2534 Lecture 6: Tractable Solutions of MDPs and POMDPs

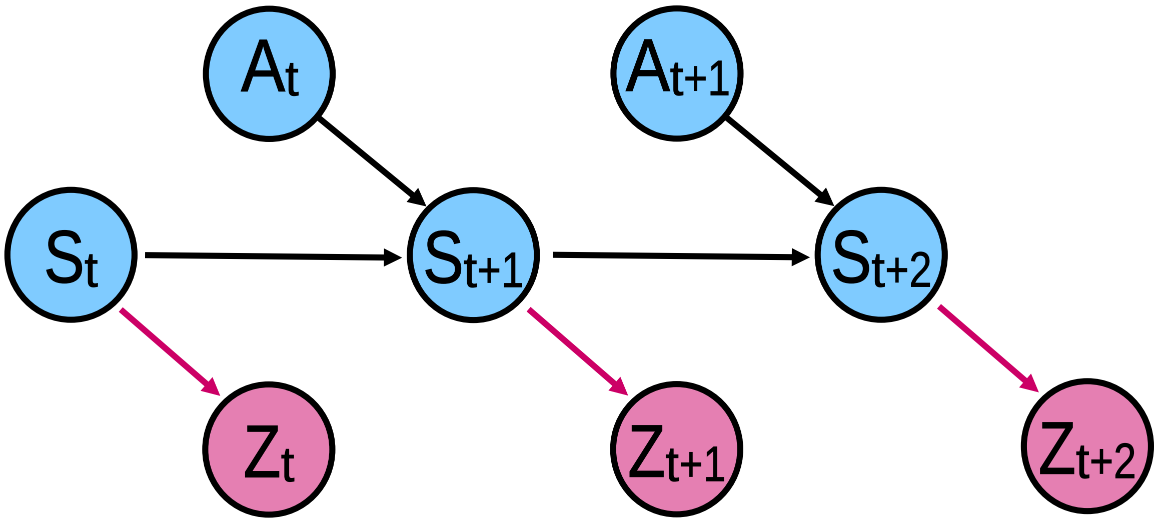
- Discuss basic algorithms for POMDPs (from last time)
- POMDPs: Point-based Value Iteration
- Structured Models of MDPs
- Announcements
  - Asst.1 due today
  - Project discussions slots on Tues, Thurs, Friday this week
    - 20 minute time slots (come prepared)

Recap: POMDPs

- ***POMDPs*** offer a very general model for sequential decision making allowing:
  - uncertainty in action effects
  - *uncertainty in knowledge of system state, noisy observations*
  - multiple (possibly conflicting) objectives
  - nonterminating, process-oriented problems
- It is the *uncertainty in system state* that distinguishes them from MDPs

Recap: POMDPs: Basic Model

- As in MDPs: S, A, p_{ij}^{a}, r_i^a, r_i^T
- Observation space: Z (or Z_a)
- Observation probabilities: p_{ijz}^{a} for $z \in Z_a$



Recap: History-based Policies

- Information available at time *t*:
  - initial distribution (belief state) $b \in \Delta(S)$
  - history of actions, observations: $a^1, z^1, a^2, z^2, \ldots, a^{t-1}, z^{t-1}$
- Thus, we can view a *policy* as a mapping:

$$\pi : \Delta(S) \times H^{t \leq T} \rightarrow A$$

- For given belief state *b*, it is a *conditional plan*

$$\textit{e.g.,} \quad MN;MN;EX;\begin{cases} if\ Def{:}IN;MN;MN\ldots \\ else{:}MN;MN;EX \begin{cases} if\ Def{:}RP;MN\ldots \\ else{:}MN\ldots \end{cases} \end{cases}$$

  - notice distinction with MDPs: can't map from *state* to actions

Recap: Belief States

- History-based policy grows exponentially with horizon
  - infinite horizon POMDPs problematic
- *Belief state* $b \in \Delta(S)$ summarizes history sufficiently **[Aoki (1965), Astrom (1965)]**
- Let *b* be belief state; suppose we take action *a*, get obs *z*
- Let *T(b,a,z)* be *updated belief state* (transition to *new b*)
- If we let b_i denote *Pr(S = i)*, we update:

$$
\begin{aligned}
T(b,a,z)_i &= Pr(i \mid a,z,b) \\
&= \alpha Pr(z \mid i,a,b) Pr(i \mid a,b) \\
&= \frac{\sum_j b_j p^a_{ji} p^a_{jiz}}{\sum_{jk} b_k p^a_{jk} p^a_{jkz}}
\end{aligned}
$$

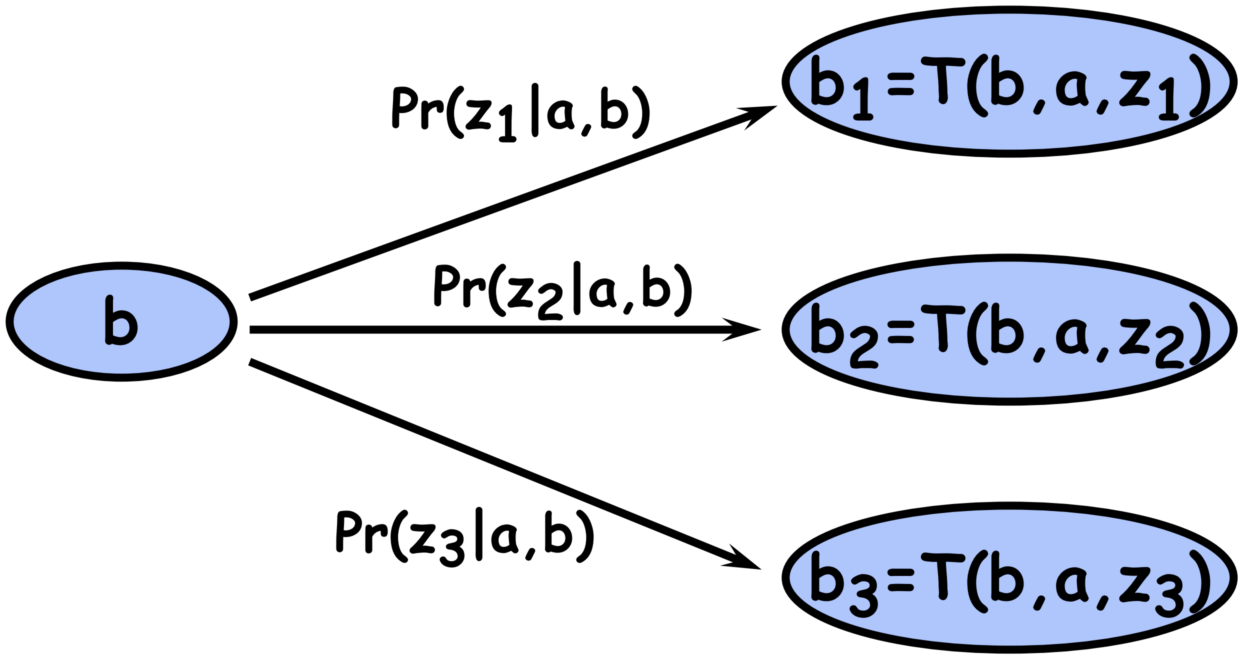
Recap: Belief State MDP

- POMDP now an MDP with state space $\Delta(S)$
- Reward: $r_b^a = b \cdot r^a = \sum_i b_i r_i^a$
- Transitions: $p_{b,b'}^a = \Pr(z \mid b, a)$ if *b' = T(b,a,z); 0 o.w.*
- Optimality Equations:

$$\boxed{\begin{aligned} Q_a^k(b) &= b \cdot r^a + \sum_{b'} p_{b,b'}^a V^{k-1}(b) \\ &= \sum_i b_i \; [r_i^a + \sum_j p_{ij}^a \sum_z p_{ijz}^a V^{k-1}(T(b,a,z))] \end{aligned}}$$

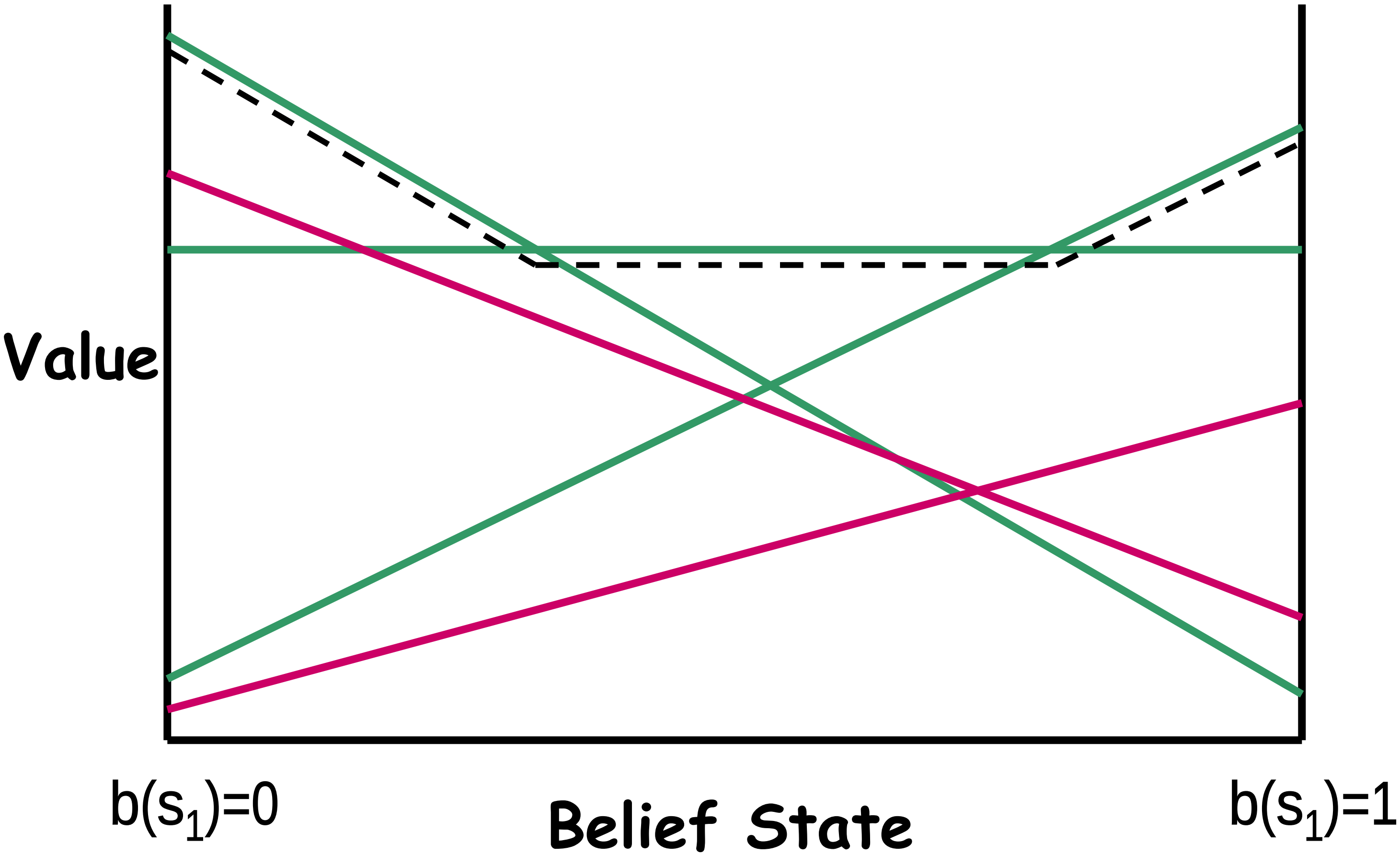
$$\boxed{V^k(b) = \max_a Q_a^k(b)} \qquad \boxed{\pi^k(b) = \arg\max_a Q_a^k(b)}$$

