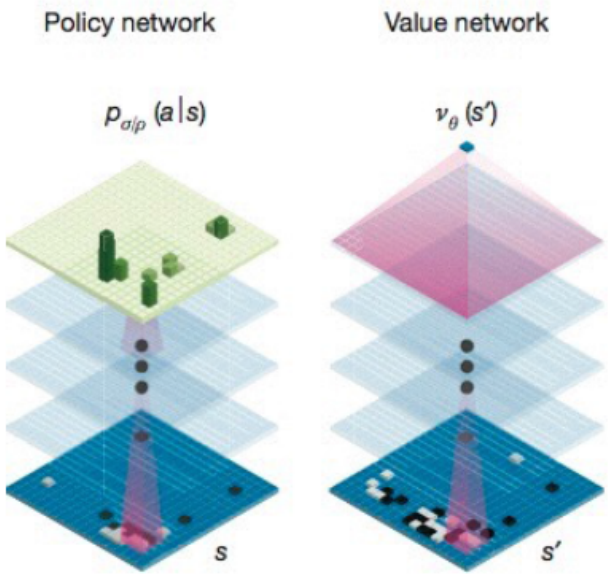
Hierarchical Reinforcement Learning

With thanks to Rich Sutton, Satinder Singh, Gheorghe Comanici, Anna Harutyunyan, Andre Barreto, David Silver, Pierre-Luc Bacon, Jean Harb, Shibl Mourad, Khimya Khetarpal, Zafarali Ahmed, David Abel, Sasha Vezhnevets, Shaobo Hou, Philippe Hamel, Eser Aygun, Diana Borsa, Justin Novosad, Will Dabney, Nicholas Heess, Remi Munos

COMP579 Lecture 18, 2024

Knowledge in AlphaGo



- *Policy*: what to do (probability of action given current "state") - ie *procedural knowledge*
- *Value function*: estimation of expected long-term return - ie *predictive knowledge*

From Reinforcement Learning to AI

- Growing knowledge and abilities in an environment
- Learning efficiently from one stream of data
- Reasoning at multiple levels of abstraction
- Adapting quickly to new situations

Building Knowledge with Reinforcement Learning

- Focusing on two types of knowledge:
  - *Procedural knowledge*: skills, goal-driven behavior
  - *Predictive, empirical knowledge*: predicting effects of actions

- Knowledge must be:
  - *Expressive*: able to represent many things, including abstractions (objects, places, high-level strategies...)
  - *Learnable*: from data, ideally without supervision (for scalability)
  - *Composable*: suitable for fast planning by assembling existing pieces

Abstraction and generalization

- An *abstract representation* ignores low-level details of the problem, or modifies the problem representation altogether

  Eg. addresses vs exact coordinates

  Eg. representing graphs through vertex position and edges vs by adjacency matrix
- *Generalization* is the ability to take knowledge acquired in some circumstances and applying it in different circumstances

  Eg. Being good at some games helps us learn other games faster
- These two concepts are related but not identical: an abstract representation may helps us to generalize
- Generalization is often achieved in AI/ML by using function approximation (eg deep nets)
- In RL, we have an extra important dimension: *time/action* - can we build abstraction/generalization here too?

What is temporal abstraction?

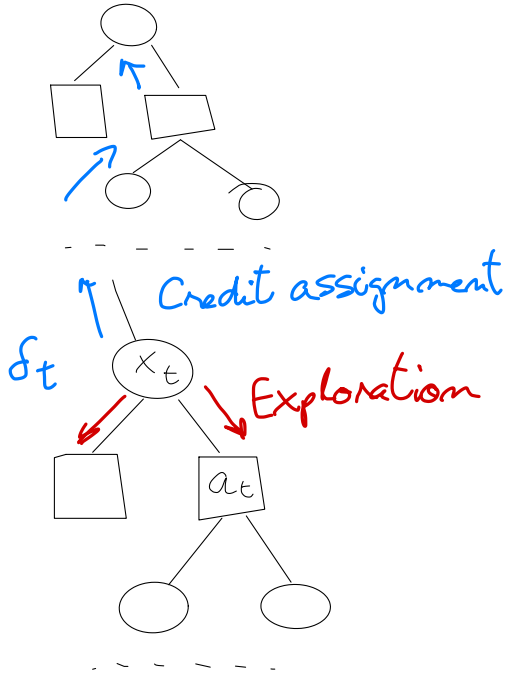
- Consider an activity such as cooking dinner

  - High-level steps: choose a recipe, make a grocery list, get groceries, cook,...
  - Medium-level steps: get a pot, put ingredients in the pot, stir until smooth, check the recipe ...
  - Low-level steps: wrist and arm movement while driving the car, stirring, ...
- All have to be seamlessly integrated!

Temporal abstraction in AI

- A cornerstone of AI planning since the 1970's:
  - Fikes et al. (1972), Newell (1972, Kuipers (1979), Korf (1985), Laird (1986), Iba (1989), Drescher (1991) etc.
- It has been shown to :
  - Generate shorter plans
  - Reduce the complexity of choosing actions
  - Provide robustness against model misspecification
  - Allows taking shortcuts in the environment
- In robotics and hybrid systems, the use of controllers provides similar benefits, and also improves interpretability and allows specifying prior knowledge

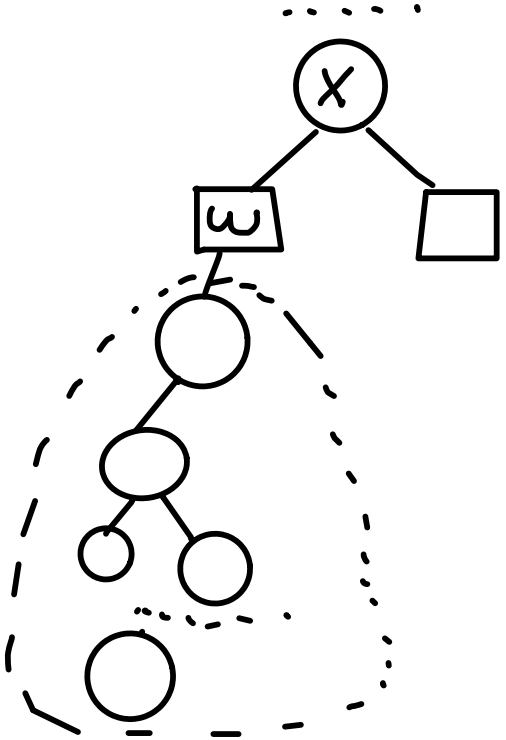
Recall: RL cartoon



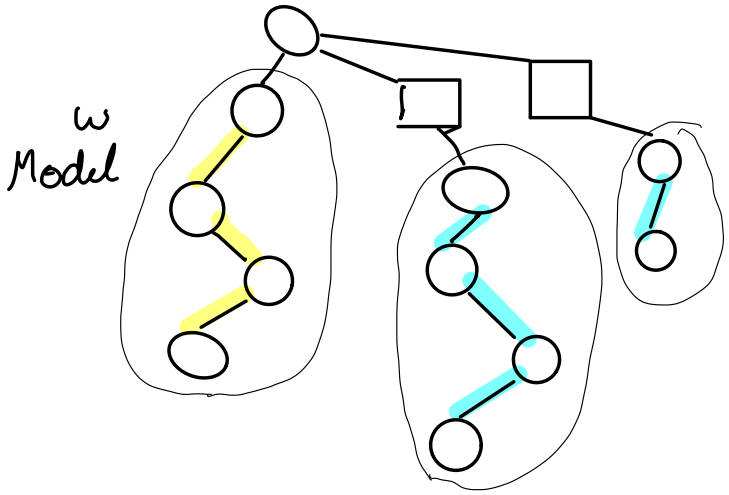
Goals of temporal abstraction:

- Reduce tree width
- Reduce tree depth
- Generalize between different branches of the tree

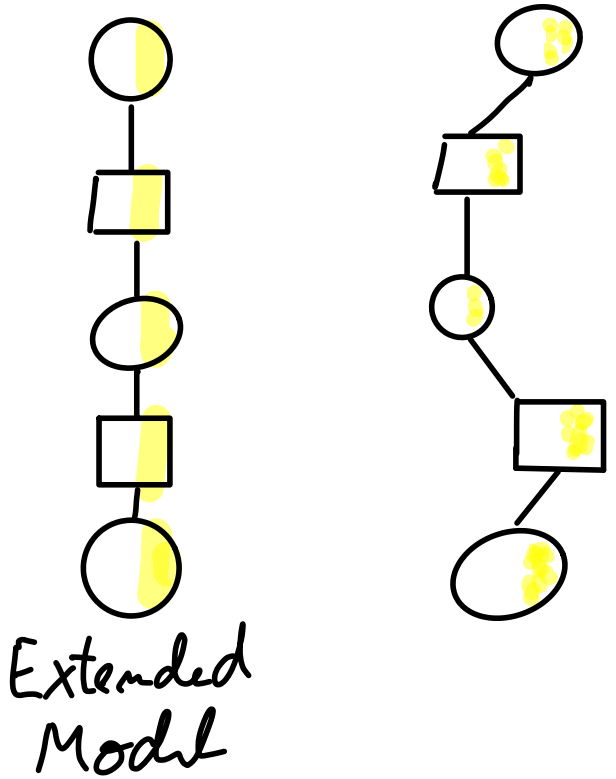
Options intuition: "package" a subtree



Options intuition: Faster updates



Options intuition: Enable cheaper/faster planning



Procedural, Temporally Abstract Knowledge: Options

- An *option* ω consists of 3 components
  - An *initiation set* $I_\omega \subseteq \mathcal{S}$ (aka precondition)
  - A *policy* $\pi_\omega : \mathcal{S} \times \mathcal{A} \to [0, 1]$
    $\pi_\omega(a|s)$ is the probability of taking a in s when following option ω
  - A termination condition $\beta_\omega : \mathcal{S} \to [0, 1]$:
    $\beta_\omega(s)$ is the probability of terminating the option ω upon entering s
- Eg., robot navigation: if there is no obstacle in front (I_ω), go forward (π_ω) until you get too close to another object (β_ω)
- Inspired from macro-actions / behaviors in robotics / hybrid planning and control

Cf. Sutton, Precup & Singh, 1999; Precup, 2000

Options as Behavioral Programs

- *Call-and-return execution*
  - When called, option ω is pushed onto the execution stack
  - During the option execution, the program looks at certain *variables (aka state)* and executes an *instruction (aka action)* until a termination condition is reached
  - The option can keep track of additional *local variables*, eg counting number of steps, saturation in certain features (e.g. Comanici, 2010)
  - *Options can invoke other options*
- *Interruption*
  - At each step, one can check if a better alternative has become available
  - If so, the option currently executing is *interrupted* (special form of concurrency)

Option models

- *Option model* has two parts:
  1. *Expected reward* $r_\omega(s)$: the expected return during ω's execution from state s
  2. *Transition model* $P_\omega(s'|s)$: specifies *where* the agent will end up after the option/program execution and *when* termination will happen
- Models are *predictions* about the future, conditioned on the option being executed
- Programming languages: preconditions (initiation set) and postconditions
- Models of options represent *(probabilistic) post-conditions*
- "Jumpy" planning is better for temporal credit assignment, accurate value estimation

What type of planning?

