Capacity-aware Sequential Recommendations

de Nijs, Frits; Theocharous, Georgios; Vlassis, Nikos; de Weerdt, Mathijs; Spaan, Matthijs

Publication date
2018

Document Version
Proof

Published in
Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems

Citation (APA)

Important note
To cite this publication, please use the final published version (if applicable). Please check the document version above.
Capacity-aware Sequential Recommendations

Frits de Nijs
Delft University of Technology
Delft, the Netherlands
f.denijs@tudelft.nl

Georgios Theocharous
Adobe Systems
San Jose, California, United States
theochar@adobe.com

Nikos Vlassis
Netflix
Los Gatos, California, United States
nvlassis@netflix.com

Mathijs M. de Weerdt
Delft University of Technology
Delft, the Netherlands
m.m.deweerdt@tudelft.nl

ABSTRACT

Personalized recommendations are increasingly important to engage users and guide them through large systems, for example when recommending points of interest to tourists visiting a popular city. To maximize long-term user experience, the system should consider issuing recommendations sequentially, since by observing the user’s response to a recommendation, the system can update its estimate of the user’s (latent) interests. However, as traditional recommender systems target individuals, their effect on a collective of users can unintentionally overload capacity. Therefore, recommender systems should not only consider the users’ interests, but also the effect of recommendations on the available capacity.

The structure in such a constrained, multi-agent, partially observable decision problem can be exploited by a novel belief-space sampling algorithm which bounds the size of the state space by a limit on regret. By exploiting the stationary structure of the problem, our algorithm is significantly more scalable than existing approximate solvers. Moreover, by explicitly considering the information value of actions, this algorithm significantly improves the quality of recommendations over an extension of posterior sampling reinforcement learning to the constrained multi-agent case. We show how to decouple constraint satisfaction from sequential recommendation policies, resulting in algorithms which issue recommendations to thousands of agents while respecting constraints.

ACM Reference Format:

1 INTRODUCTION

Personalized recommendations are an increasingly important approach to engage users and to help filter collections of objects which are otherwise too large to explore [3]. In many cases, recommendations should also take into account relations between objects and the history of the user, which requires the system to consider long-term effects of a recommendation. For example, when recommending news articles to readers, the user’s historical interests can unintentionally overload capacity. Therefore, recommendations should consider the users’ interests, but also the effect of recommendations on the available capacity.

One of the primary challenges for a recommender system is the discovery of a user’s preferences. Existing recommender systems are typically modeled as bandit models or click models. Such models aim to minimize regret incurred from taking exploratory actions [33]. Unfortunately, these models cannot anticipate the effect of a sequence of recommendations on the user [30]. To plan for long-term gains, we should instead cast the problem as a reinforcement learning problem, where we attempt to learn the dynamics of a Markov Decision Process (MDP) over time [35].

Because recommendations are targeted to the preference of an individual, their effect on a collective of users can unintentionally overload infrastructural capacity. For example, the use of an uncoordinated route guidance system can adversely affect the average waiting times in theme parks [7]. However, capacity constraints on recommended items may also serve an operational purpose: in virtual items such as news articles, limiting recommendations for naturally popular items can promote recommendation diversity.

Sharing resources is especially challenging in a system where multiple learning agents interact, because the trade-off between exploration and exploitation couples across agents: should an uncertain agent be awarded the resource in order to learn, or should another agent be allowed to use it to obtain reward with high certainty? However, recommendations provide the potential to steer users around constrained points, motivating the need for capacity-aware sequential recommendations. In this paper, we investigate how recommender systems should learn when they are constrained by resource limits restricting their joint actions.

It is critical for a recommender system to identify the true interests of a user in as few recommendations as possible, as mistakes risk losing the user’s attention. However, general reinforcement learning algorithms have a high sample complexity, requiring long interaction periods before a good policy is obtained [19]. An optimal learning policy prescribes actions which ensure that the entire learning trajectory is optimal [21]. Computing an optimal learning policy for a general reinforcement learning problem amounts to solving a continuous-state, Partially Observable MDP (POMDP; [12]). Unfortunately, these models can only be practically solved using approximate algorithms [2, 26]. Therefore, in this work we make the simplifying assumption that we can model differences between
users through parametric MDPs [11] with a finite parameter space, corresponding to a finite number of user types.

Work by Guez et al. [15] suggests two approaches to arrive at an optimal learning policy: (i) On-line sparse sampling algorithms such as Posterior Sampling Reinforcement Learning (PSRL; [34]), which uses an optimistic heuristic to eventually converge to the optimal policy, or (ii) Off-line planning of an optimal learning policy, by following Chadès et al. [6] in casting the parametric MDP to a stationary Mixed-Observable MDP (MOMDP; [23]). Unfortunately, neither approach can be applied directly to our capacity-aware recommendation problem; to the best of our knowledge no version of PSRL exists which incorporates constraints in the learning process, and it is not clear under what conditions the multi-agent case converges to a policy satisfying the constraints. On the other hand, computing an optimal policy for a MOMDP is a PSPACE-complete problem [25], limiting its practical scalability.

