Anytime Recognition of Objects and Scenes

Sergey Karayev
UC Berkeley

Mario Fritz
MPI for Informatics

Trevor Darrell
UC Berkeley

CVPR 2014

We present work on Anytime Recognition of Objects and Scenes.

Human perception is anytime & progressive

PT 40 ms — I saw a very brght object, shaped in a pyramidal shape. There was somethin black in the front, but I couldn't tell what it was. (Subject: JB)

PT 67 ms — Possibly outdoors. maybe a few ducks, or geese. Water in the background. (Subject: JL)

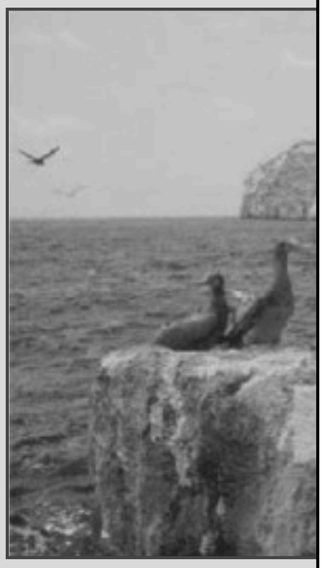
PT 500 ms — It was definately on a coast byt hte ocean with a large [r]ock in the forground and atleast three bird sitting on the rock. (Subject: CC)

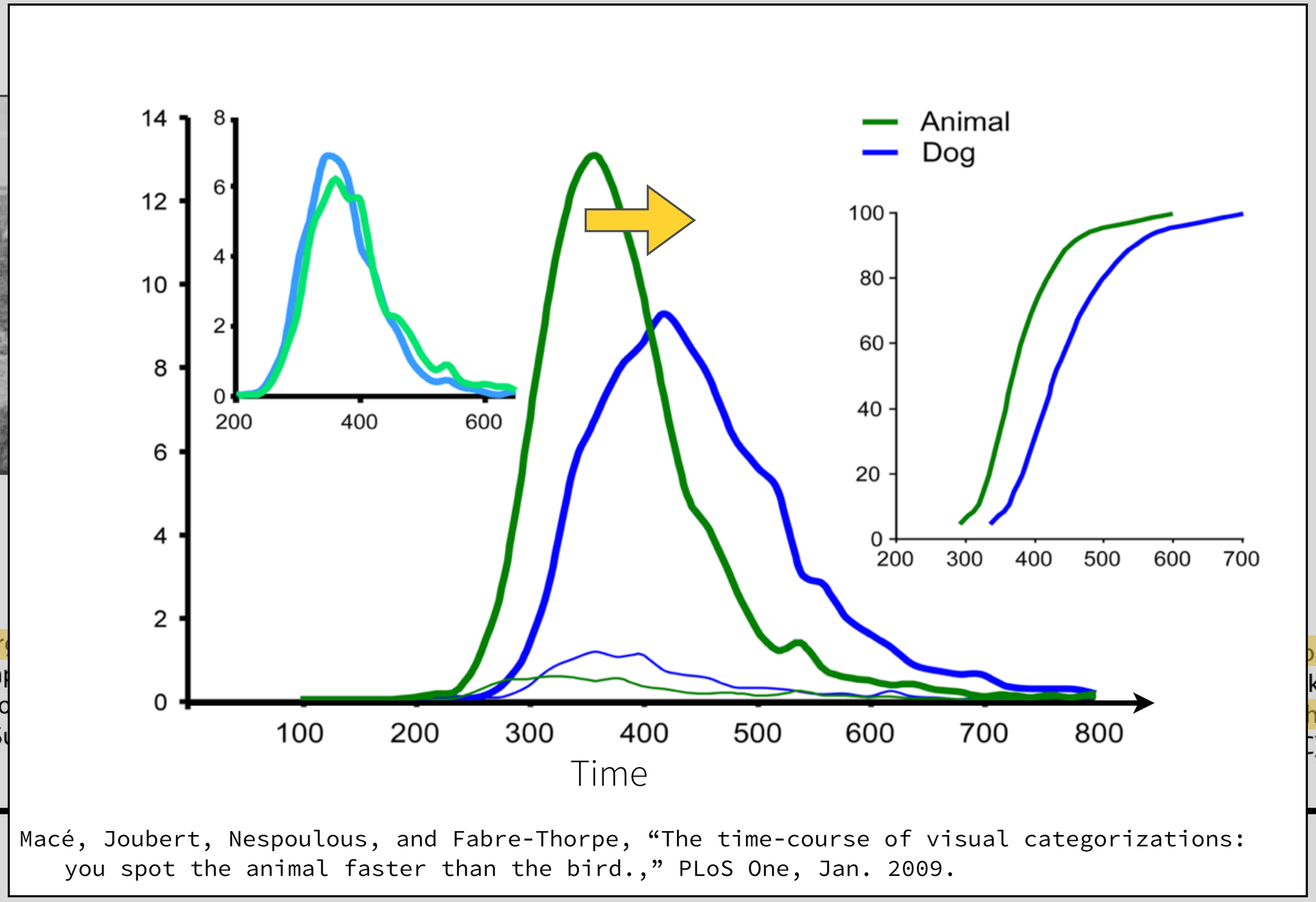
Time

Fei-Fei, Iyer, Koch, and Perona, “What do we perceive in a glance of a real-world scene?,” J. Vis., Jan. 2007.

It is a well-known fact that human perception is both anytime, meaning that a scene can be described after even a short presentation, and progressive, meaning that the quality of description increases with more time.
For example, in one well-known study, the stimulus shown was described as just “a very bright object” after 40 ms, as “possibly outdoors” after 60 ms, and as “definitely on the coast, with at least three birds” after 500 ms.

Human perception is anytime, progressive





PT
40 ms

I saw a very br
pyramidal shap
black in the fro
what it was. (Su

past byt hte
k in the
hree bird sitting
C)

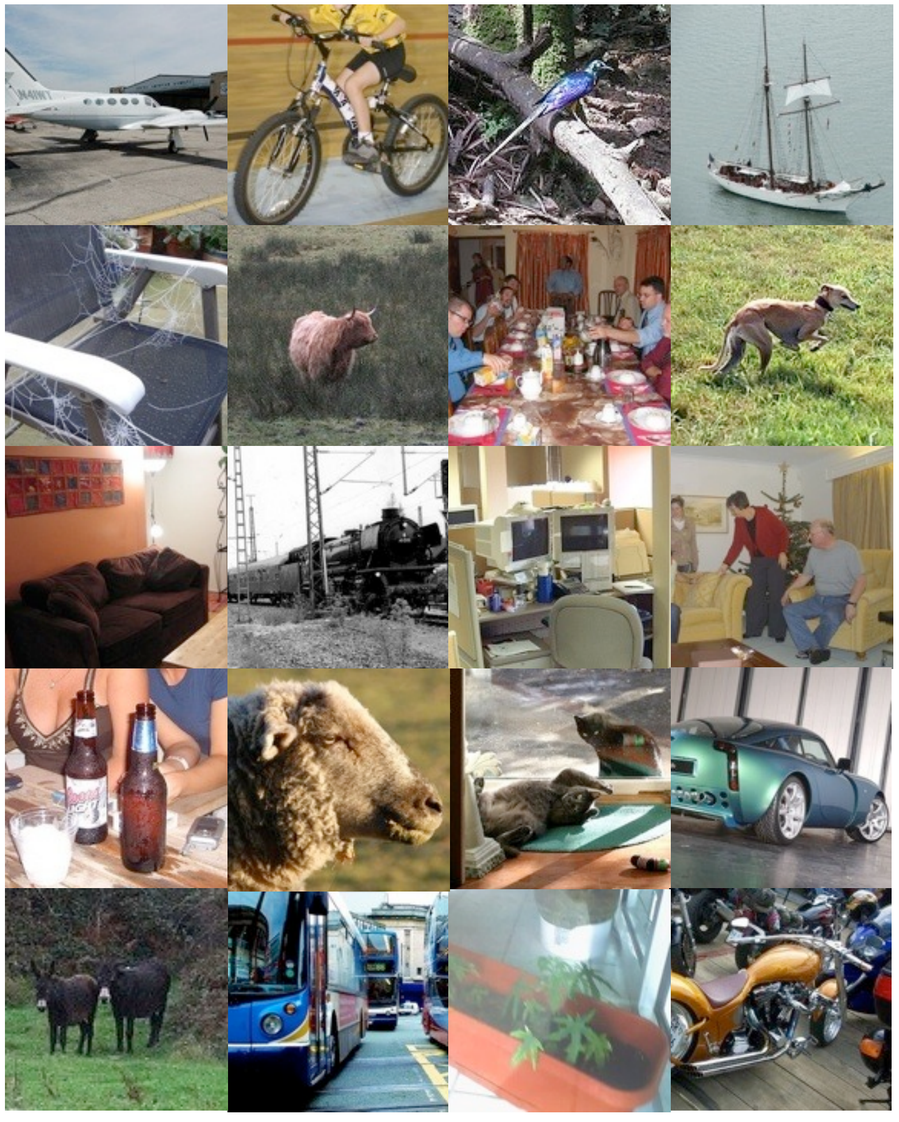
Time

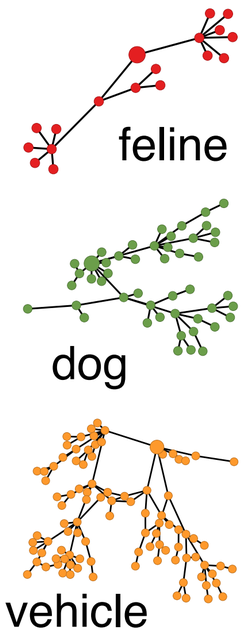
Macé, Joubert, Nespoulous, and Fabre-Thorpe, “The time-course of visual categorizations: you spot the animal faster than the bird.,” PLoS One, Jan. 2009.

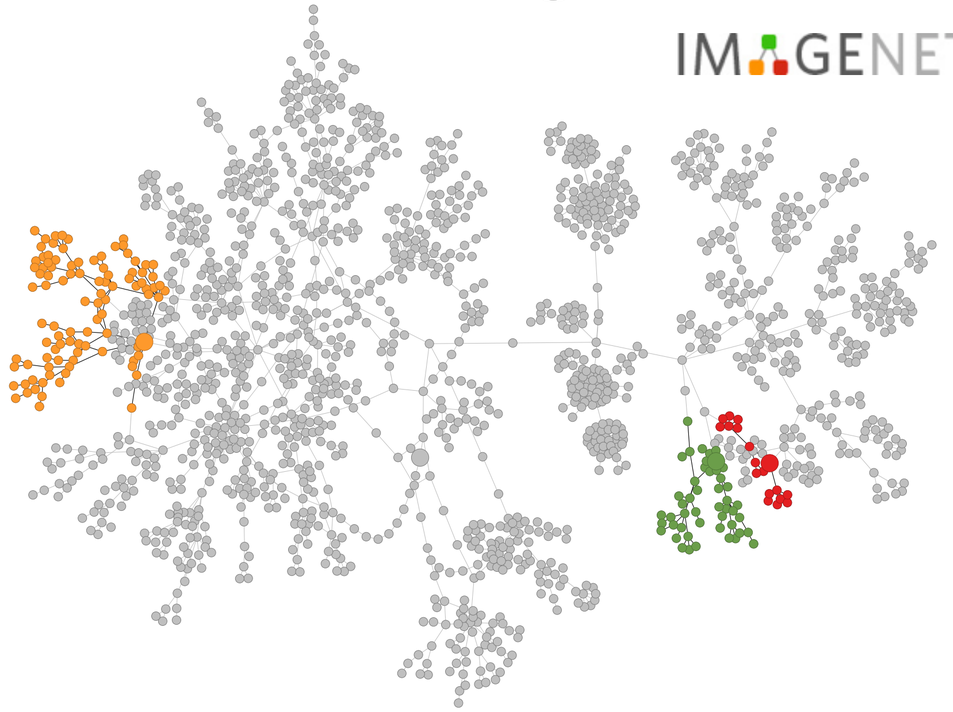
Fei-Fei, Iyer, Koch, and Perona, “What do we perceive in a glance of a real-world scene?,” J. Vis., Jan. 2007.

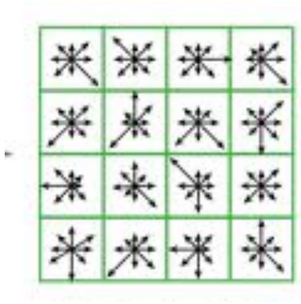
There is also evidence that the progressive enhancement of description occurs in an ontologically meaningful way, as for example, when we recognize something as an animal before recognizing it as a dog.

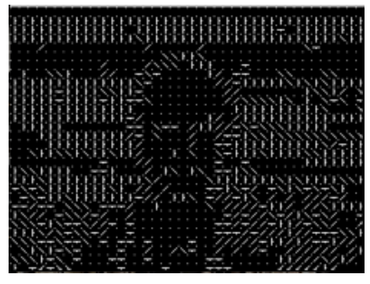
We deal with varied inputs, classes, and recognition actions

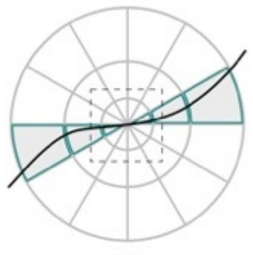


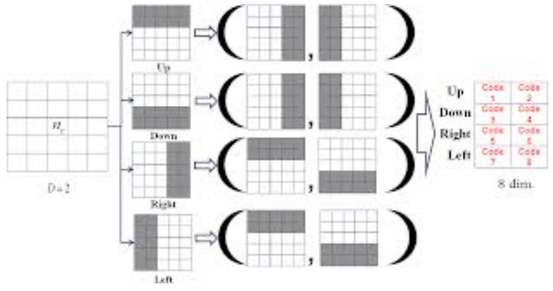


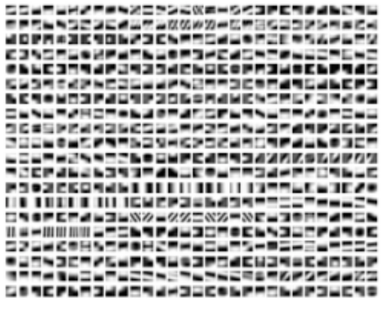










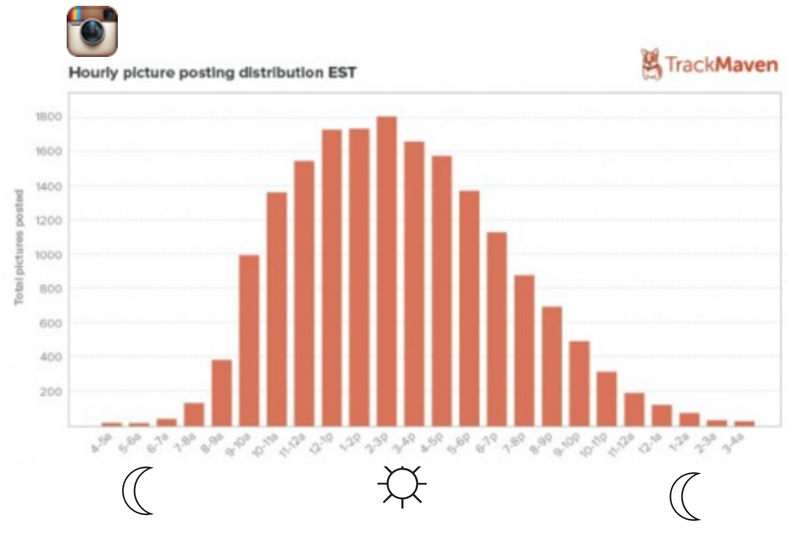


In computer vision, we deal with a variety of inputs, ranging from simple frames to highly complex scenes; a large number of classes, which may be ontologically organized; and many possible recognition actions.
Crucially, these recognition actions have different computational costs and provide different benefits.

Motivation: time-sensitive applications

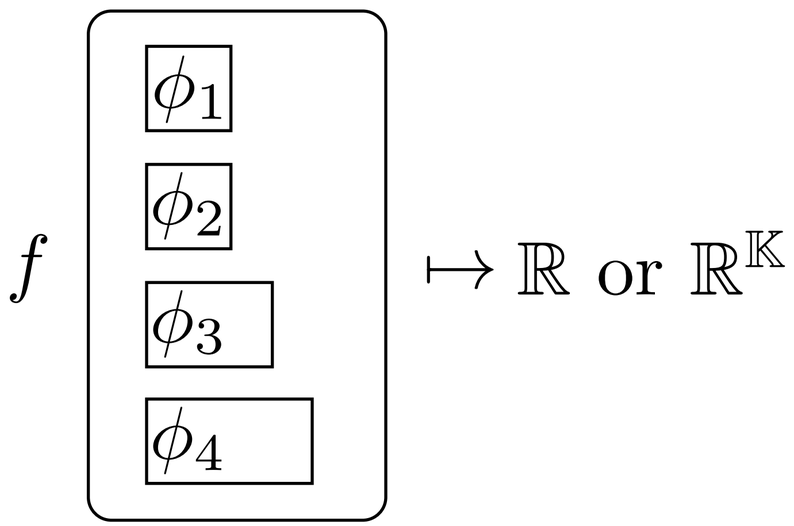
Goal is to provide the best performance, given any budget.

Even when it is flexible or unknown.



We are faced with a multitude of time-sensitive applications.
Driver assistance or replacement clearly has real-time needs, while large server farms can crank away with little time pressure.
In between are robotic applications, which need to respond to input in a time-sensitive way, and flexible deployments, where the quality of recognition may depend on the length of the job queue.
In settings like these, we may not be able to compute all possible features or recognize all classes.
To provide the best performance given any budget, we would like our recognition actions to depend on the input.

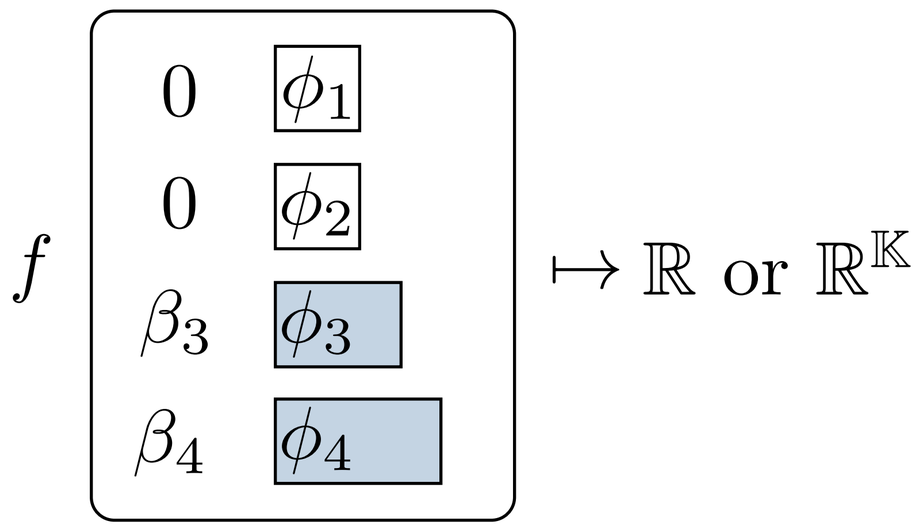
Setup: classifier



- General combination of features.
- $\boxed{\phi_4}$ is more expensive than $\boxed{\phi_1}$

Let's begin with a general classifier, which computes features of different costs, and combines them with learned weights to output a classification score.

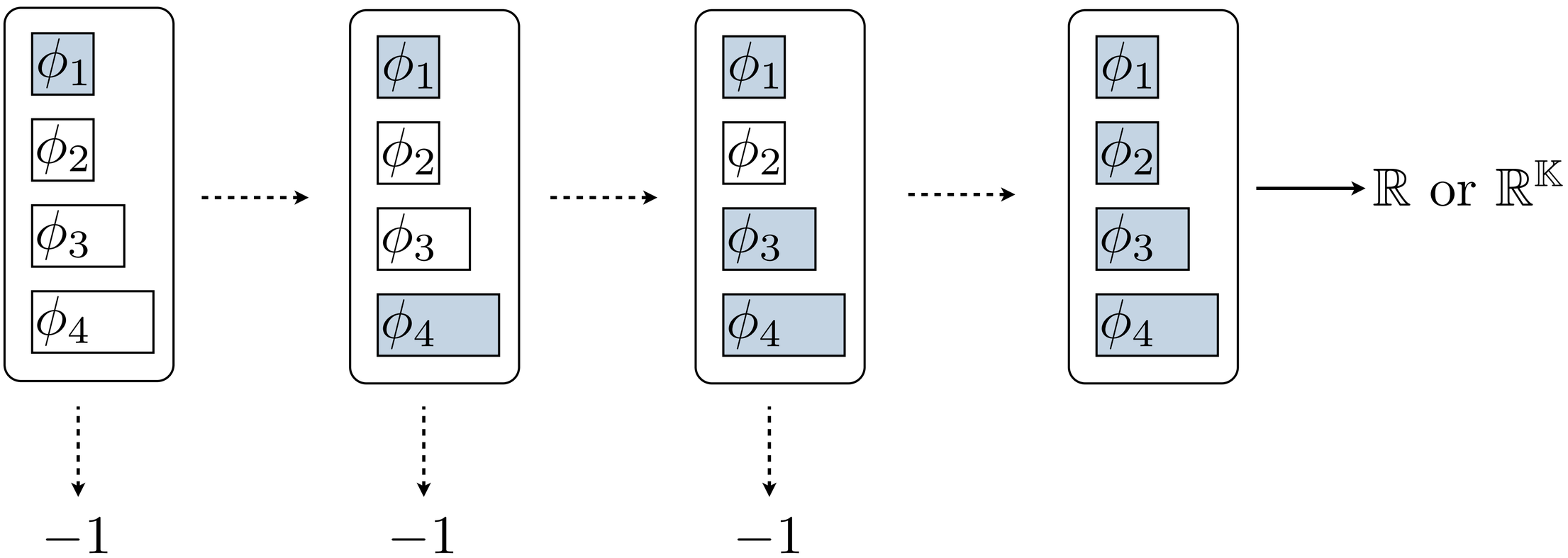
Setup: static feature selection



- Select best features for data source and average budget.
- Insufficient for Anytime performance.
- Static selection is not robust to image variety.