- *Models that are compositional can be used to plan through value iteration*
- *Sequencing*

$$\begin{aligned} \mathbf{r}_{\omega_1\omega_2} &= \mathbf{r}_{\omega_1} + P_{\omega_1}\mathbf{r}_{o_2} \\ P_{\omega_1\omega_2} &= P_{\omega_1}P_{\omega_2} \end{aligned}$$

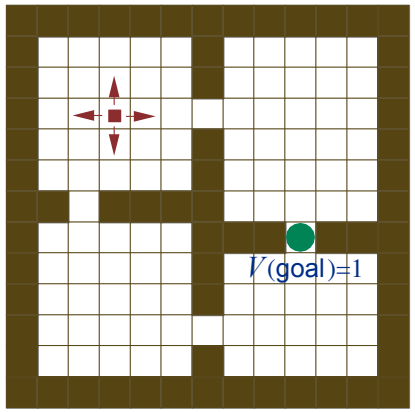
  Cf. Sutton et al, 1999, Sorg & Singh, 2011
- *Stochastic choice*: can take expectations of reward and transition models
- These are sufficient conditions to allow Bellman equations to hold
- Silver & Ciosek (2012): re-write model in one matrix, compose models to construct programs
- Model-predictive control (receding horizon planning) is also possible

Option Models Provide Semantics

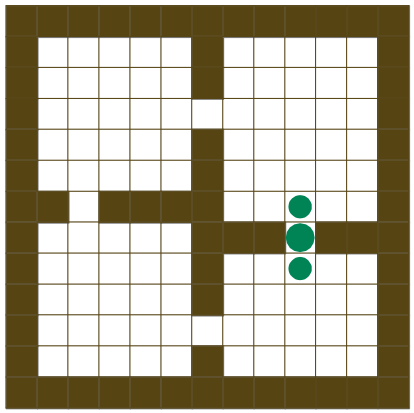
- Models of actions consist of immediate reward and transition probability to next state
- Models of options consist of reward until termination and (discounted) transition to termination state
- Models are *predictions about the future*