Recap: Belief State MDP Graphically



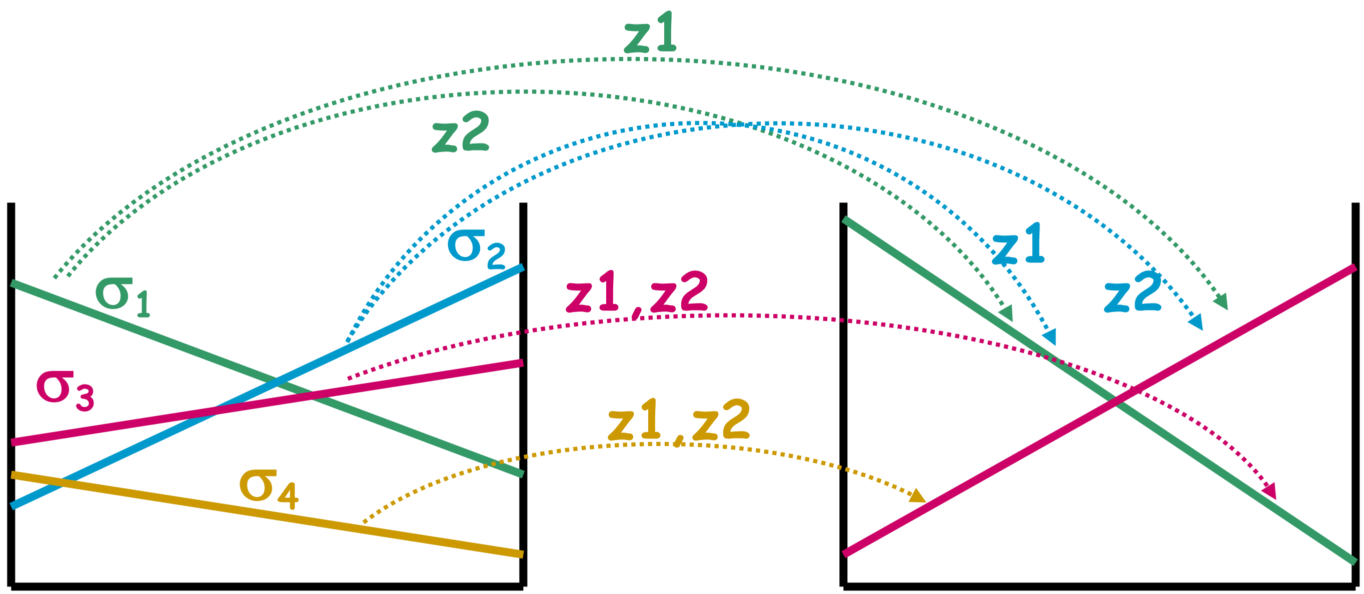
Belief State Transitions for Action a, Belief State b

Recap: PWLC Value Function



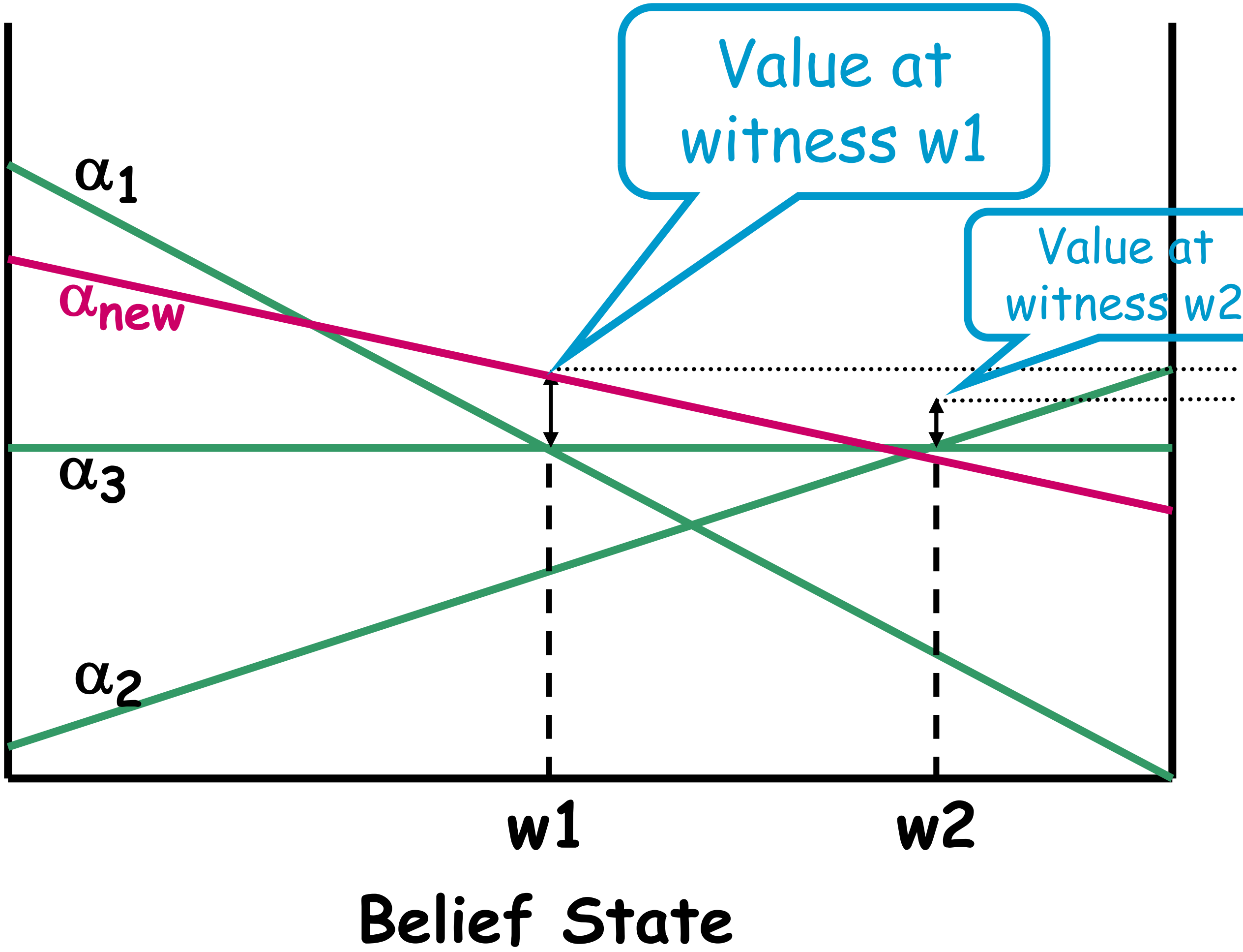
Recap: Representation of Q-function

PWLC Representation of Q_a



σ_1 corresponds to "Do(a);
if z1, do(red);
if z2, do(green)"

Recap: Linear Support Graphically



Sources of Intractability

- Size of α-vectors
  - each is size of state space (exponential in number of variables)
- Number of α-vectors
  - potentially grows exponentially with horizon
- Belief state monitoring
  - must maintain belief state online in order to implement policy using value function
  - belief state representation: size of state space

Approximation Strategies

- Sizes of problems solved exactly are quite small
  - various approximation methods developed
  - often deal with 1000 or so states, not much more
- **Grid-Based Approximations**
  - compute value at small set of belief states
  - require method to “interpolate” value function
  - require grid-selection method (uniform, variable, etc.)
  - *we’ll discuss one method (Perseus/PBVI) today*
- **Finite Memory Approximations**
  - e.g., policy as function of most recent actions, observations
  - can sometimes convert VF into finite-state controller

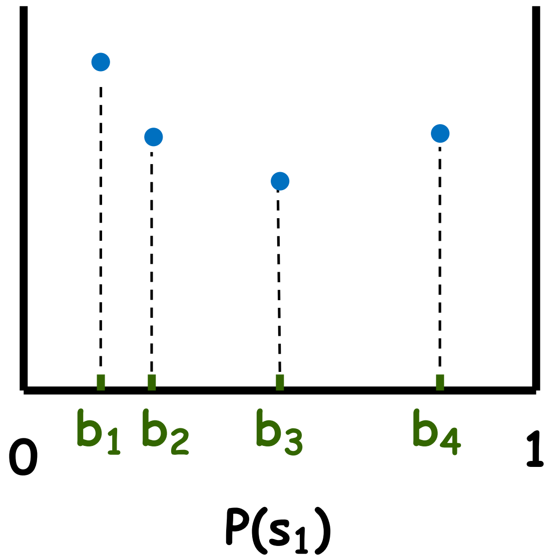
Approximation Strategies

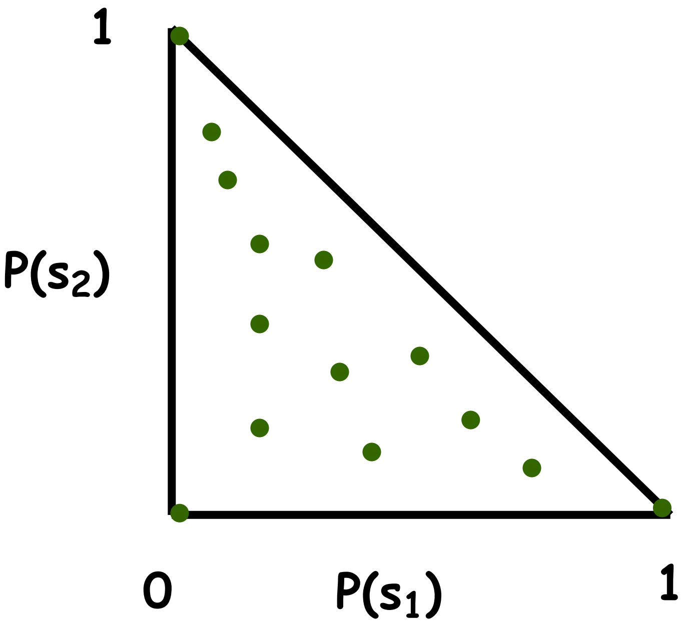
- **Learning Methods**
  - assume specific value function representation
  - e.g., linear value function, smooth approximation, neural net
  - train representation through simulation
- **Heuristic Search Methods**
  - search through belief space from initial state
  - requires good heuristic for leverage
  - heuristics could be generated by other methods
- **Structure-based Approximations**
  - E.g., based on decomposability of problem