Faced with a specific data source and some knowledge of the budget, we can select the best features to compute.
However, this is not sufficient for Anytime performance, when the budget may not be precisely known, and there may not be enough time to compute all features.

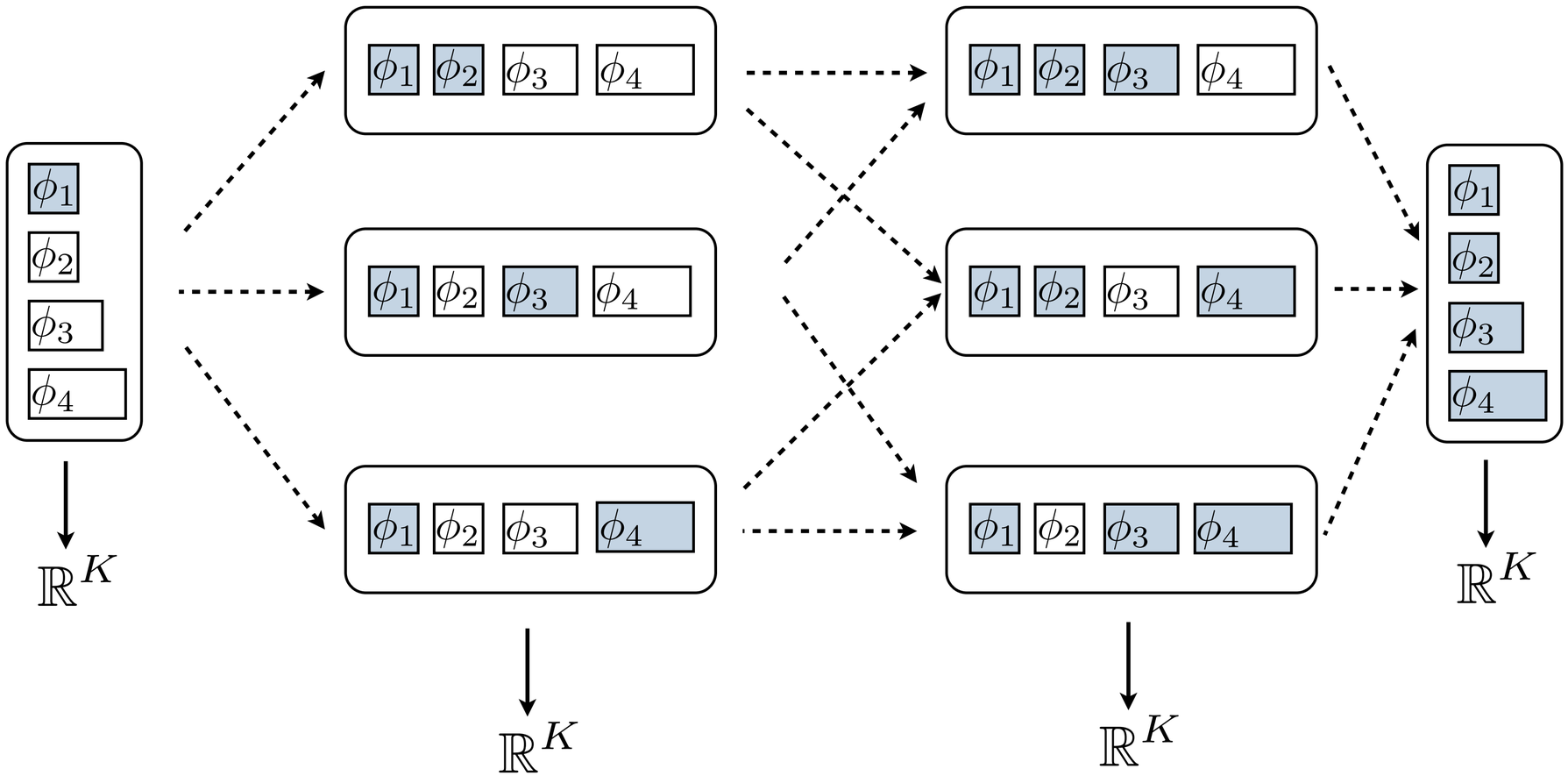
Setup: cascades



- Two **actions**: Reject and Continue.
- Anytime performance is limited.
- Fixed order is not robust to image variety.

The Cascade is the classical idea of evaluating features in sequence, as merited by the image.
In addition to the feature computation actions, the classifier is augmented with a rejection action.
The cascade is Anytime in a limited way, as only the rejection answer can be given before all features are evaluated.
Furthermore, the fixed order of the cascade is not robust to the fact that different images benefit from different features.

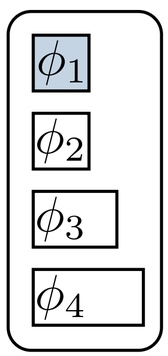
Our method



- Features are selected for computation in a flexible order.
- We find a *dynamic*, *non-myopic* policy for state traversal using reinforcement learning.
- Classification answer can be given at any point in the computation.

Our work considers not a chain but a general DAG over selected-feature subsets, which allows actions to be taken in an entirely flexible order.
Given a data source and a range of budgets, we use reinforcement learning to find a policy for navigating this graph.
The policy is dynamic (meaning that it incorporates feedback) and non-myopic (meaning that it plans ahead),
This makes it robust to the fact that different images benefit from different features.
We are also able to give the classification answer from all states, making our work truly Anytime.

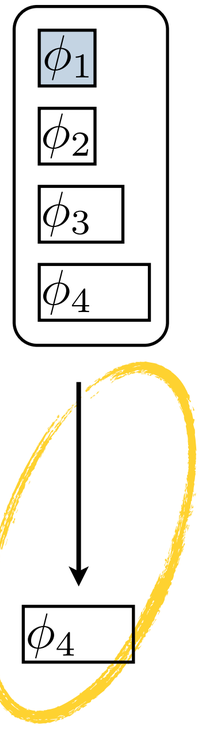
Our method



- We model the problem of **state** traversal as a Markov Decision Process.

This is how it works.
We model the problem as a Markov Decision Process, with the state consisting of the computed feature values.

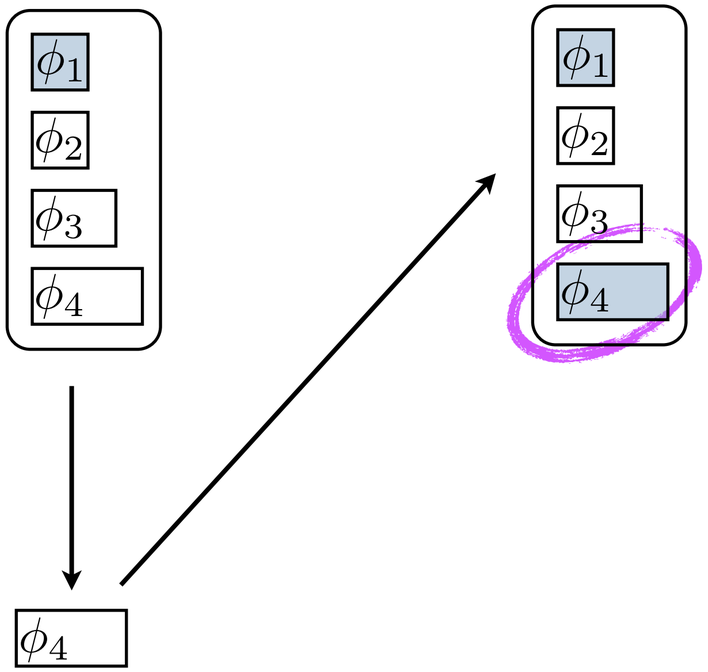
Our method



- We model the problem of **state** traversal as a Markov Decision Process.
  - For every state**,** we learn to **select action** of maximum **expected value.**

From each state, we learn to select the action of maximum expected value, where expected value depends on how we define the Reward function -- this will be explained later.

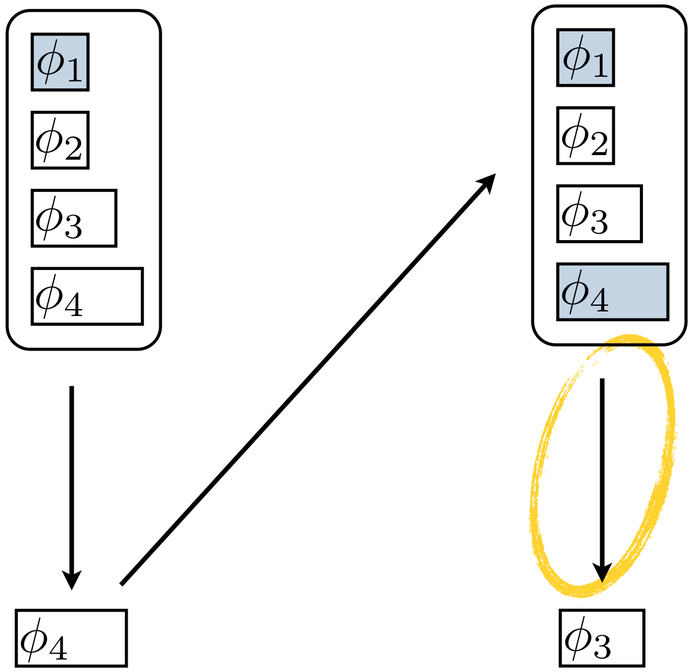
Our method



- We model the problem of **state** traversal as a Markov Decision Process.
  - For every state**,** we learn to **select action** of maximum expected value.
  - State is **updated** with the result of the selected action.

Upon taking an action and computing a feature, the state is updated with the result.

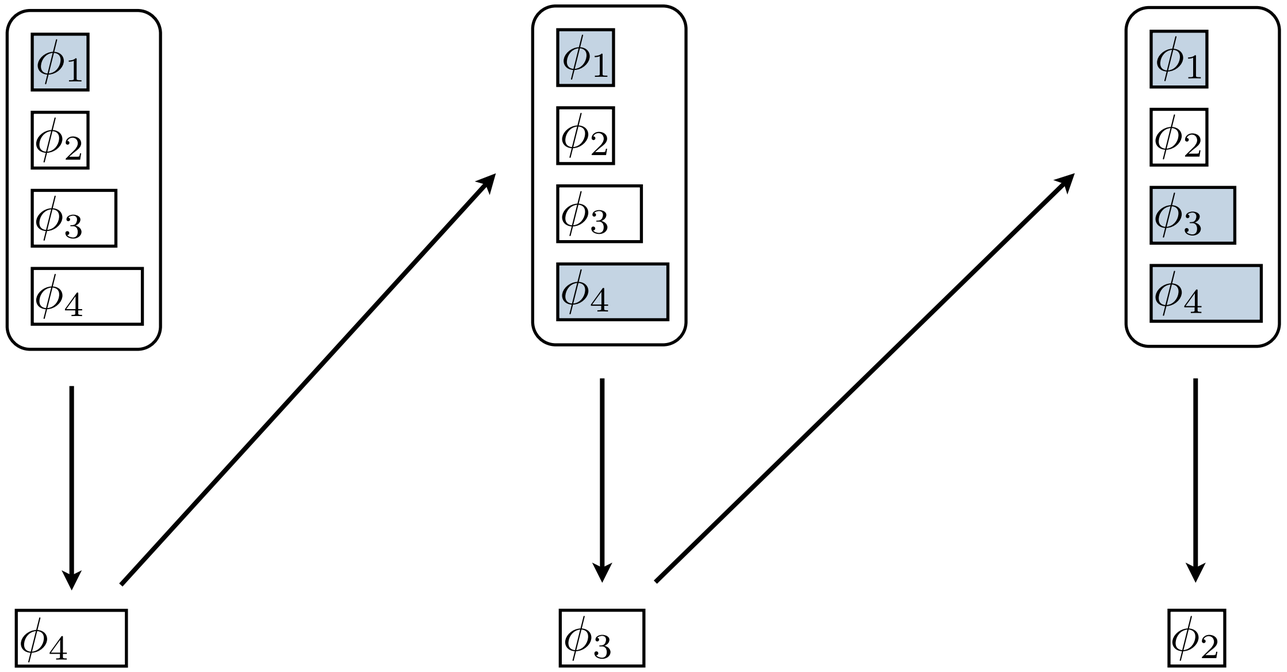
Our method



- We model the problem of **state** traversal as a Markov Decision Process.
  - For every state**,** we learn to **select action** of maximum expected value.
  - State is **updated** with the result of the selected action.

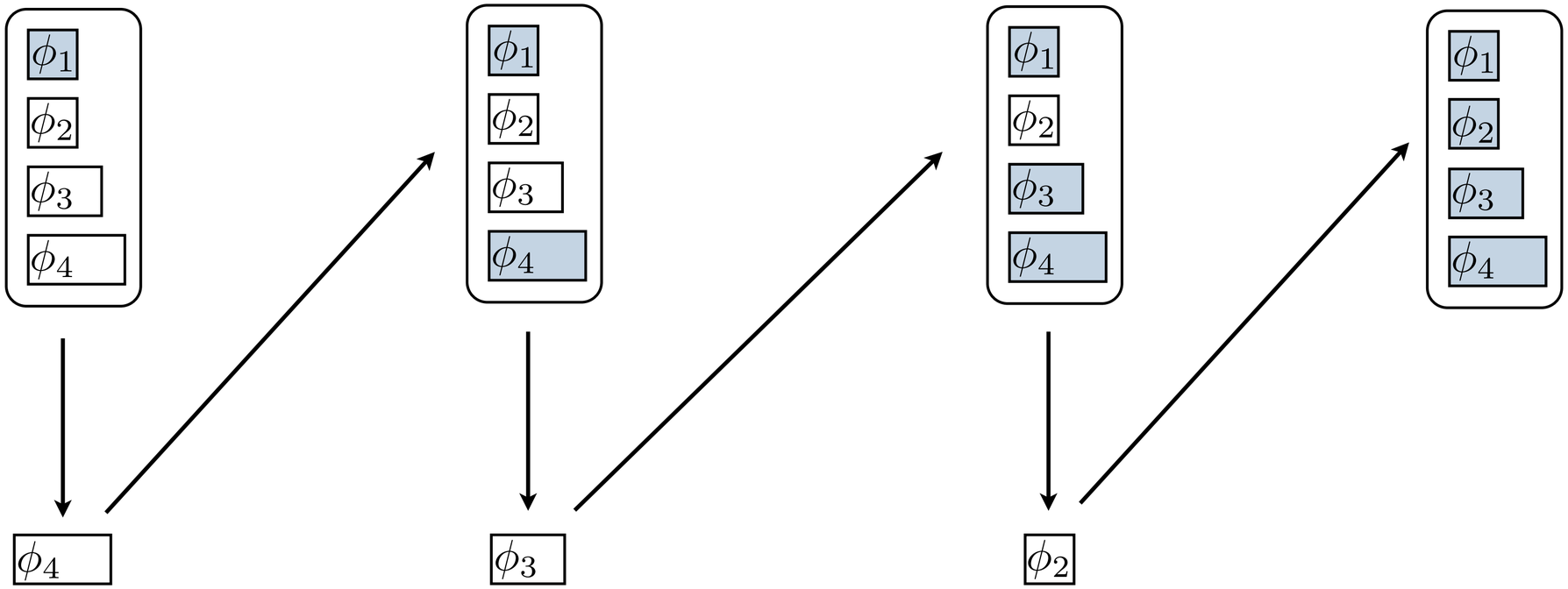
This process repeats until either all features are computed, or we have exhausted our budget.

Our method



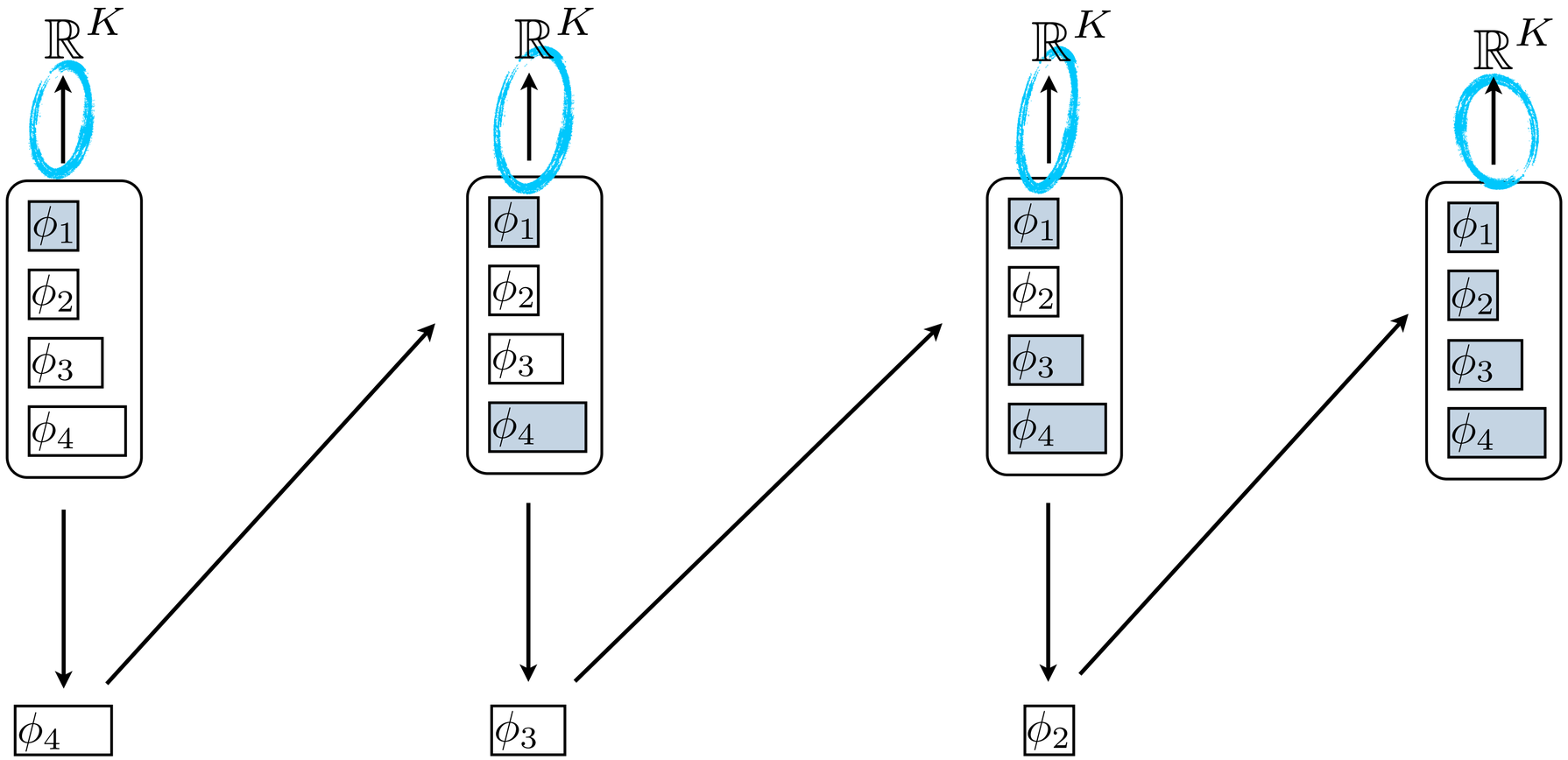
- We model the problem of **state** traversal as a Markov Decision Process.
  - For every state**,** we learn to **select action** of maximum expected value.
  - State is **updated** with the result of the selected action.

Our method



- We model the problem of **state** traversal as a Markov Decision Process.
  - For every state**,** we learn to **select action** of maximum expected value.
  - State is **updated** with the result of the selected action.

