Logical Representations and Computational Methods for Markov Decision Processes

Craig Boutilier
Department of Computer Science
University of Toronto

Course Overview

- **Lecture 1**
  - motivation; MDPs: classical model and algorithms
- **Lecture 2**
  - logical representations
  - simple abstraction methods
- **Lecture 3**
  - decision-theoretic regression
- **Lecture 4**
  - linear function approx’n
  - factored basis functions
  - approximate VI
  - solution by LPs
  - choosing basis sets
- **Lecture 5**
  - temporal logic and non-Markovian dynamics
  - wrap up; further topics
Recap and Overview

- Last time: decision-theoretic regression
  - dynamic abstraction (exact or approximate)
  - produces piecewise constant representation of VF
- Today: we’ll consider *linear function approximation* and *decomposition* methods
  - offers more flexibility in space of VFs spanned for same size of representation

Overview

- linear function approximation and approximate VI
- factored linear approximations
- basic function construction (incl. decomposition)

Function Approximation

- Common approach to solving MDPs
  - find a functional form $f(\theta)$ for VF that is tractable
    - e.g., not exponential in number of variables
  - attempt to find parameters $\theta$ s.t. $f(\theta)$ offers “best fit” to “true” VF
- Example:
  - use neural net to approximate VF
    - inputs: state features; output: value or Q-value
  - generate samples of “true VF” to train NN
    - e.g., use dynamics to sample transitions and train on Bellman backups (bootstrap on current approximation given by NN)
Linear Function Approximation

- Assume a set of basis functions $B = \{ b_1 ... b_k \}$
  - each $b_i : S \rightarrow \mathbb{R}$ generally compactly representible
- A linear approximator is a linear combination of these basis functions; for some weight vector $w$:
  $$\overline{V}(s) = \sum_i w_i b_i(s)$$

Several questions:
- what is best weight vector $w$?
- what is a “good” basis set $B$?
- what does this buy us computationally?

Flexibility of Linear Decomposition

- Assume each basis function is compact
  - e.g., refers only a few vars; $b_1(X,Y)$, $b_2(W,Z)$, $b_3(A)$
- Then VF is compact:
  $$V(X,Y,W,Z,A) = w_1 b_1(X,Y) + w_2 b_2(W,Z) + w_3 b_3(A)$$
- For given representation size (10 parameters), we get more value flexibility (32 distinct values) compared to a piecewise constant rep’n
- So if we can find decent basis sets (that allow a good fit), this can be more compact
Linear Approx: Components

- Assume basis set $B = \{ b_1 \ldots b_k \}$
  - each $b_i : S \rightarrow \mathbb{R}$
  - we view each $b_i$ as an n-vector
  - let $A$ be the $n \times k$ matrix $[ b_1 \ldots b_k ]$
- Linear VF: $V(s) = \sum w_i b_i(s)$
- Equivalently: $V = Aw$
  - so our approximation of $V$ must lie in subspace spanned by $B$
  - let $B$ be that subspace

Approximate Value Iteration

- We might compute approximate $V$ using Value Iteration:
  - Let $V^0 = Aw^0$ for some weight vector $w^0$
  - Perform Bellman backups to produce $V^1 = Aw^1; V^2 = Aw^2; V^3 = Aw^3; \text{etc...}$
- Unfortunately, even if $V^0$ in subspace spanned by $B$, $L^*(V^0) = L^*(Aw^0)$ will generally not be
- So we need to find best approximation to $L^*(Aw^0)$ in $B$ before we can proceed
Projection

- We wish to find a projection of our VF estimates into $B$ minimizing some error criterion
  - We’ll use max norm (standard in MDPs)
- Given $V$ lying outside $B$, we want a $w$ s.t:
  
  \[ \| Aw - V \| \text{ is minimal} \]

Projection as Linear Program

- Finding a $w$ that minimizes $\| Aw - V \|$ can be accomplished with a simple LP

<table>
<thead>
<tr>
<th>Vars: $w_1, ..., w_k, \phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimize: $\phi$</td>
</tr>
<tr>
<td>S.T. $\phi \geq V(s) - Aw(s)$, $\forall s$</td>
</tr>
<tr>
<td>$\phi \geq Aw(s) - V(s)$, $\forall s$</td>
</tr>
</tbody>
</table>

