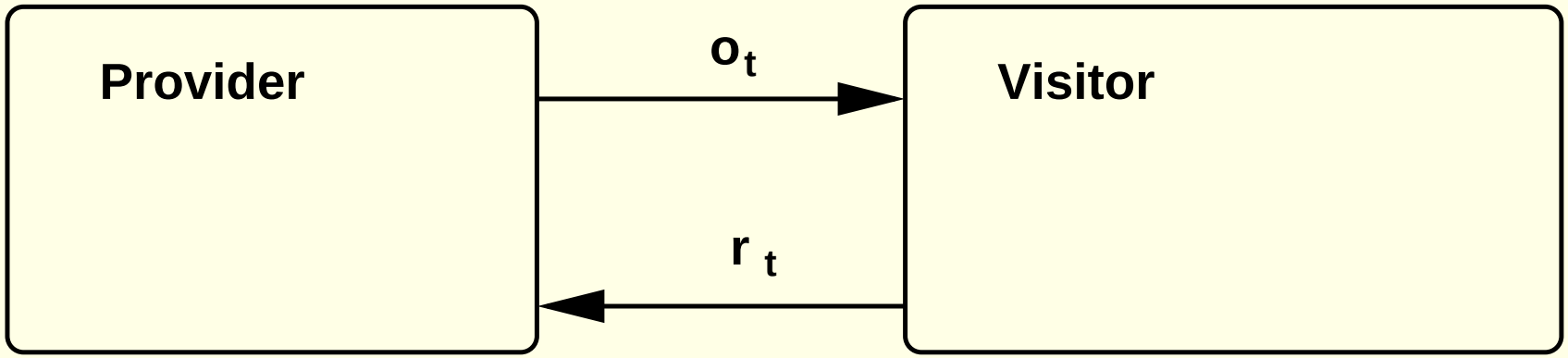
Exploration vs. Exploitation Challenge Framework

Peter Auer

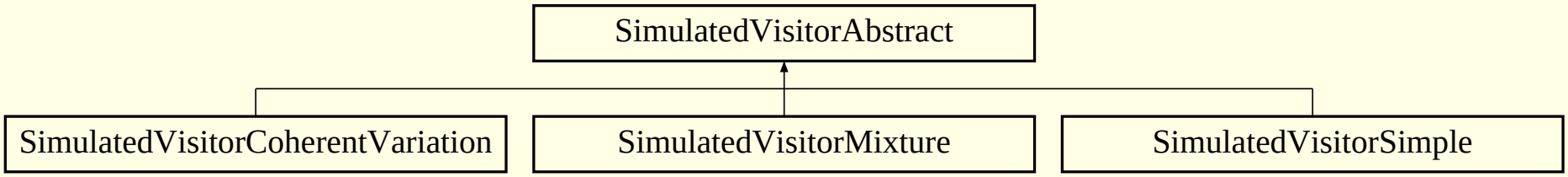
CiT, University of Leoben

Overview



- A visitor represents a bandit problem.
- A provider realizes a bandit algorithm.
- An option o presented by the provider to the visitor represents an arm of the bandit problem.
- In contrast to the bandit problem, the success probability of an option varies over time: $\mathbf{P}\{r_t = \texttt{true} | o_t\} = p_t(o_t)$.

Simulated Visitors



- Main public method:

  `boolean selectOption( int optionNo )`

- The number `optionNo` of the selected option is passed to the visitor, and the visitor returns success (`true`) or failure (`false`), based on the current success probability of the option and the private coin toss of the visitor.

Simulated Visitors

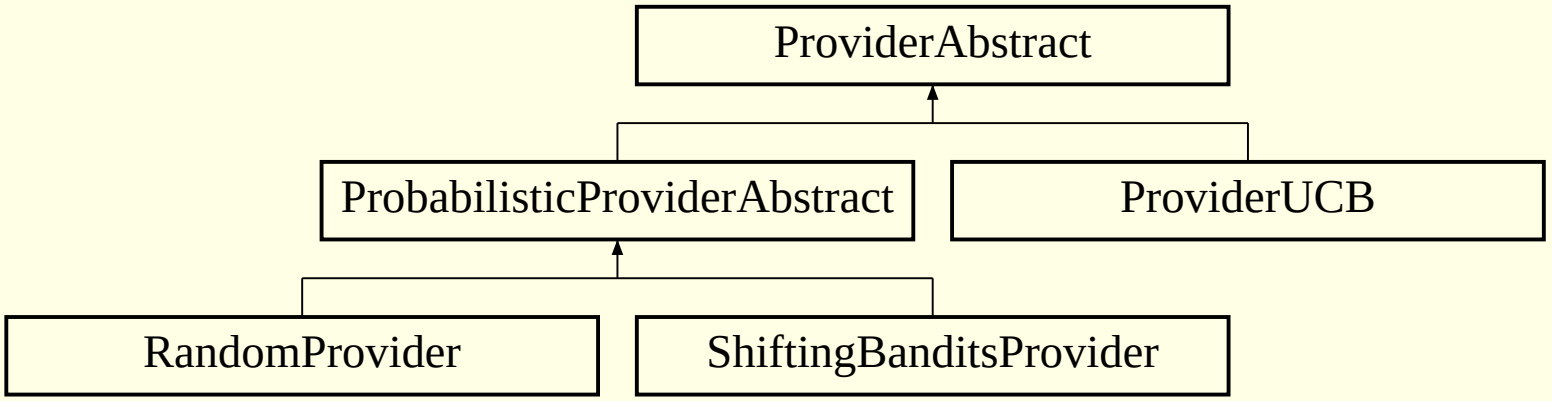
- Additional public methods:
  - `int getNumOptions()`