Grid-based Approximations

- High level motivation:
  - number of a vectors grows exponentially (even in practice) with horizon (one of biggest impediments to solving POMDPs)
  - intuitively, need optimal policies for every belief point
  - instead, we could select a finite sample (or grid) of belief points on the *n*-dimensional simplex and compute optimal value function (or policy) for those points
  - for any other belief points not on grid, use some interpolation scheme
  - can define a simple value iteration scheme based on this idea

Belief Grid (2-D, 3-D), with VF (2-D)





2 state POMDP (s_0, s_1)

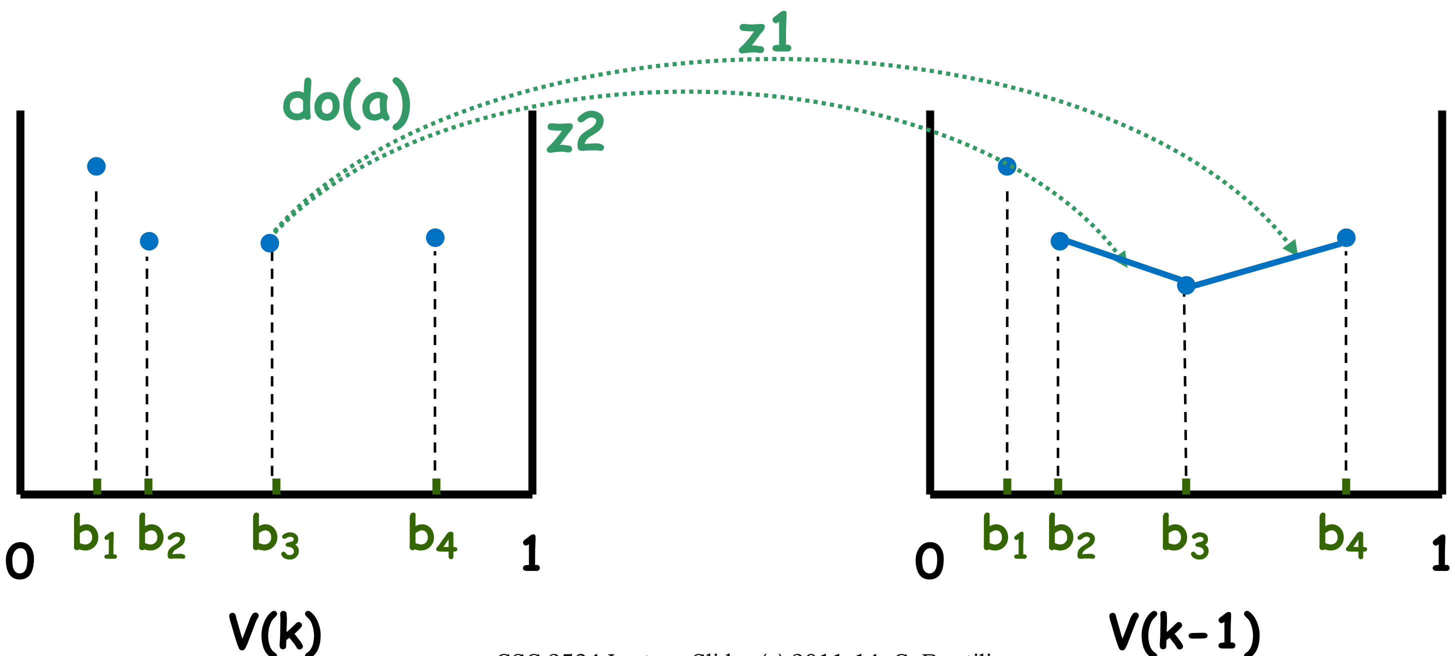
3 state POMDP (s_0, s_1, s_2)

Grid-based Value Iteration

- Given value function *V(k-1)* on grid *B*
- Compute value *V(k)* at grid points in usual way

$$\boxed{Q_a^k(b) = \sum_i b_i \; [r_i^a + \sum_j p_{ij}^a \sum_z p_{ijz}^a V^{k-1}(T(b,a,z))]}$$

- Problem: *T(b,a,z)* not usually on grid even if *b* is
- Solution: use some form of interpolation over *V(k-1)*

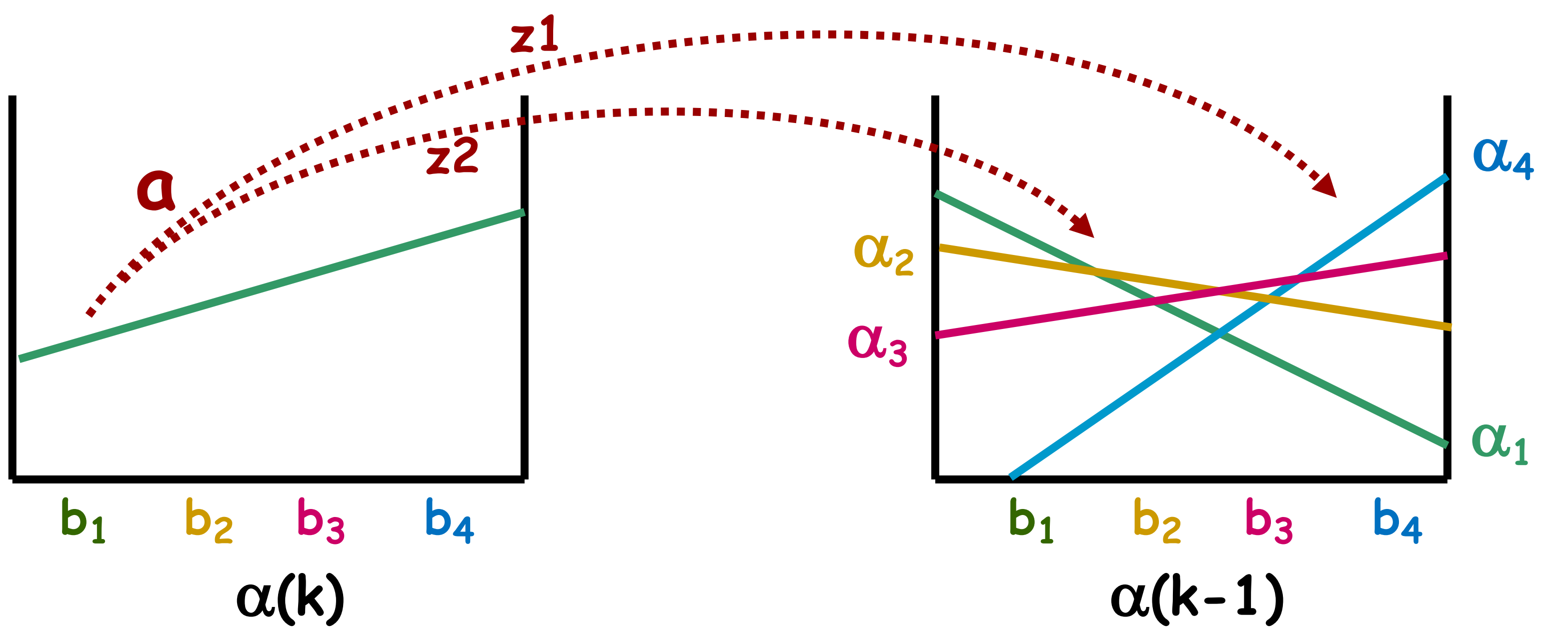


Point-based Value Iteration

- Grid-based methods expensive, performance debatable
  - Selecting suitable grid, interpolation can be expensive
- But recall approximation based on Cheng's linear support
  - just use a subset of α-vectors
- PBVI methods combine the two insights
  - select a small subset of belief points
  - but compute/backup α-vectors instead of just values
  - no interpolation, use collection of α-vectors as VF representation
- Briefly, let's look at:
  - Pineau's original *PBVI*
  - Spaan and Vlassis *Perseus*

Point-based Value Iteration

- Main idea (roughly)
  - fix a small set of belief points *B*
  - assume approximate set of α-vectors *V(k-1)*
  - do backups for each *b* in *B*, using *V(k-1)*, to construct *V(k)*
  - can prune (remove dominated vectors)
  - can expand set of belief points in an anytime fashion (add new belief points if you want, as time permits)



PBVI: Which Belief States (Grid)?

- Initial belief states B
  - starting at b_0, consider updated *T(b,z,a)* reached by taking action *a* and sampling *a* random observation *z* (sample *z* with *Pr(z|b,a)*)
  - take belief state from *one* of these actions, the one that is *greatest distance* (L1 or L2) from any belief point in the set
    - aim: trying to get maximum coverage of belief space (diversity, but informed by reachability considerations)
- Repeat as time permits, consider expanding belief set *B* by
  - using same process as above, for *each b in B*
  - double size of belief set at each iteration until you are "satisfied" with coverage (or number of belief states reaches some threshold)
- Paper discusses other methods for generating belief points
  - experiments don't show large differences except for one (large) domain

PBVI: Observations

- Time complexity: each backup takes *O(SAOVB)* $\approx$ *O(SAOB²)*
  - each backup requires *AO* belief projections
  - each projection required *V* value evaluations (to determine which vector has max value)
  - each projection/evaluation takes *O(S)* time
  - *B* points to backup (and *V* is bounded by *B*)
- Error can be bounded based on density of belief grid
  - result is straightforward, bound is a bit too loose to be useful

Theorem 1 *For any belief set B and any horizon n, the error of the PBVI algorithm* $\eta_n = \|V_n^B - V_n^*\|_\infty$ *is bounded by*

$$\eta_n \leq \frac{(R_{max} - R_{min})\epsilon_B}{(1-\gamma)^2}$$

Introduce an error by pruning away alpha vectors at each stage of:
Rmax-Rmin*eps / (1-gamma)

PBVI: Performance (works pretty well)

Method	Goal%	Reward	Time(s)	$\|B\|$
Maze33 / Tiger-Grid				
QMDP[*]	n.a.	0.198	0.19	n.a.
Grid [Brafman, 1997]	n.a.	0.94	n.v.	174
PBUA [Poon, 2001]	n.a.	2.30	12116	660
PBVI[*]	n.a.	2.25	3448	470
Hallway				
QMDP[*]	47	0.261	0.51	n.a.
QMDP [Littman *et al.*, 1995]	47.4	n.v.	n.v.	n.a.
PBUA [Poon, 2001]	100	0.53	450	300
PBVI[*]	96	0.53	288	86
Hallway2				
QMDP[*]	22	0.109	1.44	n.a.
QMDP [Littman *et al.*, 1995]	25.9	n.v.	n.v.	n.a.
Grid [Brafman, 1997]	98	n.v.	n.v.	337
PBUA [Poon, 2001]	100	0.35	27898	1840
PBVI[*]	98	0.34	360	95
Tag				
QMDP[*]	17	-16.769	13.55	n.a.
PBVI[*]	59	-9.180	180880	1334