To address these challenges, we propose two novel algorithms: the first algorithm is an extension of PSRL to the multi-agent, constrained setting, by combining it with a Column Generation technique which has proven effective at decoupling agents from global constraints [10, 40]. The second algorithm exploits the structural properties of the recommendation problem to approximately solve the MOMDP: by computing the worst-case regret of switching from one recommendation strategy over a belief over types to an optimal policy, or (ii) Off-line planning of an optimal learning policy: (i) On-line sparse sampling algorithms such as multi-agent problems, directly optimizing the resulting LP is not 'priced out' by the \( \lambda \) prices computed in the dual solution. A solution is priced out if its contribution to the objective per unit of the constraint is less than \( \lambda \). If we can generate the optimal solution to be selected on the fly, we avoid having to maintain the exhaustive set of solutions explicitly. Generating the solution comes down to optimizing an ancillary problem subject to the \( \lambda \) costs.

Yost and Washburn [41] identified that this technique can be applied when solving constrained POMDPs, by augmenting the optimality criterion of the planning problem with a term corresponding to the expected resource consumption cost \( \mathbb{E}[C_{\pi, r}] \), i.e.,

\[
\arg \max_{\pi} \left( \mathbb{E}[V_{\pi}] - \sum_{t, r} \lambda_t \mathbb{E}[C_{\pi, t, r}] \right). \tag{3}
\]

This routine is used to compute a new policy to be added to the set of potential policies \( Z \), which forms the search space of the LP. The optimal mix of policies subject to constraints is then selected by
Thus far, we have assumed the instantiation parameter θ to be known to the decision maker. However, usually these parameters are hidden; in our model, parameter θ encodes the users’ latent interests. We consider a multi-agent recommender system consisting of n agents, their models characterized by a single parametric MDP. Each agent i behaves according to the MDP instantiated from its type θi. We assume agent types to be sampled from a finite set of potential types, according to a known prior probability φ = P(θ|θ = θ). The controller for each agent must learn what the type of the agent is, while ensuring that the agents jointly satisfy the global constraints. The result is a constrained, multi-agent parametric MDP having tuple (n, φ, Θ, S, A, R, T, h, C, L). Figure 1 presents the interactions between two agents and one constraint graphically through their Dynamic Bayesian Networks (DBN; [5]). Nodes in the figure represent states and observations (circles), decisions (squares), and costs and rewards (diamonds). Solid edges represent stochastic influences, while the dotted edges indicate deterministic influence, capturing the fact that an agent’s type θ is stationary.

2.3 Multi-agent constrained learning problem

Thus far, we have assumed the instantiation parameter θ to be known to the decision maker. However, usually these parameters are hidden; in our model, parameter θ encodes the users’ latent interests. We consider a multi-agent recommender system consisting of n agents, their models characterized by a single parametric MDP. Each agent i behaves according to the MDP instantiated from its type θi. We assume agent types to be sampled from a finite set of potential types, according to a known prior probability φ = P(θ|θ = θ). The controller for each agent must learn what the type of the agent is, while ensuring that the agents jointly satisfy the global constraints. The result is a constrained, multi-agent parametric MDP having tuple (n, φ, Θ, S, A, R, T, h, C, L). Figure 1 presents the interactions between two agents and one constraint graphically through their Dynamic Bayesian Networks (DBN; [5]). Nodes in the figure represent states and observations (circles), decisions (squares), and costs and rewards (diamonds). Solid edges represent stochastic influences, while the dotted edges indicate deterministic influence, capturing the fact that an agent’s type θ is stationary.

3 MULTI-AGENT CONSTRAINED PSRL

Column Generation is an effective algorithm for constrained multi-agent MDPs when they are weakly coupled [10]. At the same time, PSRL is an effective heuristic to learn the true type of a parametric MDP. Therefore, we propose to combine these two algorithms to obtain an effective heuristic for constrained learning problems.

3.1 Posterior sampling reinforcement learning

The algorithm operates as follows: to identify the true parameters θ of an instantiated MDP, the algorithm iteratively refines a probability density over parameter space Θ, through application of Bayes’ Theorem on the likelihood of the observed state. The Thompson sampling heuristic [39] is used to select actions, by optimistically assuming that type θ sampled from the current belief over types b is the true type. The optimal policy for the assumed model πj is used to select actions for an episode of τ steps, during which the belief over Θ is updated with every observed transition to state s′.

Although the PSRL algorithm is straightforward to state and based on an optimistic heuristic, it has strong performance guarantees: the algorithm has sample complexity polynomial in the number of parameters when learning the model of factored MDPs [24], as well as the guarantee of finding the optimal policy in a logarithmic number of time steps with high probability in our on-line (non-episodic) setting [14]. The Thompson sampling heuristic has also proven effective in recommender systems, with applications in ad format selection [36] and contextual recommenders [16].