Illustration: Navigation

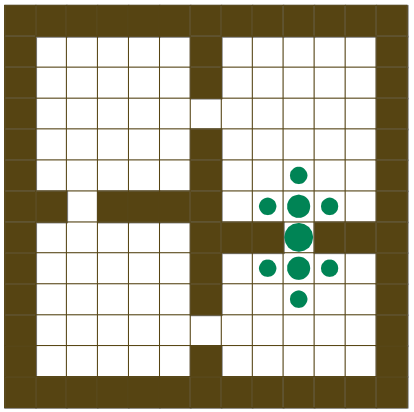
with cell-to-cell primitive actions



Iteration #0

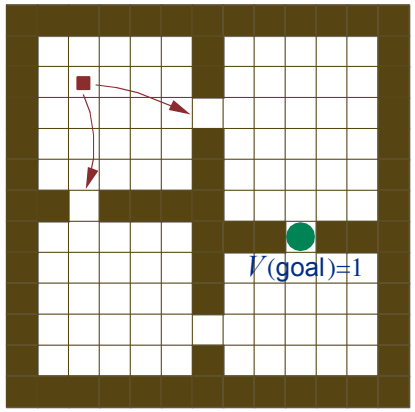


Iteration #1

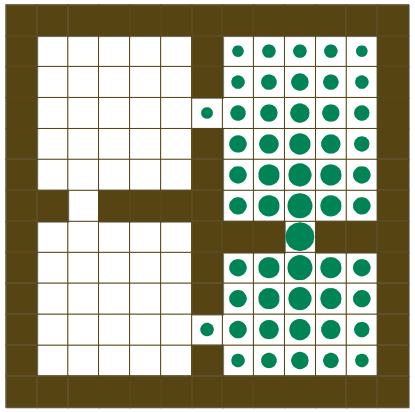


Iteration #2

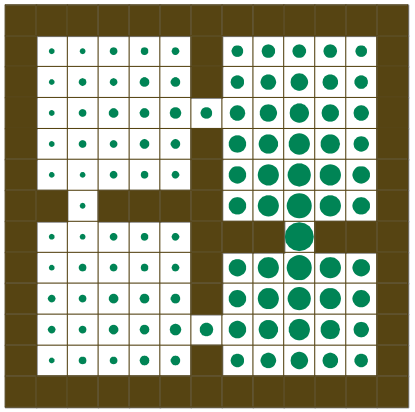
with room-to-room options



Iteration #0

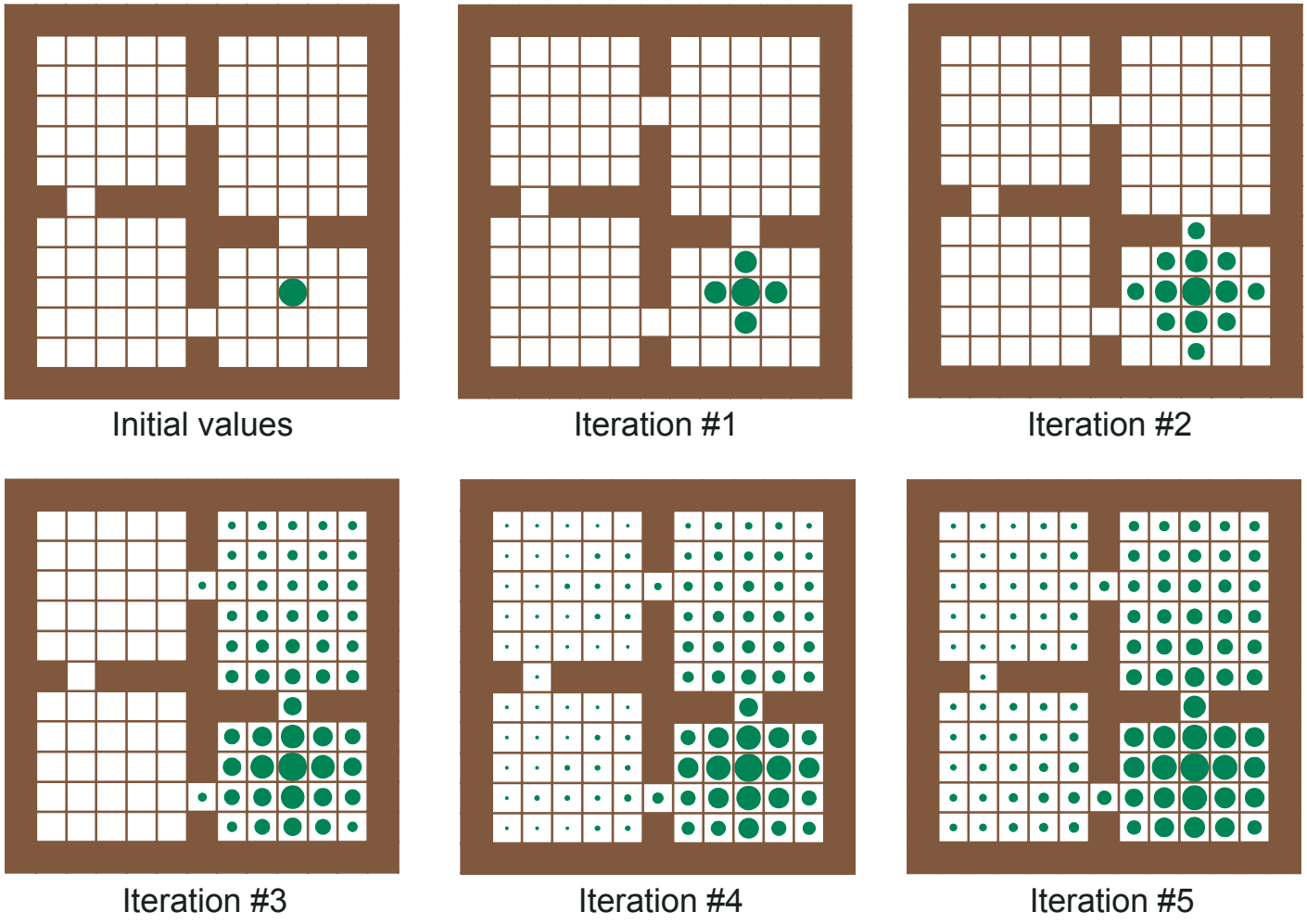


Iteration #1

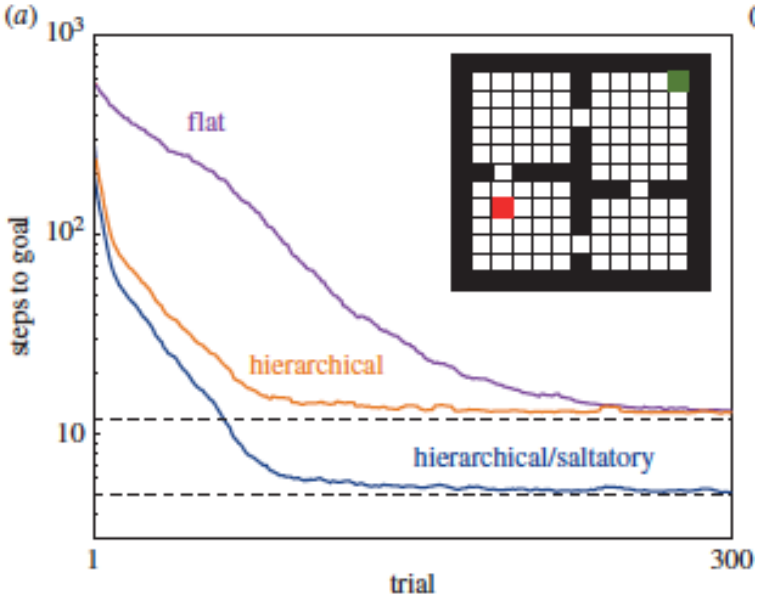


Iteration #2

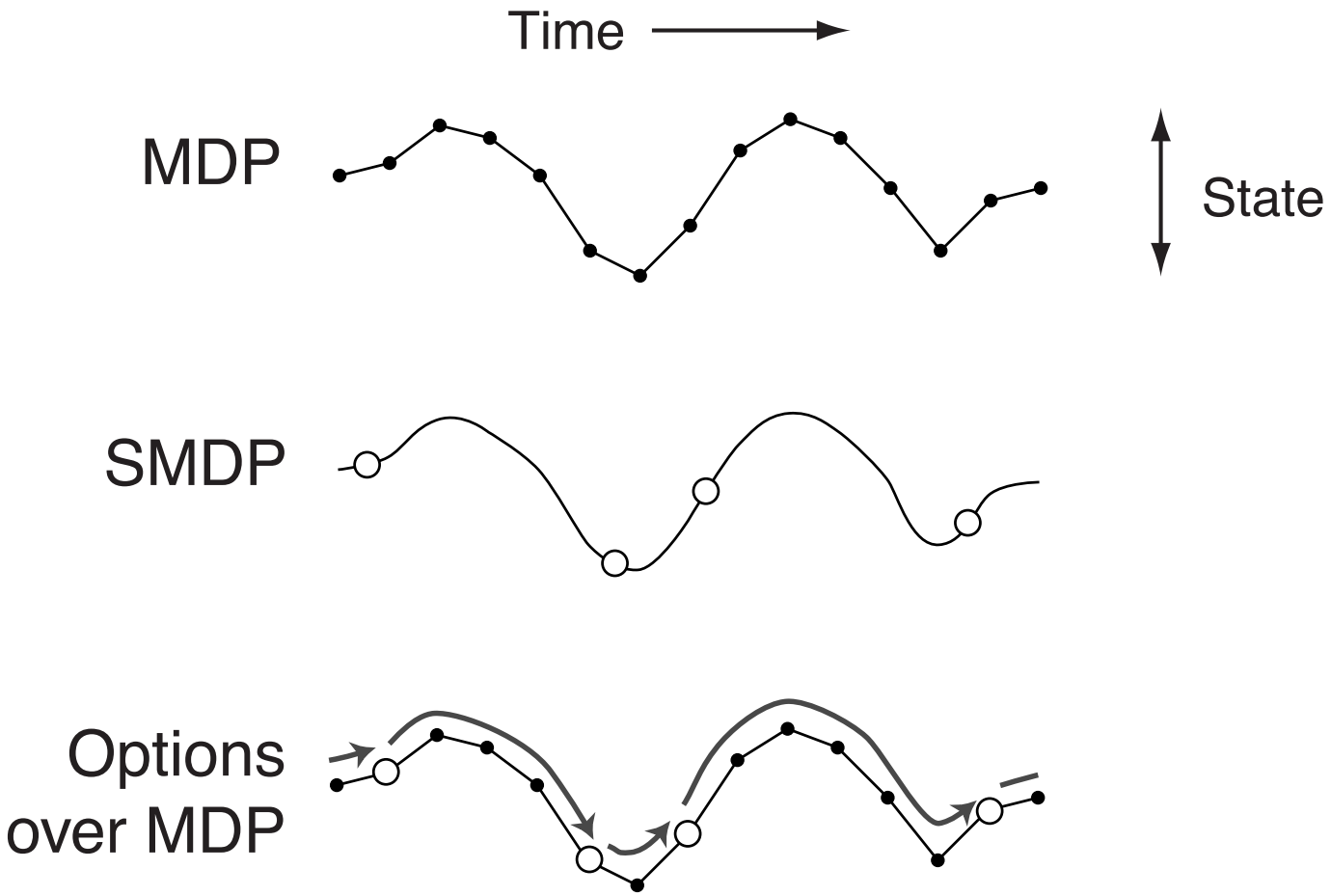
Illustration: Options and Primitives



Benefits of options (cf Botvinick & Weinstein, 2014)



Decision-Making with Options



Learning and planning algorithms are the same at all levels of abstraction!

Option-value function

- The option-value function of a policy over options π_Ω is defined as:

$$q_{\pi_\Omega}(s,\omega) = \mathbf{E}_{\pi_\Omega}\left[R_{t+1} + \gamma\beta_\omega(S_{t+1})q_{\pi_\Omega}(S_{t+1},\omega_{t+1})) \right.$$
$$\left. + \gamma((1-\beta_\omega(S_{t+1}))q_{\pi_\Omega}(S_{t+1},\omega)|S_t = s\right]$$

- One can use eg Q-learning, actor-critic,... to learn this!
- Note that if we learn/plan in an SMDP, the *contraction factor will be lower than* γ
- So fixing a set of options may allow solving the problem faster, but maybe in a slightly sub-optimal way
- Intuitively, models are more self-contained than option-value functions

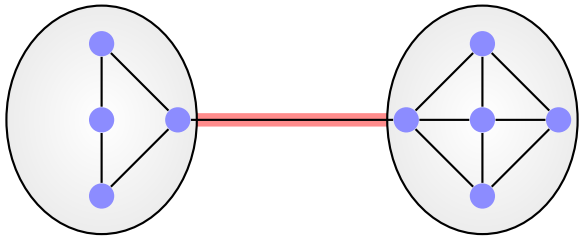
Advantages

- Easy to learn using temporal-difference-style methods, from a single stream of experience
- Planning with option models is done just like planning with primitives - *no explicit hierarchy*
- Result of planning with a set of options Ω is an option-value function, e.g. V_Ω, Q_Ω
- *But we can also use the underlying MDP structure to help in learning the options*

How Should Options Be Created?

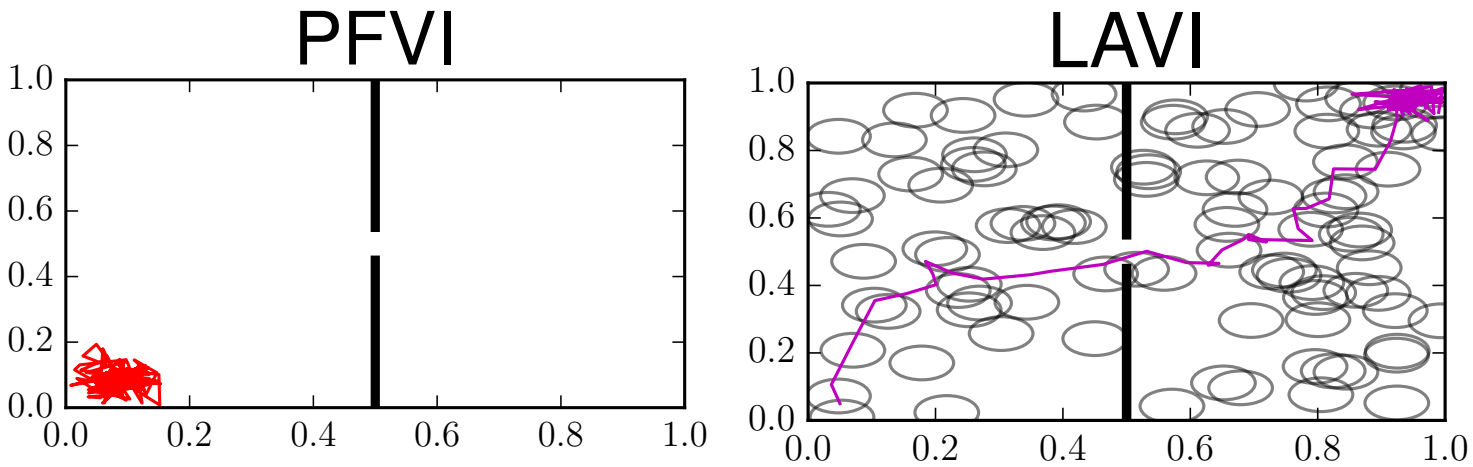
- Options can be given by a system designer (eg robotics)
- If subgoals / secondary reward structure is given, the option policy can be obtained, by solving a smaller planning or learning problem (cf. Precup, 2000)
  - Eg. acquiring certain objects in a game
  - Eg. Intrinsic motivation
- *What is a good set of subgoals / options?*
- This is a *representation discovery* problem
- Studied a lot over the last 15 years
- Bottleneck states and change point detection currently the most successful methods