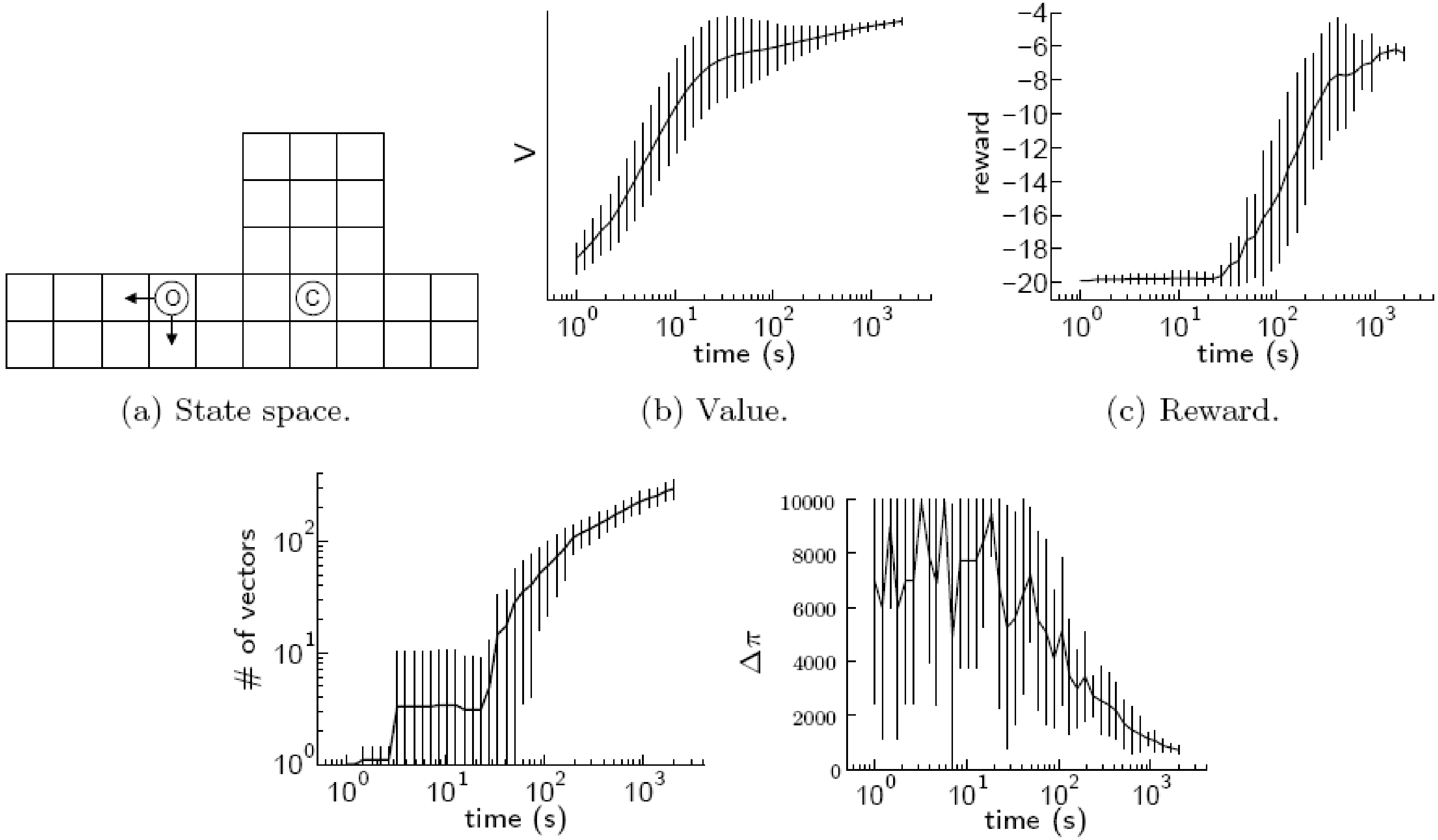
n.a.=not applicable n.v.=not available

Name	$\|S\|$	$\|O\|$	$\|A\|$
Tiger-grid	33	17	5
Hallway	57	21	5
Hallway2	89	17	5
Tag	870	30	5

PERSEUS

- Perseus makes a small but useful tweak on PBVI
  - fixes a set of belief states B
  - given *V(k-1)*, does not update *all* belief states to get *V(k)*, instead:
    - select a random *b* from *B*
    - do a point-based backup to get a new α-vector α*(b)* for *b*
      - *if new α-vector not improving, use best old one from V(k-1)*
    - if α*(b)* improves any other *b'* in *B*, then do not backup *b'*
    - continue until all belief states *b'* in *B* have "improved", either through *their own* backup or by that of *some other b*
- Simple idea: don't waste backups on *b* in *B* if other backups have improved its value anyway
  - little you can prove about this, but it keeps the size of the sets *V(k)* of α-vectors *much* smaller in practice

Perseus Performance (TAG domain)



(a) State space. (b) Value. (c) Reward.

(d) Nr. of vectors. (e) Policy changes.

Figure 2: Tag: (a) state space with chasing and opponent robot; (b)–(e) performance of PERSEUS.

Perseus Performance (Comparative)

Tiger-grid	R	$\lvert\pi\rvert$	T
HSVI	2.35	4860	10341
Perseus	2.34	134	104
PBUA	2.30	660	12116
PBVI	2.25	470	3448
BPI w/b	2.22	120	1000
Grid	0.94	174	n.a.
Q_{MDP}	0.23	n.a.	2.76

(a) Results for Tiger-grid.

Hallway	R	$\lvert\pi\rvert$	T
PBVI	0.53	86	288
PBUA	0.53	300	450
HSVI	0.52	1341	10836
Perseus	0.51	55	35
BPI w/b	0.51	43	185
Q_{MDP}	0.27	n.a.	1.34

(b) Results for Hallway.

Hallway2	R	$\lvert\pi\rvert$	T
Perseus	0.35	56	10
HSVI	0.35	1571	10010
PBUA	0.35	1840	27898
PBVI	0.34	95	360
BPI w/b	0.32	60	790
Q_{MDP}	0.09	n.a.	2.23

(c) Results for Hallway2.

Tag	R	$\lvert\pi\rvert$	T
Perseus	-6.17	280	1670
HSVI	-6.37	1657	10113
BPI w/b	-6.65	17	250
BBSLS	≈ -8.3	30	10^5
BPI n/b	-9.18	940	59772
PBVI	-9.18	1334	180880
Q_{MDP}	-16.9	n.a.	16.1

(d) Results for Tag.

State Space Explosion

- For MDPs/POMDPs, state space explosion is a key issue
  - MDPs, POMDPs: transition, reward, obs rep'n are $O(S^2)$, $O(S)$
  - MDPs: value functions and policies: $O(S)$
  - POMDPs: each α-vector (just a VF): $O(S)$
- Most problems (in AI especially) are feature-based
  - *S* is exponential in number of variables
  - Specification/representation of problem in state form impractical
  - Explicit state-based dynamic programming impractical
- Require structured representations
  - exploit regularities in probabilities, rewards
- Require structured computation
  - exploit regularities in policies, value functions
  - can aid in approximation (anytime computation)

Structured Representation

- States decomposable into *state variables*

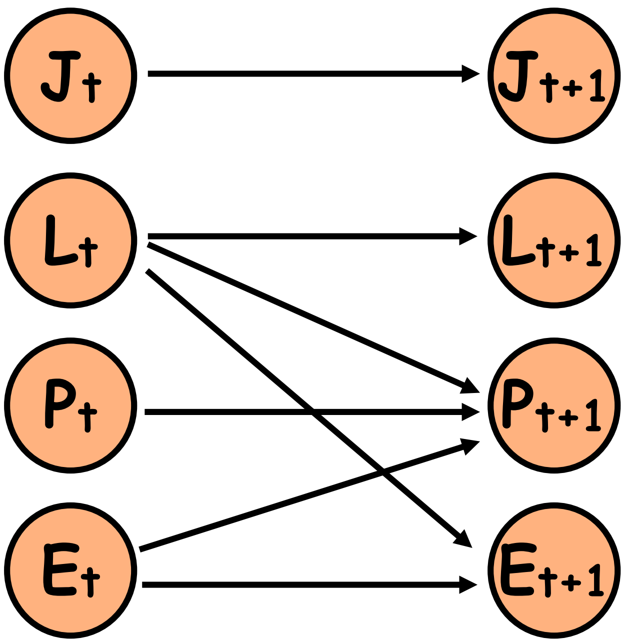
$$S = X_1 \times X_2 \times \ldots X_n$$

- *Structured* representations the norm in AI
  - STRIPS, Sit-Calc., Bayesian networks, etc.
  - Describe *how actions affect/depend on features*
  - Natural, concise, can be exploited computationally
- Same ideas can be used for MDPs
  - actions, rewards, policies, value functions, etc.
  - dynamic Bayes nets **[DeanKanazawa89,BouDeaGol95]**
  - decision trees and diagrams **[BouDeaGol95,Hoeyetal99]**

Action Representation – DBN/ADD

J – Joe needs coffee
L – robot in printer room
P – robot has printout
E – robot gripper empty

Pickup Printout



J	J(t+1)	$\overline{J(t+1)}$
T	1.0	0.0
F	0.0	1.0

$f_J(J_t, J_{t+1})$

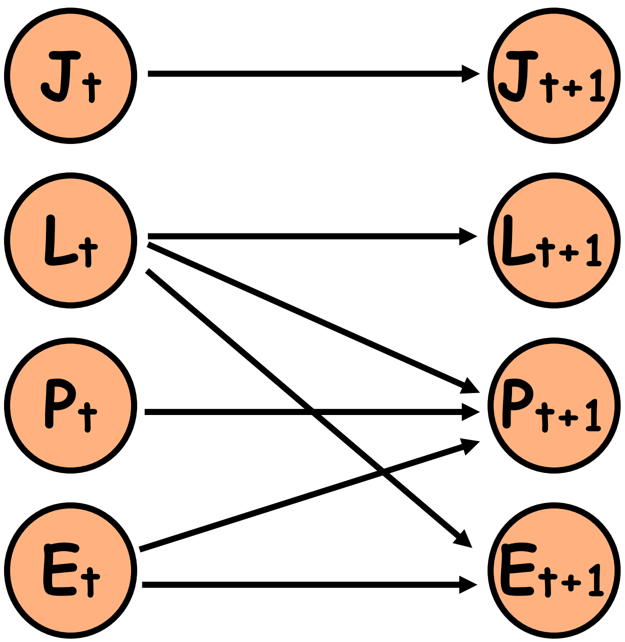
L	E	P	P(t+1)	$\overline{P(t+1)}$
T	T	T	1.0	0.0
F	T	T	1.0	0.0
T	F	T	1.0	0.0
F	F	T	1.0	0.0
T	T	F	0.8	0.2
F	T	F	0.0	1.0
T	F	F	0.0	1.0
F	F	F	0.0	1.0

$f_P(L_t, P_t, E_t, P_{t+1})$

Action Representation – DBN/ADD

$$Pr(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1} \mid J_t, L_t, P_t, E_t)$$

$$= f_J(J_t, J_{t+1}) * f_P(L_t, P_t, E_t, P_{t+1}) * f_L(L_t, L_{t+1}) * f_E(E_t, E_{t+1})$$