    Returns the number of available options, $\{0, \ldots,$ `getNumOptions()` $- 1\}$.
  - `float getRegret()`

    Actual regret as compared to the sum of success probabilities of the best option in each trial:

$$\sum_t \max_o p_t(o) - \#\{t : r_t = \texttt{true}\}.$$

Baseline Solutions



- A provider presents options to a visitor.
- The base classes and exemplary learners have been implemented.
- Public constructor:

```
ProviderAbstract(
    SimulatedVisitorAbstract visitor)
```

ProviderAbstract Interface

- `void satisfy()`

  Asks the provider to present an option `o=getCurrentChoice()` to the visitor.

  The response `r` of the visitor is received and the state of the provider is updated by `updateState(o,r)`.

- `abstract int getCurrentChoice()`

  `abstract void updateState( int o, boolean r )`

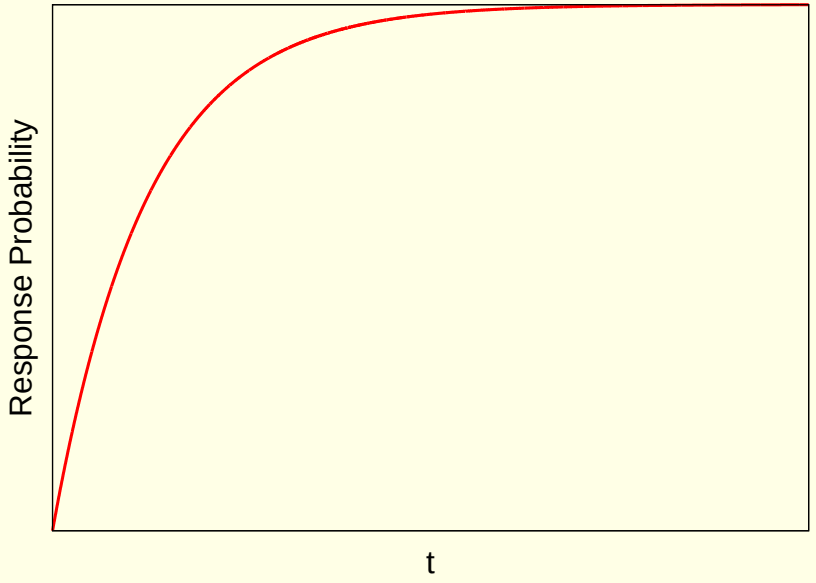
  have to be implemented by the derived classes.

Derived Provider classes

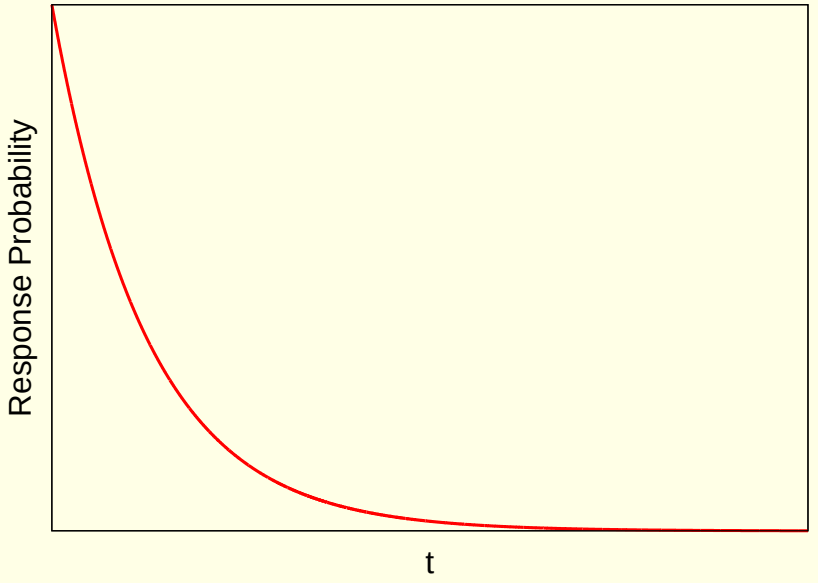
- `ProviderUCB`:
  Deterministic algorithm for random bandit problems.
- `ProbabilisticProviderAbstract`:
  Base class for learners which provide a probability distribution over the options.
  It implements `getCurrentChoice()` by choosing according to this probability distribution.
  Derived classes have to generate this probability distribution in `updateState()`.
- `RandomProvider`:
  Chooses uniformly among the options.
- `ShiftingBanditsProvider`:
  Calculates the probability distribution according to the worst case bandit algorithm with shifting bandits.