Bottleneck States



- Perhaps the most explored idea in options construction
- A bottleneck allows "circulating" between many different states
- Lots of different approaches!
  - Frequency of states (McGovern et al, 2001, Stolle & Precup, 2002)
  - Graph partitioning / state graph analysis (Simsek et al, 2004, Menache et al, 2004, Bacon & Precup, 2013)
  - Information-theoretic ideas (Peters et al., 2010)
- People seem quite good at generating these (cf. Botvinick, 2001, Solway et al, 2014)
- *Main drawback: expensive both in terms of sample size and computation*

Random Subgoals Also Help



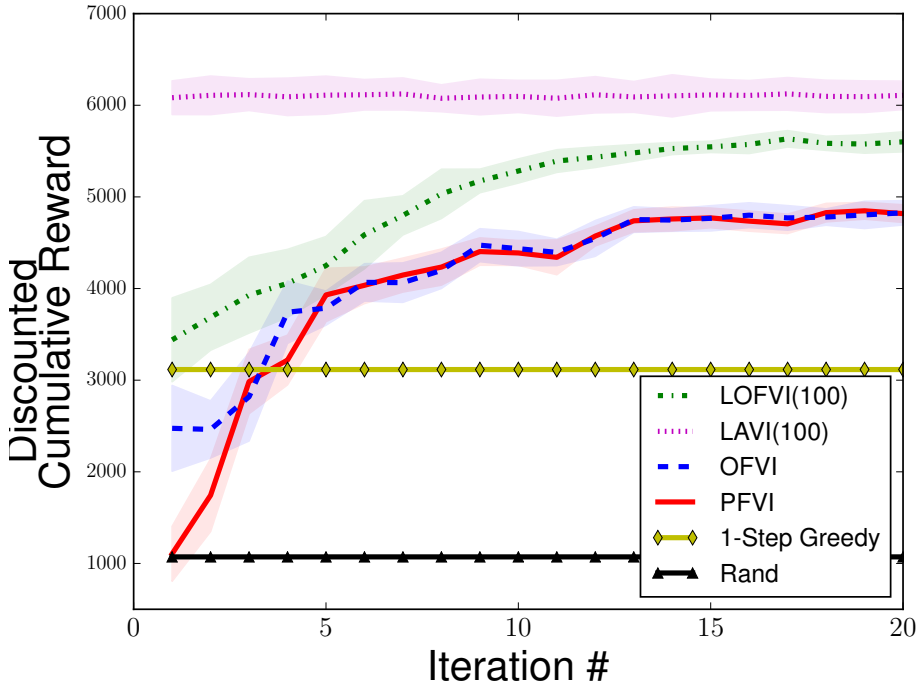
Cf. Mann, Mannor & Precup, 2015

Inventory management application

- Manage a warehouse that can stock 8 different commodities
- At most 500 items can be stored at any given time
- Demand is stochastic and depends on time of year
- Negative rewards are given for unfulfilled orders and for the cost of ordered items
- Hand-crafted options: order nothing until some threshold is crossed
- Primitive actions: specify amount of order for each item

Inventory management results

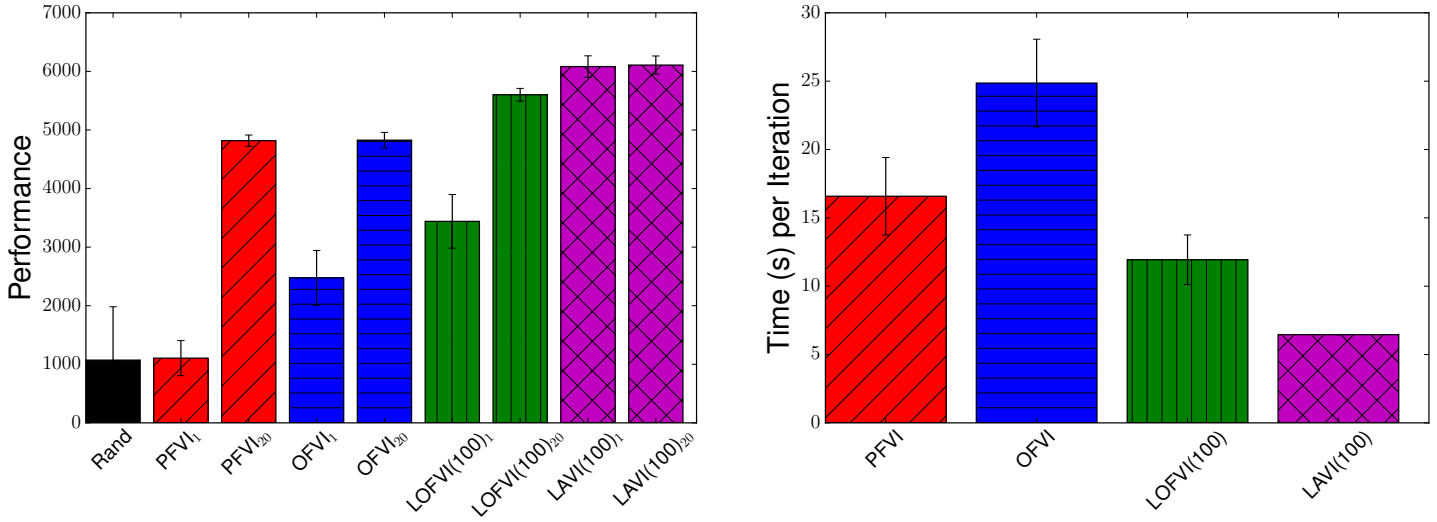
- Comparing a random policy and a 1-step greedy choice with using just primitives (PFVI) using primitives and hand-crafted options (OFVI), using "landmarks" (LOFVI) and using landmarks and only computing values for landmarks states (LAVI)



- *Randomly generated landmarks perform much better*

Performance and time evaluation

- Performance of initial and final policy (left) and running time (right) averaged offer 20 independent runs

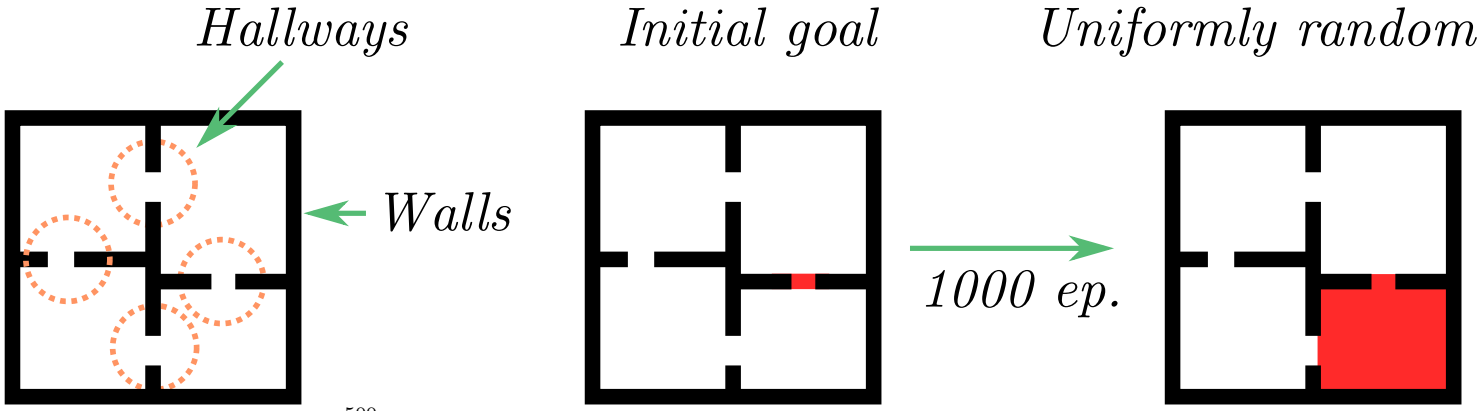


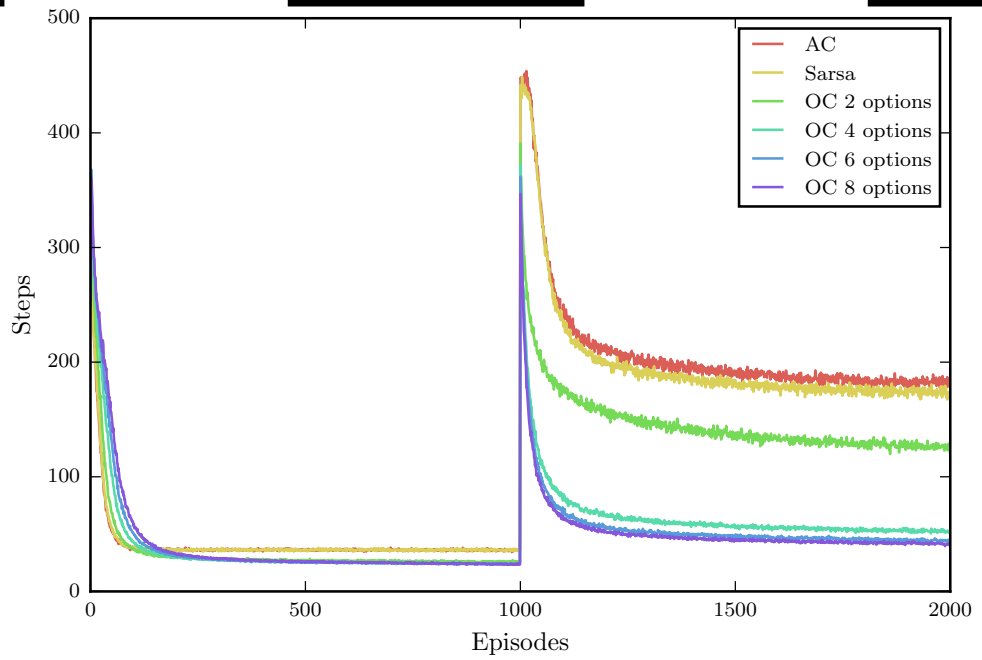
- Computing values only at landmark states yields a good policy almost immediately
- Handcrafted options are better than primitives in the beginning but slightly worse in the long run but *randomly generated landmarks are much better*

Option-Critic: Learn Options that Optimize Return

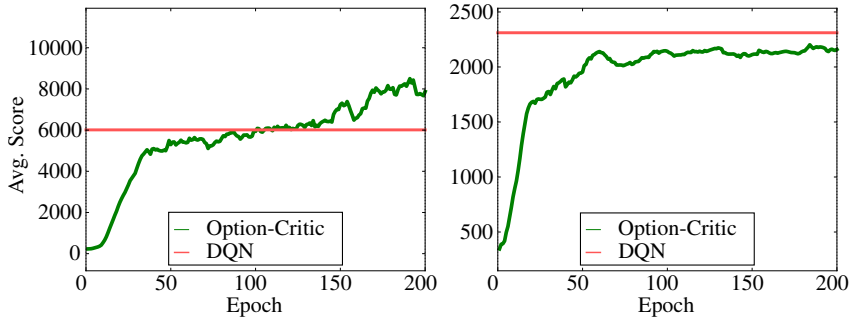
- Explicitly state an *optimization objective* and then solve it to find a set of options
- Handle both *discrete and continuous* set of state and actions
- Learning options should be *continual* (avoid combinatorially-flavored computations)
- Options should provide *improvement within one task* (or at least not cause slow-down...)