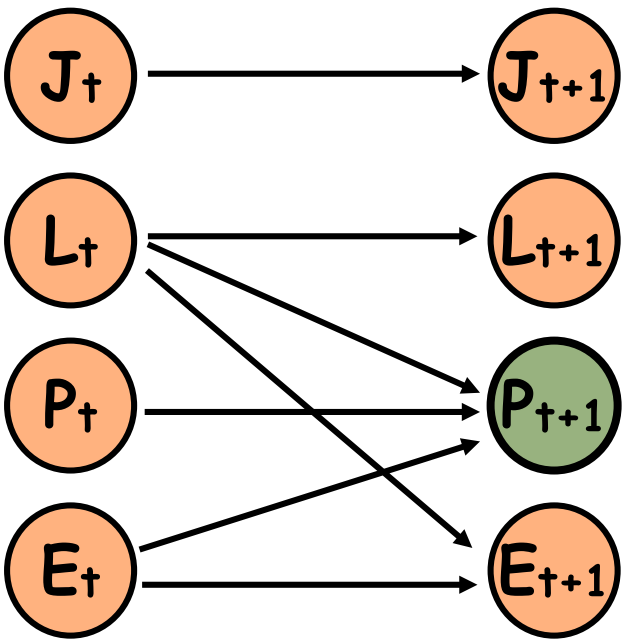
- Only 28 parameters vs. 256 for matrix

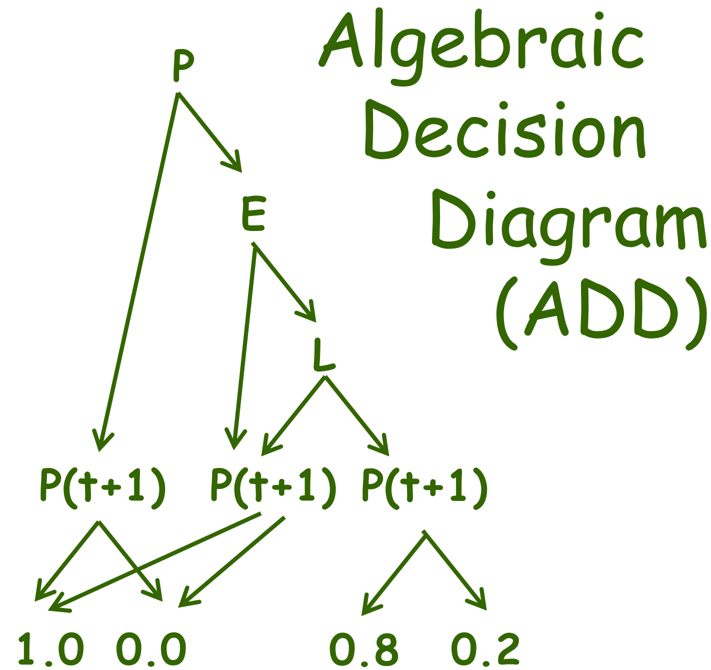
	s_1	s_2	...	s_{256}
s_1	0.9	0.05	...	0.0
s_2	0.0	0.20	...	0.1
$\vdots$				
s_6	0.1	0.0	...	0.0

-Removes global exponential dependence

Action Representation – DBN/ADD

Pickup Printout





- ADDs, decision trees, Horn rules,
- both compact and natural

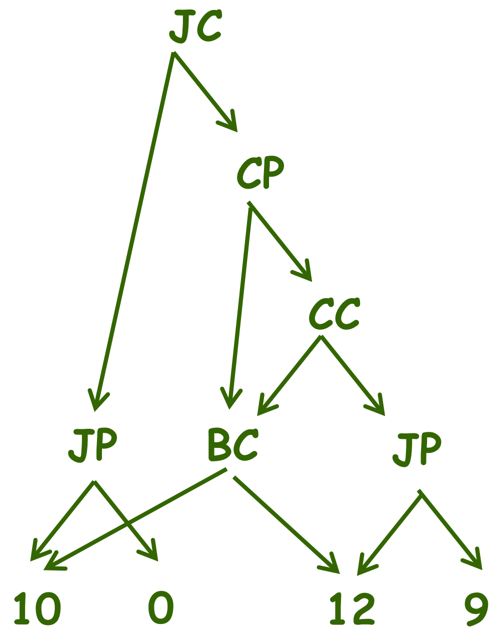
DBN Remarks

- Dynamic Bayes net action representation
  - each state variable occurs at time *t* and *t+1*
  - dependence of time *t+1* variables on time *t* variables
    - can also depend on other time *t+1* variables (provided the DBN remains acyclic) to capture correlations in action effects
  - *no quantification* of time *t* variables is specified (since we don't care about prior)
    - so DBN represents *a family of conditional distributions* over the time *t+1* variables given the time *t* variables
  - compact representation of CPTs using trees, ADDs, Horn rules exploits *context-specific independence* **[BFGK96]**

Reward Representation

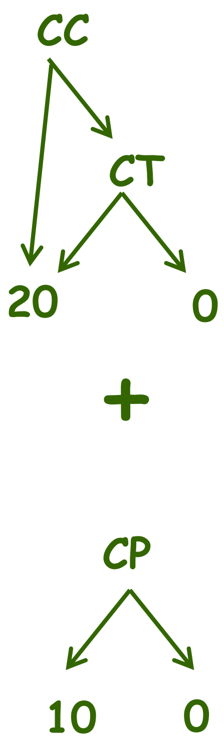
- Rewards represented similarly
  - save on 2^n size of vector rep'n

JC – Joe has coffee
JP – Joe has printout
CC – Craig has coffee
CP – Craig has printout
BC- Battery charged



Reward Representation

- Rewards represented similarly
  - save on 2^n size of vector representation
- Additive independent (or GAI) reward also very common
  - as in multi-attribute utility theory
  - offers more natural and concise representation for many types of problems



Structured Computation

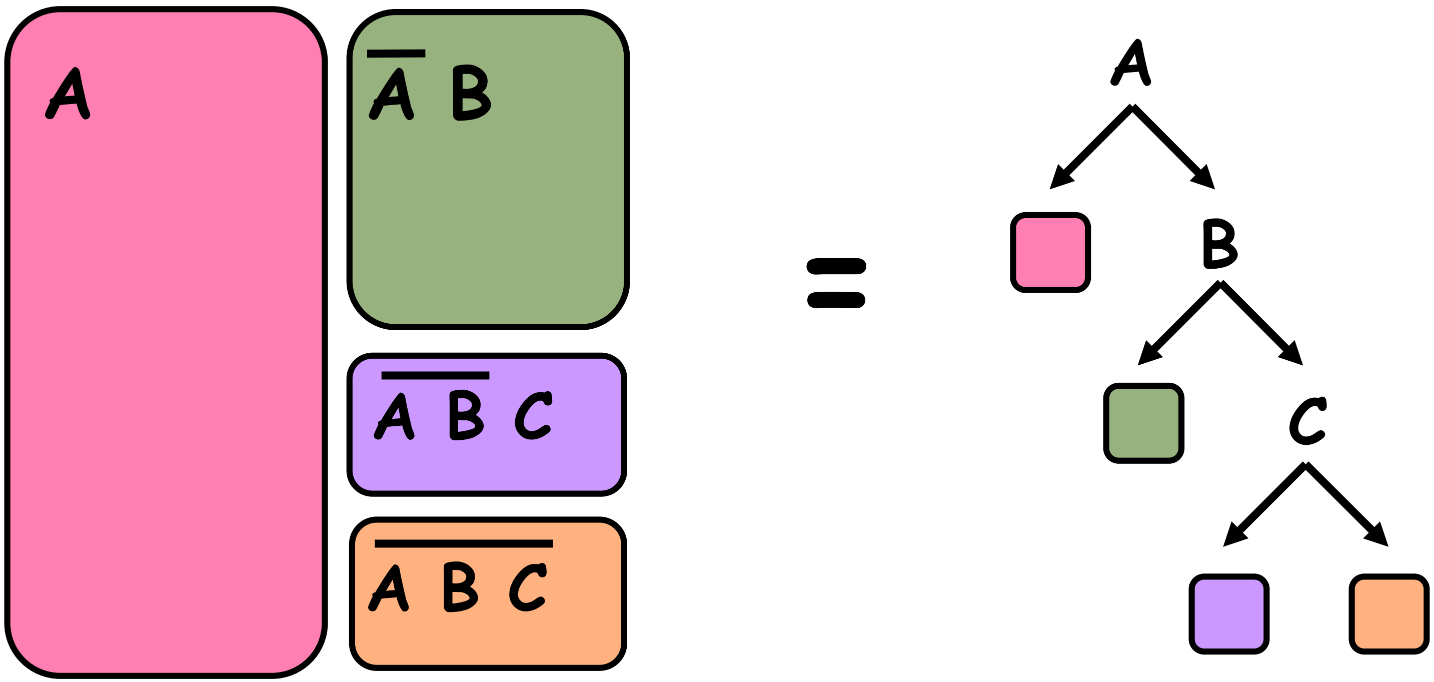
- Given compact representation, can we solve MDP without explicit state space enumeration?
- Can we avoid $O(|S|)$-computations by exploiting regularities made explicit by DBNs/ADDs?

State Space Abstraction

- General method: *state aggregation*
  - group states, treat aggregate as single state
  - commonly used in OR **[SchPutKin85, BertCast89]**
  - viewed as automata minimization **[DeanGivan96]**

- *Abstraction* is a specific aggregation technique
  - aggregate by ignoring details (features)
  - ideally, focus on *relevant* features

Graphical View of Abstraction



Value function (or policy choice) depends only on a small subset of variables (A,B,C) and not others (D,E,F,…); and may do so in a "structured" fashion.

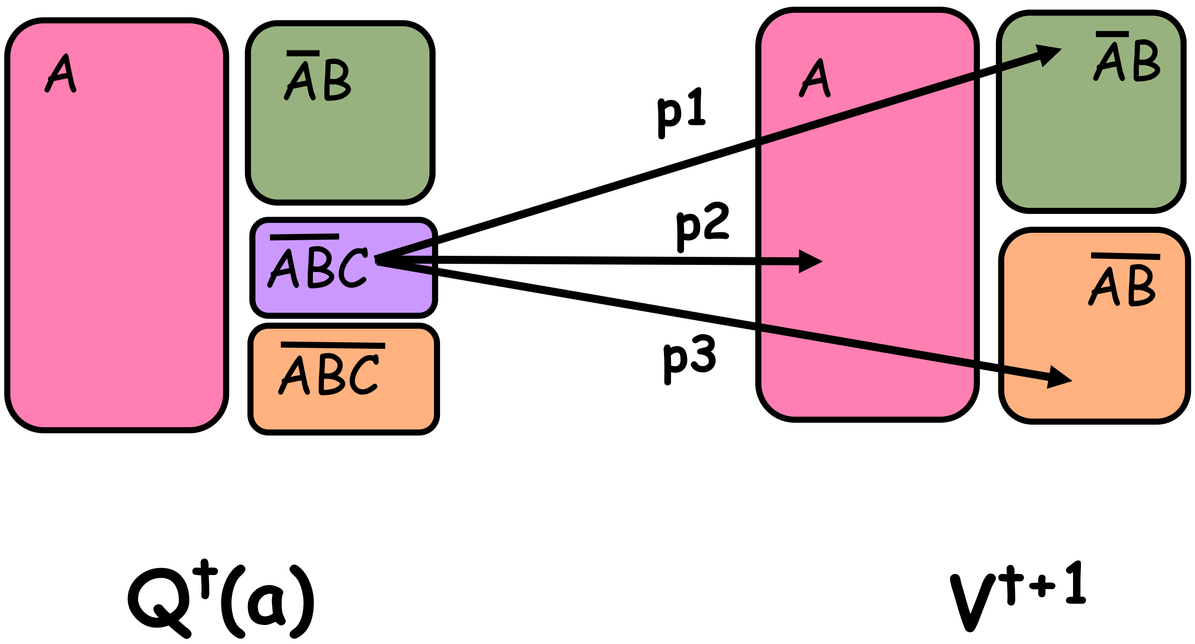
Decision-Theoretic Regression

- *Goal regression* a classical abstraction method
  - *Regr(G,a)* is a logical condition *C* under which *a* leads to *G* (aggregates *C* states and ~*C* states)