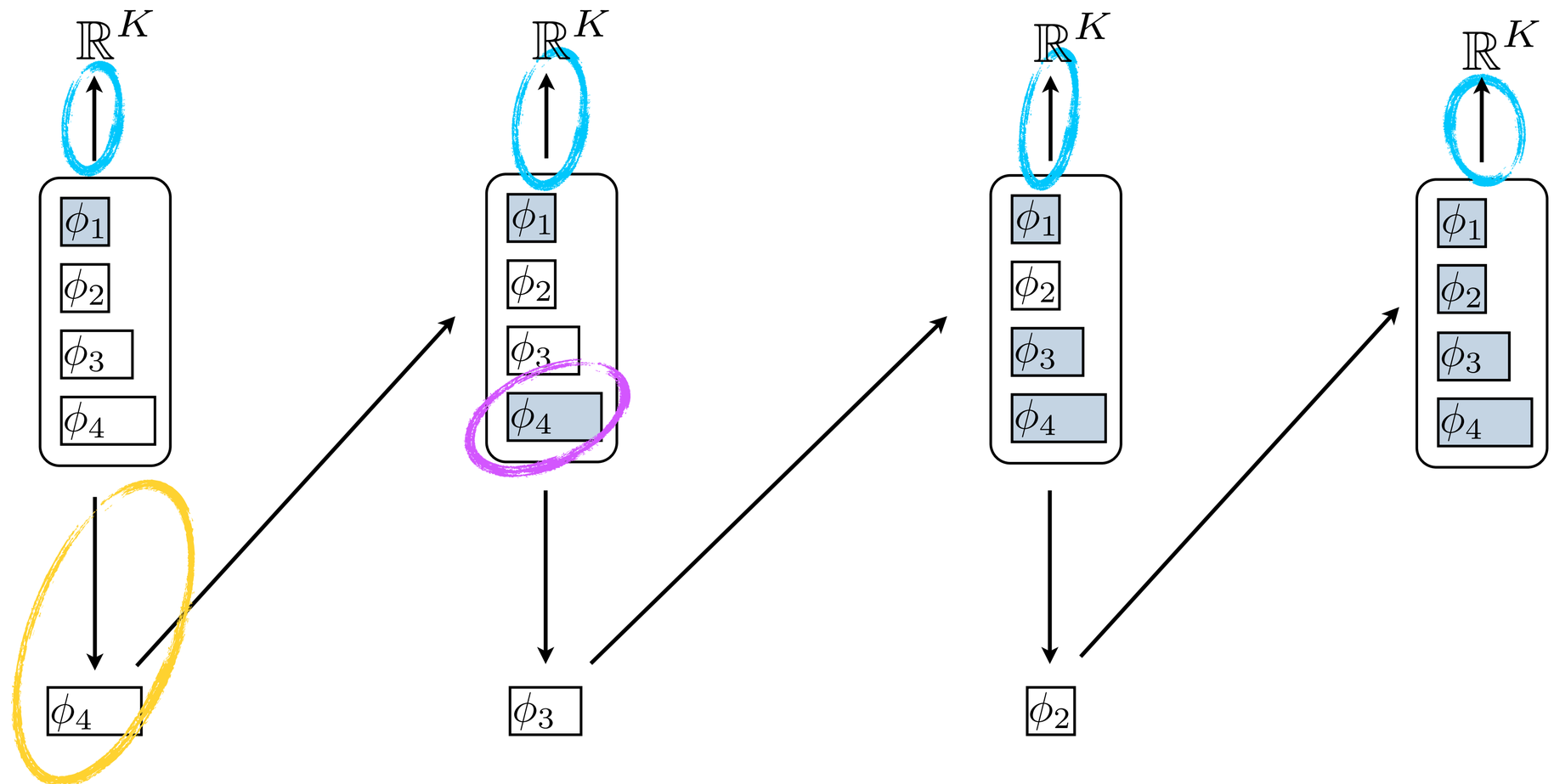
Our method



- We model the problem of **state** traversal as a Markov Decision Process.
  - For every state**,** we learn to **select action** of maximum expected value.
  - State is **updated** with the result of the selected action.
- We train **classifier** on subsets of features, to give answer at any time.

We train a classifier on all subsets of features encountered in running the policy; this allows us to give Anytime answers.

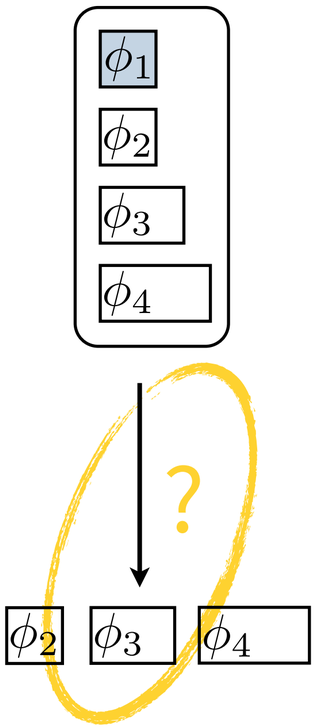
Our method



- We model the problem of **state** traversal as a Markov Decision Process.
  - For every state**,** we learn to **select action** of maximum expected value.
  - State is **updated** with the result of the selected action.
- We train **classifier** on subsets of features, to give answer at any time.

Now all the components are in place.
So how do we learn to select actions?

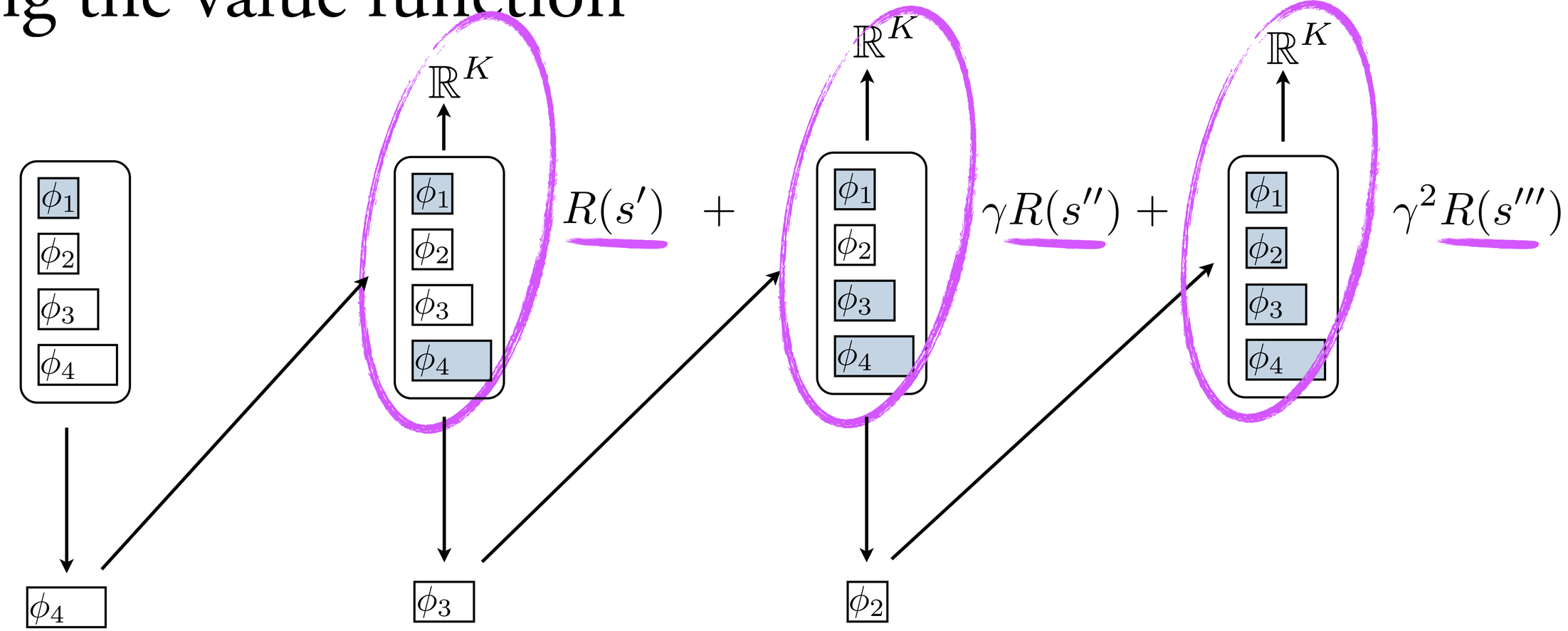
Learning the value function



- Action selection is done by the **policy** function: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$

Our goal is to learn a policy function that will follow a simple rule: for each state, pick the action that maximizes the *value function*.
If we can learn the value function, we have our policy.

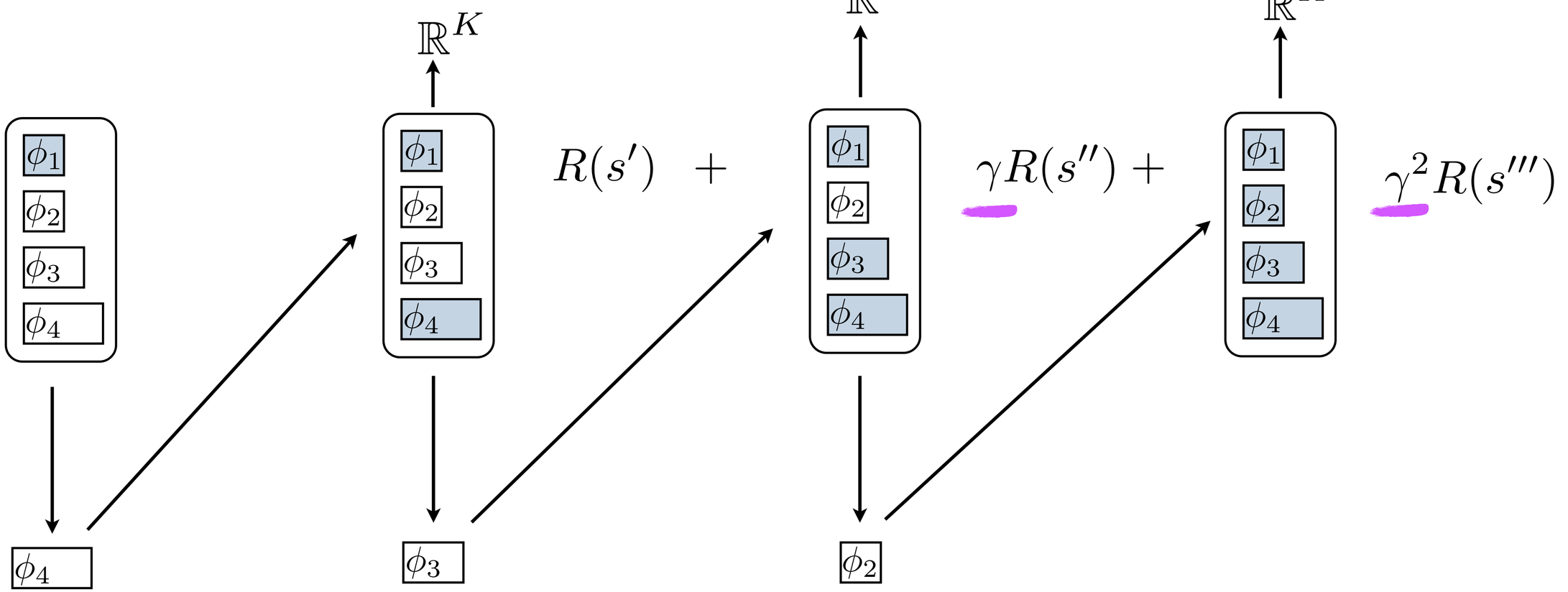
Learning the value function



- Action selection is done by the **policy** function: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$
- $\mathrm{Q(s,a)}$ gives the expected **sum of rewards** to the end of the episode.

The Q-value function gives the expected sum of rewards of taking the action in the state, then following the policy to the end of the episode.
The reward is a function of the state, and has to be manually defined -- I will explain how we define it in a few slides.

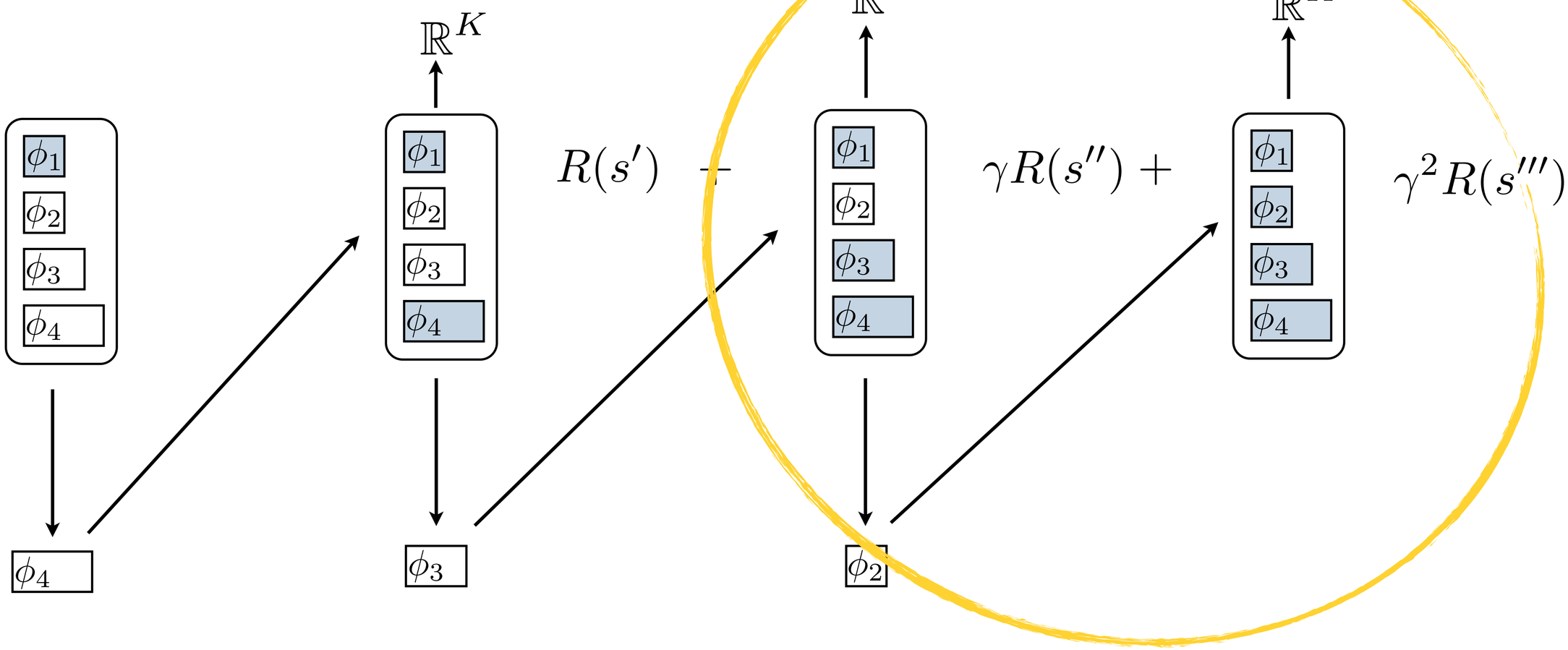
Learning the value function



- Action selection is done by the **policy** function: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$
- $\mathrm{Q(s,a)}$ gives the expected **sum of rewards** to the end of the episode.
- Parameter γ controls the amount of lookahead.

Note that the discount parameter gamma controls how much future rewards contribute to the value of taking an action now.
With gamma equal to 0, only the next step reward is considered.
With gamma equal to 1, future rewards are weighed equally to next-step rewards.
This can vary a policy from greedy to non-myopic.

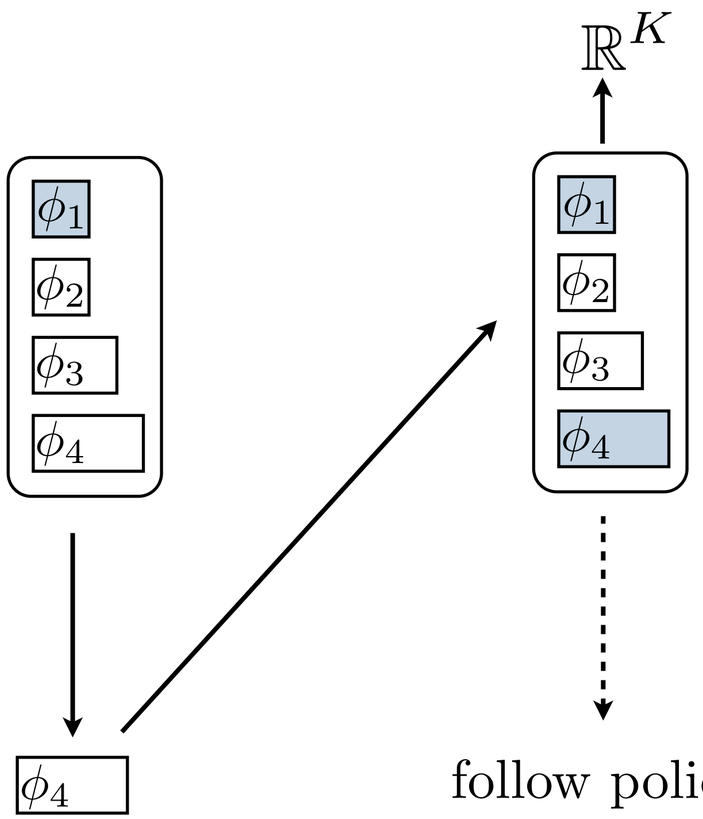
Learning the value function



- Action selection is done by the **policy** function: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$
- $\mathrm{Q(s,a)}$ gives the expected **sum of rewards** to the end of the episode.
- Parameter γ controls the amount of lookahead.

The Q-value function may be written recursively, as simply the next-step reward plus the discounted value of following the policy from that state.

Learning the value function



$$R(s') \quad + \quad \gamma Q^{\pi}(s', \pi(s'))$$

- Action selection is done by the **policy** function: $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$
- $\mathrm{Q(s,a)}$ gives the expected **sum of rewards** to the end of the episode.
- Parameter γ controls the amount of lookahead.
- $Q^{\pi}(s, a) = \mathbb{E}_{s'}\left[R(s') + \gamma Q^{\pi}(s', \pi(s'))\right]$

The Q-value function may be written recursively, as simply the next-step reward plus the discounted value of following the policy from that state.

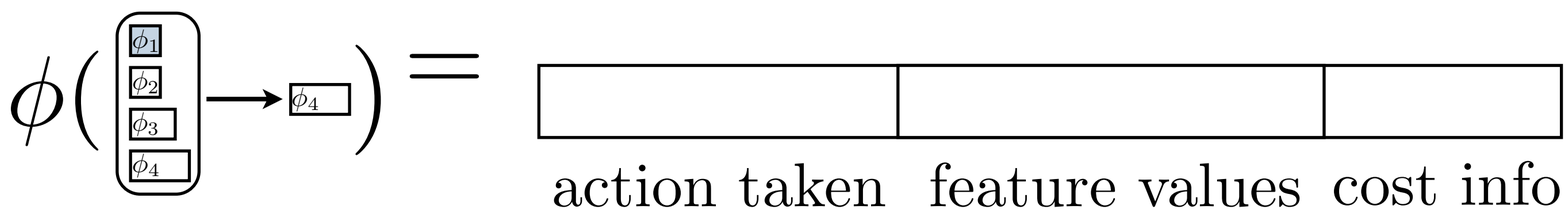
Learning the value function

- We sample $Q^\pi(s,a) = \mathbb{E}_{s'}\left[R(s') + \gamma Q^\pi(s', \pi(s'))\right]$ by running the policy.

- Due to infinite number of states, we learn an approximation:

$$Q^\pi(s,a) = \theta^T \phi(s,a)$$

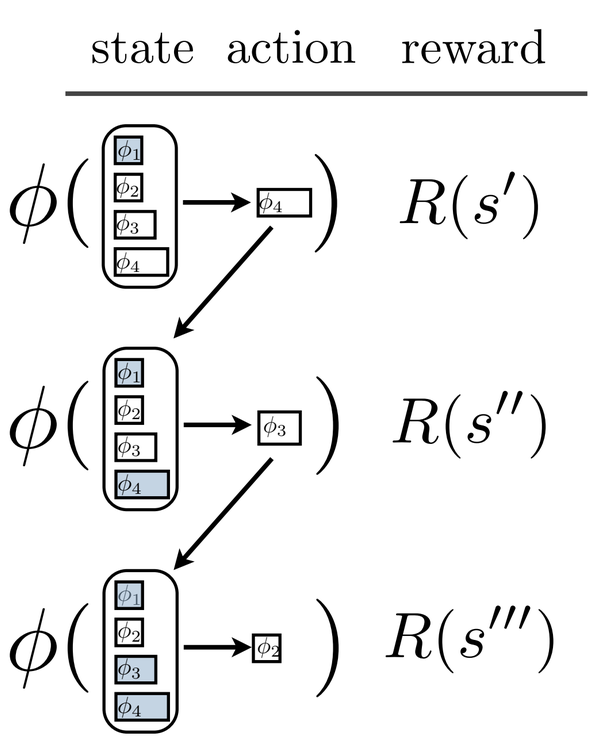
- State-action pairs are featurized with relevant values:



The Q-function is an expectation, and can be sampled by running the policy many times.
Because there is an infinite number of states, we cannot tabulate the Q-function.
Instead, we learn a linear approximation of it, for which we need a featurization of the state-action pair.
We featurize the state-action with the actions that have been taken, feature values that have been observed, and information about the action cost and remaining budget.