Results: Transfer in Rooms Domain



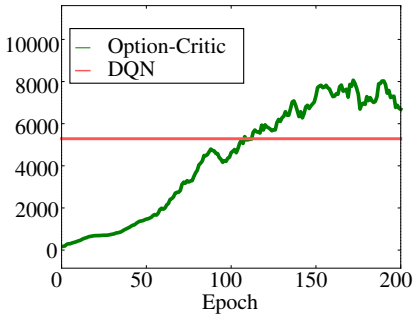


Quantitative and qualitative results in Atari games

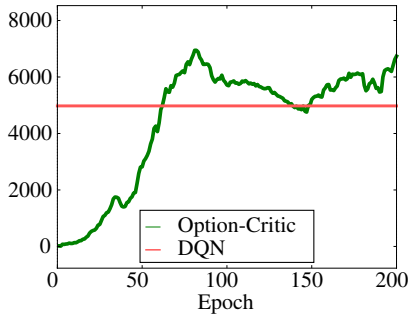


(a) Asterix

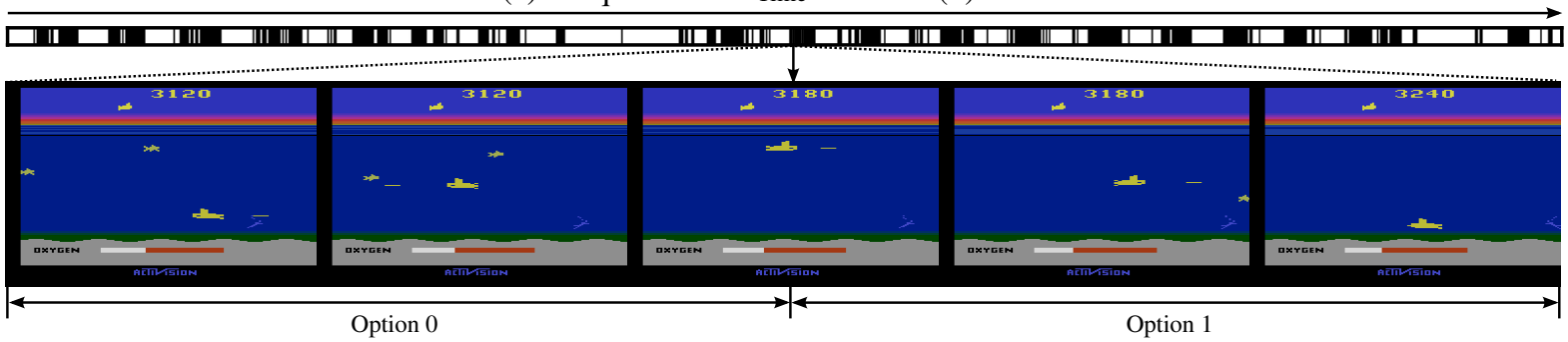
(b) Ms. Pacman



(c) Seaquest



(d) Zaxxon



Preserving Procedural Knowledge over Time

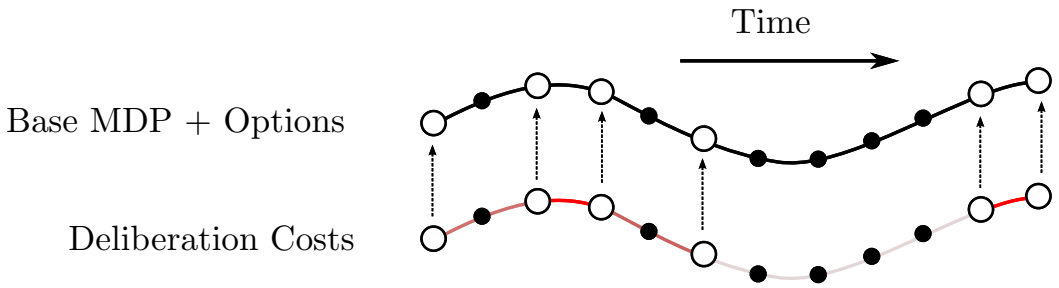
- Successful simultaneous learning of terminations and option policies
- But, as expected, *options shrink over time* unless additional regularization is imposed

  Cf. time-regularized options, Mann et al, (2014)
- Intuitively, using longer options increase the speed of learning and planning (but may lead to a worse result in call-and-return execution)
- Diverse options are useful for exploration in continual learning setting

Bounded Rationality as Regularization

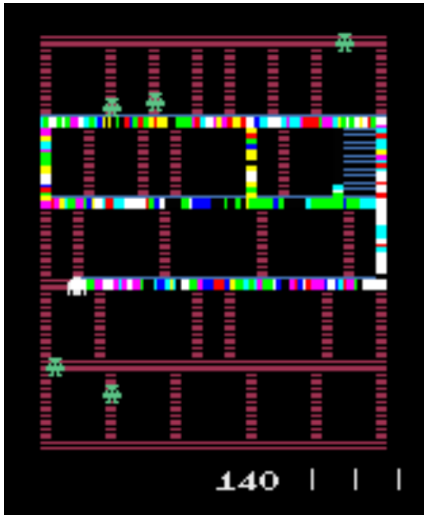
- Problem: optimizing return leads to option collapse (primitive actions are sufficient for optimal behaviour)
- Bounded rationality: reasoning about action choices is expensive (energy consumption and missed-opportunity cost)

  Eg Russell, 1995, Lieder & Griffiths, 2018
- Idea: switching options incurs an additional cost

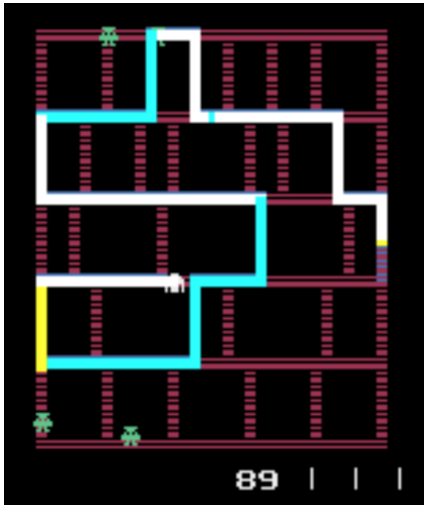


- Can be shown equivalent to requiring that *advantage exceeds a threshold* before switching

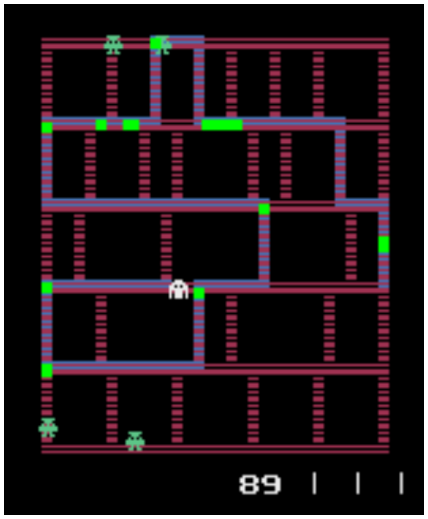
Illustration: Amidar



(a) Without a deliberation cost, options terminate instantly and are used in any scenario without specialization.



(b) Options are used for extended periods and in specific scenarios through a trajectory, when using a deliberation cost.



(c) Termination is sparse when using the deliberation cost. The agent terminates options at intersections requiring high level decisions.

- Deliberation costs prevent options from becoming too short
- Terminations are intuitive

Should All Option Components Optimize the Same Thing?