3.2 Combining Column Generation and PSRL

Because the Thompson sampling heuristic samples hypotheses MDPs from the parametric description which are eventually correct, we may compute policies for these converged MDPs using Column Generation to obtain a joint policy which eventually satisfies the constraints. While belief has not converged, the expected consumption of an agent’s policy may not be attained because its true type does not match the sampled type. Nevertheless, we expect this strategy to work well in practice because every correctly identified agent behaves according to its constraint-respecting policy, and eventually all agents converge to their type.

Algorithm 1 presents the proposed approach. Column generation is called on line 1 to compute the optimal mix of resource-satisfying policies over the expected number of agents of each type. Because our agents behave according to homogeneous types, agents of the same type can be added together [41]. Therefore, the master LP is solving the following LP:

\[
\begin{align*}
\max_{x_j} \sum_{\pi_j \in Z_j} x_j E[V_{\pi_j}], \\
\text{s.t.} \quad \sum_{\pi_j \in Z_j} x_j E[C_{\pi_j, \tau, r}] \leq L(r) \quad \forall r, \forall t, \\
\sum_{\pi_j \in Z_j} x_j = 1, \text{ and } x_j \geq 0 \quad \forall j.
\end{align*}
\]

Algorithm 1 Multi-agent constrained PSRL

1. Given prior φ = P(θ), epoch length τ, initial state s1
2. Set time t = 1. For all i, set state si ← s1, belief b1 ← φ
3. plan (x, Z) = ColGen(MDPθ, n, φ)
4. for episode k = 1 → |z1| do
5. sample ∀i: δ1 ∼ b1
6. sample joint π by πi ∼ (xδ1, Zδ1)
7. for timestep l = 1 → τ and t ≤ h do
8. select joint action a = π(t, x)
9. observe next state vi: s′ i ∼ P(· | δ1, x, a, s1) → Agent par. δ1
10. update b1 by Bayes’ rule, ∀i: P(b1 | s, a, s1, b)
11. end for
12. end for
The relative frequencies $x_{i,j}$ computed by column generation define a probability distribution over policies: for a policy $π_{i,j}$ in set $Z_t$, $P(π_{i,j}) = \frac{x_{i,j}}{π_i Z_t}$. The policy the agent will use is sampled according to this probability distribution on line 4, choosing $Z_t$ according to the agents’ hypothetical MDP type sampled on line 3. The remaining structure of the algorithm follows from PSRL directly, accounting for the multiple agents in each step.

At the start and while converging there may be overconsumption due to incorrectly hypothesized agent types. However, as the number of agents of true type $\hat{θ}_i$ is in expectation $n \cdot P(\hat{θ}_i)$, provided the prior $φ$ is accurate, the sampled set of agents eventually converges to the distribution used to compute the constraint-satisfying policies. If prior $φ$ is inaccurate or the number of agents $n$ is too small to rely on the expectation, column generation can instead be invoked on the sampled types, after line 3.

### 4 A MOMDP APPROACH TO SEQUENTIAL RECOMMENDATIONS

Because PSRL uses the Thompson sampling heuristic to choose policies, the trajectory leading up to convergence may use sub-optimal actions resulting in unexpected resource violations. In order to control the consumption at all times, we need to compute an optimal learning policy, which amounts to solving a constrained Mixed-Observable MDP (MOMDP; [23]). Solving a general MOMDP model to optimality is a hard problem. However, our models are built out of a parametric MDP, which enables exploiting its structure during solving. We propose a novel algorithm for these problems, which obtains a bounded approximation error by switching from belief-space MOMDP policy to a regular MDP policy at belief points where the regret of such a switch is low. Because the resulting policy will be used in Column Generation to satisfy the constraints, we need to take special care that the expected values computed by this algorithm remain correct for these approximate solutions, which we address in the following section.

#### 4.1 Optimal learning of Parametric MDPs

Although PSRL eventually converges to the optimal policy, its trajectory leading up to convergence may be sub-optimal as a result of using a heuristic. For example, if there exists an action which is not part of the optimal policy for any $θ$, this action will never be chosen by PSRL. This is the case even if this action immediately reveals the true parameters of the MDP. In order to reason about such information gathering actions, a learning algorithm should explicitly consider the decision-theoretic value of information [17].

To our knowledge, Silver [31, Ch. 2] is the first to investigate how to make decisions when the true transition matrix of such a ‘multi-matrix’ MDP must be identified, while keeping the reward function fixed. Chadès et al. [6] extend the scope to our setting of identifying the true model of a hidden-model MDP, consisting of a set of candidate MDPs each with their own transition and reward function. In order to leverage existing algorithms to compute an optimal policy for hidden-model MDPs, the authors convert the problem to a MOMDP.

