implementation is quite expensive computationally – $O(N^3)$ on a sequence of N observations:
  - N^2 matches are considered, and the length of the matching subsequences backwards and forwards can be as long as N elements.
  - The complexity of finding the MST given the similarity matrix of N^2 elements is $O(N^3)$, if Prim's algorithm is used, and $O(N^2 \log N)$, if Kruskal's algorithm is employed in combination with a fast sorting routine.
  - The clique labeling stage takes computations in the order of N^3, if matrix multiplications are used to determine adjacency between members within a clique.

Algorithmic improvements to SMTC

- Several algorithmic improvements can bring the worst-case running time down to $O(N^2)$ and the average running time to $O(N \log N)$:
  - The algorithm of Leung et al. for DNA sequencing uses bucket arrays and linked lists to find all matches of subsequences of a particular length k in time linear in the number of observations N.
  - The linked lists produced by the above algorithm are a very convenient data structure for finding the minimum spanning tree by means of Kruskal's algorithm, which results in finding the MST in average time $O(N)$.
  - The complexity of labeling the states with clique numbers is dominated by the time of sorting the *remaining* edges in the MST, which is $O(N \log N)$.
- Novel iterative clustering algorithms have been proposed that lead to optimal clustering under certain conditions (Kenneth Rose, 1991; Buhmann and Hofmann, 1995.

Experimental comparision on a Nomad 150 simulator