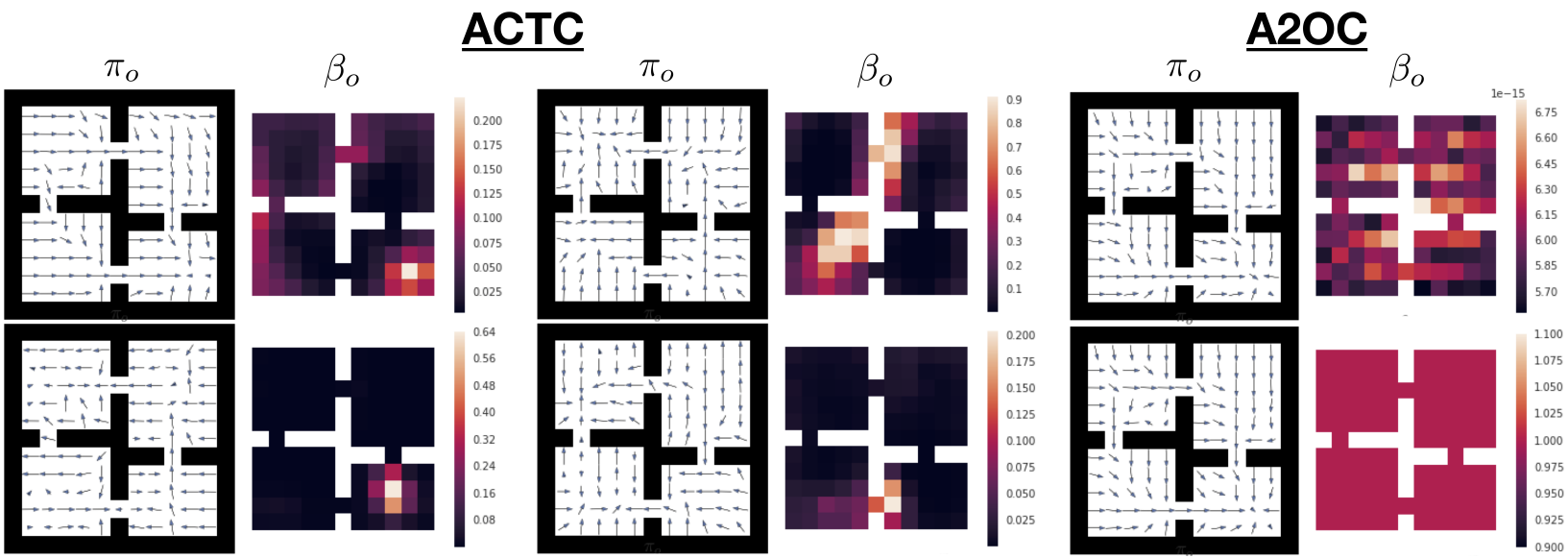
- Deliberation cost can be viewed as associated specifically with termination
- Rewards could be optimized mainly by the internal policy of the option
- Can we generalize this idea to other optimization criteria?

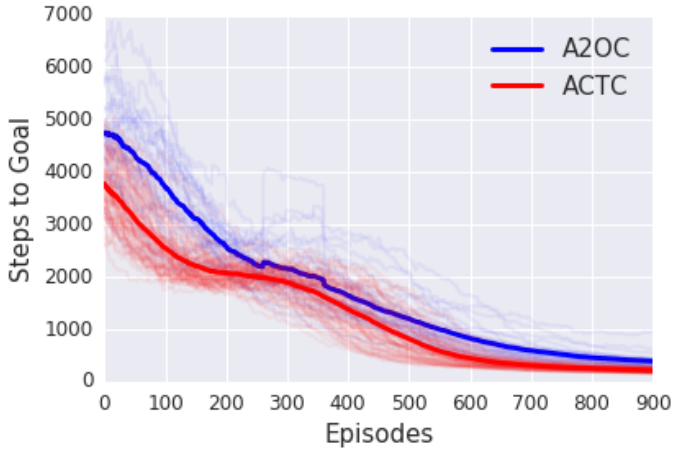
Termination-Critic

- *Optimize the termination condition independently of the policy inside the option*
- Option termination should focus on *predictability* ie finding "funnelling states"
- Interesting side effect: if each option ended at a funelling state, expectation and distribution model would be almost identical and the option would be almost deterministic
- Implementation: minimize the entropy of the option transition model P_ω

cf. Harutyunyan et al, AISTATS'2019

Illustration: Rooms environment





Predictive knowledge: Value Function

- Given a policy π, a discount factor γ and a reward function r, the value function of the policy is given by:

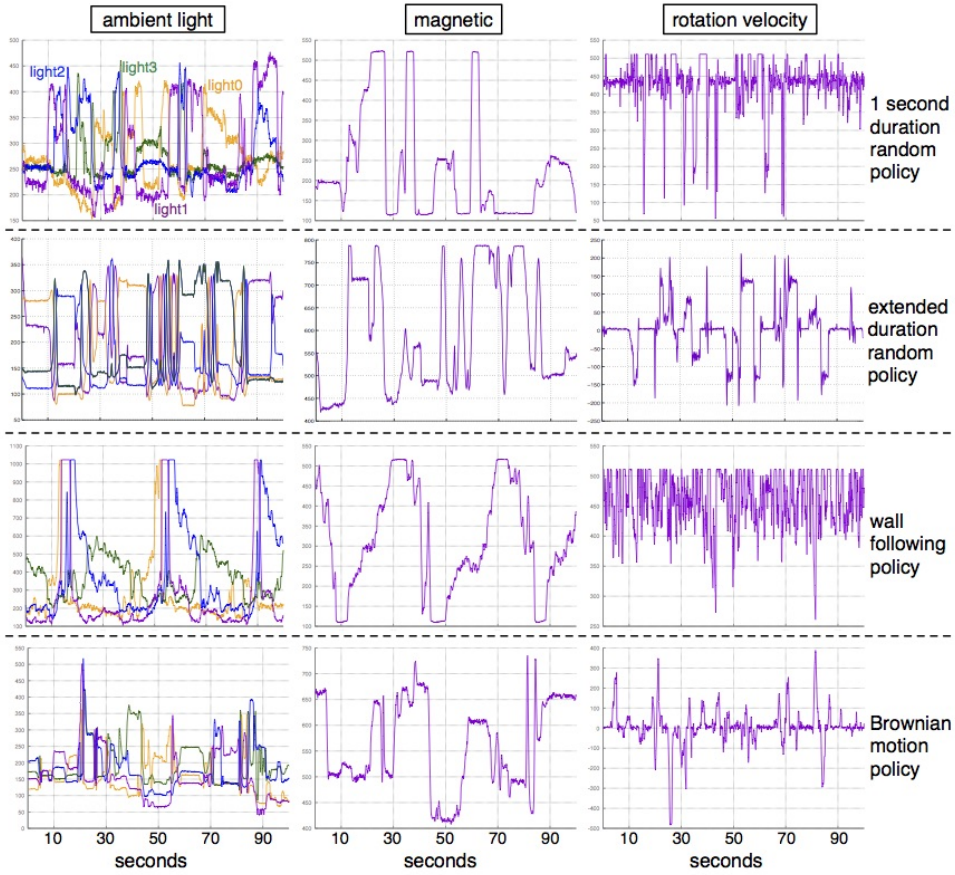
$$\begin{aligned} v_\pi(s) &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k)\gamma^{k-t} | S_t = s, A_{t:\infty} \sim \pi] \\ &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k) \prod_{i=t+1}^{k} \gamma | S_t = s, A_{t:\infty} \sim \pi] \end{aligned}$$

- r is the *signal of interest* for the prediction
- γ defines the time scale over which we want to make the prediction (in a very crude way)
- Optimal value function: given a discount factor γ and a reward function r, compute v_{π^*} and π^*, the optimal policy wrt γ, r
-

Focusing on value function

- Definition allows us to leverage great tools: *bootstrapping* (as in dynamic programming) and *sampling*
- We have good ideas for how to learn value functions from data using temporal-difference methods, off-policy learning...
- Usual objection: this is restricted to one reward function and usually a fixed time scale (discount)
- An agent may need to make predictions about many different things and at many different time scales

There are many things to learn! (Adam White's thesis)



Sensory stream of Critterbot robot about different sensors for different policies
Can we learn about all these signals in parallel from one stream of data?

Temporally Abstract Predictions: General Value Functions (GVFs)

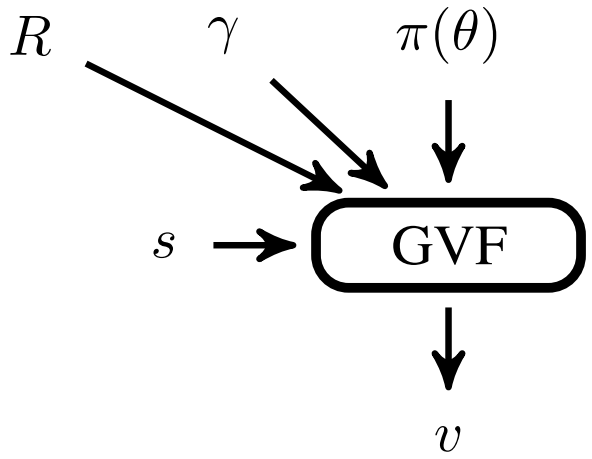
- Given a cumulant function c, state-dependent continuation function γ and policy π, the General Value Function $v_{\pi,\gamma,c}$ is defined as:

$$v_{\pi,c,\gamma}(s) = \mathbf{E}\left[\sum_{k=t}^{\infty} c(S_k, A_k, S_{k+1}) \prod_{i=t+1}^{k} \gamma(S_i) | S_t = s, A_{t:\infty} \sim \pi\right]$$

- *Cumulant* c can output a vector (even a matrix)
- *Continuation function* γ maps states to [0,1] (further generalizations are possible)
- Cf. Horde architecture (Sutton et al, 2011); Adam White's thesis; inspiration from Pandemonium architecture
- Special case: policy is optimal wrt c, γ, $v^*_{c,\gamma}$ - Universal Value Function approximation (UVFA) (Schaul et al, 2015)

- No single task is required, just a multitude of cumulants and time scales!