The state space of a MOMDP model factors into a fully observable factor $x \in X$ and a partially observable factor $y \in Y$, each with their own transition functions, $T_X(x' \mid x, y, a)$ and $T_Y(y' \mid x, y, a, s')$. As in the partially observable case, an observation function $Ω(o \mid a, y')$ exists to inform the decision maker about transitions of the hidden factor. However in addition to the observations, the decision maker also conditions his policy $π(t, x, o)$ on the observable factor $x$. Given a finite parametric MDP $(Ω, S, A, R, T, h)$, we derive an equivalent stationary MOMDP $(X, Y, A, Ω, T_X, T_Y, R, Ω, h)$ having elements

$$X = S,$$

$$Y = Θ,$$

$$O = \{o_{\text{NULL}}\},$$

$$Ω(o_{\text{NULL}} \mid a, y') = 1,$$

$$T_X(s' \mid s, θ, a) = T_θ(s'),$$

$$R(s, θ, a) = R_θ(s, a),$$

$$T_Y(θ' \mid s, θ, a, s') = \begin{cases} 1 & \text{if } θ = θ', \\ 0 & \text{otherwise.} \end{cases}$$

Figure 2 presents the two models graphically, through their dynamic Bayesian networks. The dotted edge in the HMDP model captures the notion of stationarity in the type given by equation (7). Although the HMDP appears to be a much less general model, Chadès et al. [6] prove that computing an optimal policy for HMDPs falls in the same PSPACE complexity class as POMDPs [25].

Casting parametric MDPs to MOMDPs has the advantage that existing theory and algorithms can be leveraged. Most algorithms for POMDPs and MOMDPs make use of a celebrated result by Sonik [32] that the optimal value function is piecewise linear and convex, and can be represented by a set of $α$-vectors, each giving expected values associated with taking an action $a$ in belief $b$. Martin et al. [22] exploit the stationary property of the MOMDP to compute a set of $α$-vectors which form a lower bound on the optimal value function. They propose to compute the optimal MDP policy for each type, and subsequently apply each optimal policy to all other types to construct an $α$-vector per policy. Initializing existing solvers with this lower bound speeds up their convergence, by providing tighter bounds for pruning computed vectors.

Unfortunately, existing solvers for MOMDPs typically assume the discounted infinite-horizon case, which incurs approximation errors on non-stationary problems, even if we annotate the state space with an additional time factor (thereby increasing its size by a factor $h$). Additionally, the complexity of solving a MOMDP necessitates computing approximate solutions, however approximate $α$-vector based solvers return expected values which do not correspond with the true expected value of the policy. This is problematic because we need true expectations for the integration with Column Generation.

![Figure 2: Comparison of HMDP and MOMDP models.](image-url)
4.2 Computing exact expectations for a reduced belief space

To avoid the drawbacks of existing solvers, we propose a new algorithm for stationary MOMDPs based on explicitly reasoning about reachable belief states. A belief state \( b \) records a probability distribution over the possible (unobserved) states \( s \), with \( b(s) \) indicating how likely the agent expects to be in state \( s \) [18]. Given a belief state \( b \), the action taken \( a \), and the observation received \( o \), the subsequent belief state \( b'(s') \) can be obtained using application of Bayes’ theorem. For a finite-horizon POMDP planning problem, the number of reachable belief states \( B \) is also finite, as in the worst case they form a tree of depth \( h \) with a branching factor of \( |A||O| \) at each node. This belief-state tree can be used as the state space of a belief-state MDP that is equivalent to the POMDP, which can in principle be solved by an application of (1), although the tractability of this approach is limited by the exponential growth of \( B \) in the horizon \( h \). Therefore, approximation algorithms generally attempt to reduce the size of \( B \), focusing on a subset of the space \( B' \).

Because the belief state space \( B' \) is an approximation of the exact state space \( B \), we expect to obtain potentially suboptimal policies. Nevertheless, we require exact expectations of a (suboptimal) policy’s consumption to use in the Column Generation program, as the satisfaction of the constraints depends on the selected policies using the resources to the reported levels. This can be achieved if we know the exact expected values of the policy at each ‘missing’ belief point not in \( B' \). We propose to use the stationary structure of the model to compute an approximate continuation from every reachable belief point.

The belief points \((t, s, b)\) of our MOMDP are factored into a time \( t \), MDP state \( s \), and belief \( b \) over possible types \( \theta \). For states at the corners of the belief where \( b(\theta_j) = 1 \) (and \( b(\theta_j) = 0 \) for \( i \neq j \)), the stationary condition ensures that the optimal continuation is the optimal MDP policy computed for the model instantiated with parameter \( \theta_i \). Thus, the expected value of such a corner-point immediately follows; if \( \pi^*_j \) is the optimal policy for MDP \( \theta_j \), then \( V^*(t, s, b) = V_{b, \theta_j; \pi^*_j}(t, s) \). We propose to approximate missing belief points using the same principle, by selecting the best policy from the optimal policies of each type. Intuitively this follows from the idea that for points which are very close to a corner, choosing policy \( \pi^*_j \) will almost always be correct. In the rare case this choice is incorrect, policy \( \pi^*_j \) is instead applied to another MDP \( \theta_j \), resulting in value \( V_{\theta_j, \pi^*_j}(t, s) \). The probability that this occurs is \( b(\theta_j) \). Thus, the total value of choosing policy \( \pi^*_j \) in belief point \((t, s, b)\) is

\[
\hat{V}(t, s, b) = \sum_{j=1}^{\Theta} b(\theta_j) \cdot V_{\theta_j, \pi^*_j}(t, s). \tag{8}
\]

The optimal value of using a fixed policy in point \((t, s, b)\) is then

\[
\hat{V}(t, s, b) = \max_{\pi} \hat{V}(t, s, b). \tag{9}
\]

While the expected value \( \hat{V}(t, s, b) \) is a lower bound on the optimal expected value \( V^*(t, s, b) \), it remains a correct expectation because it is based on the belief state \( b \) and the exact MDP expectations. Therefore we can use the value of \( \hat{V} \) as approximation for any belief point \((t, s, b) \notin B'\).