- Decision-theoretic analog: given "logical description" of V^{t+1}, produce such a description of V^t or optimal policy (e.g., using ADDs)

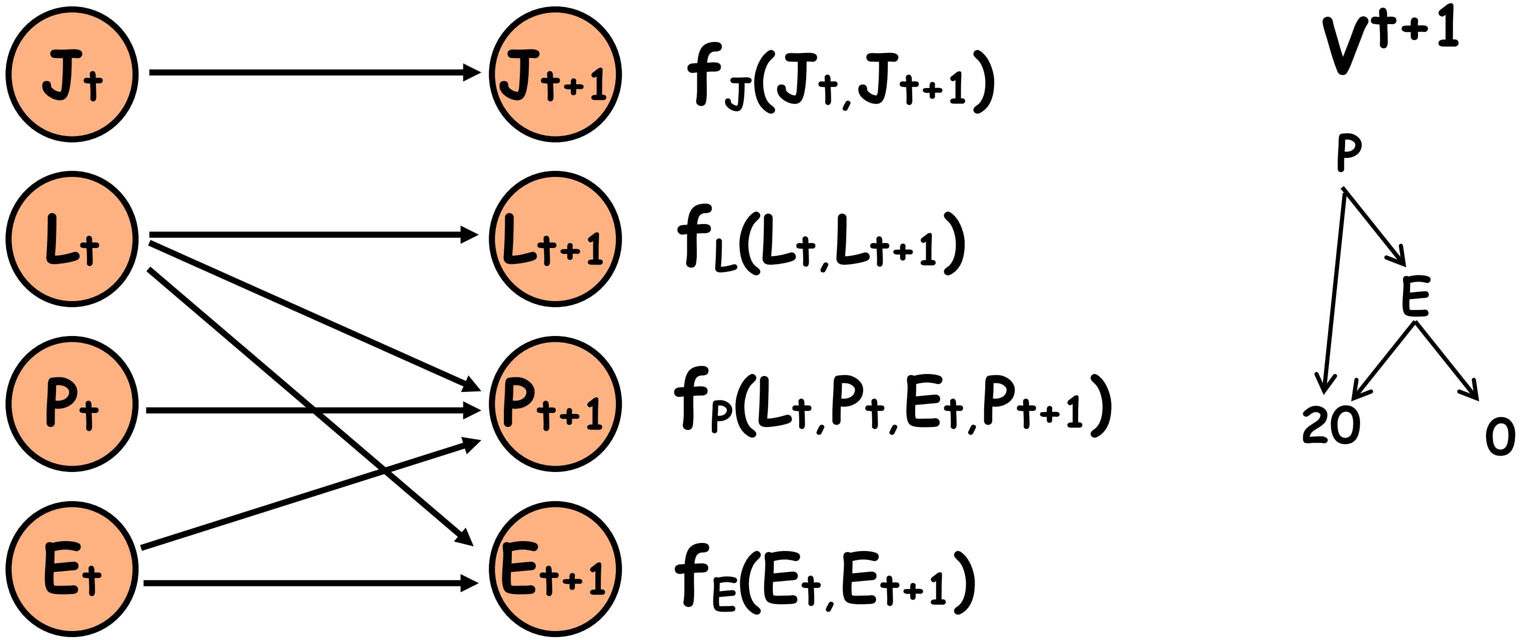
- Cluster together states at any point in calculation with *same best action* (policy), or with *same value* (VF)

A Graphical View of DTR



Functional View of DTR

- Generally, V^{t+1} depends on only a subset of variables (usually in a structured way)
- *What is value of action a at time t (at any s)?*



Functional View of DTR

- Assume VF V^{t+1} is structured: what is value of doing action *a* at time *t* ?
- Use variable elimination!

Functional View of DTR

- Assume VF V^{t+1} is structured: what is value of doing action *a* at time *t* ? (Use variable elimination!)

$Q^a_t(J_t,L_t,P_t,E_t)$

Functional View of DTR

- Assume VF V^{t+1} is structured: what is value of doing action *a* at time *t* ? (Use variable elimination!)

$Q^a_t(J_t,L_t,P_t,E_t)$

$= R + \Sigma_{J,L,P,E(t+1)} \Pr^a(J_{t+1},L_{t+1},P_{t+1},E_{t+1} \mid J_t,L_t,P_t,E_t)\ V_{t+1}(J_{t+1},L_{t+1},P_{t+1},E_{t+1})$

Functional View of DTR

- Assume VF V^{t+1} is structured: what is value of doing action *a* at time *t* ? (Use variable elimination!)

$Q^a_t(J_t, L_t, P_t, E_t)$

$= R + \Sigma_{J,L,P,E(t+1)} \Pr^a(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1} \mid J_t, L_t, P_t, E_t) \; V_{t+1}(J_{t+1}, L_{t+1}, P_{t+1}, E_{t+1})$

$= R + \Sigma_{J,L,P,E(t+1)} f_J(J_t, J_{t+1}) \; f_P(L_t, P_t, E_t, P_{t+1}) \; f_L(L_t, L_{t+1}) \; f_E(E_t, E_{t+1}) \; V_{t+1}(P_{t+1}, E_{t+1})$

Functional View of DTR

- Assume VF V^{t+1} is structured: what is value of doing action *a* at time *t* ? (Use variable elimination!)

$$Q^a_t(J_t,L_t,P_t,E_t)$$

$$= R + \Sigma_{J,L,P,E(t+1)} \Pr^a(J_{t+1},L_{t+1},P_{t+1},E_{t+1} \mid J_t,L_t,P_t,E_t)\ V_{t+1}(J_{t+1},L_{t+1},P_{t+1},E_{t+1})$$

$$= R + \Sigma_{J,L,P,E(t+1)} f_J(J_t,J_{t+1})\ f_P(L_t,P_t,E_t,P_{t+1})\ f_L(L_t,L_{t+1})\ f_E(E_t,E_{t+1})\ V_{t+1}(P_{t+1},E_{t+1})$$

$$= R + \Sigma_{L,P,E(t+1)} f_P(L_t,P_t,E_t,P_{t+1})\ f_L(L_t,L_{t+1})\ f_E(E_t,E_{t+1})\ V_{t+1}(P_{t+1},E_{t+1})$$

Functional View of DTR

$Q^a_t(J_t,L_t,P_t,E_t)$

$= R + \Sigma_{J,L,P,E(t+1)} Pr^a(J_{t+1},L_{t+1},P_{t+1},E_{t+1} \mid J_t,L_t,P_t,E_t)\ V_{t+1}(J_{t+1},L_{t+1},P_{t+1},E_{t+1})$

$= R + \Sigma_{J,L,P,E(t+1)} f_J(J_t,J_{t+1})\ f_P(L_t,P_t,E_t,P_{t+1})\ f_L(L_t,L_{t+1})\ f_E(E_t,E_{t+1})\ V_{t+1}(P_{t+1},E_{t+1})$

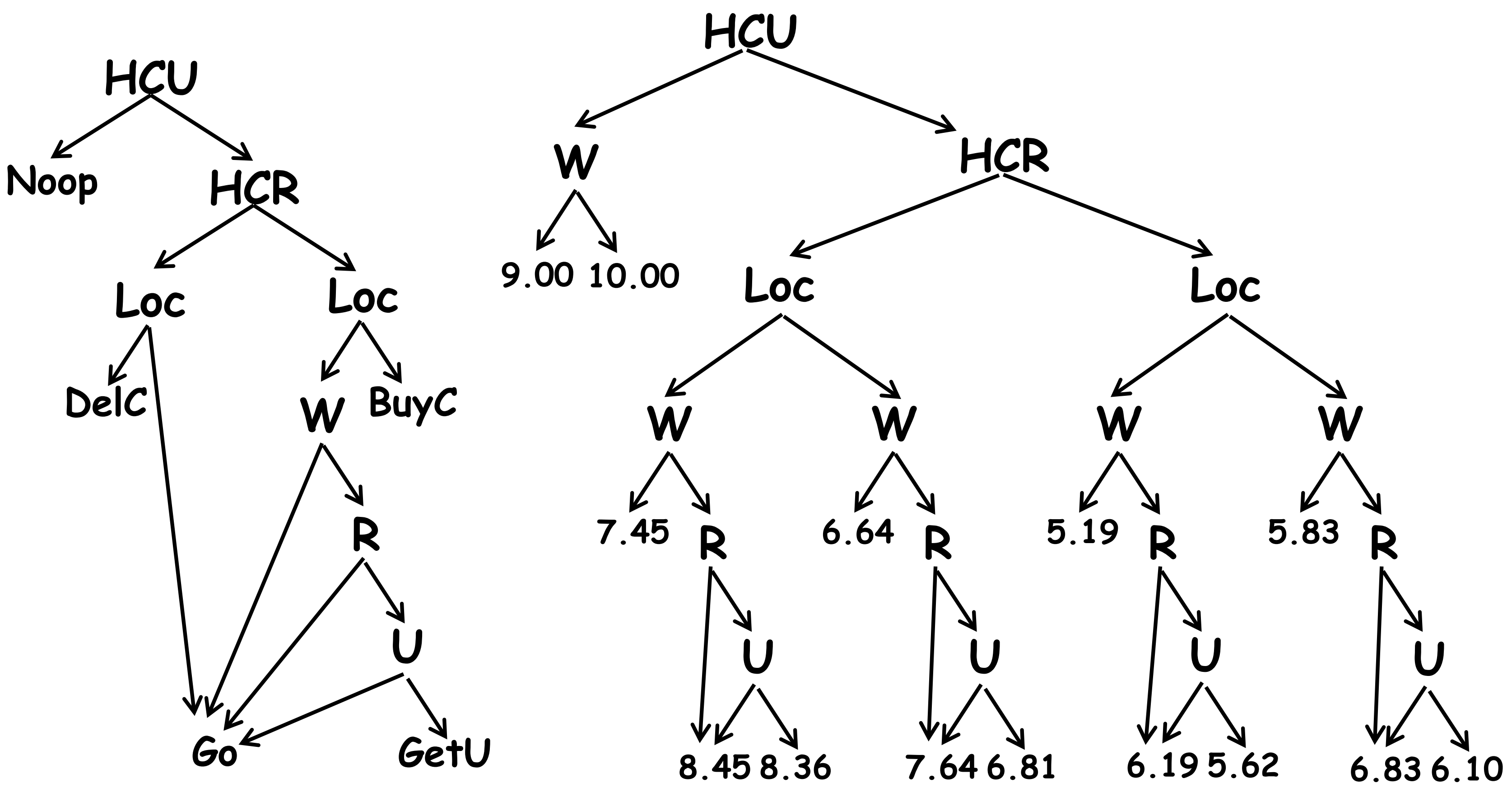
$= R + \Sigma_{L,P,E(t+1)} f_P(L_t,P_t,E_t,P_{t+1})\ f_L(L_t,L_{t+1})\ f_E(E_t,E_{t+1})\ V_{t+1}(P_{t+1},E_{t+1})$