Learning the value function

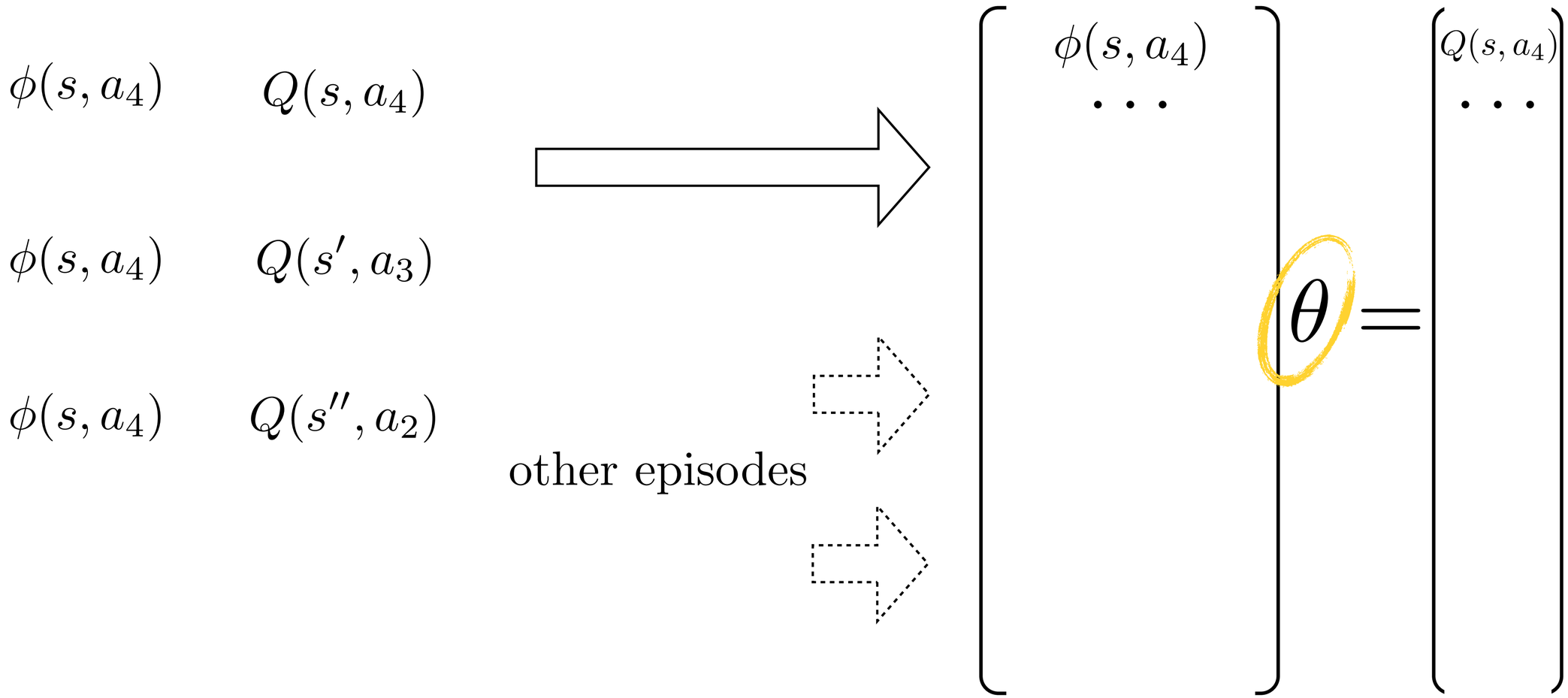
- We sample $Q^\pi(s,a) = \mathbb{E}_{s'}\left[R(s') + \gamma Q^\pi(s', \pi(s'))\right] = \theta^T \phi(s,a)$ by running the policy over many images.



To learn the approximation, we generate many recognition episodes by following a policy.`
Here, we illustrate just one episode, but there are many.
Running the policy, we gather state-action-reward tuples, and featurize state-action pairs.
At the end of the episode, we know not only the reward for each tuple, but the sampled Q-function values, obtained by adding up rewards.

Learning the value function

- We sample $Q^{\pi}(s,a) = \mathbb{E}_{s'}\left[R(s') + \gamma Q^{\pi}(s', \pi(s'))\right] = \theta^T \phi(s,a)$ by running the policy over many images.
- Minimize prediction error of θ.



We gather a large training set by running episodes on many images, and update the linear approximation of the Q-value function, theta, by minimizing its prediction error.

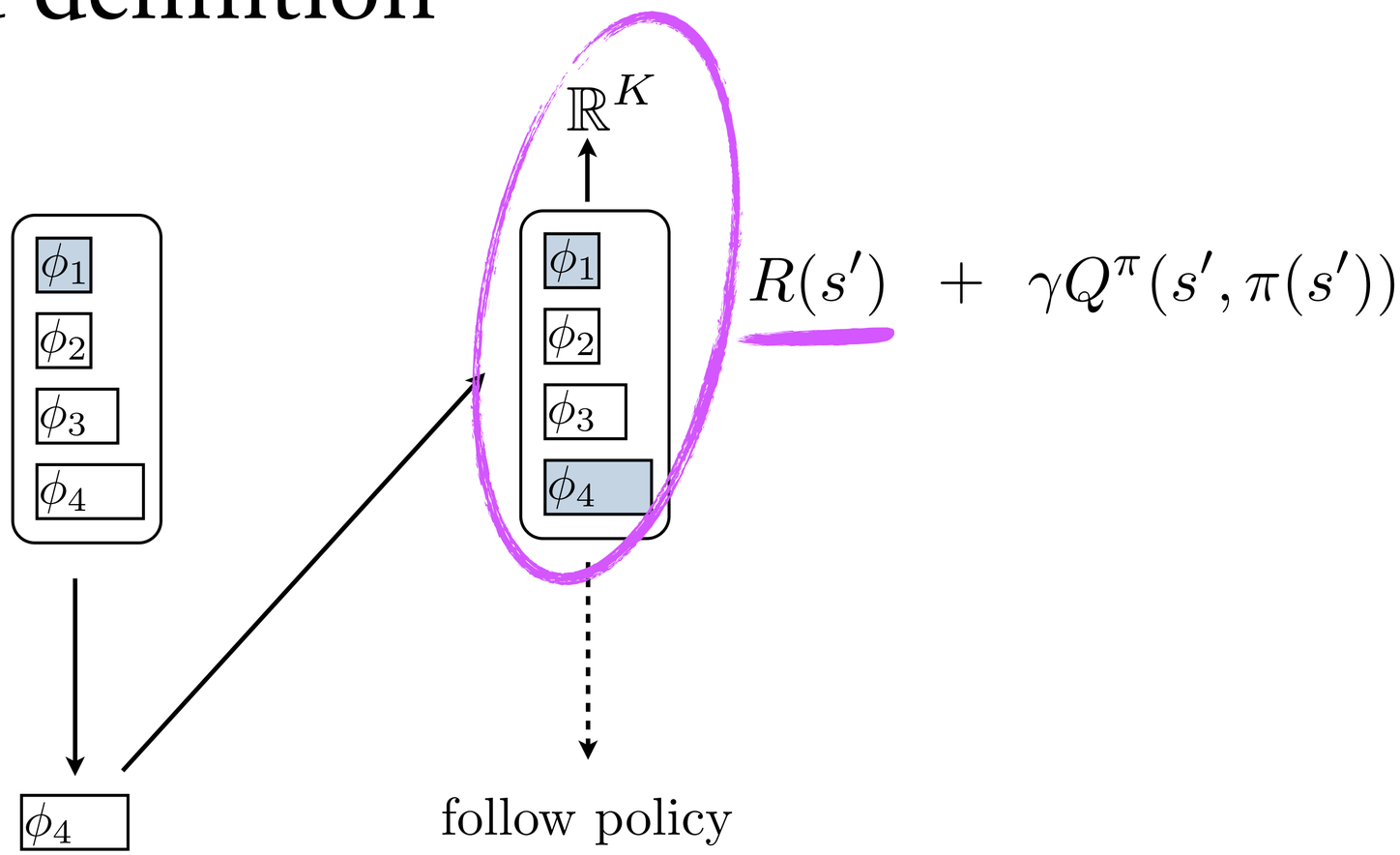
Training algorithm

$\pi_0 \leftarrow$ random;
for $i \leftarrow 1$ **to** *max_iterations* **do**
  States, Actions, Costs, Labels $\leftarrow$ `GatherSamples(`$\mathcal{D}$, π_{i-1}`)`;
  Values $\leftarrow$ `ComputeValues(`*States*, *Costs*, *Labels*, $g_i, \mathcal{L}_{\mathcal{B}}, \gamma$`)`;
  $\pi_i \leftarrow$ `UpdatePolicy(`*States*, *Actions*, *Values*`)`;
end

The overall training algorithm is therefore as follows:
- First, initialize policy randomly
- Gather samples by running recognition episodes with current policy.
- Compute the sampled values for every state-action tuple gathered.
- Update the Q-value function, which updates the policy.

We repeat the above until the values converge, or we're out of iterations. This is a variant of reinforcement learning algorithm called Q-iteration.

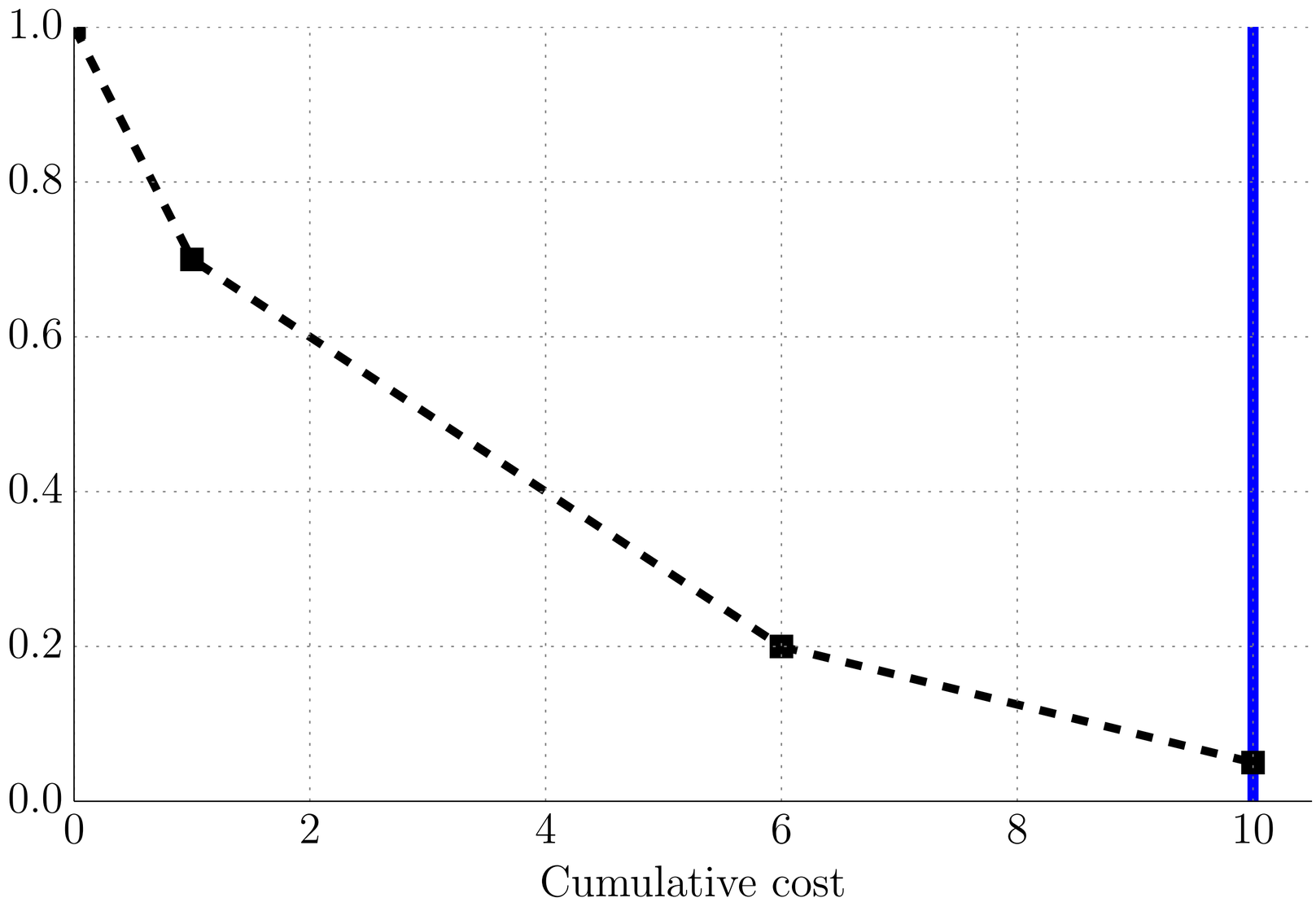
Reward definition



- Still one piece missing: what is the **reward**?

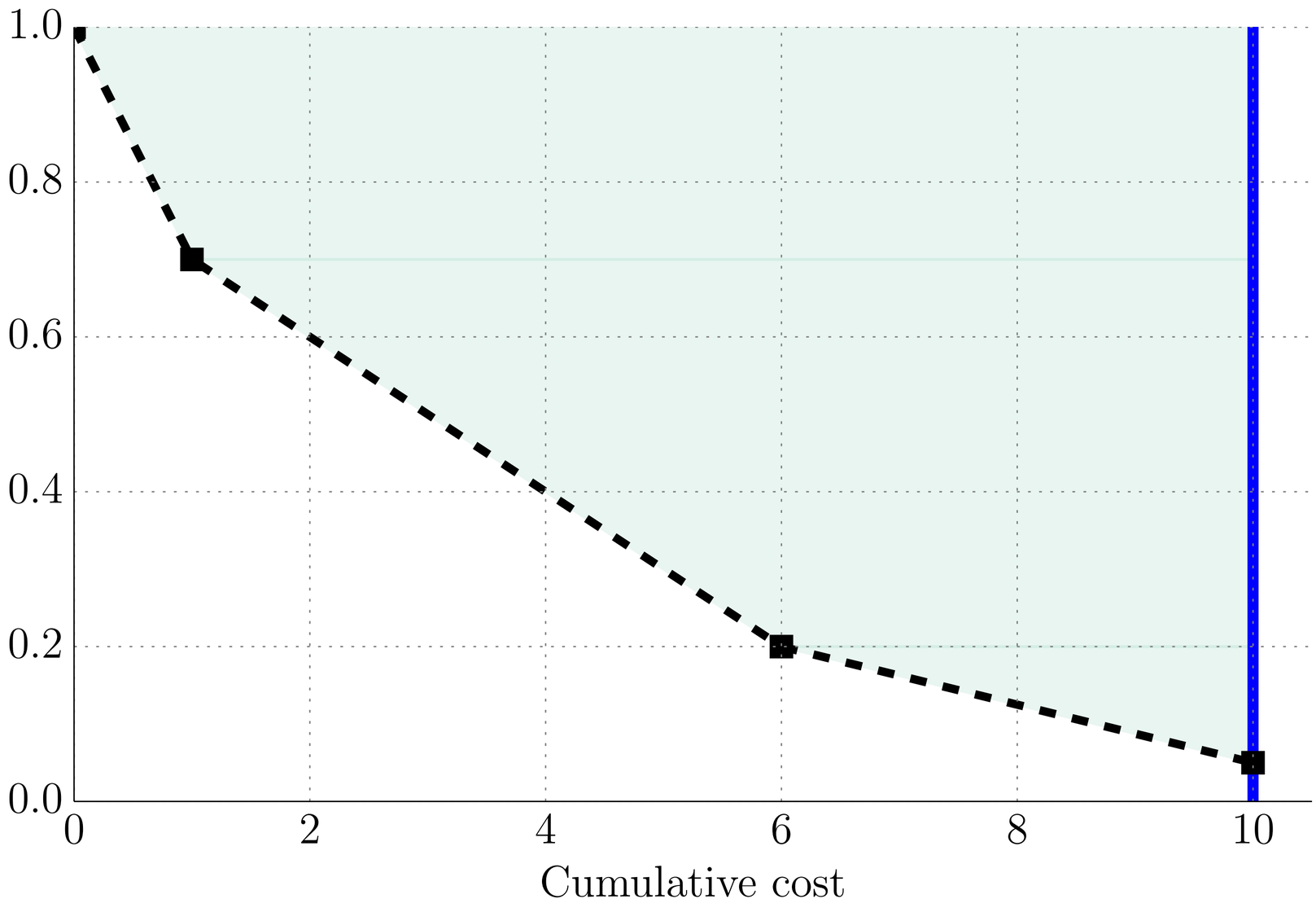
There's still one piece missing in the story: how shall we define the reward function?

Reward definition



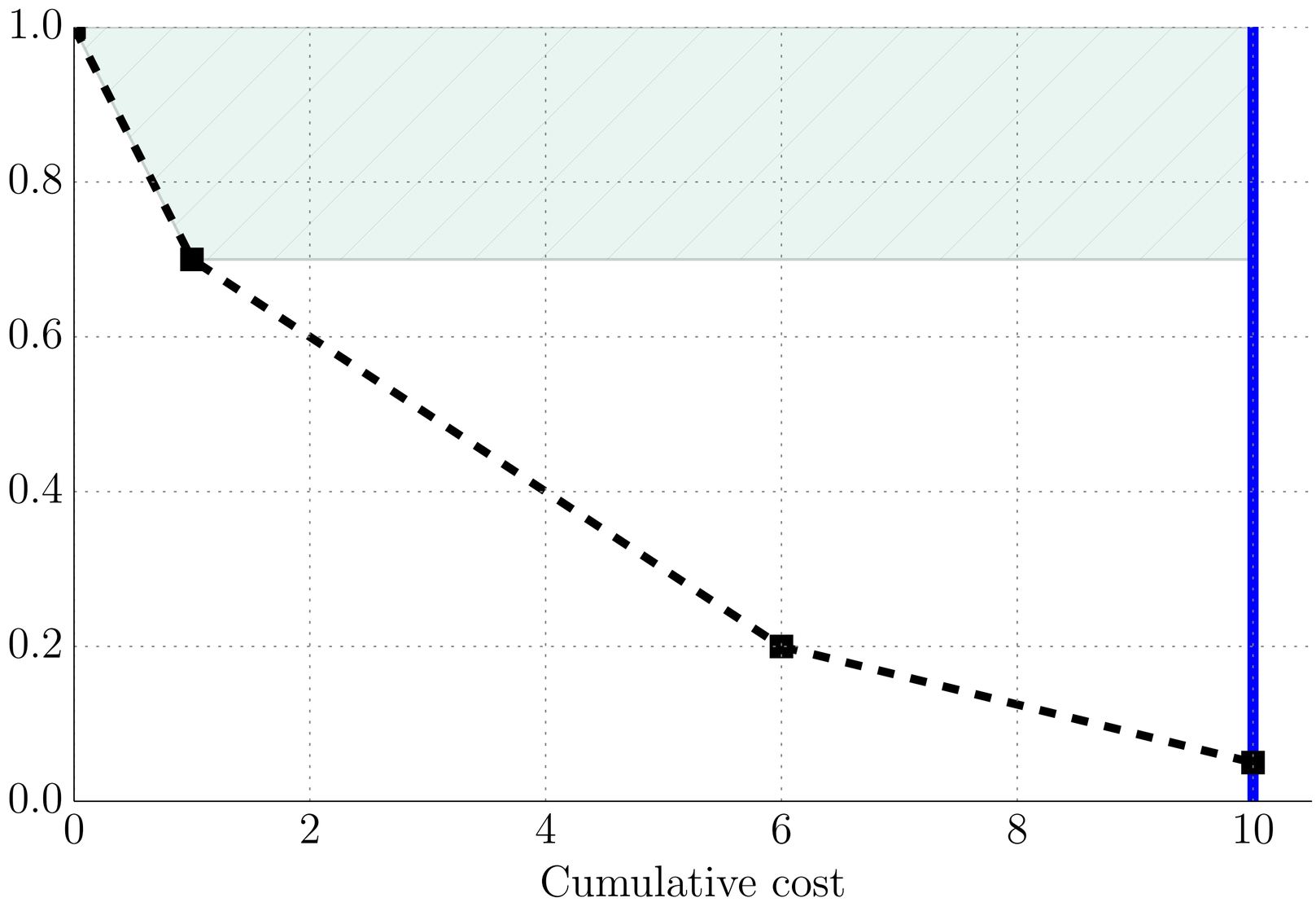
The goal of the Anytime classifier is to predict as correctly as possible, as cheaply as possible.