Algorithm 2 Bounded belief state space planning.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Plan ( \pi_j^* ) for all ( j ), compute ( V_{\theta_i, \pi_j^*} ) for all ( i, j )</td>
</tr>
<tr>
<td>2</td>
<td>Create policy ( \pi[b] )</td>
</tr>
<tr>
<td>3</td>
<td>for time ( t = h \rightarrow 1 ) do</td>
</tr>
<tr>
<td>4</td>
<td>for belief point ( b \in B'(t) ) do</td>
</tr>
<tr>
<td>5</td>
<td>( V[b] = \infty )</td>
</tr>
<tr>
<td>6</td>
<td>for action ( a \in A ) do</td>
</tr>
<tr>
<td>7</td>
<td>( Q[b, a] = R(b, a) )</td>
</tr>
<tr>
<td>8</td>
<td>for observed next state ( s' \in S ) do</td>
</tr>
<tr>
<td>9</td>
<td>( b' = \text{updateBelief}(b, a, s') )</td>
</tr>
<tr>
<td>10</td>
<td>if ( b' \in B' ) then</td>
</tr>
<tr>
<td>11</td>
<td>( Q[b, a] = Q[b, a] + P(s' \mid b, a) \cdot V[b'] )</td>
</tr>
<tr>
<td>12</td>
<td>else</td>
</tr>
<tr>
<td>13</td>
<td>( \pi[b', \pi_j^<em>] = \arg \max \pi_j^</em> Q[b', \pi_j^*] )</td>
</tr>
<tr>
<td>14</td>
<td>( Q[b, a] = Q[b, a] + P(s' \mid b, a) \cdot \hat{V}[b'] )</td>
</tr>
<tr>
<td>15</td>
<td>end if</td>
</tr>
<tr>
<td>16</td>
<td>end for</td>
</tr>
<tr>
<td>17</td>
<td>if ( Q[b, a] &gt; V[b] ) then</td>
</tr>
<tr>
<td>18</td>
<td>( V[b] = Q[b, a] )</td>
</tr>
<tr>
<td>19</td>
<td>( \pi[b] = a )</td>
</tr>
<tr>
<td>20</td>
<td>end if</td>
</tr>
<tr>
<td>21</td>
<td>end for</td>
</tr>
<tr>
<td>22</td>
<td>end for</td>
</tr>
<tr>
<td>23</td>
<td>end for</td>
</tr>
<tr>
<td>24</td>
<td>return ((\pi, V[b]))</td>
</tr>
</tbody>
</table>

In principle we could compute \( \hat{V}(t, s, b) \) exactly, however this would come down to computing an MDP policy for every belief point \( b \) that is reachable from the points in \( B' \). We can avoid this computational burden by the following observation: for points which are very close to corner \( i \), policy \( \pi^*_j \) will be the optimal policy with high probability. If we take care to construct \( B' \) such that the reachable points are close to corners, we can limit our search to the optimal policies of each type,

\[
\hat{V}(t, s, b) = \max_{\theta_j \in \Theta} Q(t, s, b, \pi_j^*). \tag{10}
\]

As the number of types is fixed, this comes down to computing \( |\Theta| \) MDP policies initially, and determining for each of these policies the expected values of applying it to the other types.

Algorithm 2 lists the exact expectation belief space planner. It starts by computing the optimal MDP policy \( \pi_j^* \) for each type \( \theta_j \) on line 1, followed by determining the exact expected values \( V_{\theta_i, \pi_j^*} \) of these policies for every other type \( \theta_i \). The remainder of the algorithm computes expected values at each of the generated belief points backwards over time, according to the typical dynamic programming algorithm, except in case a value is needed for a missing belief point on line 12. In case of a missing point \( b' \), the best policy \( \pi_j^* \) is selected on line 13, and the expected value of using this policy is computed according to the belief state.

The resulting policy returned on line 24 consists of two stages. For every belief point \( b \) in the collection \( B' \), the maximally valued action stored in \( \pi[b] \) on line 19 is selected. However, in case a \( b' \notin B' \) is reached during execution, the policy \( \pi_j^* \) stored on line 13.
is used as replacement for $\pi[b']$. Because the expected value of the MDP policies is exact, and $b'$ describes the state distribution that is reached in expectation \cite{kaelbling2018}, the expected value at any such `missing' belief state is also exact. Therefore, the values computed for the prior $b_0 = (1, s_1, \phi)$ are the true expectations of the (potentially suboptimal) values obtained by executing the policy computed by Algorithm 2. Therefore, this algorithm avoids all three weaknesses of existing approximate MOMDP solvers: it is a finite horizon solver without discounting, it computes exact expectations, and it remains tractable by operating on a reduced belief space by using the properties of our models.