GVFs as building blocks of knowledge



- Note that one can take the output of a GVF and make it an input to another GVF
- Or, the output of a GVF could become part of the "state" for another GVF

Successor states and successor features are GVFs

- *Successor features* (Barreto et al, 2017, 2018) are a natural extension of successor states (Dayan, 1992)
- Successor states give the expected occupancy of future states
- If states are defined by a feature vector $\phi(s)$, successor features give the expected, discounted sum of future feature vectors from a state.
- In GVF terms, the *cumulant is* $c = \phi$, and there is a fixed policy and discount
- Interesting property highlighted in Barreto et al:

$$v_{\pi,\mathbf{w}^T c,\gamma}(s) = \mathbf{w}^T v_{\pi,c,\gamma}(s)$$

which leads to one-shot computation of new GVFs

Option models are GVFs

- The reward model for an option ω is defined as:

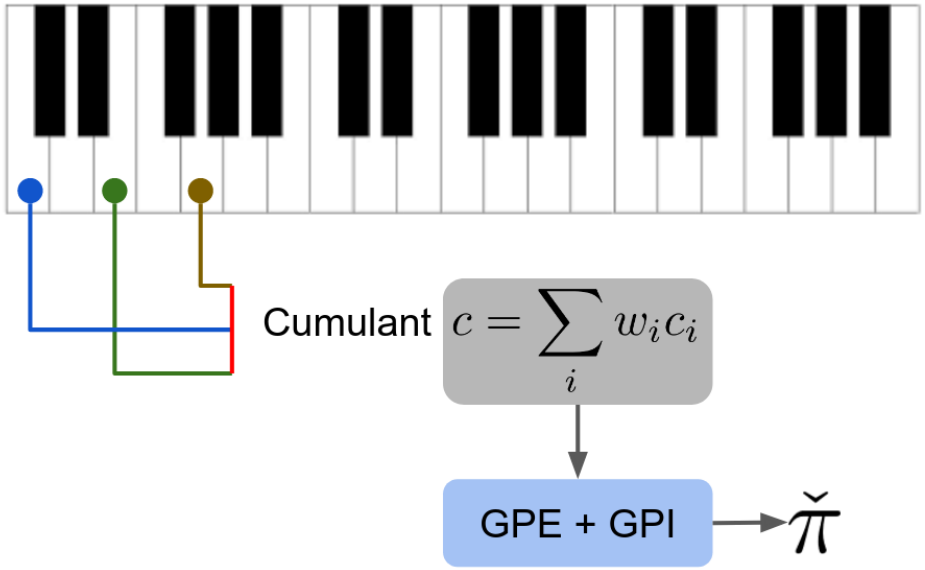
$$r_\omega(s) = \mathbb{E}_\omega[r(S_t, A_t) + \gamma(1 - \beta_\omega(S_{t+1}))r_\omega(S_{t+1})|S_t = s]$$

- This means the option reward model is a GVF:
  - policy is π_ω
  - *cumulant* is the environment reward r
  - *continuation function* is $\gamma(1 - \beta_\omega)$
- Option transition model can be similarly written as a GVF

Many other approaches that can be expressed as GVFs

- Option-value functions (Precup, 2000; Sutton, Precup & Singh, 1999)
- Feudal networks (Dayan, 1994; Vezhnevets et al, 2017)
- Value transport (Hung et al, 2018)
- Auxilliary tasks (Jaderberg et al, 2016)
- *Are GVFs just an interesting insight or can they be useful?*

GVFs for synthesizing new behaviors



Option-keyboard - Barreto et al, 2019, based on ideas of Rich Sutton

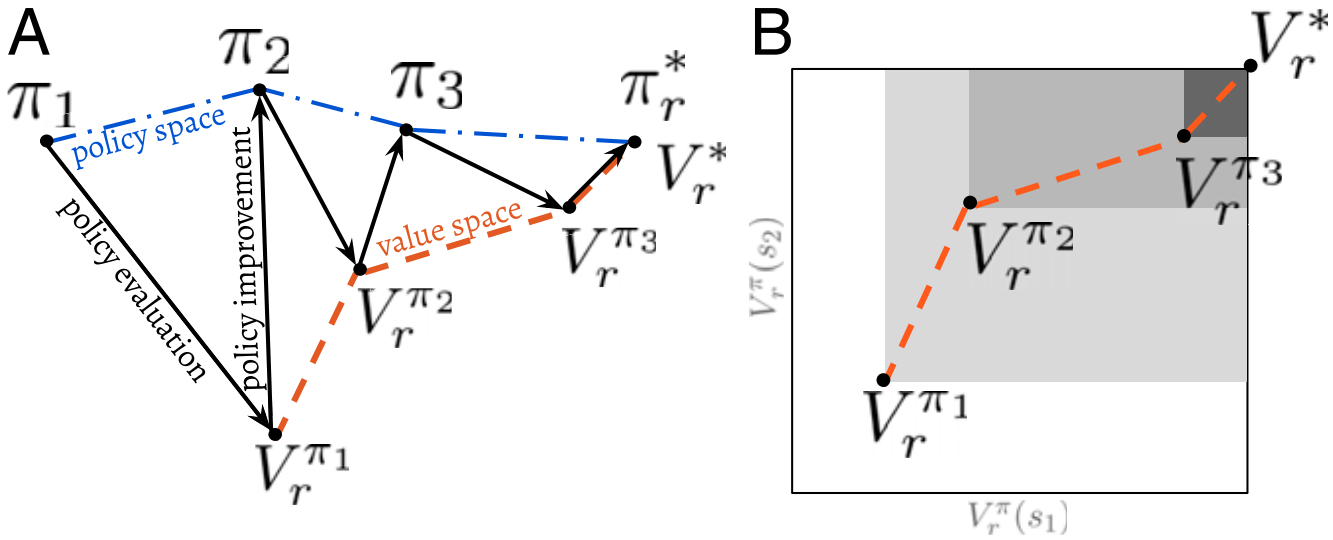
Policy Evaluation and Policy Improvement

- Consider a Markov Decision Process $\langle \mathcal{S}, \mathcal{A}, P, r \rangle$ and a policy $\pi : \mathcal{S} \rightarrow Dist(\mathcal{A})$
- Classic dynamic programming relies on two basic operations:
  - *Policy evaluation*: given policy π, compute the value function V_r^π and/or Q_r^π
  - *Policy improvement*: given value function Q_r^π, compute an improved policy: $\pi'(s) = \arg\max_{a' \in \mathcal{A}} Q_r^\pi(s, a')$
- Policy improvement guarantee:

$$Q_r^{\pi'}(s, a) \geq Q_r^\pi(s, a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

- Dynamic programming: interleave these steps (executed exactly)
- Reinforcement learning: carry out these steps approximately

Visualizing Policy Evaluation and Policy Improvement



- Generalize this process to *multiple reward functions (ie tasks)* $r \in \mathcal{R}$ and *multiple policies* $\pi \in \Pi$

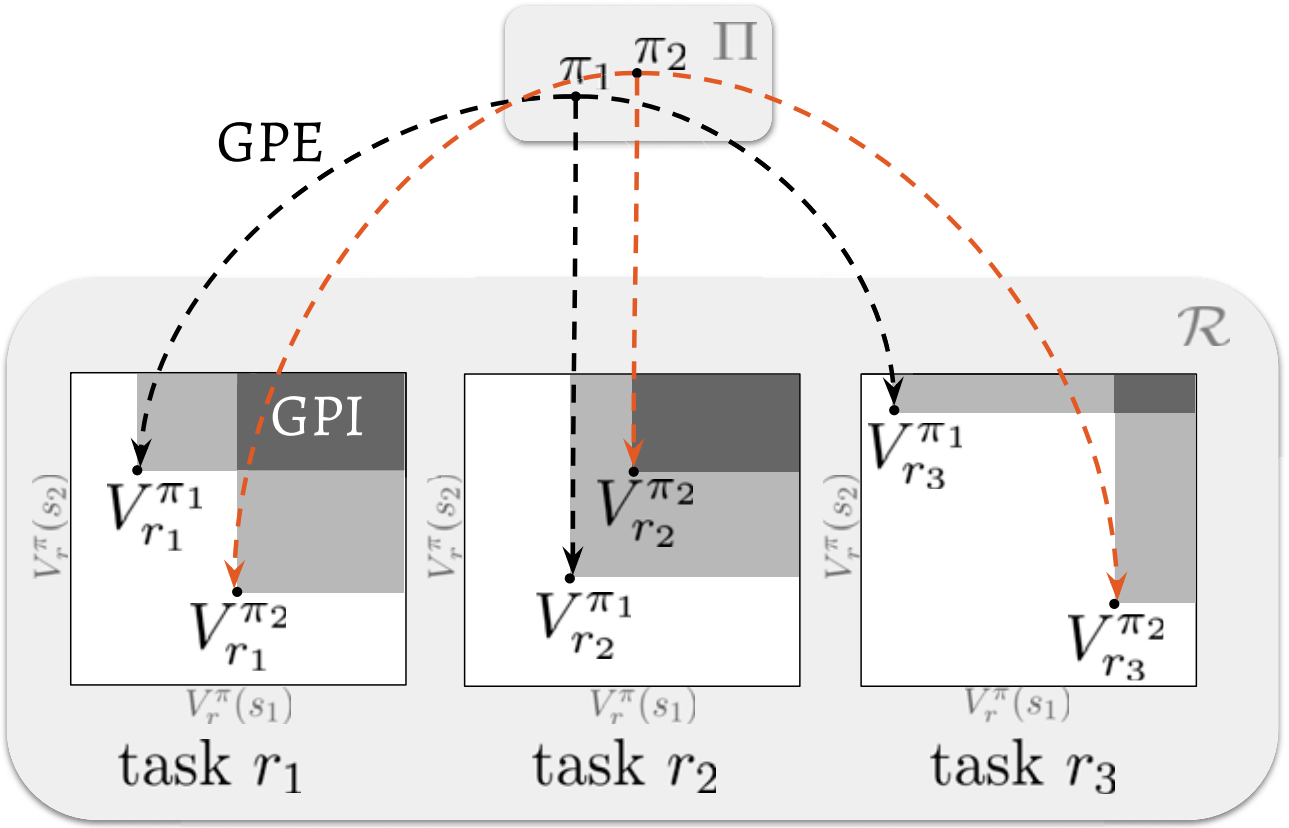
Generalized Policy Updates

- *Generalized policy evaluation (GPE)*: compute the value of a policy π on a set of reward functions $\mathcal{R}$
- *Generalized policy improvement (GPI)*: given a set of policies Π and a reward function r, compute a new policy such that:

$$Q_r^{\pi'}(s,a) \geq \sup_{\pi \in \Pi} Q_r^{\pi}(s,a),\ \forall s \in \mathcal{S} \forall a \in \mathcal{A}$$