- When V^{t+1} depends on subset of variables:
  - $Q^t(a)$ usually depends on subset of variables as well
  - Computation can be structured without exponential blowup (VE)
  - Further enhancements: Each function represented as ADD
  - … and ADD operations allow structure to be preserved

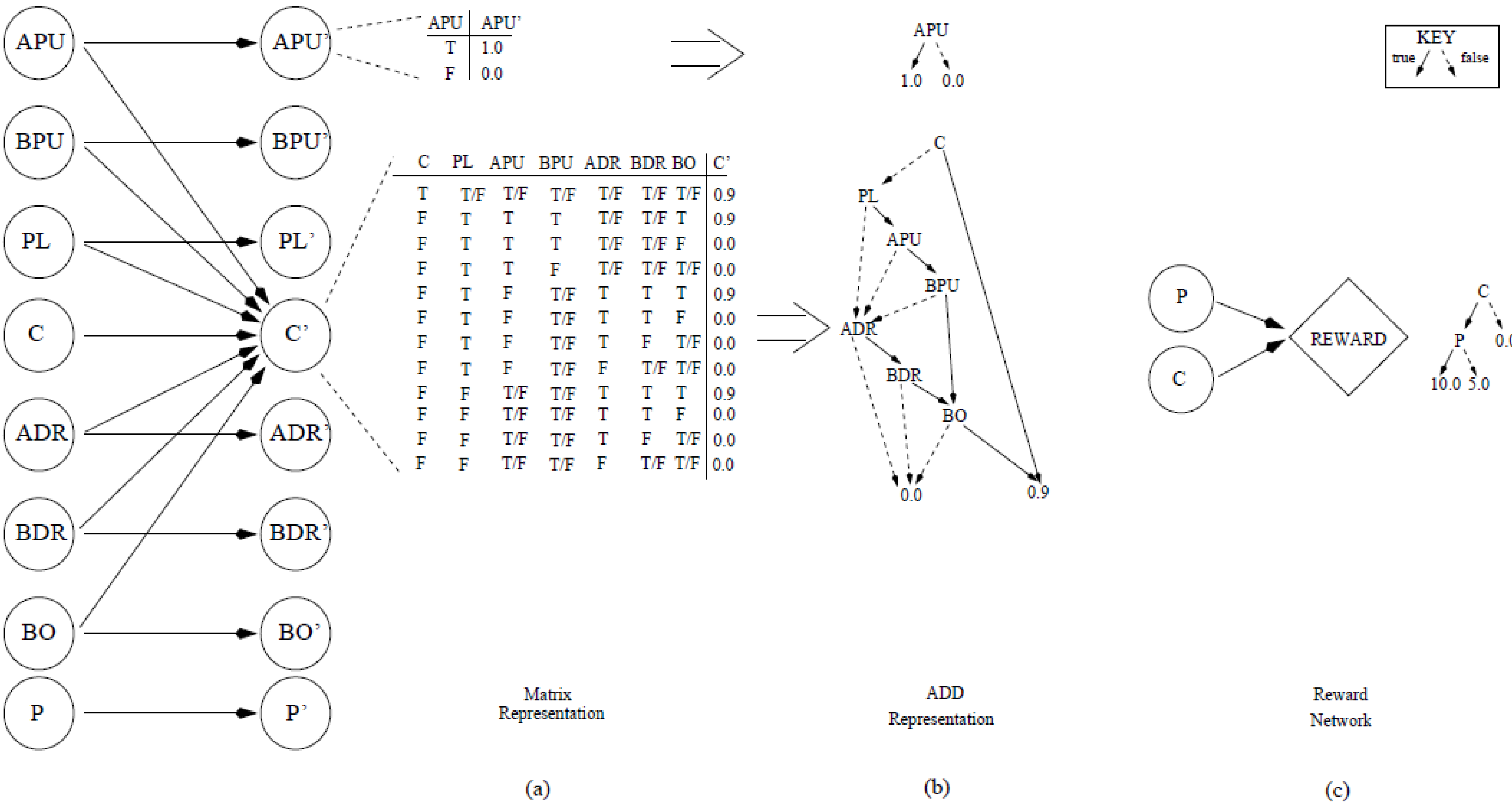
Structured Value Iteration

- Assume compact representation of V^k
  - start with *R* at stage-to-go 0 (say)
- For each action a, compute Q^{k+1} using variable elimination on the two-slice DBN
  - eliminate all *k*-stage-to-go variables, leaving only *k+1* variables
  - use ADD operations when initial representation *(Pr, R)* are ADDs
- Compute $V^{k+1} = max_a\ Q^{k+1}$
  - use ADD operations again to preserve structure, efficiency
- Policy iteration can be approached similarly

Structured Policy and Value Function



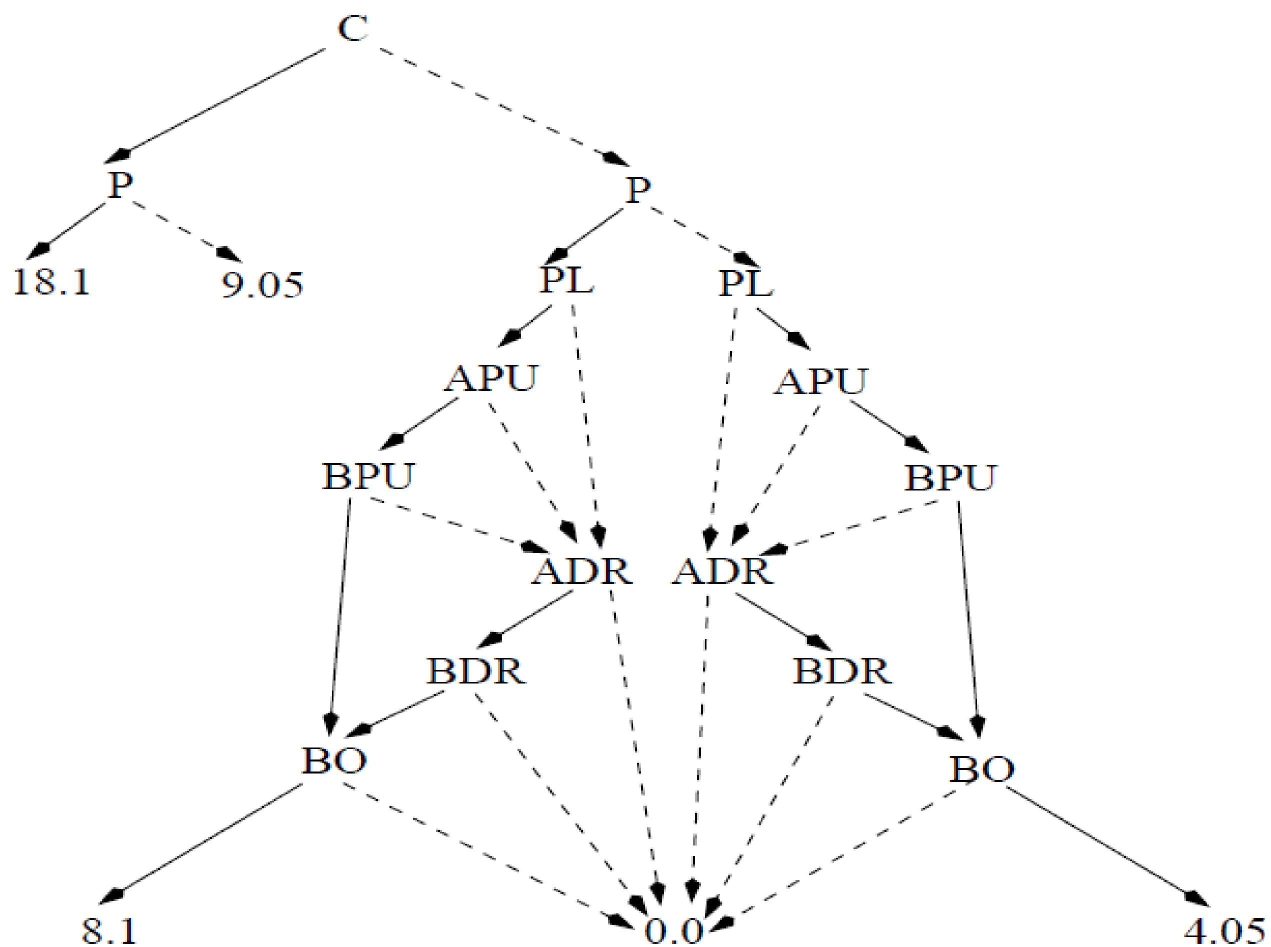
Example Action Reward/Representation



APU	APU'
T	1.0
F	0.0

C	PL	APU	BPU	ADR	BDR	BO	C'
T	T/F	T/F	T/F	T/F	T/F	T/F	0.9
F	T	T	T	T/F	T/F	T	0.9
F	T	T	T	T/F	T/F	F	0.0
F	T	T	F	T/F	T/F	T/F	0.0
F	T	F	T/F	T	T	T	0.9
F	T	F	T/F	T	T	F	0.0
F	T	F	T/F	T	F	T/F	0.0
F	T	F	T/F	F	T/F	T/F	0.0
F	F	T/F	T/F	T	T	T	0.9
F	F	T/F	T/F	T	T	F	0.0
F	F	T/F	T/F	T	F	T/F	0.0
F	F	T/F	T/F	F	T/F	T/F	0.0

ADD: Example Value Function



SPUDD Results

Example Name	State space size: variables ternary	variables total	states	SPUDD - Value: time (s)	internal nodes	leaves	equiv. tree leaves	SPI - Value: time (s)	internal nodes	leaves	ratio of tree nodes: ADD nodes
factory	3	14	55296	-	-	-	-	2210.6	6721	7879	8.12
	0	17	131072	78.0	828	147	8937	2188.23	9513	9514	11.48
factory0	3	16	221184	-	-	-	-	5763.1	15794	18451	13.89
	0	19	524288	111.4	1137	147	14888	6238.4	22611	22612	19.89
factory1	3	18	884736	-	-	-	-	14731.9	31676	37315	14.60
	0	21	2097132	279.0	2169	178	49558	15430.6	44304	44305	20.43
factory2	3	19	1769472	-	-	-	-	14742.4	31676	37315	14.60
	0	22	4194304	462.1	2169	178	49558	15465.0	44304	44305	20.43
factory3	4	21	10616832	-	-	-	-	98340.0	138056	168207	29.31
	0	25	33554432	3609.4	4711	208	242840	112760.1	193318	193319	41.04
factory4	4	24	63700992	-	-	-	-	-	-	-	-
	0	28	268435456	14651.5	7431	238	707890	-	-	-	-