Currently implemented options

AsymptoticGrowthOption



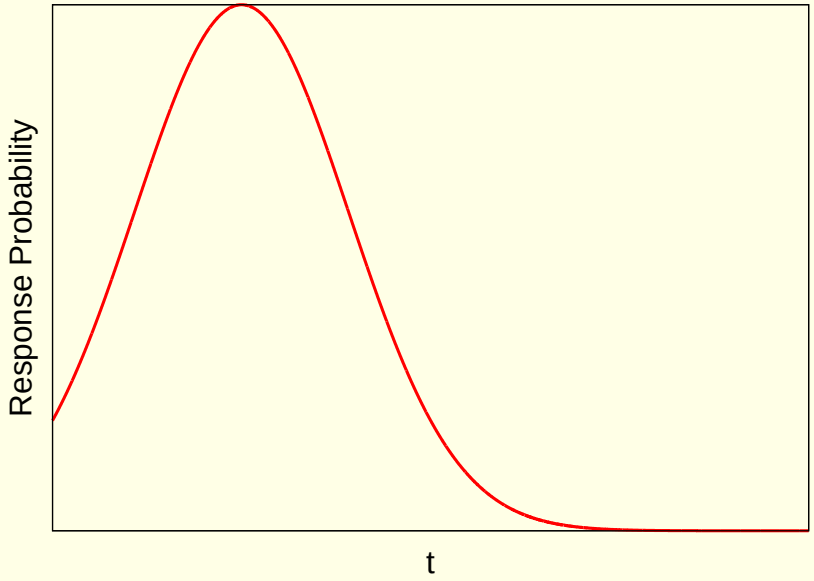
ExponentialDecayOption



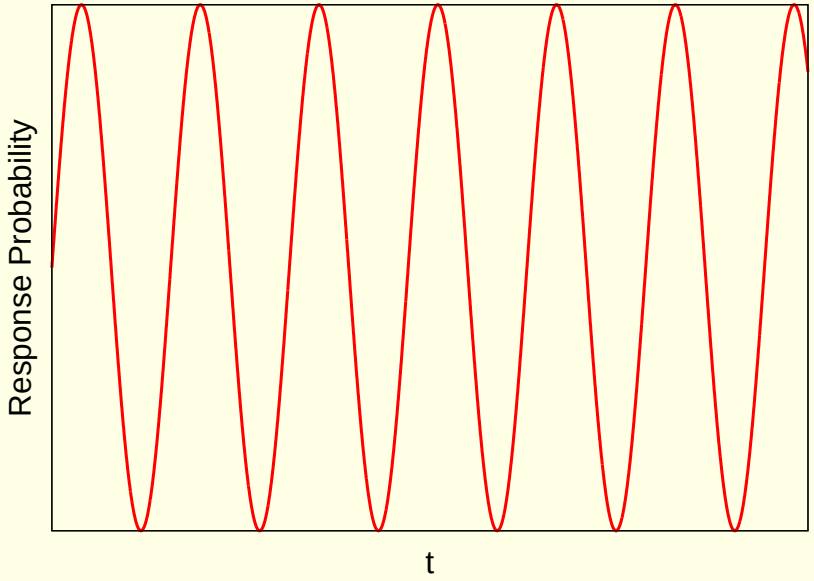
Parameters:
time constant, maximum probability