Reward definition



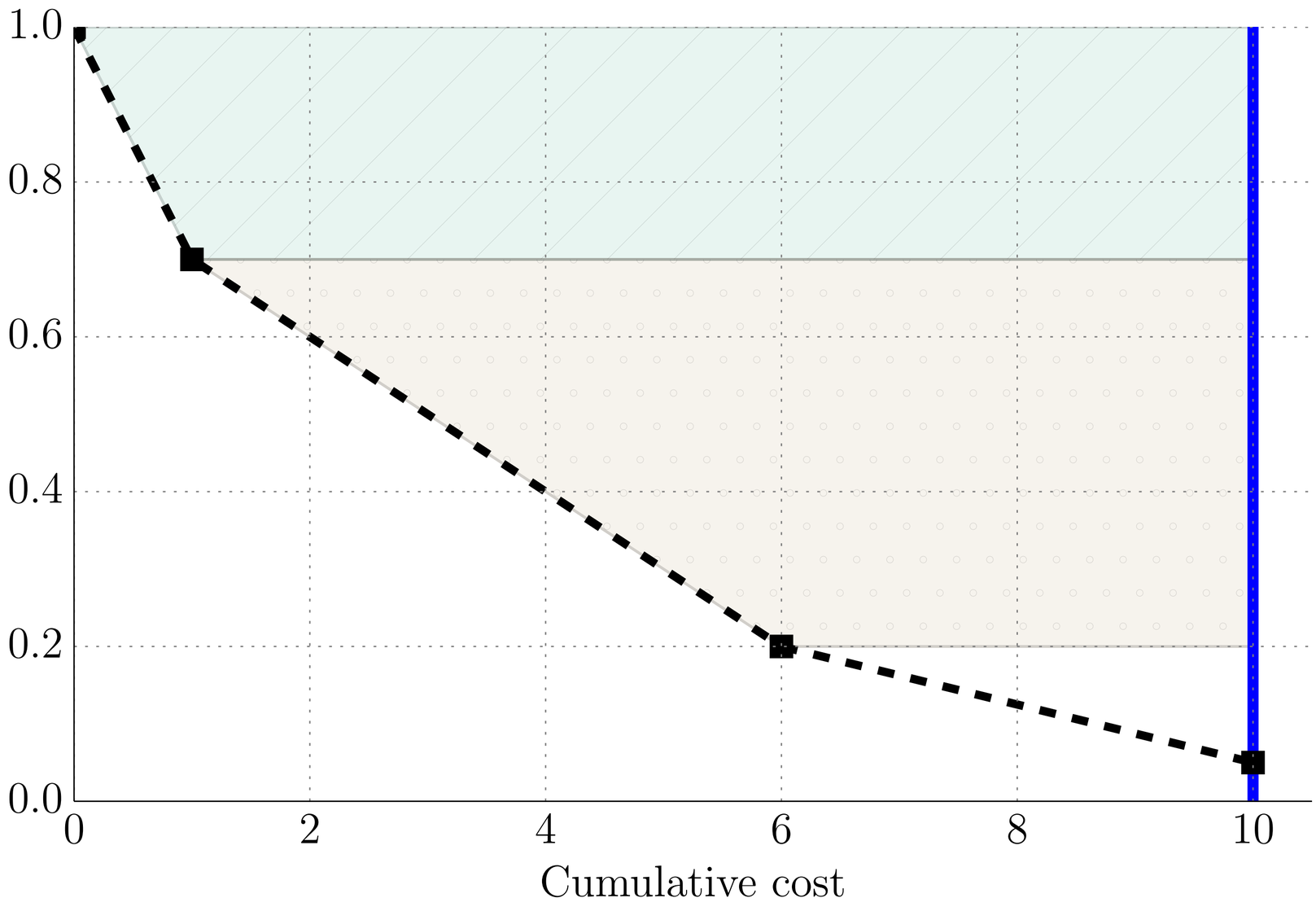
This corresponds to maximizing the area above the Error vs. Cost curve.

Reward definition



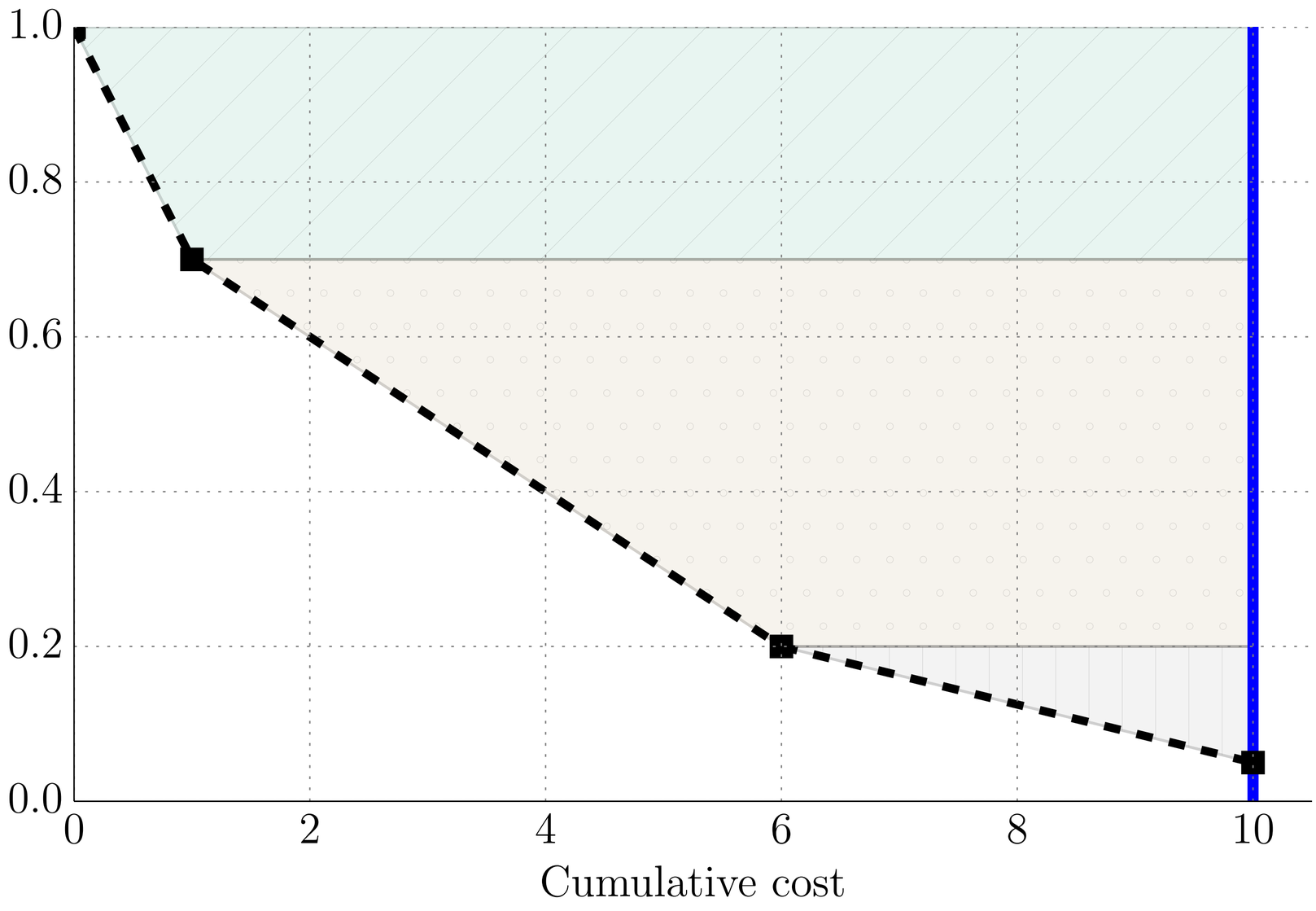
Each action contributes a horizontal "slice" of this area, and that is the intuition behind how we define its reward.

Reward definition

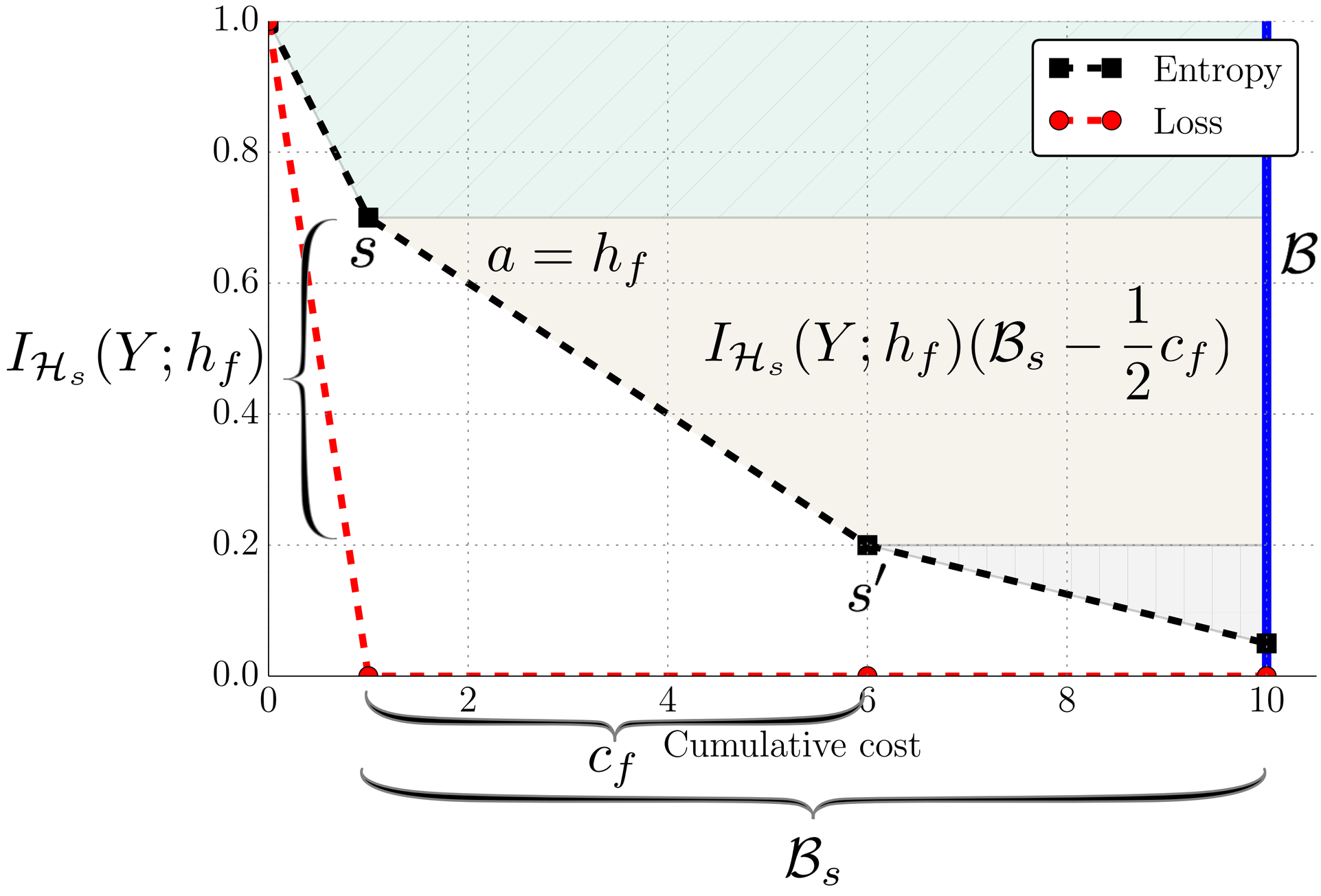


Essentially, the larger the slice, the larger the reward.

Reward definition

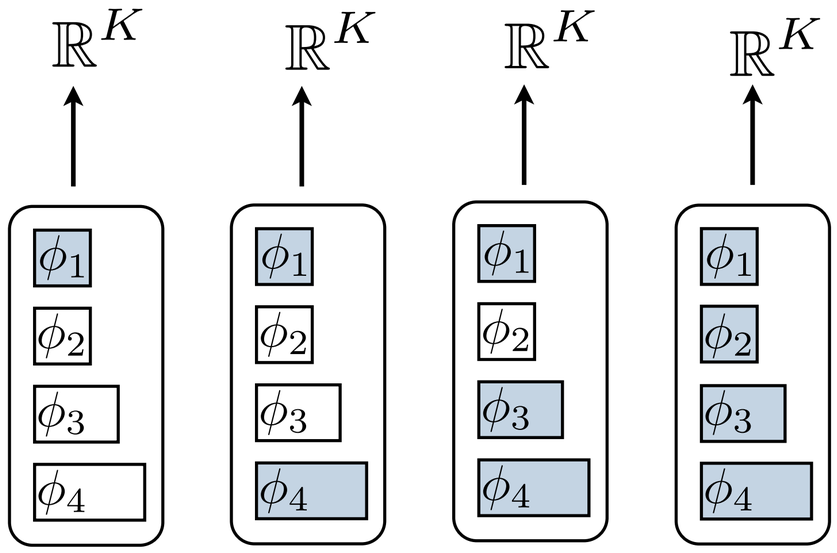


Reward definition

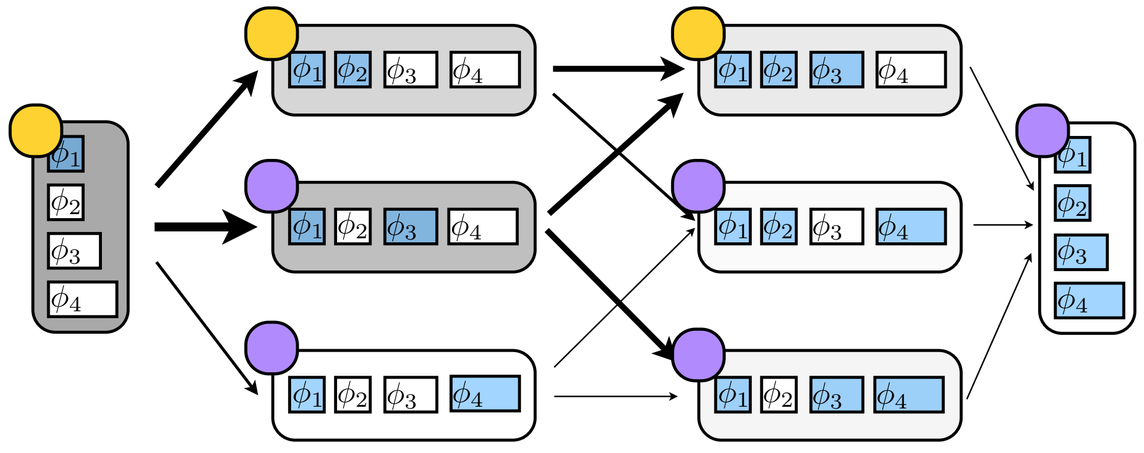


We found that it made for a more stable reward function to use the empirical entropy of the prediction rather than the error.
Reward then corresponds to a function of the empirical information gain.

Training classifier



- We use logistic regression.
- Classifier must be robust to range of possible feature subsets.
  - Different missing-value imputation methods.
  - Mixture of classifiers.



We see that the reward function depends on the performance of the classifier. How does that work?
First of all, because our method is all about conserving computation, we only consider linear classifiers.
The classifier must be robust to a range of possible feature subsets.
We consider several missing value imputation methods, and [click] use a mixture of classifiers so as to cover different feature subsets.
We find that simple mean imputation and single components generally work best.

Final training algorithm

$\pi_0 \leftarrow$ random;
for $i \leftarrow 1$ **to** *max_iterations* **do**
    States, Actions, Costs, Labels $\leftarrow$ `GatherSamples`($\mathcal{D}$, π_{i-1});
    $g_i \leftarrow$ `UpdateClassifier`(*States*, *Labels*);
    Values $\leftarrow$ `ComputeValues`(*States*, *Costs*, *Labels*, $g_i, \mathcal{L}_{\mathcal{B}}, \gamma$);
    $\pi_i \leftarrow$ `UpdatePolicy`(*States*, *Actions*, *Values*);
end

Since the distribution of feature subsets depends on the policy, our final training algorithm has an added classifier update step.

Evaluation

Baselines:

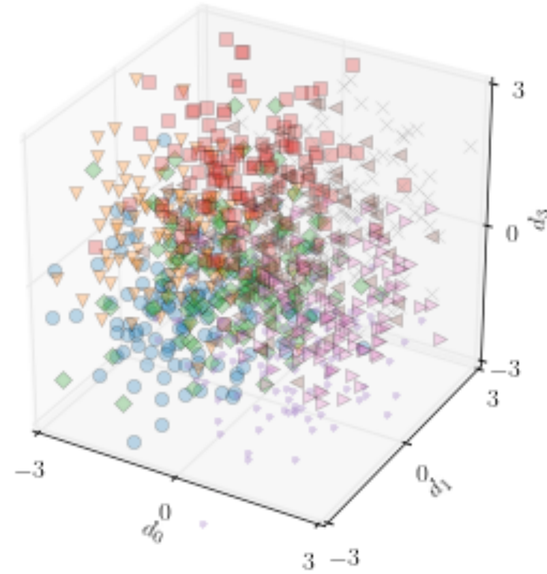
- **Static, greedy**: do not observe feature values, select actions greedily
- **Static, non-myopic**: do not observe feature values, but use MDP with $\gamma = 1$ to plan ahead.
- **Dynamic, greedy**: observe feature values, but select actions greedily.

Our method **Dynamic, non-myopic**: observe feature values, and plan ahead.

Our evaluation works as follows.

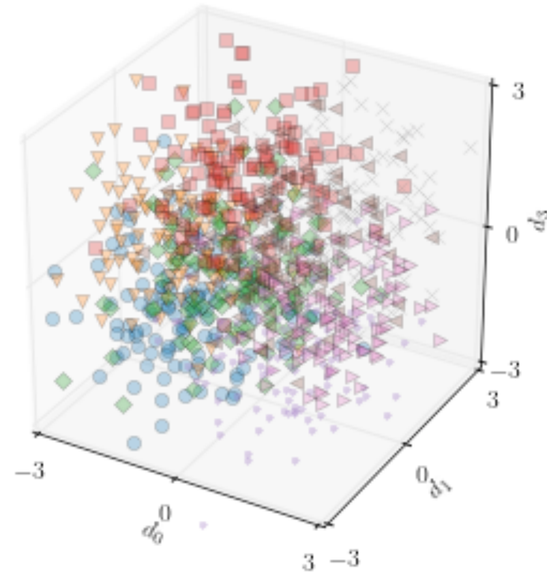
Evaluation: Synthetic Example

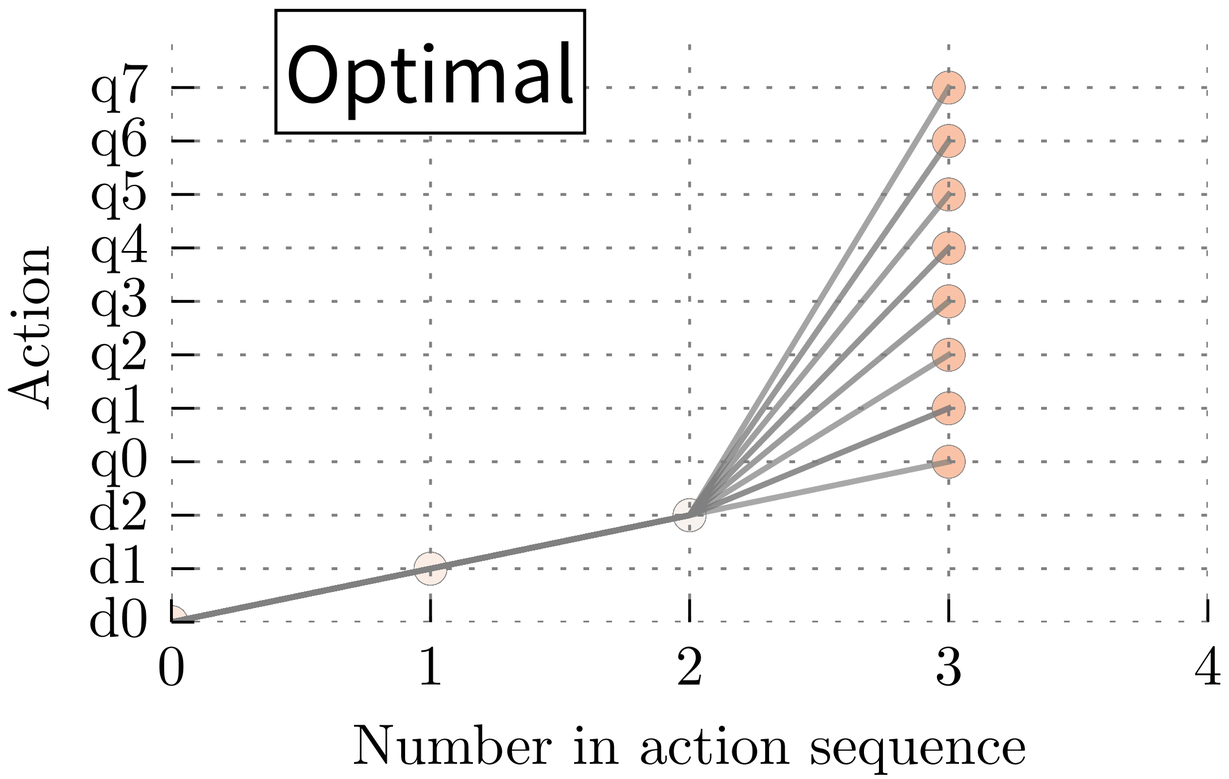
- We construct a task where the optimal policy is known, and has to be dynamic and non-myopic.