- Number of variables is small ($k+1$); but number of constraints is large ($2$ per state)
  - this defeats the purpose of function approximation
  - but let’s ignore for the moment
Approximate Value Iteration

- Run value iteration; but after each Bellman backup, project result back into subspace $B$
- Choose arbitrary $w^0$ and let $V^0 = Aw^0$
- Then iterate
  - Compute $V^t = L^*(Aw^{t-1})$
  - Let $V^t = Aw^t$ be projection of $V^t$ into $B$
- Error at each step given by $\phi$
  - final error, convergence not assured
- Analog for policy iteration as well

Factored MDPs

- Suppose our MDP is represented using DBNs and our reward function is compact
  - can we exploit this structure to implement approximate value iteration more effectively?
- We’ll see that if our basis functions are “compact”, we can implement AVI without state enumeration
  - we’ll exploit principles we’ve seen in abstraction methods
Assumptions

- State space defined by variables $X_1, \ldots, X_n$
- DBN action representation for each action $a$
  - assume small set $\text{Par}(X_i)$
- Reward is sum of components
  - $R(X) = R_1(W_1) + R_2(W_2) + \ldots$
  - each $W_i \subseteq X$ is a small subset
- Each basis function $b_i$ refers to a small subset of vars $C_i$
  - $b_i(X) = b_i(C_i)$

Factored AVI

- AVI: repeatedly do Bellman backups, projections
- With factored MDP and basis representations
  - $Aw$ and $V$ are functions of variables $X_1, \ldots, X_n$
  - $Aw$ is compactly representable
    - $Aw = w_1 b_1(C_1) + \ldots + w_k b_k(C_k)$
    - each $W_i \subseteq X$ is a small subset
  - So $V^t = Aw^t$ (projection of $V^t$ into $B$) is compact
- So we need to ensure that:
  - each $V^t$ (nonprojected Bellman backup) is compact
  - we can perform projection effectively
Compactness of Bellman Backup

- Bellman backup: \( V^t(s) = \max_a Q^t(a, s) \)
- Q-function:

\[
Q^t(x, s) = R(x) + \sum_{x'} \Pr(x, a, x') \cdot V^{t-1}(x') = R_i(w_1) + R_2(w_2) + ... + \sum_x \Pr(x, a, x') \cdot \left[ w^{t-1}_i b_i(c_i') + ... + w^{t-1}_k b_k(c_k') \right] = R_i(w_1) + R_2(w_2) + ... + w^{t-1}_i \sum_{c_i} \Pr(c_i' \mid Par(c_i')) \cdot b_i(c_i') + ... + w^{t-1}_k \sum_{c_k} \Pr(c_k' \mid Par(c_k')) \cdot b_k(c_k')
\]

- So Q-functions are (weighted) sums of a small set of compact functions:
  - the rewards \( R_i(W_i) \)
  - the functions \( f_i(Par(C_i)) \) – each of which can be computed effectively (sum out only vars in \( C_i \))
  - note: backup of each \( b_i \) is decision-theoretic regression

- Maximizing over these to get VF straightforward
  - Thus we obtain compact rep’n of \( V^t = L^*(A(w^t)^{-1}) \)

- Problem: these new functions don’t belong to the set of basis functions
  - need to project \( V^t \) into \( B \) to obtain \( V^t \)
Factored Projection

- We have $V^t$ and want to find weights $w^t$ that minimize $\|Aw^t - V^t\|$.
  - We know $V^t$ is the sum of compact functions.
  - We know $Aw^t$ is the sum of compact functions.
  - Thus, their difference is the sum of compact functions.
- So we wish to minimize $\|\sum f_j(Z_j; w^t)\|$.
  - Each $f_j$ depends on small set of vars $Z_j$ and possibly some of the weights $w^t$.
- Assume weights $w^t$ are fixed for now.
  - Then $\|\sum f_j(Z_j; w^t)\| = \max \{ \sum f_j(z_j; w^t) : x \in X\}$