Currently implemented options

GaussianOption



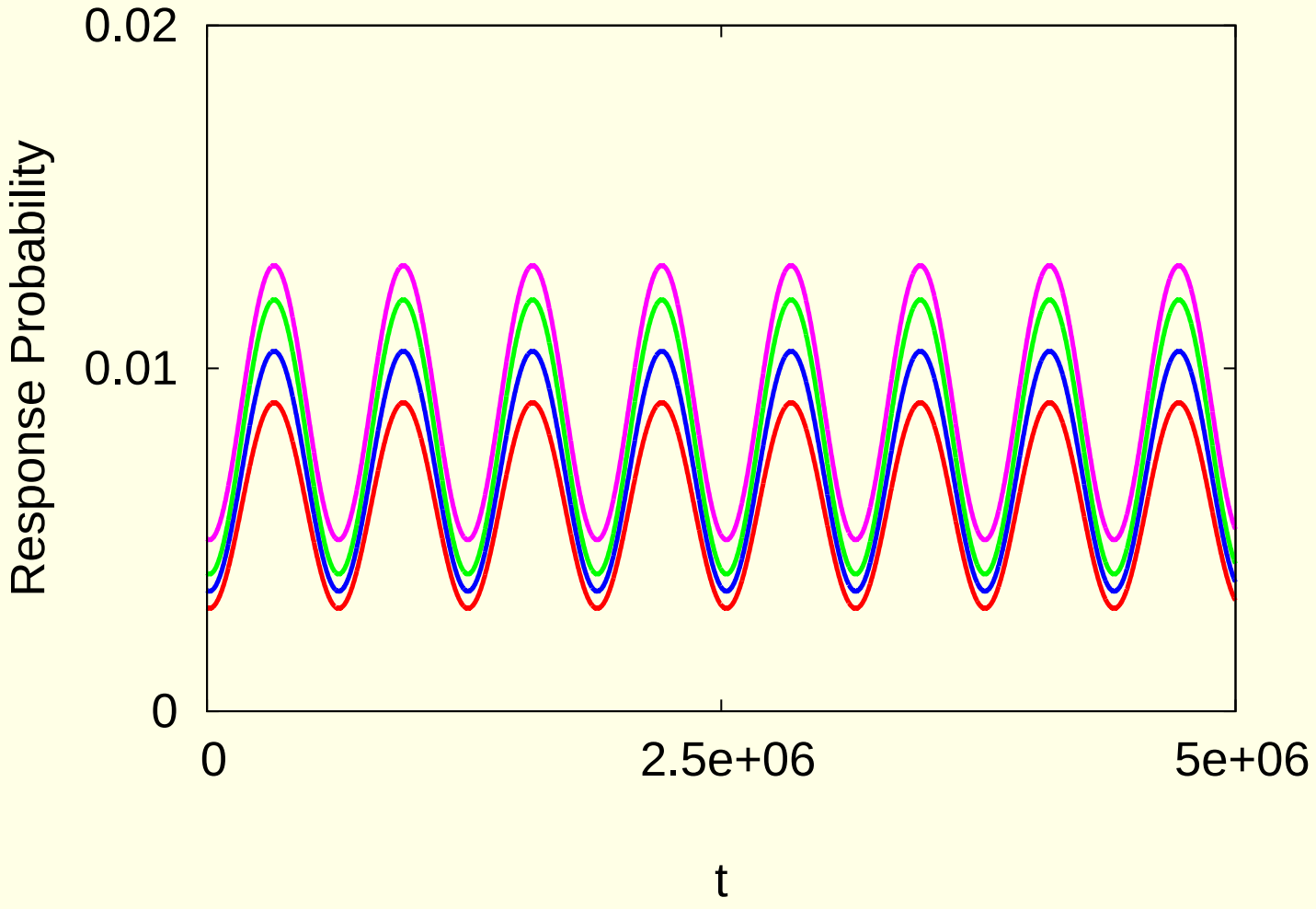
SinusoidalOption



Parameters:
mean, variance, maximum probability
phase offset, frequency, amplitude, amplitude offset