\section{Using expected regret to bound the belief state space}

To determine an approximate belief space $B'$ for Algorithm 2, we use the expected regret of switching to a fixed MDP policy as a criterion for pruning a belief point. As we have seen, at the corners of the belief space, the optimal policy is the MDP policy computed for model instantiated on $\theta_i$, at which point there is no regret. While we could develop the belief state space until a corner is reached, it computes exact expectations, and it remains tractable by operating on a reduced belief space by using the properties of our models.

\subsection{Using expected regret to bound the belief state space}

To determine an approximate belief space $B'$ for Algorithm 2, we use the expected regret of switching to a fixed MDP policy as a criterion for pruning a belief point. As we have seen, at the corners of the belief space, the optimal policy is the MDP policy computed for model instantiated on $\theta_i$, at which point there is no regret. While we could develop the belief state space until a corner is reached, it computes exact expectations, and it remains tractable by operating on a reduced belief space by using the properties of our models.

\subsection{Using expected regret to bound the belief state space}

To determine an approximate belief space $B'$ for Algorithm 2, we use the expected regret of switching to a fixed MDP policy as a criterion for pruning a belief point. As we have seen, at the corners of the belief space, the optimal policy is the MDP policy computed for model instantiated on $\theta_i$, at which point there is no regret. While we could develop the belief state space until a corner is reached, it computes exact expectations, and it remains tractable by operating on a reduced belief space by using the properties of our models.

\subsection{Using expected regret to bound the belief state space}

To determine an approximate belief space $B'$ for Algorithm 2, we use the expected regret of switching to a fixed MDP policy as a criterion for pruning a belief point. As we have seen, at the corners of the belief space, the optimal policy is the MDP policy computed for model instantiated on $\theta_i$, at which point there is no regret. While we could develop the belief state space until a corner is reached, it computes exact expectations, and it remains tractable by operating on a reduced belief space by using the properties of our models.

\subsection{Using expected regret to bound the belief state space}

To determine an approximate belief space $B'$ for Algorithm 2, we use the expected regret of switching to a fixed MDP policy as a criterion for pruning a belief point. As we have seen, at the corners of the belief state space, the optimal policy is the MDP policy computed for model instantiated on $\theta_i$, at which point there is no regret. While we could develop the belief state space until a corner is reached, it computes exact expectations, and it remains tractable by operating on a reduced belief space by using the properties of our models.

\subsection{Using expected regret to bound the belief state space}

To determine an approximate belief space $B'$ for Algorithm 2, we use the expected regret of switching to a fixed MDP policy as a criterion for pruning a belief point. As we have seen, at the corners of the belief space, the optimal policy is the MDP policy computed for model instantiated on $\theta_i$, at which point there is no regret. While we could develop the belief state space until a corner is reached, it computes exact expectations, and it remains tractable by operating on a reduced belief space by using the properties of our models.

\section{Capacity-aware sequential recommendations domain}

We evaluate the algorithms proposed in the previous sections on a tourist recommendation problem modeled on data of visitors to Melbourne, derived from a dataset\footnote{Original dataset publicly available on https://github.com/arongdari/flickr-photo} of photograph meta-data from tourists visiting the city \cite{flickr}. Given a finite set of locations $l$ to be viewed one at a time, we model a system recommending a user the next item to view. Although each user has its own goals in visiting, we assume that tourists’ interests can be clustered into a set of discrete user types $\theta \in \Theta$. Each type $\theta$ defines a valuation for the items, awarding value according to a reward function $R_\theta(l)$ for seeing item $l$. We first cluster the historic visitor data into types $\theta$ based on the types of points photographed, setting the value $R_\theta(l)$ of visiting a point $l$ by the relative frequency with which $l$ is visited by visitors in cluster $\theta$.

From the perspective of a recommender system, the user’s interactions result in a history of user actions. At one point, a user may have first seen item $l_i$, followed by $l_j$, resulting in a history $\langle \ldots, l_i, l_j \rangle$. Such a history may be summarized in a higher level ‘context state’ $s_k$. Given a current context, we assume that the next item user of type $\theta$ will visit can be modeled by a probability distribution over the items $P_\theta(l | s_k)$.