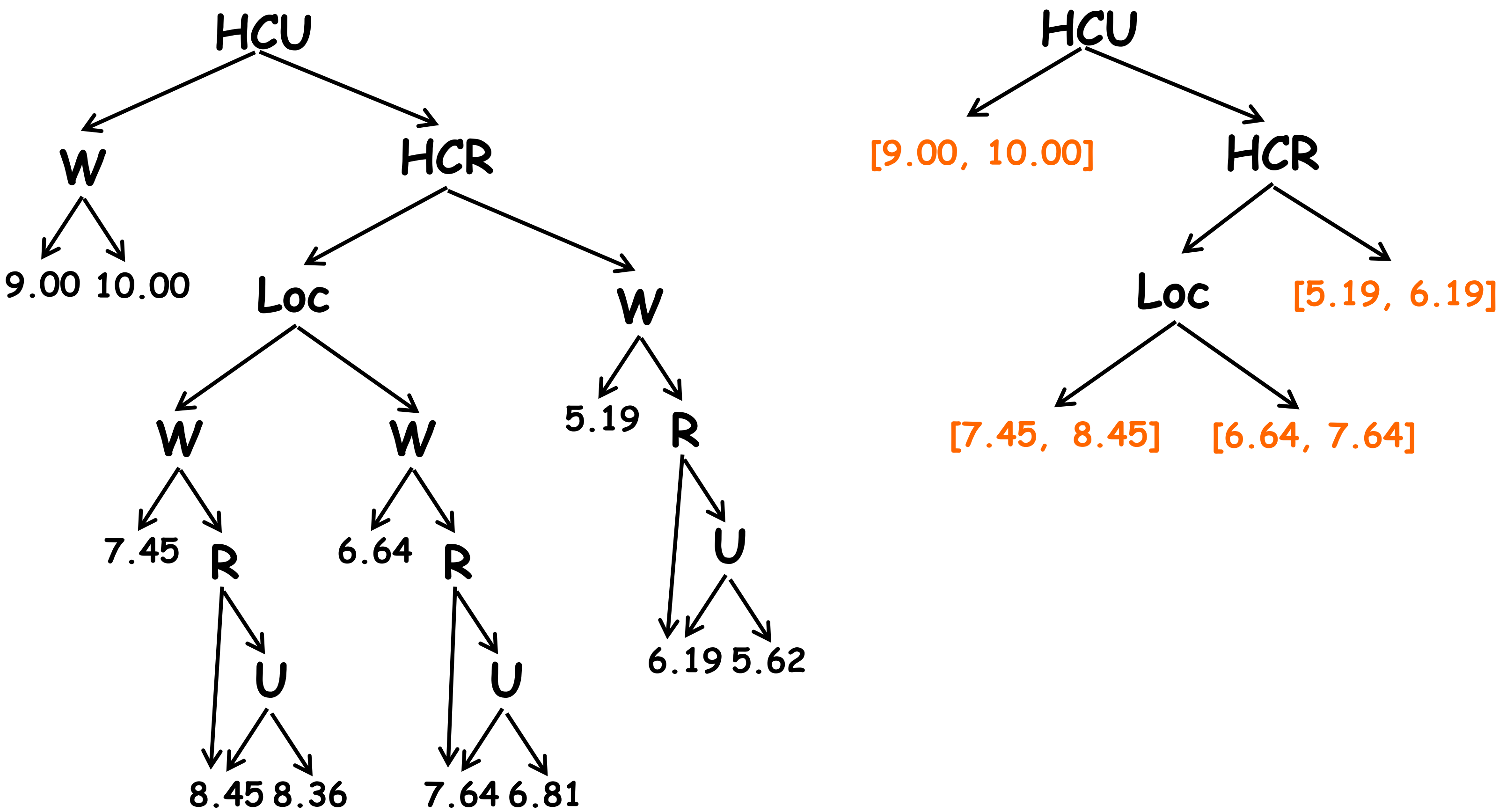
Decision-theoretic Regression: Relative Merits

- Adaptive, nonuniform, exact abstraction method
  - provides exact solution to MDP
  - much more efficient on certain problems (time/space)
  - see SPUDD package
- Some drawbacks
  - produces piecewise constant VF
  - some problems admit no compact solution representation (though ADD overhead “minimal”)
  - approximation may be desirable or necessary

Approximate Decision-theoretic Regression

- Straightforward to approximate solution using DTR
- Simple *pruning* of value function
  - Can prune trees **[BouDearden96]** or ADDs **[StAubinHoeyBou00]**

A Pruned Value ADD



Approximate Decision-theoretic Regression

- Straightforward to approximate solution using DTR
- Simple *pruning* of value function
  - Can prune trees **[BouDearden96]** or ADDs **[StAubinHoeyBou00]**
- Gives regions of *approximately same value*
- Can derive simple error bounds as well
  - e.g., for pruned versions of value iteration (with discount factor β, stopping criterion ε and maximum approximation span δ:

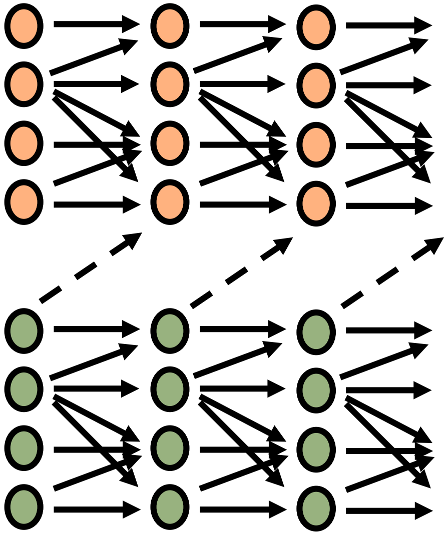
$$\left\| V^* - V_\pi \right\| \leq \frac{2\beta(2\delta + \varepsilon)}{1-\beta}$$

Approximate DTR: Relative Merits

- Relative merits of ADTR
  - fewer regions implies faster computation
  - can provide leverage for **optimal** computation
    - e.g., start with aggressive pruning, then relax (exploit contraction)
  - allows fine-grained control of time vs. solution quality with dynamic *(a posteriori)* error bounds
  - technical challenges: variable ordering, convergence, fixed vs. adaptive tolerance, etc.
- Some drawbacks
  - (still) produces piecewise constant VF
  - doesn't exploit additive structure of VF at all
- Many other ways of exploiting structure, DBNs, etc.
  - function approximation (especially linear approximations)
  - decompositions (sub-problem structure, etc.)
  - …

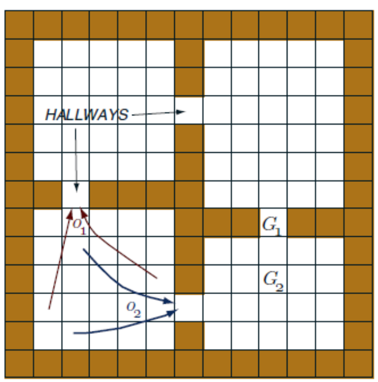
State-based Decomposition

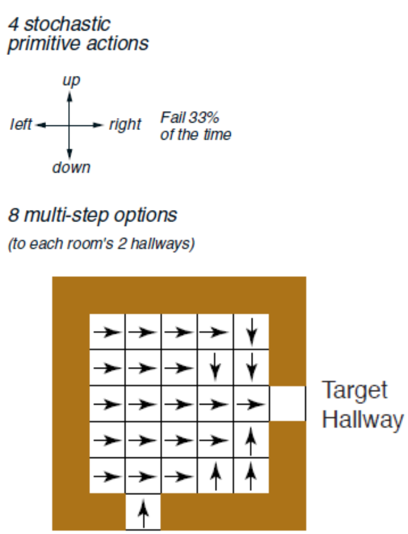
- MDP may have weakly or non-interacting subcomponents
  - E.g., policy for running several assembly lines, robots, …
    - Actions taken for one may have no (or little) impact on others
    - Can solve for policies independently if no interaction
    - If some interaction, use “independent” policies and values to guide the coordination (e.g., interaction limited to occasional assignment of resources to each assembly line)



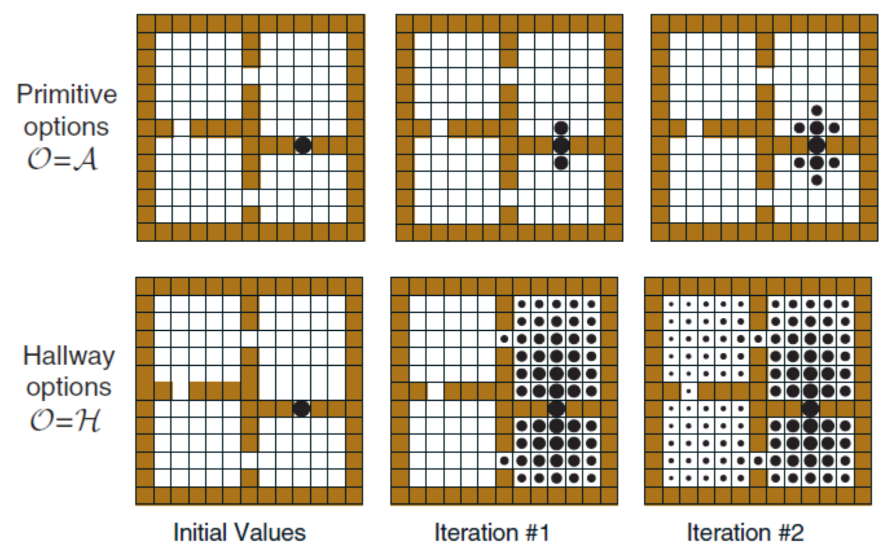
Temporal Abstraction

- Solve local MDPs over specific “regions” of state space
  - Macro-actions, “local policies,” temporally-extended actions
  - Use the local policies as actions in an smaller abstract MDP
  - Fast value propagation, small abstract MDP, prior knowledge, …
  - Issues: which macros, computing macro-models (state space), transferability/reuse for new domains/objectives, …





From Sutton, Precup, Singh, AIJ-99



Linear Value Function Approximation

- Set of *basis functions*: $B = \{b_1, b_2, \ldots, b_k\}$
  - Each $b_i: S \rightarrow \mathbb{R}$ assigns value to states, compact (e.g., depends only on a few state features)
- Approx. *V* with linear combination: $\tilde{V}(s) = \sum_i w_i b_i(s)$
  - Compact representation: weight vector $\boldsymbol{w}$ and small basis f'ns
  - Limits VF to fall within space spanned by *B*
- Approx. value iteration: sequence $\boldsymbol{w}^{(k)}$ of *k*-stage-to-go VFs
  - Run Bellman back up on $\boldsymbol{w}^{(k)}$to produce $\boldsymbol{w}^{(k+1)} = L(\boldsymbol{w}^{(k)})$
  - Trick: $\boldsymbol{w}^{(k+1)}$ usually falls out of *B*-space, but still compact; project back into *B*-space before moving to next iteration
  - Issues: good set of basis functions? Keeping computation tractable (Bellman backup, projection), e.g., exploiting DBNs? etc.
- Policy iteration, etc. can also be used