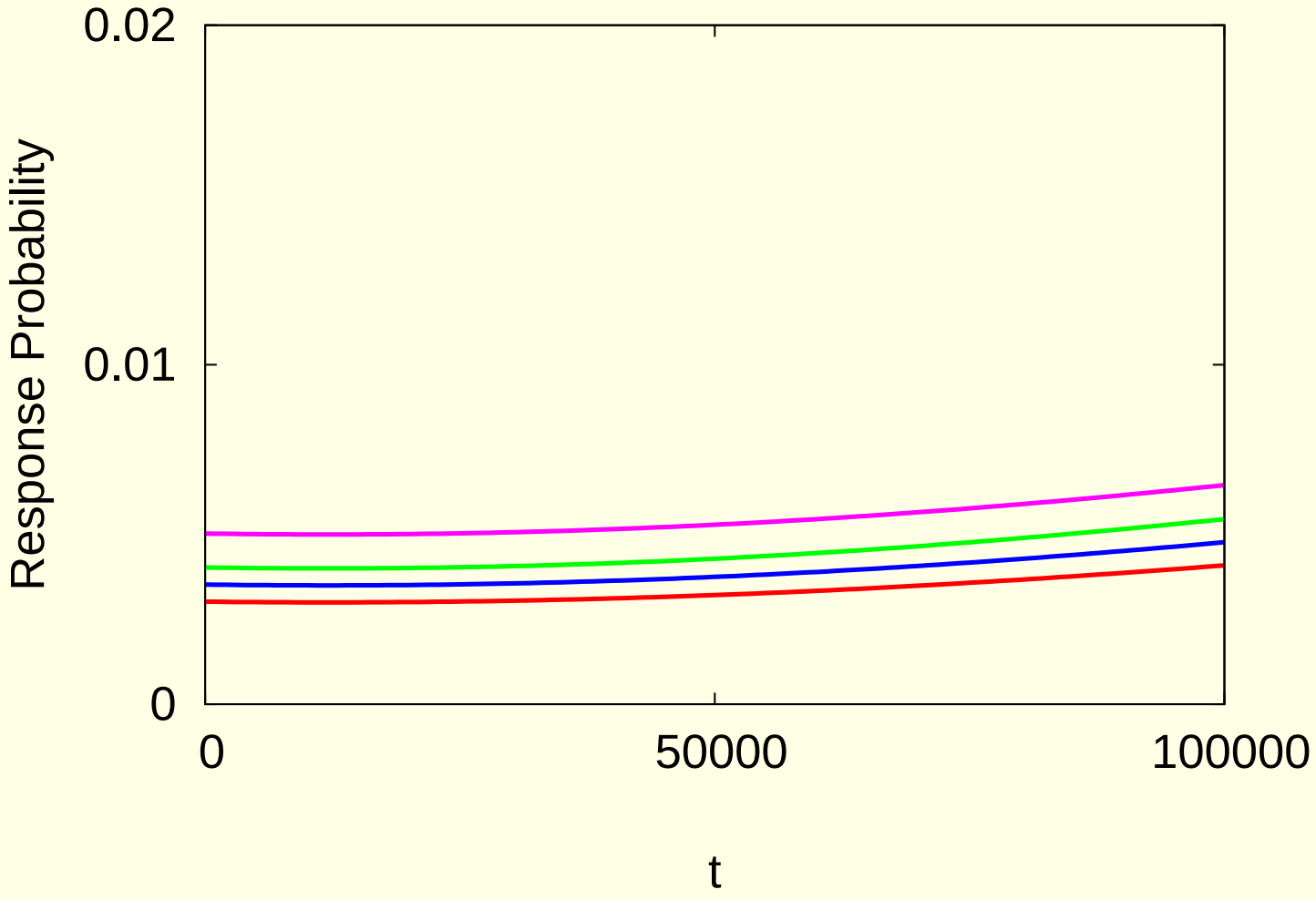
Visitors implemented for the challenge

SimulatedVisitorCoherentVariation



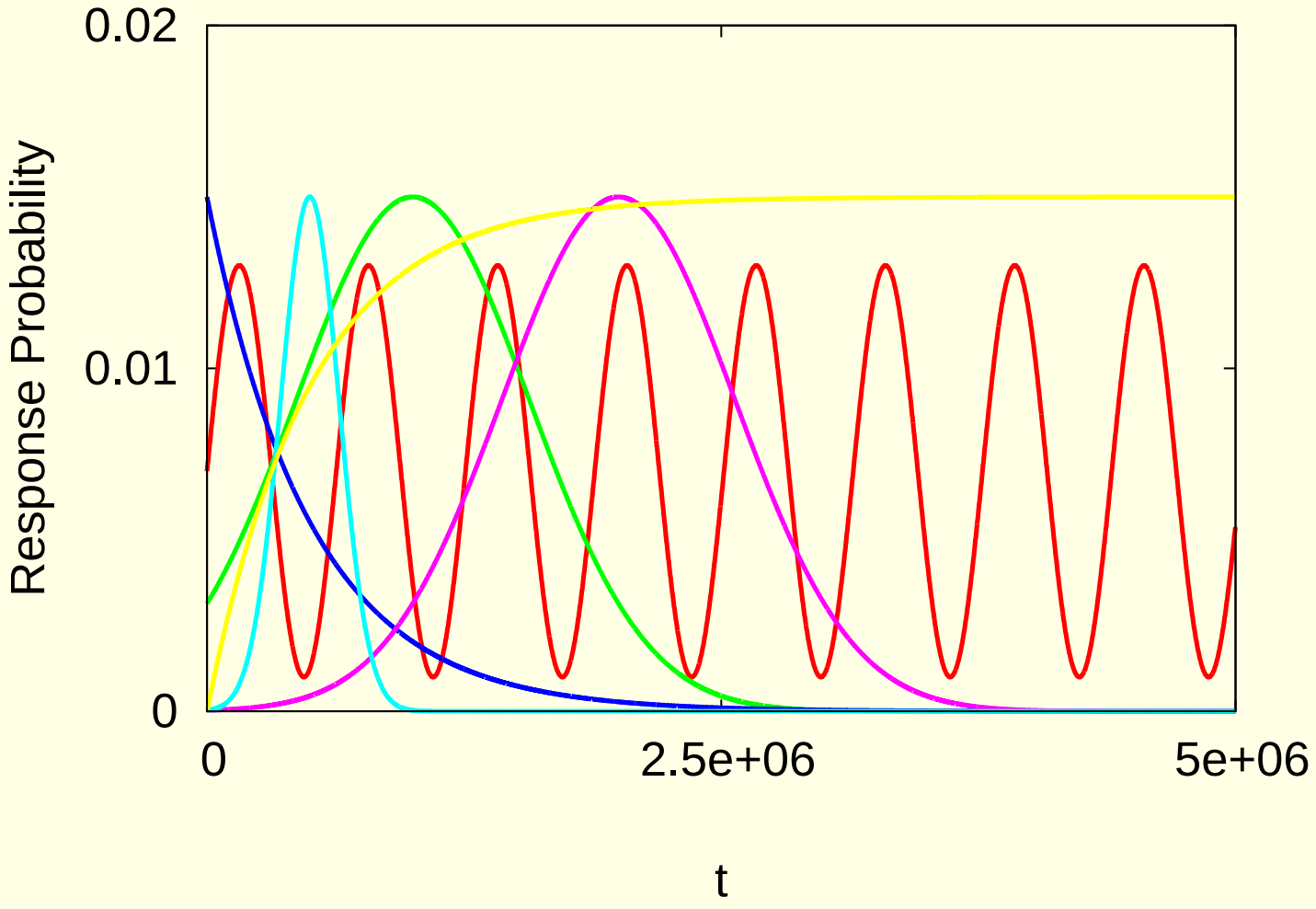
Visitors implemented for the challenge

SimulatedVisitorCoherentVariation



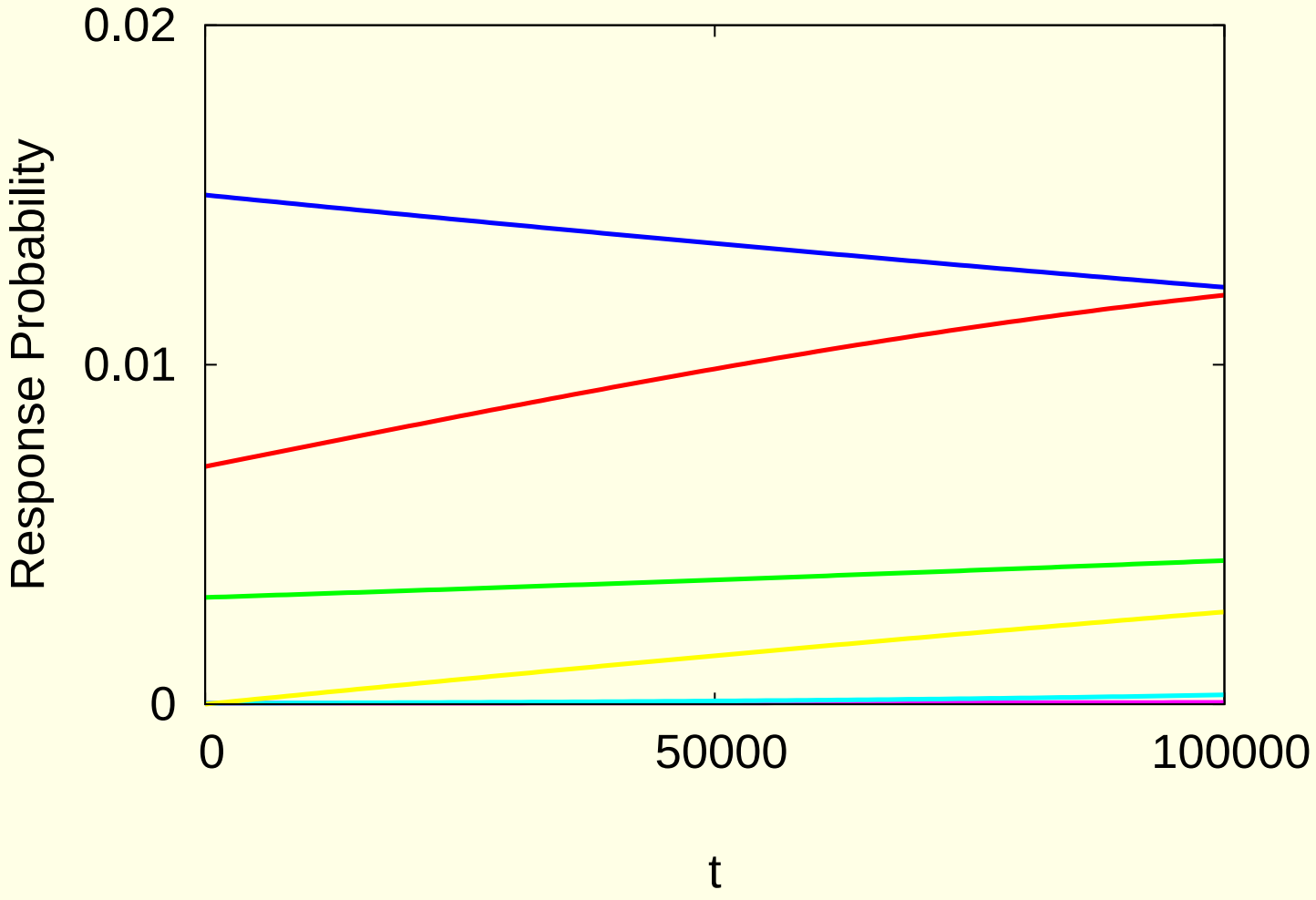
Visitors implemented for the challenge

SimulatedVisitorMixture



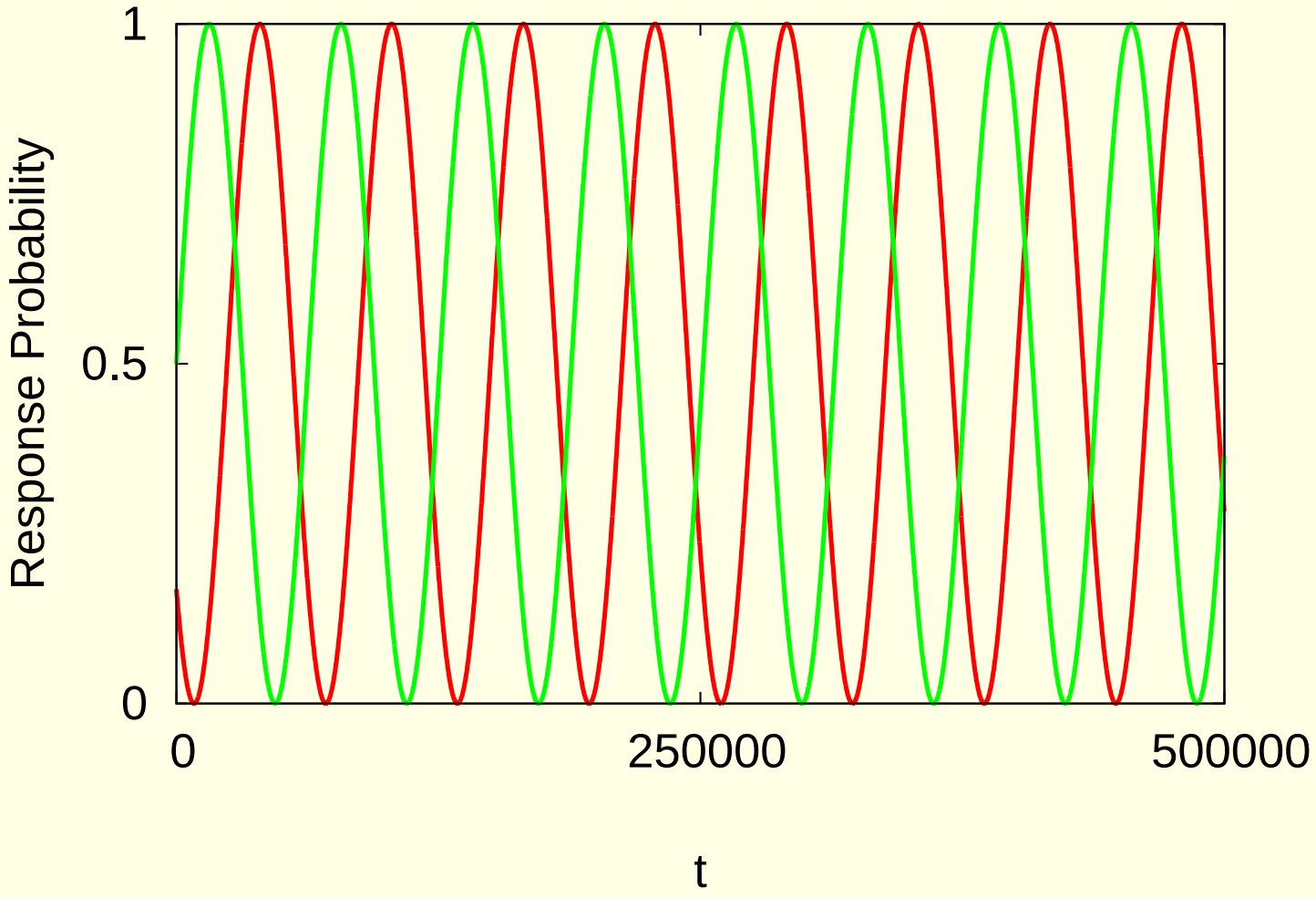
Visitors implemented for the challenge

SimulatedVisitorMixture



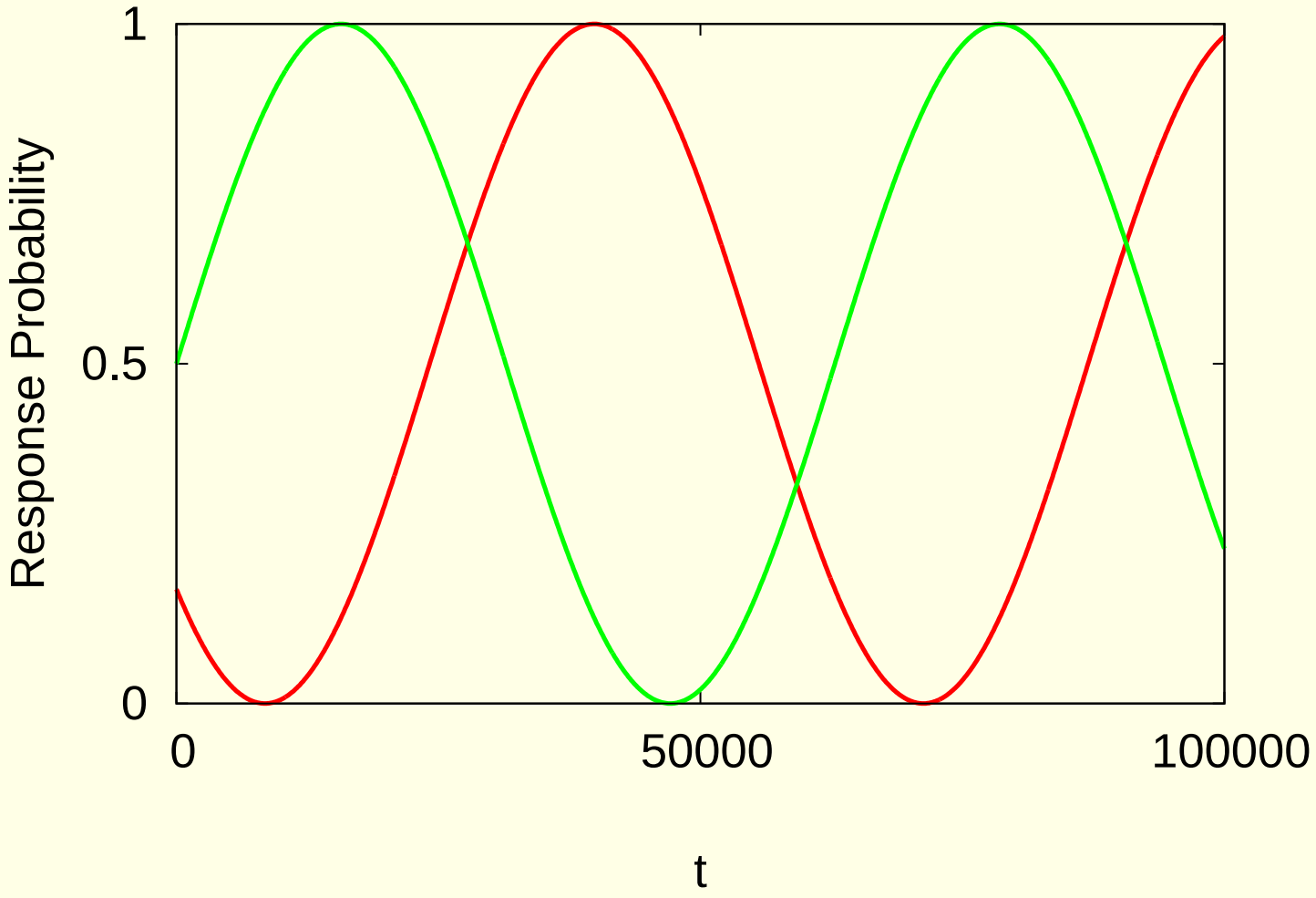
Visitors implemented for the challenge

SimulatedVisitorSimple



Visitors implemented for the challenge

SimulatedVisitorSimple



Shifting adversarial bandit algorithm

Adversarial bandits vs. random bandits:

- No statistical assumptions on how responses r_t are generated.
- Suitable for unknown, time varying success probabilities.