In order to obtain $P_\theta$ from the dataset, we fit a Probabilistic Suffix Tree (PST) to each cluster of users. A PST predicts the probability of observing the next symbol in a sequence, conditional on a variable-length, bounded history of previously observed symbols \cite{clustering}. Such a PST defines a Markov Chain over the set of possible history states $S$, which is finite by the maximum depth of the PST. We write $s_{l,k}$ for a history-state recording the sequence $\langle l_i, l_j \rangle$, specifying a user which is now at $l_j$ after first visiting $l_i$. State $s_0$ represents the initial empty history $\epsilon$. Then, after fitting a PST of depth 2, we construct a closed Markov chain $T_\theta$:

\begin{align*}
T_\theta(s_i | s_0) &= \text{PST}_\theta(l_i | \epsilon) \quad \forall l_i \in P, \\
T_\theta(s_{i,j} | s_k) &= \text{PST}_\theta(l_j | l_i) \quad \forall l_j \in P, \\
T_\theta(s_{j,k} | s_{i,j}) &= \text{PST}_\theta(l_k | l_i, l_j) \quad \forall l_k \in P.
\end{align*}

In order to control the total size of the state space, we have two options: (i) we can select the number of locations to consider, by limiting to the top-$x$ most frequently visited points in the dataset, and (ii) we can limit the depth of the PST, thereby reducing the number of history states induced over the $x$ locations.

The Markov chain defined by (14) is transformed into a Markov Decision Process by including recommendation actions. An important challenge in designing a recommender system is that it is typically not known how agents will change their behavior when
receiving a recommendation, because no such recommendation system is in place yet to observe the effect of recommendations on users. We follow Theocharous et al. [37] in assuming that users boost their probability of viewing recommended item $l_i$ in accordance to a (type-specific) propensity to listen $\mu(\theta)$.

We consider two models of sequential recommendation systems: a ‘take-it-or-leave-it’ model which issues at most a single recommendation at a time, and an ‘alternatives’ model in which the system can issue at most two recommendations. In both cases, the set of potential recommendation actions $A$ contains a ‘no recommendation’ action $a_0$, which behaves as the original Markov chain, and a recommendation action $a_i$ for each item $l_i$. The ‘alternatives’ model also contains dual recommendation actions $a_{i,j}$ recommending the visitor to select either item $l_i$ or $l_j$. In case the user receives a dual recommendation, the user behaves as if it received the recommendation for the more valued of the two, thus

$$T_0(s' | s, a_0) = T_0(s' | s)$$

$$T_0(s' | s, a_i) = \begin{cases} T_0(s' | s, a_0)^{\gamma} & \text{if } l_i \text{ selected in } s' \\ T_0(s' | s, a_0)/z & \text{otherwise} \end{cases}$$

(15)

$$T_0(s' | s, a_{i,j}) = \begin{cases} T_0(s' | s, a_i) & \text{if } R_0(l_i) \geq R_0(l_j) \\ T_0(s' | s, a_j) & \text{otherwise} \end{cases}$$

In this equation $z$ is a normalizing factor to ensure $T$ remains a probability distribution.

The value of a recommendation depends on its quality; good recommendations send the user to locations with a high $R_0(l)$ value, while avoiding locations that the user has recently visited. Therefore, we shape the reward of issuing a recommendation by multiplying with a shape function $\sigma(\mu(a_i))$, where $\mu$ is an index function computing the number of $R_0(l_i) > R_0(l_j)$. To prevent the system issuing repeat recommendations, we add a penalty term $\rho(s, a)$ when recommendation $a$ is present in the history $s$.

$$\rho(s_h, a_i) = \begin{cases} \sigma(0) \max_{i \in h} R_0(l_i) & \text{if } i \in h \\ 0 & \text{otherwise} \end{cases}$$

(16)

Finally, we formalize the constraints by letting $L_{i,t}$ be the maximum number of users allowed to simultaneously view item $l_i$ at a time. Then, because a user’s state reports its current location, we can derive consumption function by letting $C_l(s_{i,j}) = 1$ if state $s_{i,j}$ sees the user currently viewing $l$.

6 EXPERIMENTAL EVALUATION

In this section we empirically evaluate our proposed algorithms on the tourist location recommendation problem. Our objective is to assess the scalability and solution quality of our proposed algorithms. Therefore, we compare our algorithms against state-of-the-art approximate MOMDP planner SARSOP [20]; for our experiments we used the implementation available on-line. Because SARSOP is an infinite horizon solver, we take care to explicitly include time in the state space as an observable factor. In addition, we must choose an appropriate value for the discount factor $\gamma$. The choice of $\gamma$ affects the amount of look-ahead that the solver performs, effectively trading off computation time for more myopic behavior. Therefore, we compare two settings: (i) $\gamma = 0.95$, resulting in essentially optimal policies for all solvable horizon lengths, and (ii) $\gamma = 0.5$, resulting in significantly reduced computation time at the cost of potentially myopic policies. To integrate SARSOP with Column Generation, we must determine the expected value and expected consumption of the policy. We obtain estimates of these expected values through simulation, computing means over 100,000 Monte Carlo samples.

We compare the algorithms on an instance of the tourist recommendation problem consisting of 5 locations, 3 user types, 50 users and PST depth 1. For this experiment we measure the quality of the policy as the mean over 1,000 simulations per trial, solving 5 instances per setting. The computation time is measured by mean elapsed wall-clock time per setting, with a 30 minute timeout. Based on preliminary experiments, we set the regret bounding parameters to $\alpha = 500$ and $p = 0.005$, which resulted in a good trade-off between state-space size and eventual bounding of growth.