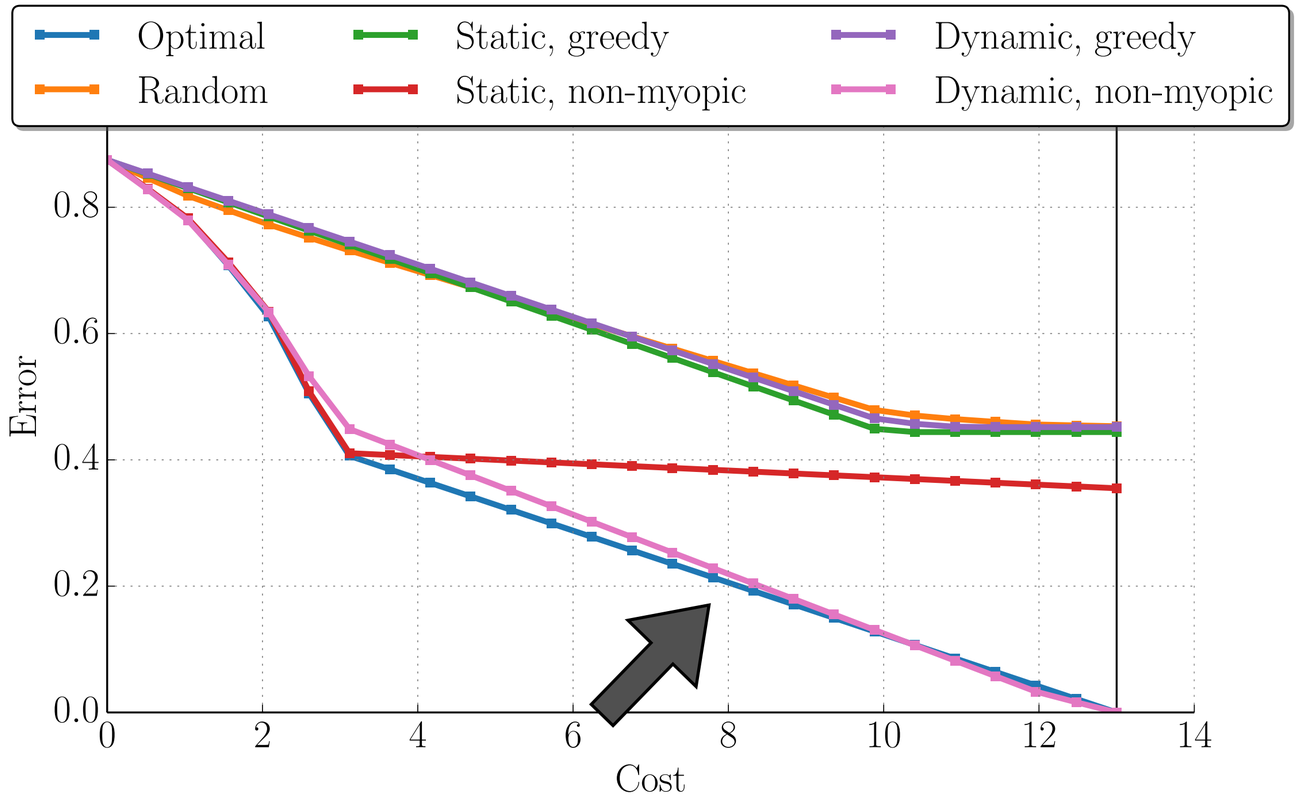


Evaluation: Synthetic Example

- We construct a task where the optimal policy is known, and has to be dynamic and non-myopic.

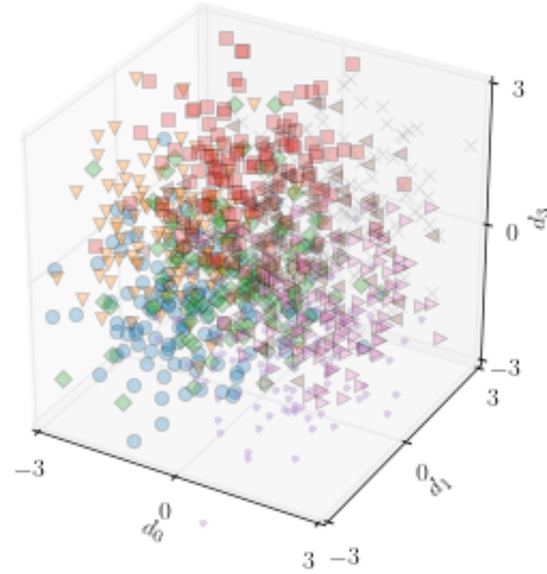


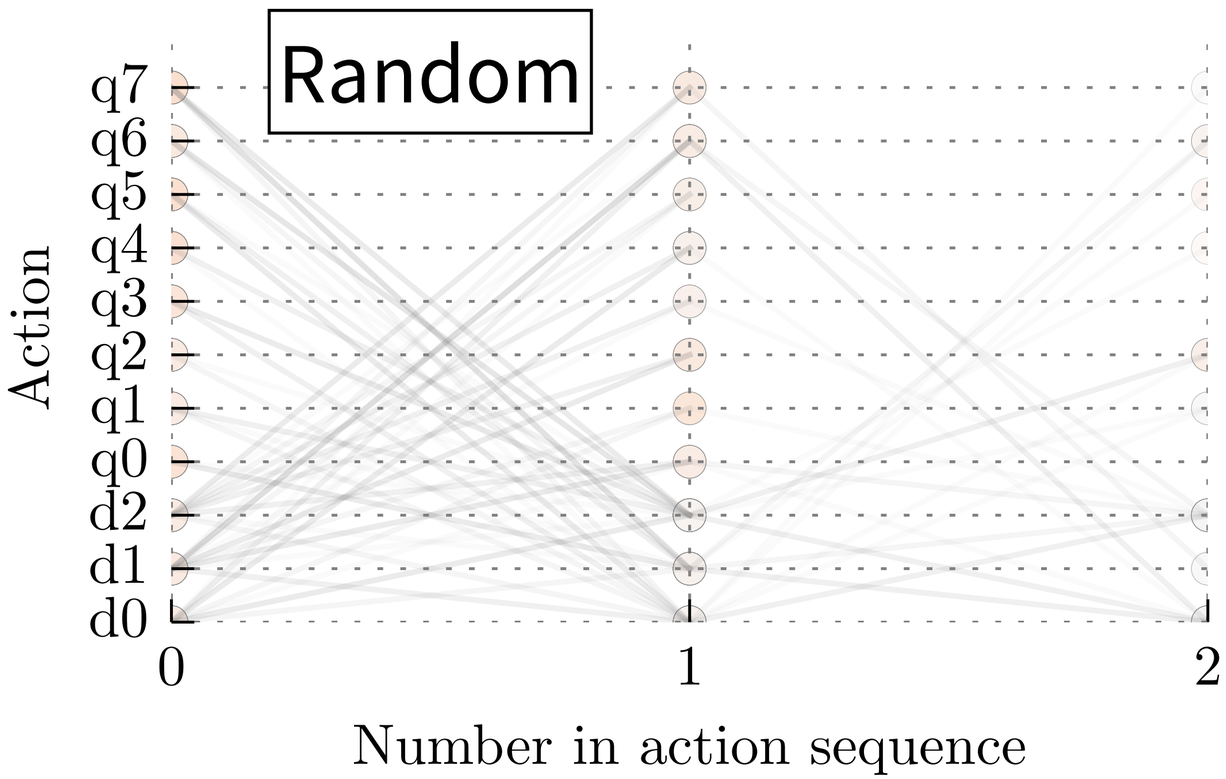


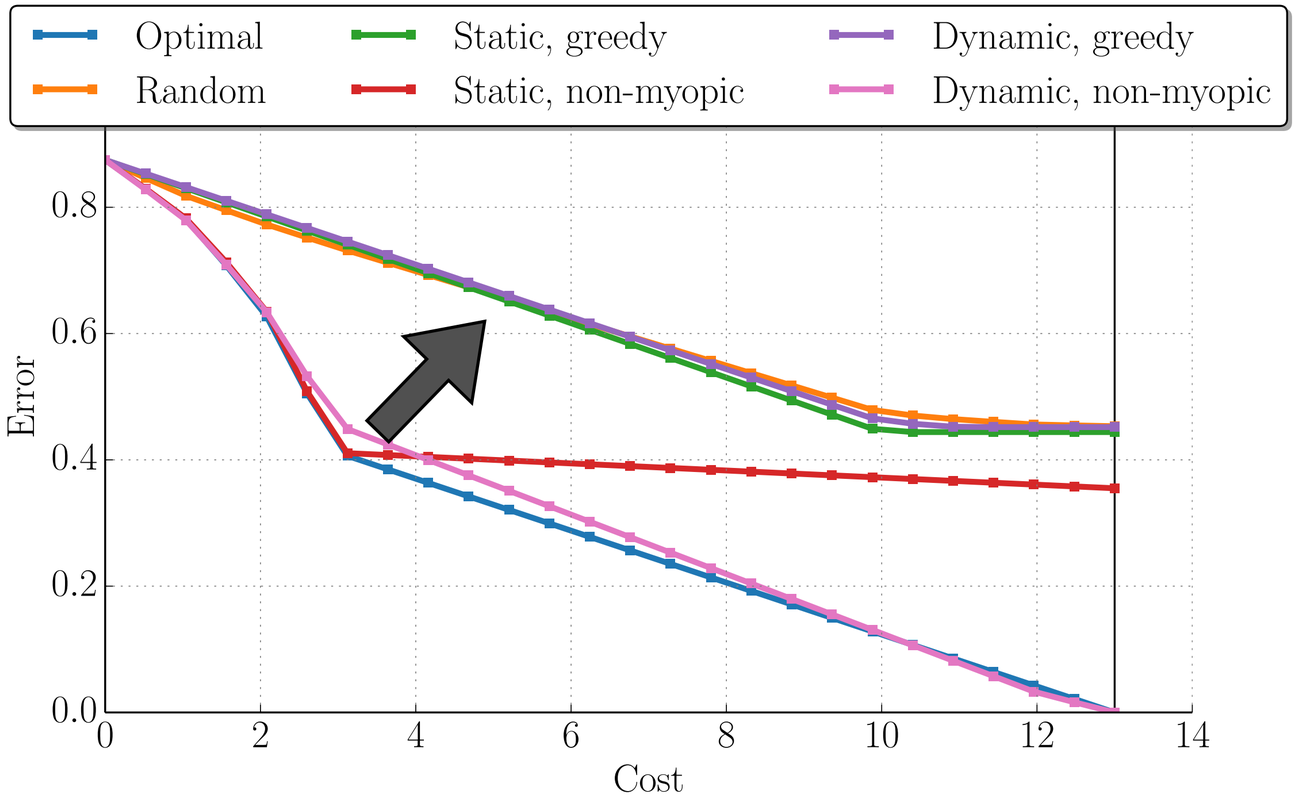


Evaluation: Synthetic Example

- We construct a task where the optimal policy is known, and has to be dynamic and non-myopic.

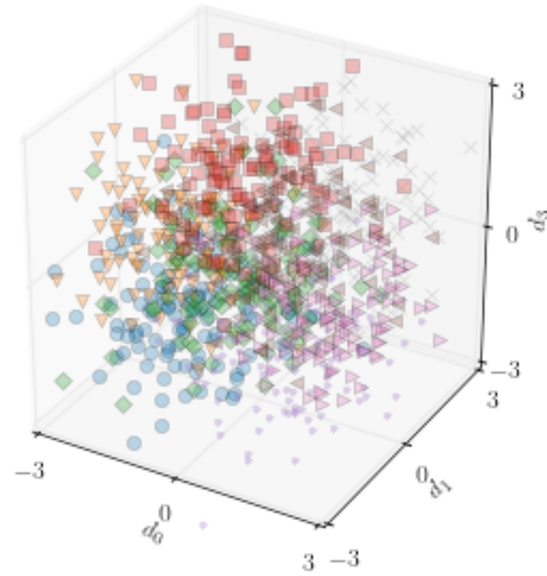


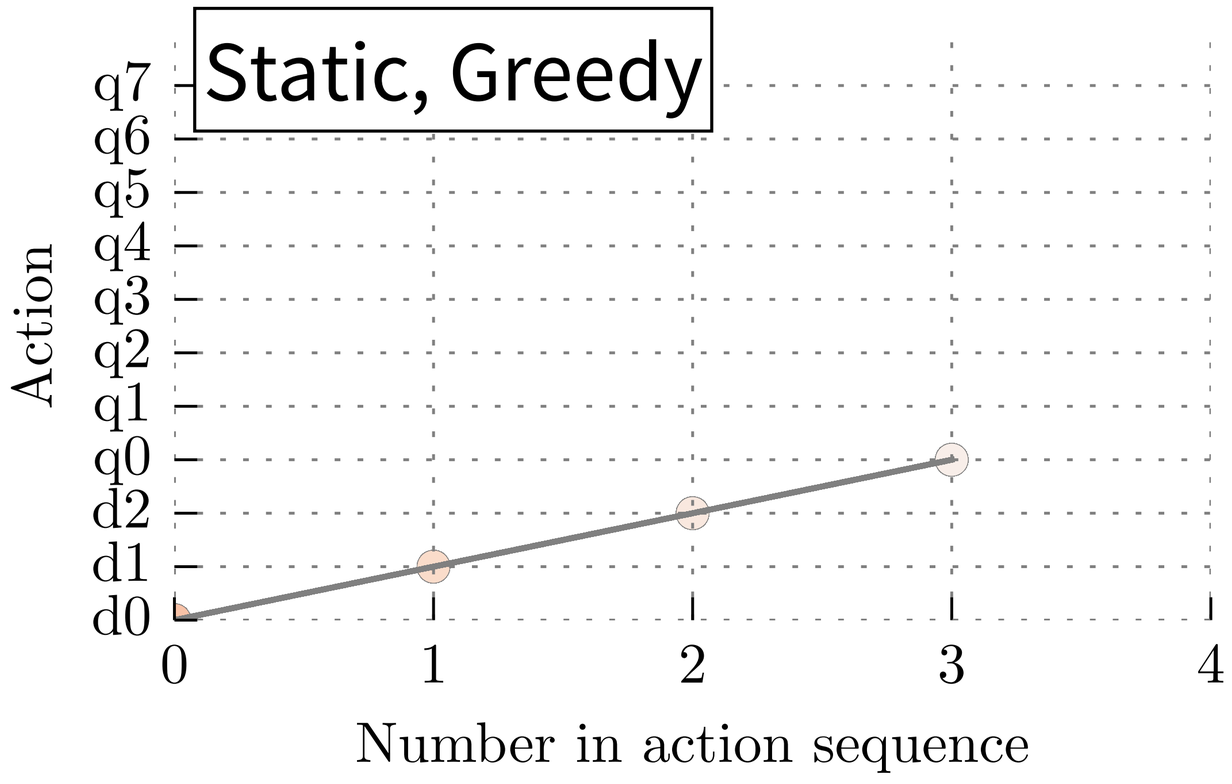


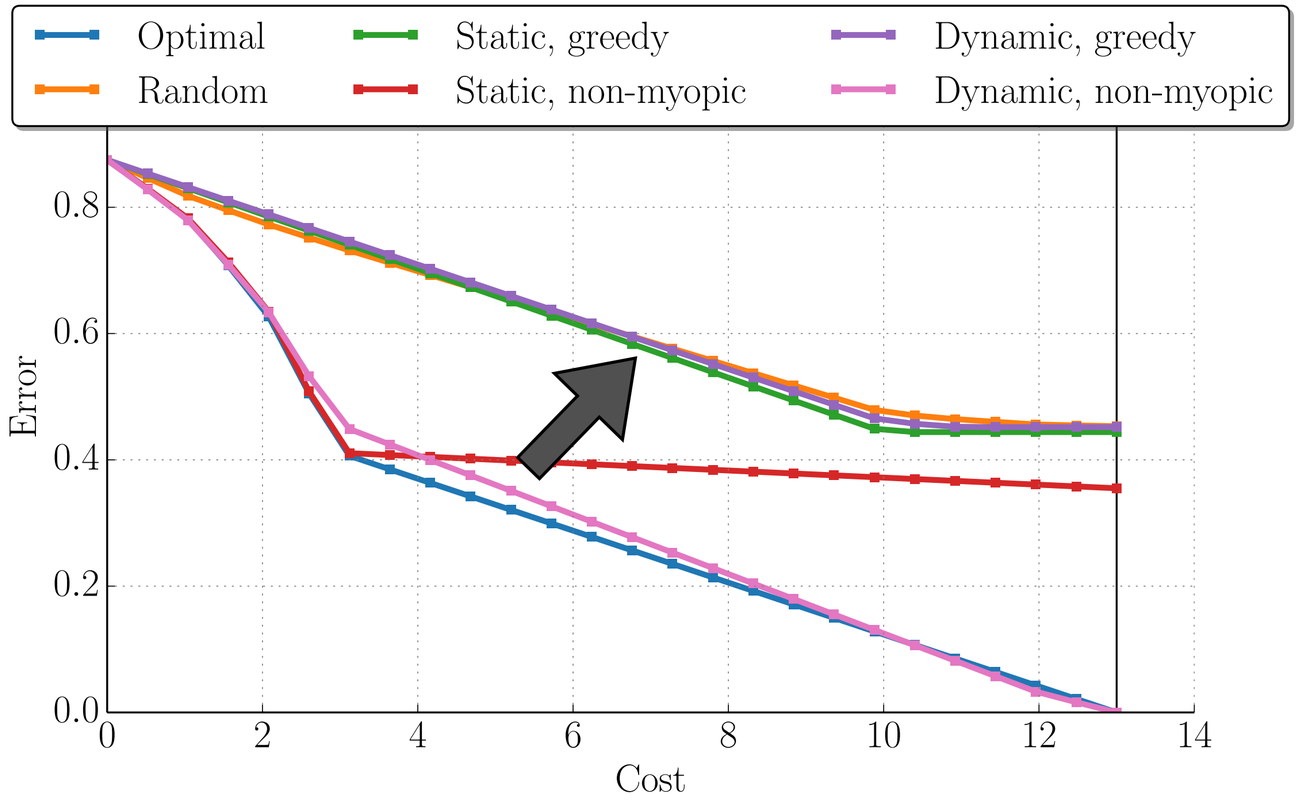


Evaluation: Synthetic Example

- We construct a task where the optimal policy is known, and has to be dynamic and non-myopic.

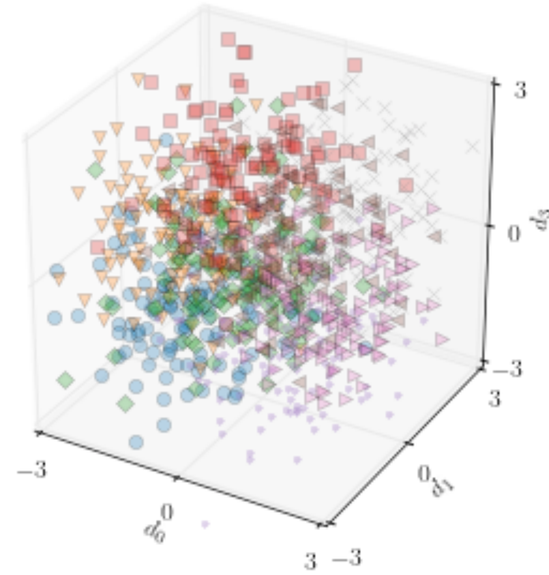


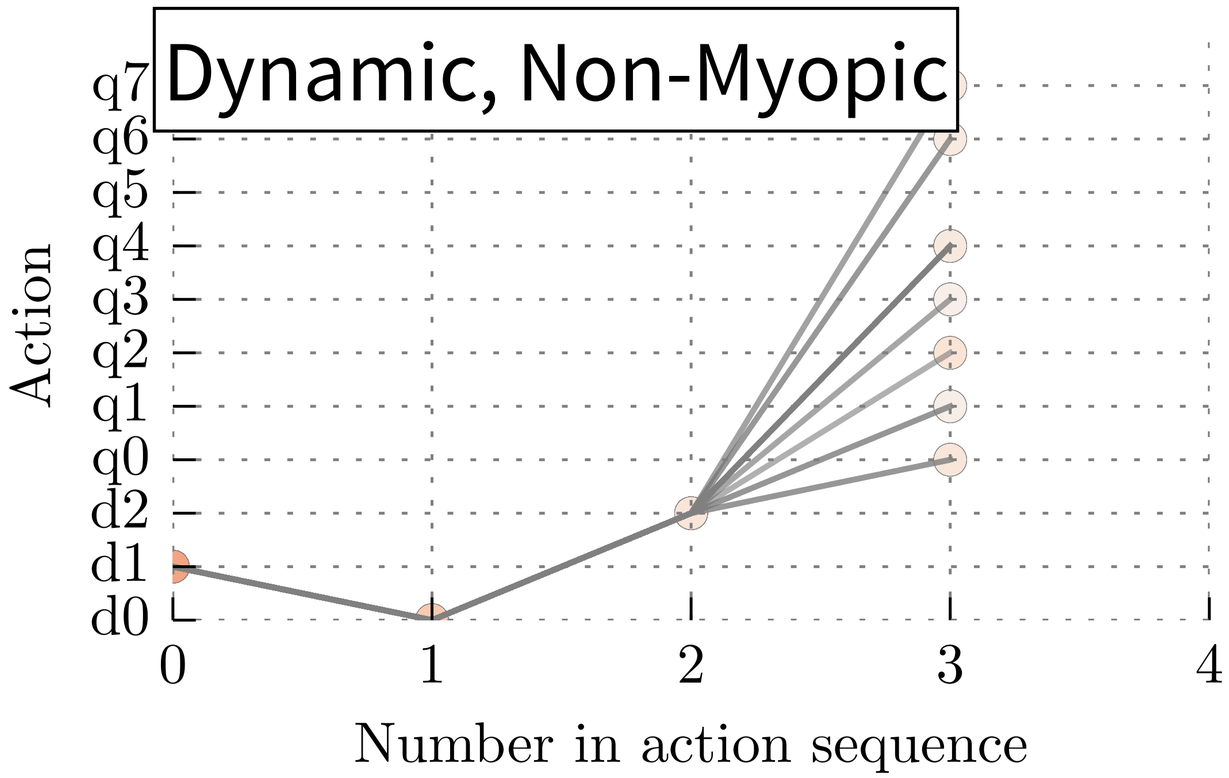


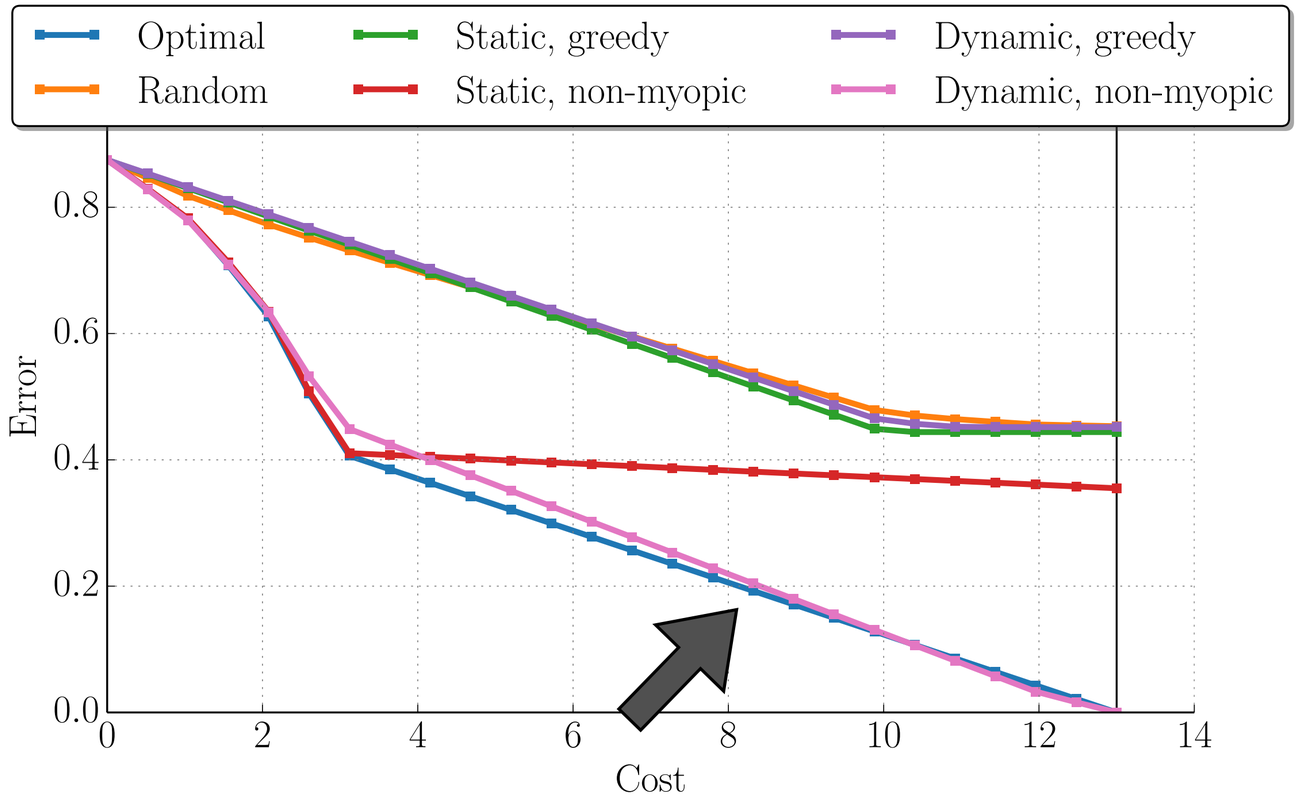


Evaluation: Synthetic Example

- We construct a task where the optimal policy is known, and has to be dynamic and non-myopic.
- Our method successfully recovers it.