- The algorithm may again use upper confidence bounds.
- It keeps track of *shifts*, comparing the successes of the algorithm with the best *schedule* $t_1, \ldots, t_S$:

$$\max_{0=t_0<t_1<\cdots<t_S=T} \left(\sum_{s=1}^{S} \left[\max_{o} \sum_{t=t_{s-1}+1}^{t_s} p_t(o) \right] \right) - \sum_{t=1}^{T} r_t.$$

- Regret [ACFS'2002]: $\tilde{O}\left(\sqrt{TS}\right)$ instead of $O\left(\log T\right)$.

Shifting adversarial bandit algorithm

Parameter: S, T.

Initialization:

$\gamma = \sqrt{SK \log(KT)/T}$, $w(o) = 1$, $W = \sum_{o=0}^{K-1} w(o)$.

Repeat for $t = 1, \ldots, T$

1. Choose o_t according to $q(o) = (1-\gamma)\frac{w(o)}{W} + \frac{\gamma}{K}$.
2. If $r_t =$`true` then $w(o_t) \leftarrow w(o_t) \cdot \exp\left\{\frac{\gamma}{K}\frac{1}{q(o_t)}\right\}$.
3. $w(o) \leftarrow w(o) + W/T$.

Shifting adversarial bandit algorithm

Parameter: S, T.

Initialization:

$\gamma = \sqrt{SK \log(KT)/T}$, $w(o) = 1$, $W = \sum_{o=0}^{K-1} w(o)$.

Repeat for $t = 1, \ldots, T$

1. Choose o_t according to $q(o) = (1-\gamma)\frac{w(o)}{W} + \frac{\gamma}{K}$.
2. If $r_t =$`true` then $w(o_t) \leftarrow w(o_t) \cdot \exp\left\{\frac{\gamma}{K}\frac{1}{q(o_t)}\right\}$.
3. $w(o) \leftarrow w(o) + W/T$.

- By $w(o)$ an exponentially weighted "unbiased" estimate of the performance of option o is represented.

Shifting adversarial bandit algorithm

Parameter: S, T.

Initialization:

$\gamma = \sqrt{SK \log(KT)/T}$, $w(o) = 1$, $W = \sum_{o=0}^{K-1} w(o)$.

Repeat for $t = 1, \ldots, T$

1. Choose o_t according to $q(o) = (1-\gamma)\frac{w(o)}{W} + \frac{\gamma}{K}$.
2. If $r_t =$`true` then $w(o_t) \leftarrow w(o_t) \cdot \exp\left\{\frac{\gamma}{K}\frac{1}{q(o_t)}\right\}$.
3. $w(o) \leftarrow w(o) + W/T$.

- The term γ/K accounts for exploration.

Shifting adversarial bandit algorithm

Parameter: S, T.

Initialization:

$\gamma = \sqrt{SK \log(KT)/T}$, $w(o) = 1$, $W = \sum_{o=0}^{K-1} w(o)$.

Repeat for $t = 1, \ldots, T$

1. Choose o_t according to $q(o) = (1-\gamma)\frac{w(o)}{W} + \frac{\gamma}{K}$.
2. If $r_t =$`true` then $w(o_t) \leftarrow w(o_t) \cdot \exp\left\{\frac{\gamma}{K}\frac{1}{q(o_t)}\right\}$.
3. $w(o) \leftarrow w(o) + W/T$.

- The term W/T keeps weights sufficiently large such that a shift to a new best expert can be performed quickly.

Summary

- Presentation of the framework structure
- Introduction of
  - Baseline Solutions
  - Options
  - Simulated Visitors
- Identification of some problems to be addresses for phase 2 of the challenge.
- Sketch of the Shifting Bandit algorithm.