Variable Elimination

- Max of sum of compact functions: variable elim.

\[
\max X_1X_2X_3X_4X_5X_6 \{ f_1(X_1X_2X_3) + f_2(X_3X_4) + f_3(X_4X_5X_6) \}
\]

\begin{align*}
\text{Elim } X_1: \text{ Replace } f_1(X_1X_2X_3) \text{ with } \\
& f_4(X_2X_3) = \max X_1 \{ f_1(X_1X_2X_3) \}
\end{align*}

\begin{align*}
\text{Elim } X_3: \text{ Replace } f_2(X_3X_4) \text{ and } f_4(X_2X_3) \text{ with } \\
& f_5(X_2X_4) = \max X_3 \{ f_1(X_1X_2X_3) + f_4(X_2X_3) \}
\end{align*}

etc. (eliminating each variable in turn until maximum value is computed over entire state space)

- Complexity determined by size of intermediate factors (and elim ordering)
**Factored Projection: Factored LP**

- VE works for fixed weights
  - but $w^i$ is what we want to optimize
  - Recall LP for optimizing weights:

  \[
  \begin{align*}
  \text{Vars: } & w_1, \ldots, w_k, \phi \\
  \text{Minimize: } & \phi \\
  \text{S.T.: } & \phi \geq V(s) - Aw(s), \forall s \\
  & \phi \geq Aw(s) - V(s), \forall s
  \end{align*}
  \]

- \(\phi \geq V(s) - Aw(s), \forall s\)
  - equiv. to \(\phi \geq \max \{V(s) - Aw(s) \mid s \in S\}\)
  - equiv. to \(\phi \geq \max \{\Sigma f_j(z_j \mid w) \mid x \in X\}\)

---

**Factored Projection: Factored LP**

- The constraints: \(\phi \geq \max \{\Sigma f_j(z_j \mid w) \mid x \in X\}\)
  - exponentially many
  - but we can “simulate” VE to reduce the expression of these constraints in the LP
  - the number of constraints (and new variables) will be bounded by the “complexity of VE”
Factored Projection: Factored LP

- Choose an elimination ordering for computing
  \[
  \max \{ \Sigma f_j(z_j ; w) , x \in X \}
  \]
  - note: weight vector \( w \) is unknown
  - but structure of VE remains the same (actual numbers can’t be computed)

- For each factor (initial and intermediate) \( e(Z) \)
  - create a new variable \( u(e,z_1,\ldots,z_n) \) for each instantiation \( z_1,\ldots,z_n \) of the domain \( Z \)
  - number of new variables exponential in size (\#vars) of factor

Factored Projection: Factored LP

- For each initial factor \( f_j(Z_j ; w) \), pose constraint:
  \[
  u(f_j,z_1,\ldots,z_n,w) = f_j(z_1,\ldots,z_n;w) , \forall z_1,\ldots,z_n
  \]
  - though the \( w \) are vars, \( f_j(Z_j ; w) \) linear in \( w \)
Factored Projection: Factored LP

- For elim step where $X_k$ removed, let
  - $g_k(Z_k ; w) = \max x_k g_{k1}(Z_{k1} ; w) + g_{k2}(Z_{k2} ; w) + ...$
  - here each $g_{kj}$ a factor including $X_k$ (and is removed)
- For each intrm factor $f_k(Z_k ; w)$, pose constraint:

\[
\begin{align*}
  u(g_k, z_1, \ldots, z_n, w) &\geq \\
  g_{k1}(z_1, \ldots, z_{n1}; w) + g_{k1}(z_1, \ldots, z_{n1}; w) + \ldots + \forall x_k, \forall z_1, \ldots, z_n
\end{align*}
\]

- constraints linear in $w$
- force $u$-values for each factor to be at least max over $X_k$ values
- number of constraints: size of factor * $|X_k|

Factored Projection: Factored LP

- Finally pose constraint
  - $\phi \geq u_{\text{final}}()$
- This ensures:

\[
\phi \geq \max \{ \sum f_i(z_i ; w) , x \in X\} = \max \{ V(s) - A w(s) , s \in S\}
\]