Figure 3 presents the results, with the left-hand graphs corresponding to the setting where at most a single recommendation can be issued at a time, while the right-hand graphs are for the domain allowing recommendations with an alternative. The top row presents the observed mean reward per agent, while the bottom row presents the plan time in minutes. We note that we observe all the expected trends in the figure; we highlight three main observations: (i) For these constrained finite-horizon problems, SARSOP quickly becomes intractable, even when the discount factor is set very low. (ii) PSRL indeed returns nearly optimal solutions for the (low information value) single recommendation instances, in a fraction of the time of the other solvers. On the dual recommendation problem it incurs larger regret, but less than the approximate SARSOP solution.
at $h = 20$. (iii) Bounded-regret finds essentially optimal policies, while at the same time remaining tractable through its effective bounding condition on the state space growth. We note that its runtime stops increasing significantly beyond $h = 20$, as a result of the bounded growth of the state space.

To demonstrate the effect of considering constraints on the crowd dynamics, we perform an experiment on a large-scale problem. Figure 4 shows a simulation of the number of visitors at three different points of interest, with the red line indicating the constraint level, on a problem with 10 locations, 3 types, PST depth 2 and 5000 visitors during the entire day. The constraint-satisfying policy is able to redirect visitors effectively from crowded points 1 and 9 to 7. While computing this policy required solving over a thousand MOMDPs, by using the Bounded-regret algorithm the capacity-aware recommendation policy was computed within one hour.

7 RELATED WORK

Zhang et al. [42] study a multi-agent problem where agents compute policies which are guaranteed to satisfy commitments, despite the fact that agents have uncertainty about their model. Their model uncertainty also distributes over a finite number of types, however their constraints are over the achievement of specific states with a minimum probability. While commitments could in principle be used to satisfy resource constraints, their solution framework uses a Mixed-Integer Linear Program having number of binary variables equal to the number of knowledge states, resulting in an exponential complexity in the number of knowledge states.

Our model combining the possible agent types is a constrained POMDP. Poupart et al. [27] propose an algorithm to solve constrained POMDPs directly, by casting the problem as an (approximate) constrained belief state MDP. The framework of Constrained MDPs [1] consists of a single large LP which combines the satisfaction of constraints with computing the policy. Such a solution maintains a variable for each (belief state, action)-pair, which quickly grows intractably large. Although their method permits approximate belief spaces, our decoupling into MDP policies could not be applied in their method, because there the satisfaction of constraints is coupled with the computation of the policy. Recently, Walraven and Spaan [40] proposed a novel approximate algorithm for constrained POMDPs on the basis of Column Generation, which solves the expected-value problem by converting $\alpha$-vector policies to policy graphs. This algorithm is directly applicable to our domain, however because it does not consider the stationarity and mixed-observability inherent in our domain, we expect this approach to be less scalable than our Bounded-regret algorithm.

8 CONCLUSIONS AND FUTURE WORK

Recommender systems should use sequential interactions with agents to optimally refine their knowledge about the user, and should plan recommendations which satisfy the user’s long-term interests. Many times, the items being recommended are also subject to capacity limitations; in this work we present two novel algorithms for computing capacity-aware sequential recommendations for large-scale recommendation problems, resulting in the following contributions:

(i) We integrate PSRL with Column Generation to obtain an efficient heuristic constrained learning algorithm (Section 3).

(ii) We exploit the stationary structure of the MOMDP in computing an approximate continuation for any belief point, based on the minimal regret MDP policy. We show that these solutions can be embedded in Column Generation to compute a constrained optimal learning policy for our model (Section 4).

(iii) We use the expected regret to propose an efficient belief space truncating condition, which results in a highly scalable approximation algorithm for stationary MOMDPs (Section 4.3).

(iv) We show how to construct a constrained multi-agent recommender system from passive data, having recommendation actions that allow an alternative (Section 5).

We demonstrate that constrained PSRL finds high-quality policies quickly when the problem considered does not exhibit information gathering actions. However, in case a model does allow information gathering, such when issuing recommendations with alternatives, our results show that we are better off casting the problem as a constrained optimal learning problem. This problem can be solved tractably by planning a MOMDP over a reduced space of beliefs, derived from the regret of switching to an MDP policy.

In future work, we want to investigate methods to ensure fairness between users under constraints. Currently, column generation may decide to structurally give one type of user lower quality recommendations, in order to satisfy the constraints. In this case, we need to consider the recommendation problem as a multi-objective decision problem [28], to compute a set of policies trading off resource consumption with expected value. Another avenue of future work considers how to recommend users when only a subset of them uses the recommender system. In this case the behavior of the uncontrolled users, under stochastic influences such as the weather, impacts the capacity constraint stochastically [9]. This is especially challenging when we can not measure crowds reliably, making the available capacity itself a partially observable quantity. Finally, we intend to evaluate our algorithms on other constrained (multi-agent) learning domains, such as adaptive management problems [22] and smart-grid applications [8].
REFERENCES