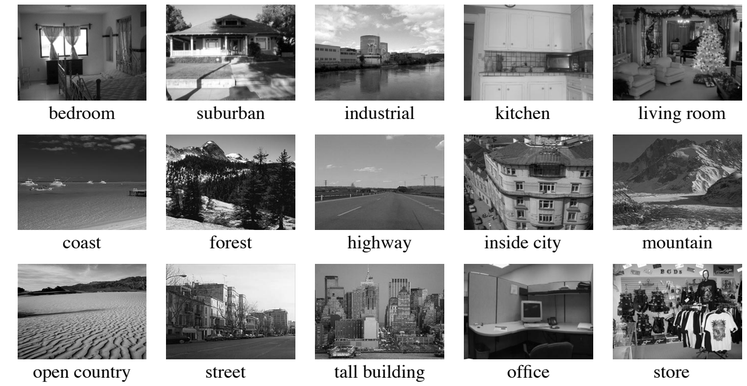


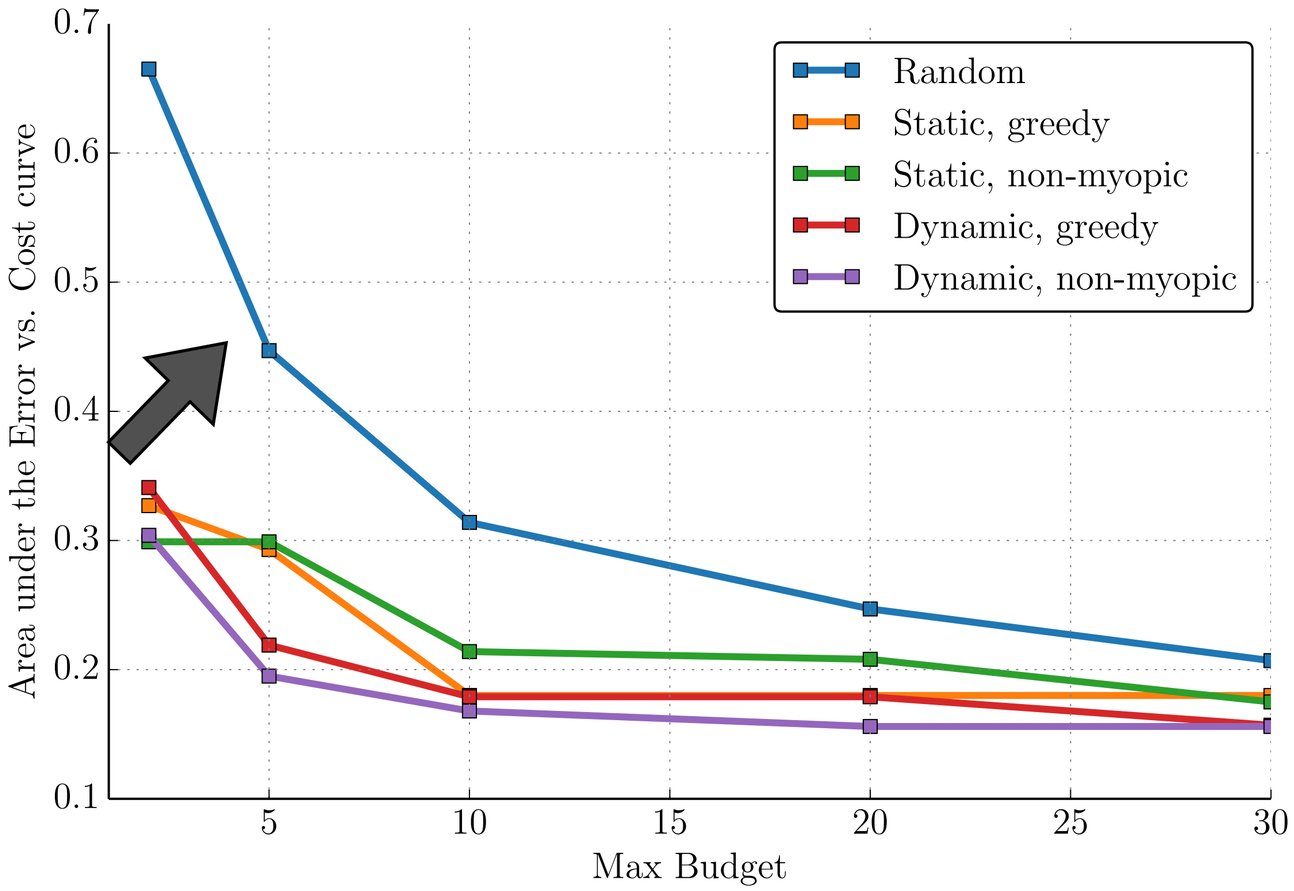




Anytime recognition of scenes

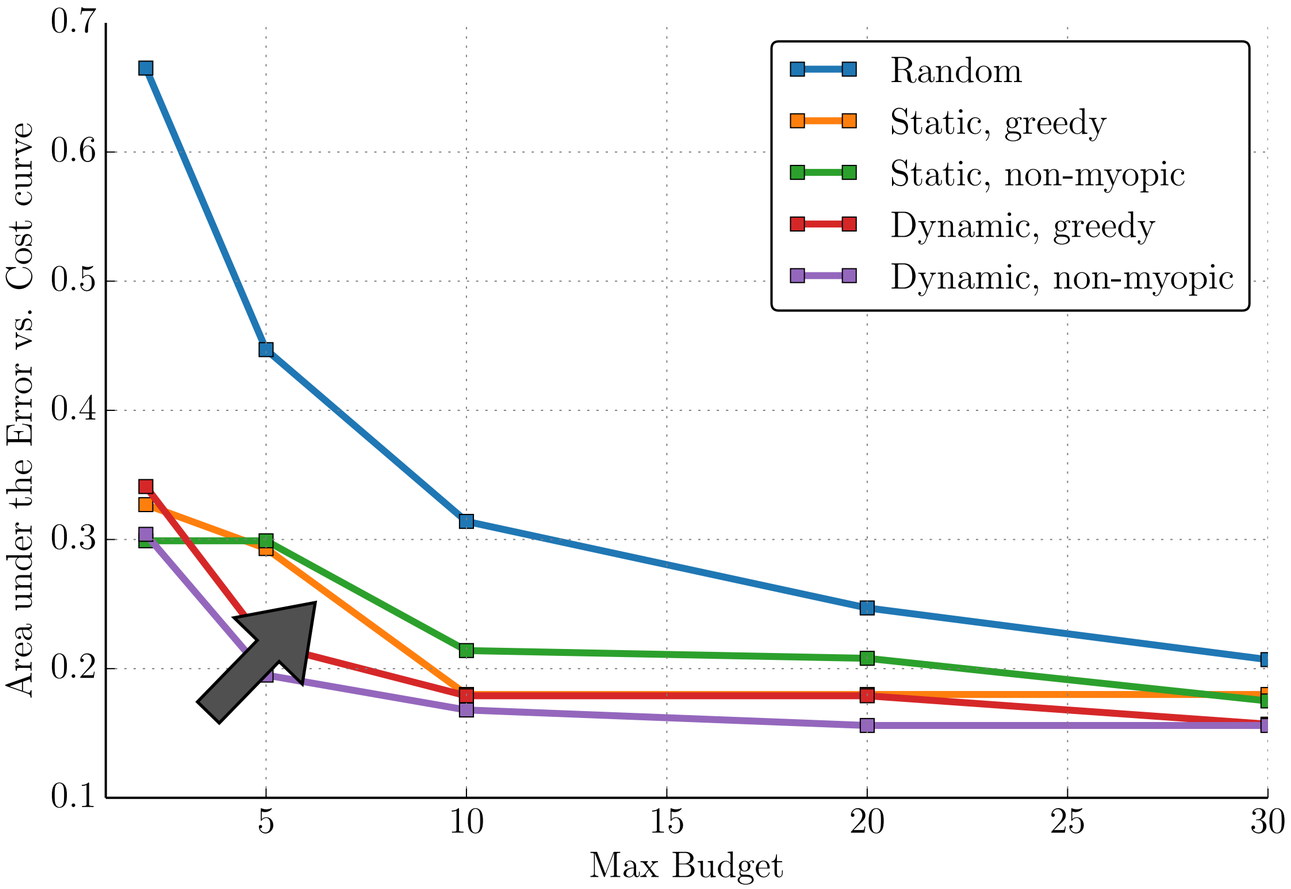
- Scenes-15 dataset.
- Actions are feature computations.

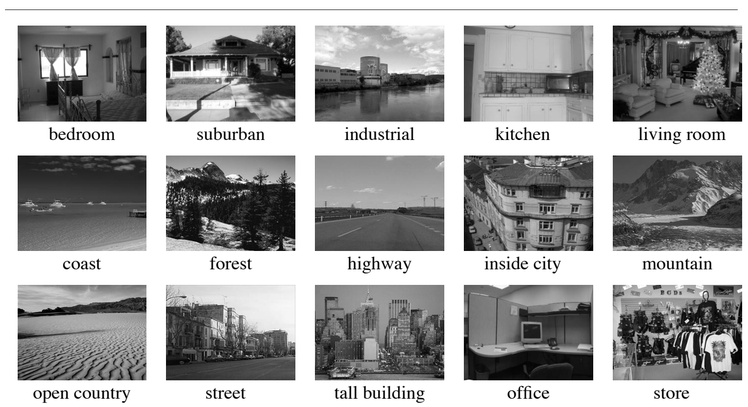


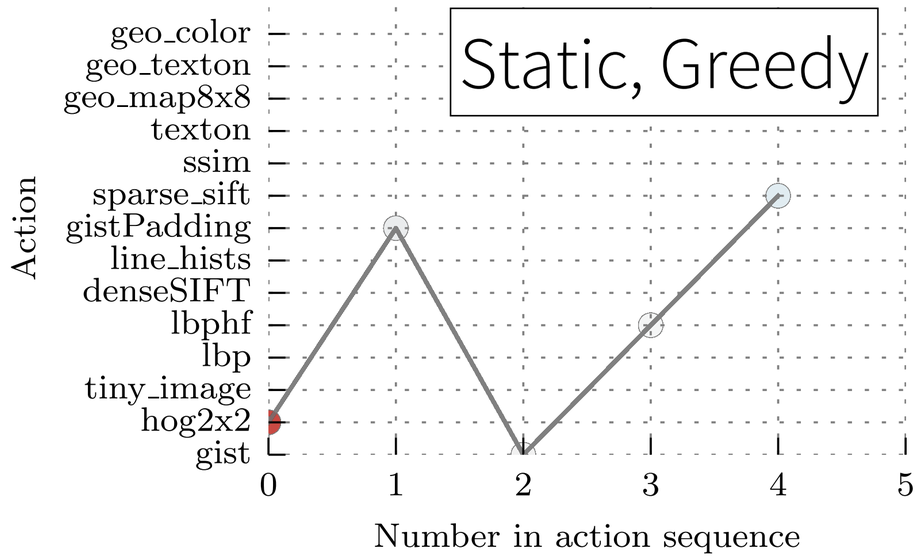


Anytime recognition of scenes

- Scenes-15 dataset.
- Actions are feature computations.

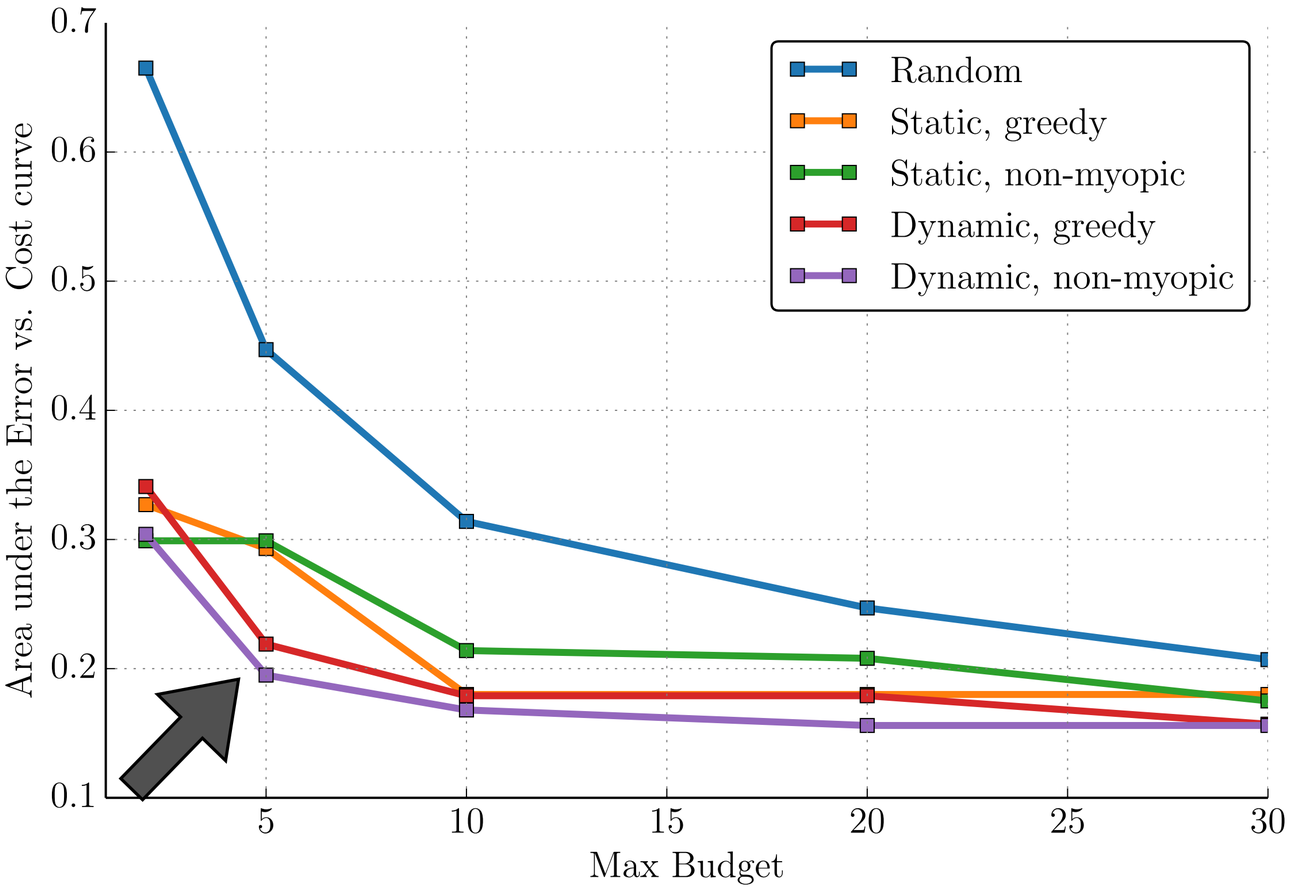


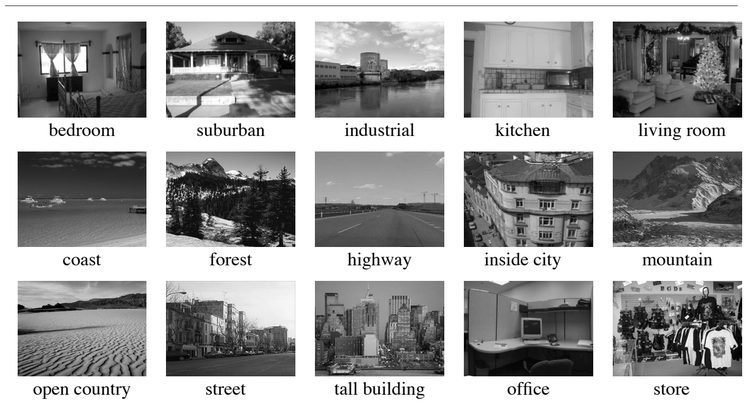


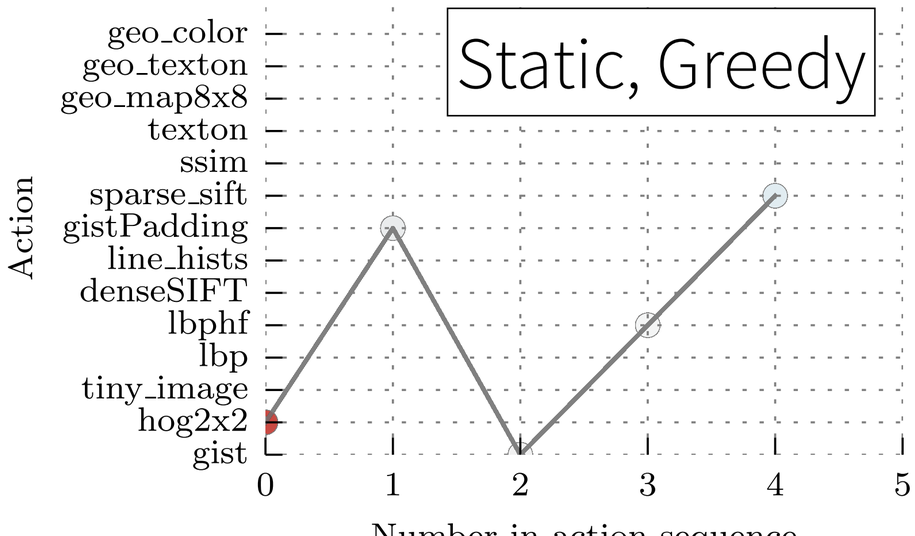


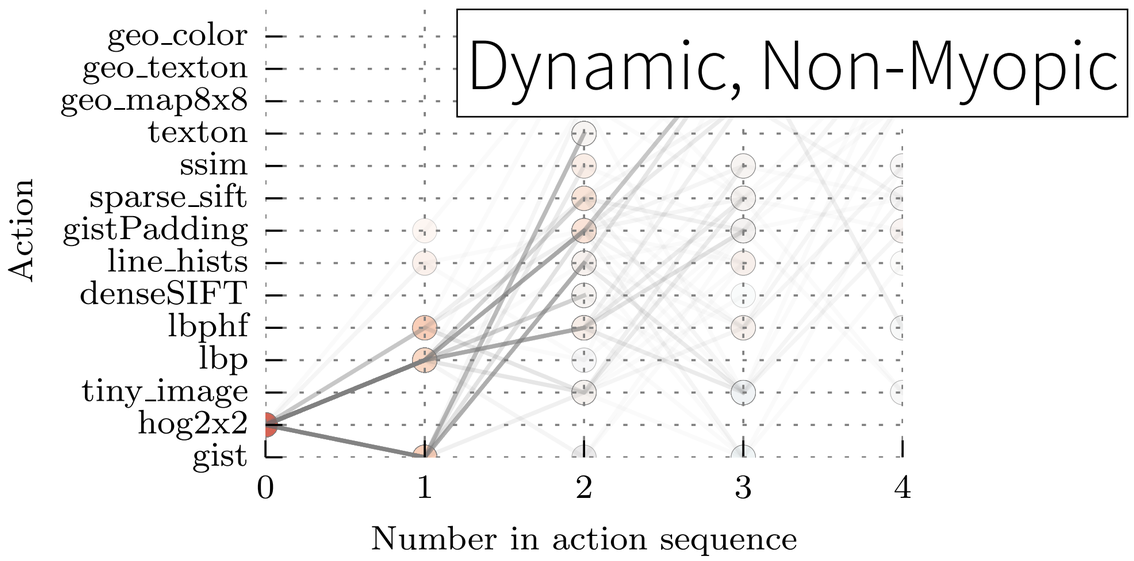
Anytime recognition of scenes

- Scenes-15 dataset.
- Actions are feature computations.