- Note: objective function in LP minimizes $\phi$
  - so constraints are satisfied at the max values
- In this way
  - we optimize weights at each iteration of Vallter
  - but we never enumerate the state space
  - size of LPs bounded by total factor size in VE
Some Results [GKP-01]

- characteristic functions over single variables
- characteristic functions over pairs of variables

Basis sets considered:

Some Results [GKP-01]

Computation Time
Some Results [GKP-01]

Computation Time

Some Results [GKP-01]

Relative error wrt optimal VF (small problems)
Linear Approximation: Summary

- Results seem encouraging
  - 40 variable problems solved in a few hours
  - simple basis sets seem to work well for “network” problems
- Open issues:
  - are tighter (a priori) error bounds possible?
  - better computational performance?
  - where do basis functions come from?
    - what impact can good/poor basis set have on solution quality?
  - are there “nonlinear” generalizations?

An LP Formulation

- AVI requires generating a large number of constraints (and solving multiple LPs/cost nets)
- But normal MDP can be solved by an LP directly:
  - \((L^aV)(s)\) is linear in values/vars \(V(s)\)

\[
\begin{align*}
\text{Vars: } V(s) \\
\text{Minimize: } \sum_s V(s) \\
\text{S.T. } V(s) \geq (L^aV)(s), \forall a, s
\end{align*}
\]
Using Structure in LP Formulation

- These constraints can be formulated without enumerating state space using cost network as before [SchPat-00]
  - by not iterating, great computational savings possible
    - a couple orders of magnitude on “networks”
  - techniques like constraint generation offer even more substantial savings

Good Basis Sets

- A good basis set should
  - be reasonably small and well-factored
  - be such that a good approximation to \( V^* \) lies in the subspace \( B \)
- Latter condition hard to guarantee
- Possible ways to construct basis sets
  - use prior knowledge of domain structure
    - e.g., problem decomposition
  - search over candidate basis sets
    - e.g., sol’n using a poor approximation might guide search for an improved basis
Parallel Problem Decomposition

- Decompose MDP into parallel processes
  - product/join decomp.
  - each refers to subset of relevant variables
  - actions affect each

- Key issues:
  - how to decompose?
  - how to merge sol’ns?

- Contrast serial decomposition
  - macros [Sutton95, Parr98]

Generating SubMDPs

- Components of additive reward: subobjectives
  - often combinatorics due to many competing objectives
  - e.g., logistics, process planning, order scheduling
  - [BouBrafmanGeib97, SinghCohn97, MHKPDB98]

- Create subMDPs for subobjectives
  - use abstraction methods discussed earlier to find subMDP relevant to each subobjective
  - solve using standard methods, DTR, etc.
Generating SubMDPs

Dynamic Bayes Net over Variable Set

Generating SubMDPs

Green SubMDP (subset of variables)
Generating SubMDPs

Red SubMDP (subset of variables)

Composing Solutions

- Existing methods piece together solutions in an online fashion; for example:
  1. Search-based composition [BouBrafmanGeib97]:
     - VFIs used in heuristic search
     - partial ordering of actions used to merge
  2. Markov Task Decomposition [MHPKDB98]:
     - has ability to deal with large actions spaces
     - MDPs with thousands of variables solvable
Search-based Composition

- Online action selection: standard expectimax search
  [DB94,97,BBS95,KS95,BG98,KMN99,...]
- Decomposed VFs viewed as heuristics (reduce requisite search depth for given error)
- E.g., given subVFs $f_1, ..., f_k$

\[ V(s) \leq f_1(s) + f_2(s) + \ldots + f_k(s) \]
\[ V(s) \geq \max \{ f_1(s), f_2(s), \ldots, f_k(s) \} \]
Offline Composition

- These subMDP solutions can be “composed” by treating subMDP VFVs as a basis set
- Approx. VF is a linear combination of the subVFVs
- Some preliminary results [Patrascu et al. 02] suggest this technique can work well
  - for decomposable MDPs, subVFVs offer better solution quality than simple characteristic functions
  - often piecewise linear combinations work better than linear combinations [Poupart et al. 02]

Next Time

- Non-Markovian processes
- Temporal logic
- Conversion to Markov process
References


References (con’t)