- If we have only one r and one π, we recover usual policy evaluation and policy improvement

Visualizing Generalized Policy Updates



Fast Generalized Policy Evaluation

- If we had a nice map from r to Q^π_r, GPE could be efficient
- Consider the class of reward functions that are linear in some feature space $\phi(s,a)$:

$$r_{\mathbf{w}}(s,a) = \mathbf{w}^T\phi(s,a) \text{ and } \mathcal{R}_\phi = \{r_{\mathbf{w}} | \mathbf{w} \in \mathbb{R}^d\}$$

  Note that ϕ can be learned and non-linear
- *Successor features*: $\psi^\pi(s,a) = \mathbf{E}_\pi[\sum_{t=1}^\infty \gamma^t \phi(s_t,a_t) | s_0 = s, a_0 = a]$
- Then the value function for a specified reward function can be easily computed as a function of the successor features:

$$Q^\pi_{\mathbf{w}}(s,a) = \mathbf{w}^T\psi^\pi(s,a)$$

- *Successor features can be pre-computed for π once and re-used thereafter (a form of model!)*
- Connections to hippocampus representations

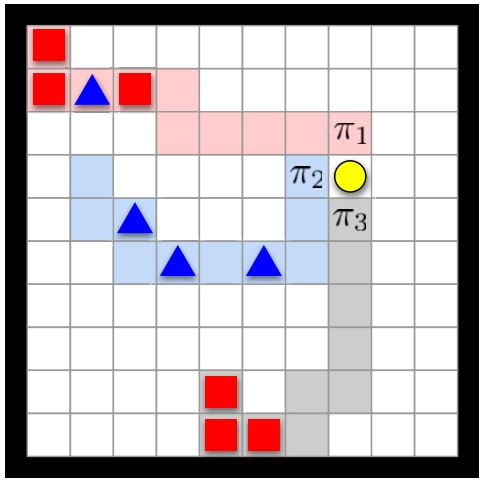
Fast Generalized Policy Improvement

- Compute the improved policy as:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \max_{\pi \in \Pi} Q_r^{\pi}(s, a)$$

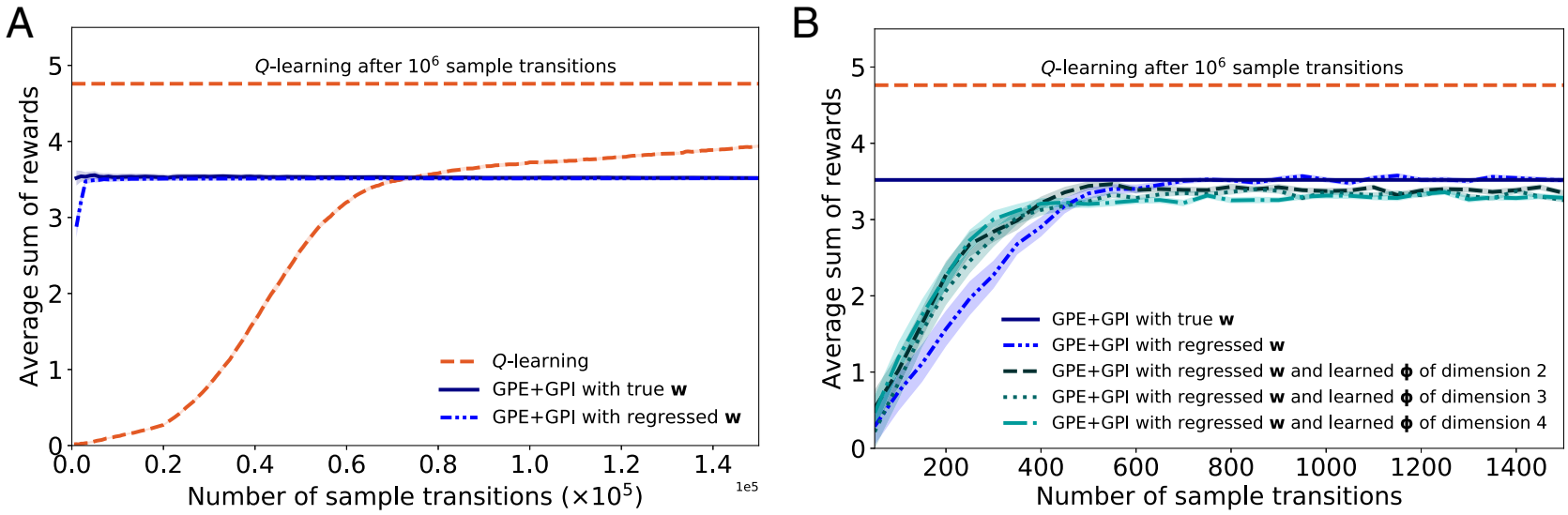
- Note that π' *could choose actions that are not chosen by any of the* π
- The process takes only *one iteration*, after which no further change to the policy π' would happen
- In contrast with iterative policy improvement...

Illustration



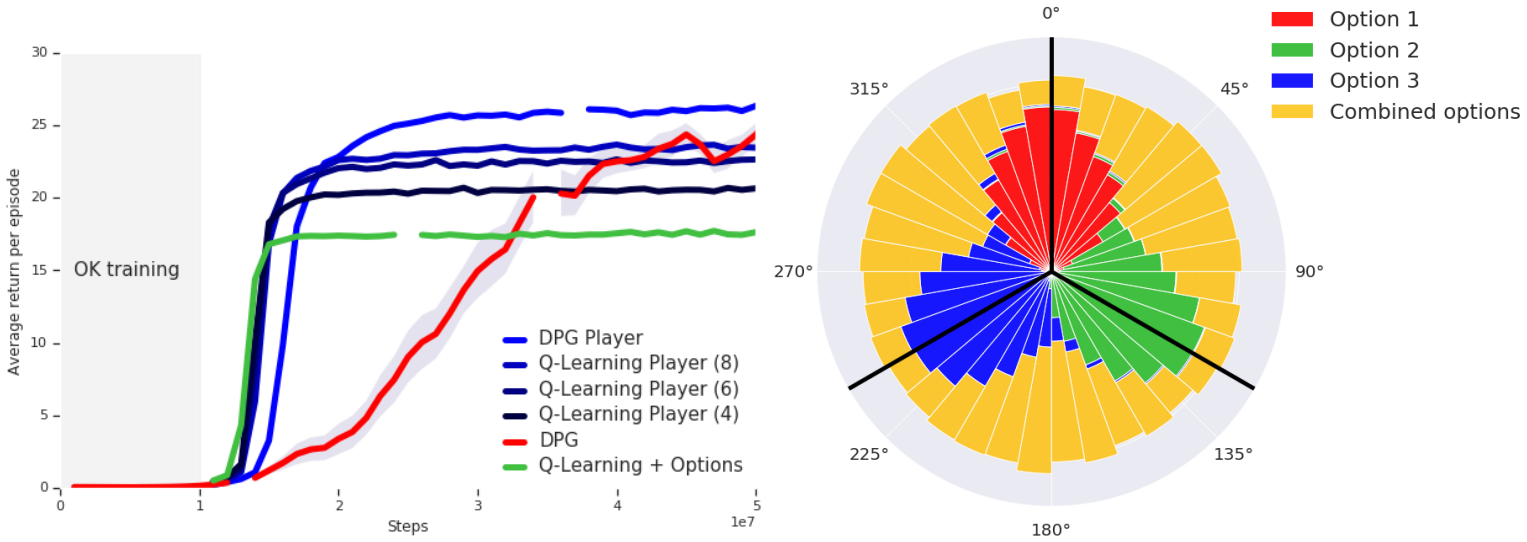
- The three policies correspond to three weight vectors: like red ($\mathbf{w}_1 = [1, 0]^T$), like blue ($\mathbf{w}_2 = [0, 1]^T$) and like red not blue ($\mathbf{w}_3 = [1, -1]^T$)
- *Note that $\mathbf{w}$ can be viewed as a preference function over features!*
- We can pre-train the policies that optimize for each preference, and train their successor features as well
- Then just do GPE/GPI!

Illustration: Results



- Training the successor features for $\mathbf{w}_1$, $\mathbf{w}_2$ over 5×10^5 samples then GPE/GPI for $\mathbf{w}_3$
- GPE/GPI with successor features achieves *75x improvement in sample size* compared to Q-learning
- Obtaining $\mathbf{w}$, ϕ by learning almost as good as knowing these in advance

Option-Keyboard for Moving Target Arena



General way to synthesize quickly new behavior for combinations of reward functions!

Generalizing Initiation Sets: Affordances

Why is temporal abstraction useful for complex RL tasks

- *Advantages to planning*
  - Need to generate shorter plans
  - Improves robustness to model errors
  - Might need to look at fewer states, since the abstract actions have pre-defined termination conditions
  - Discretize the action space in continuous problems
- *Advantages to learning*
  - Improves exploration (can travel in larger leaps)
  - Gives a natural way of using a single stream of data to learn many things (off-policy learning)
- *Advantages to interpretability*:
  - Focusing attention: Sub-plans ignore a lot of information
  - Improves readability of both models and resulting plans
  - Reduces the problem size

Towards General AI Agents Built with Reinforcement Learning

- Reinforcement learning suggests very powerful tools for knowledge representation
  - *Options* are a way to encode procedural knowledge
  - *General Value Functions* are a way to encode predictive knowledge
  - Both can be *combined as building blocks* to quickly solve new problems
- Open questions:
  - Can these ideas lead to *build integrated lifelong learning AI*?
  - *How should we evaluate empirically lifelong learning AI*?