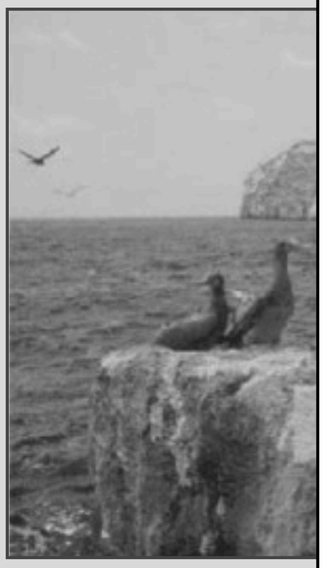


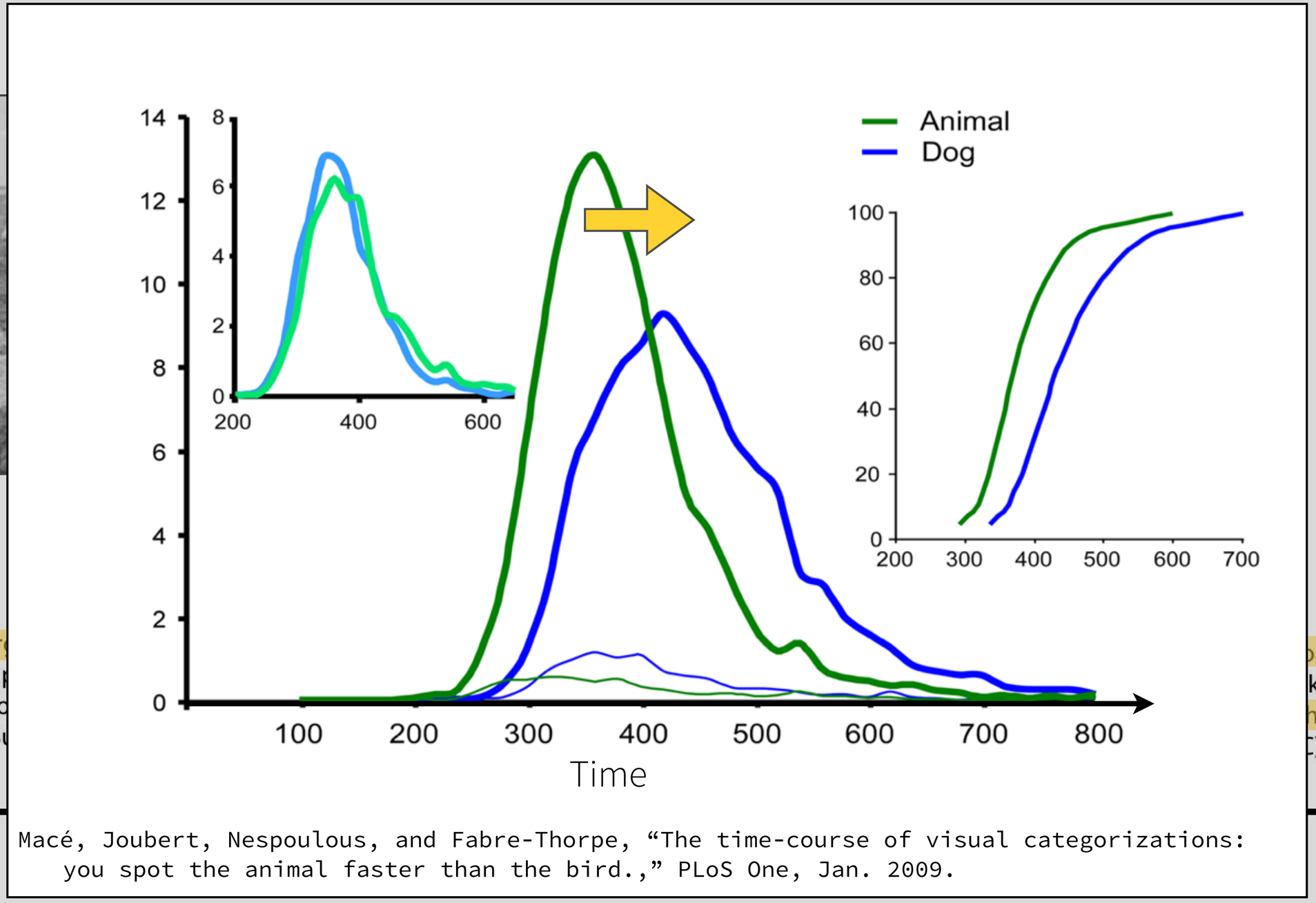






Human perception is anytime, progressive





PT
40 ms

I saw a very br
pyramidal shap
black in the fro
what it was. (Su

past byt hte
k in the
hree bird sitting
C)

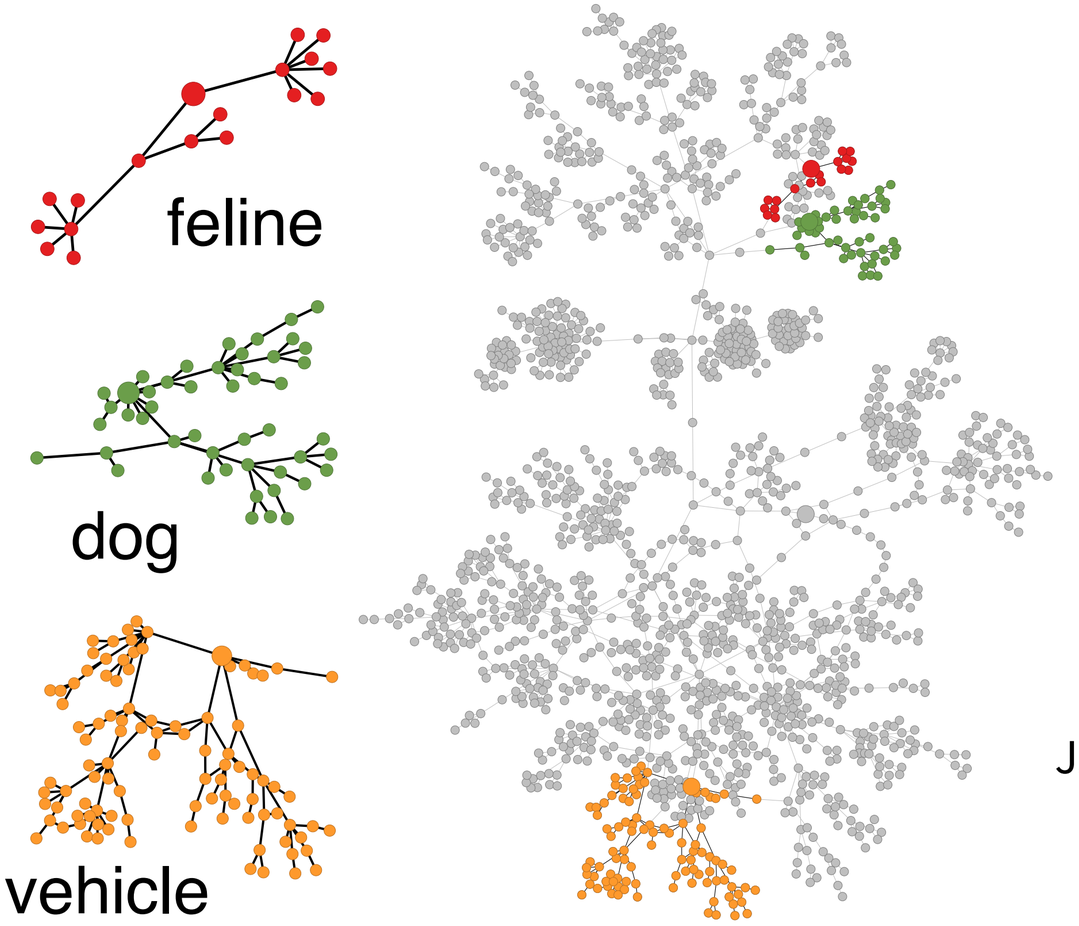
Time

Macé, Joubert, Nespoulous, and Fabre-Thorpe, “The time-course of visual categorizations: you spot the animal faster than the bird.,” PLoS One, Jan. 2009.

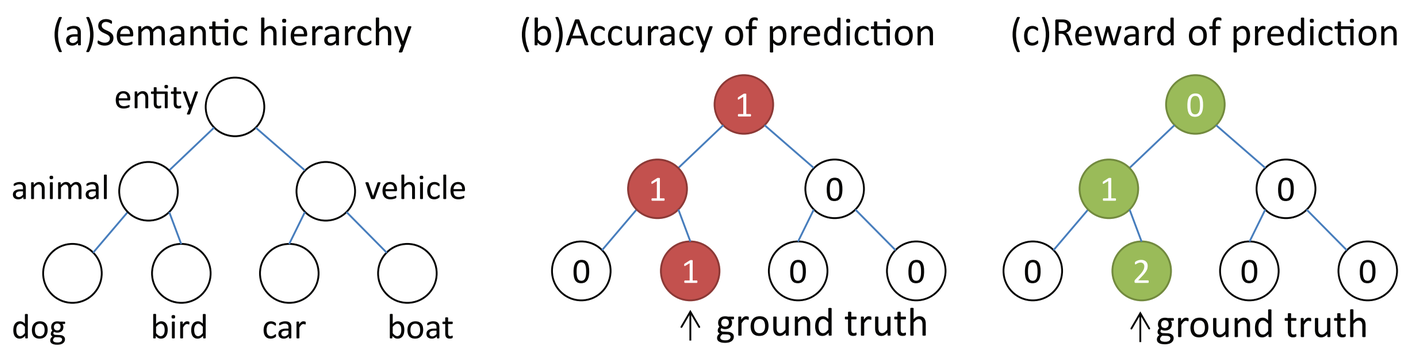
Fei-Fei, Iyer, Koch, and Perona, “What do we perceive in a glance of a real-world scene?,” J. Vis., Jan. 2007.

Recall our initial motivation of progressively specific visual perception.

Anytime recognition of objects



IMAGENET

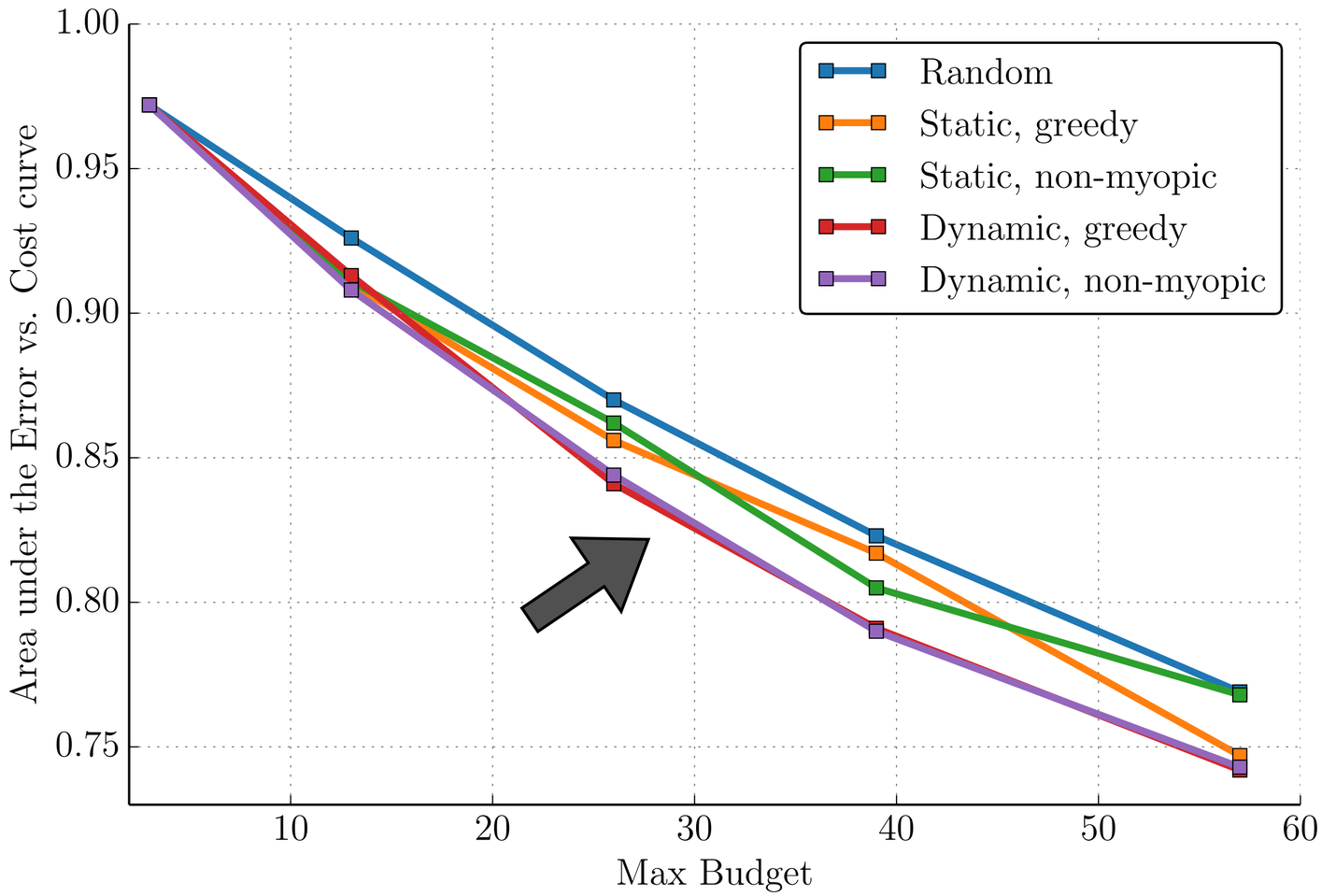


J. Deng, J. Krause, A. C. Berg, and L. Fei-fei, “Hedging Your Bets: Optimizing Accuracy-Specificity Trade-offs in Large Scale Visual Recognition,” in CVPR, 2012.

The ImageNet dataset provides a full hierarchy of classes, and lets us leverage prior work in trading off accuracy and specificity.

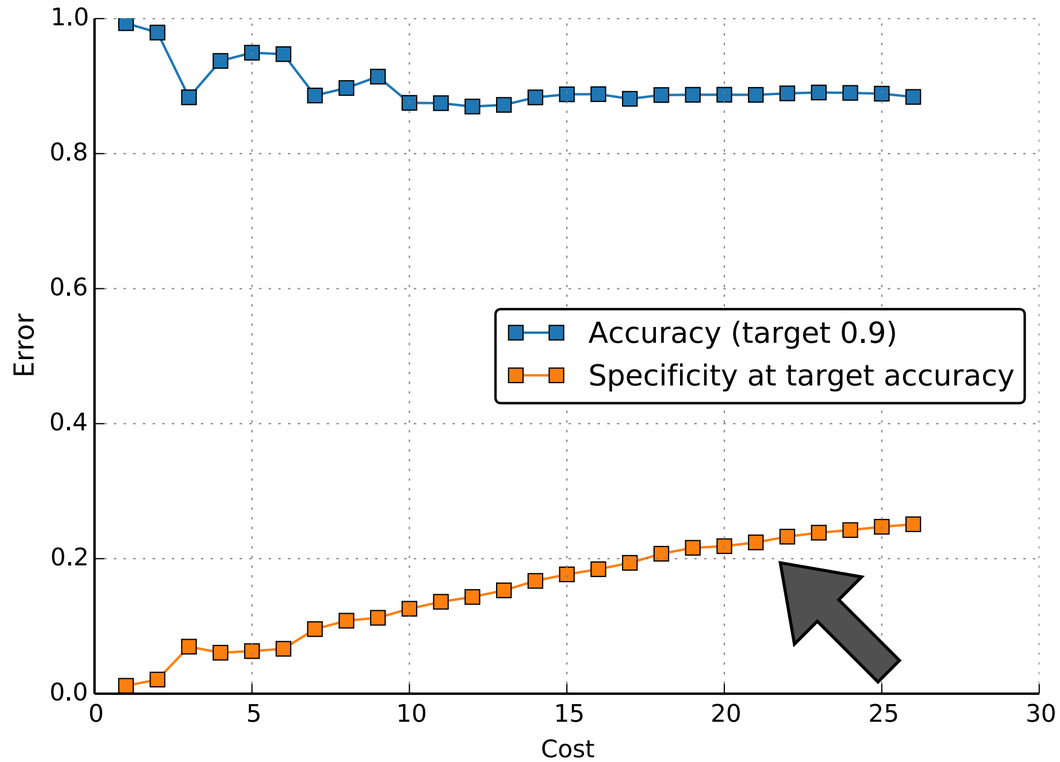
Anytime recognition of objects

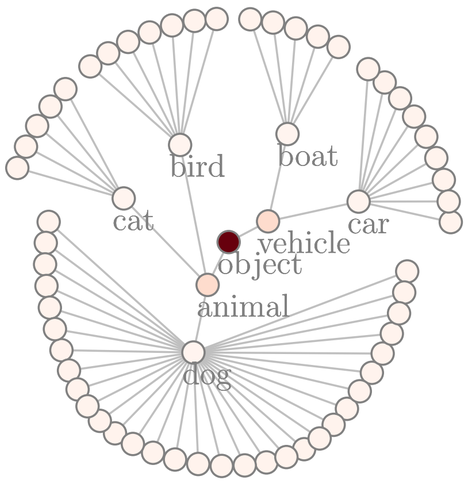
- Actions are class evaluations.



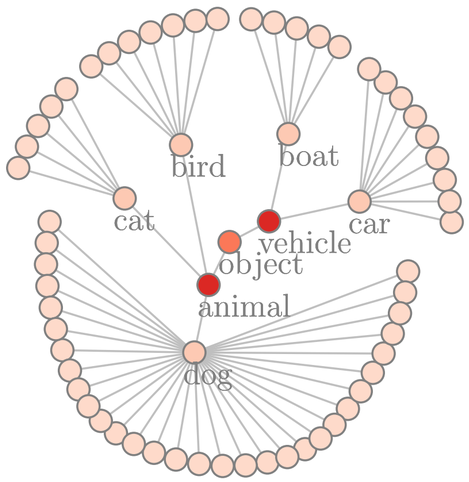
The actions on this task are class evaluations.
Our dynamic, non-myopic method also outperforms the baselines on this task.

Anytime recognition of objects with increasing specificity

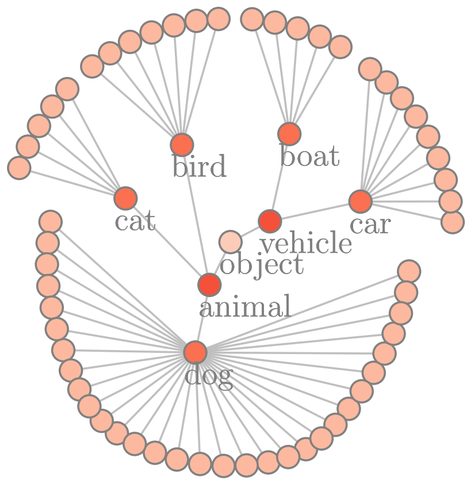




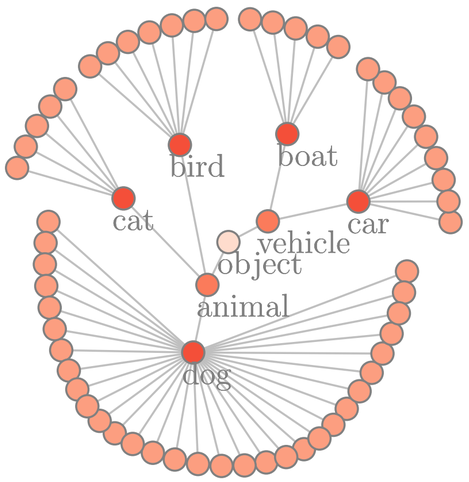
Cost: 1



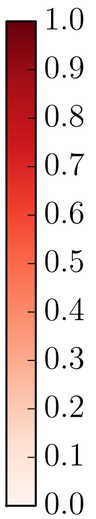
Cost: 9



Cost: 17



Cost: 26



Cost

Conclusions

- Method for dynamic, non-myopic selection of recognition actions for Anytime performance.
- Formulation accommodates many types of recognition actions.
- Avenue for further study of time-course of perception.
- Code and more details available at

http://sergeykarayev.com/recognition-on-a-budget/