A New Dog Learns Old Tricks: RL Finds Classic Optimization Algorithms

Weiwei Kong, Christopher Liaw, Aranyak Mehta, D. Sivakumar

Slides: Bulat Ibragimov

What is the work about

There are several well-known combinatorial optimization problems and their online analogues.

Is RL able to find optimal solutions for them and how to make agent explore it?

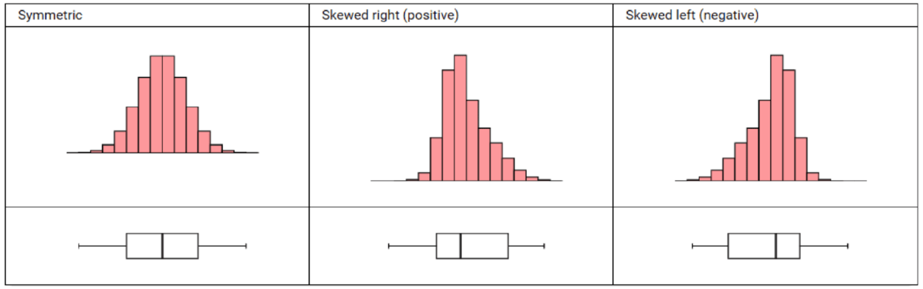
Problem 1: arbitrary input size

Use of online algorithms:

- Input arrives in small units
- Clear notion of reward at each step
- The goal is to optimize overall reward

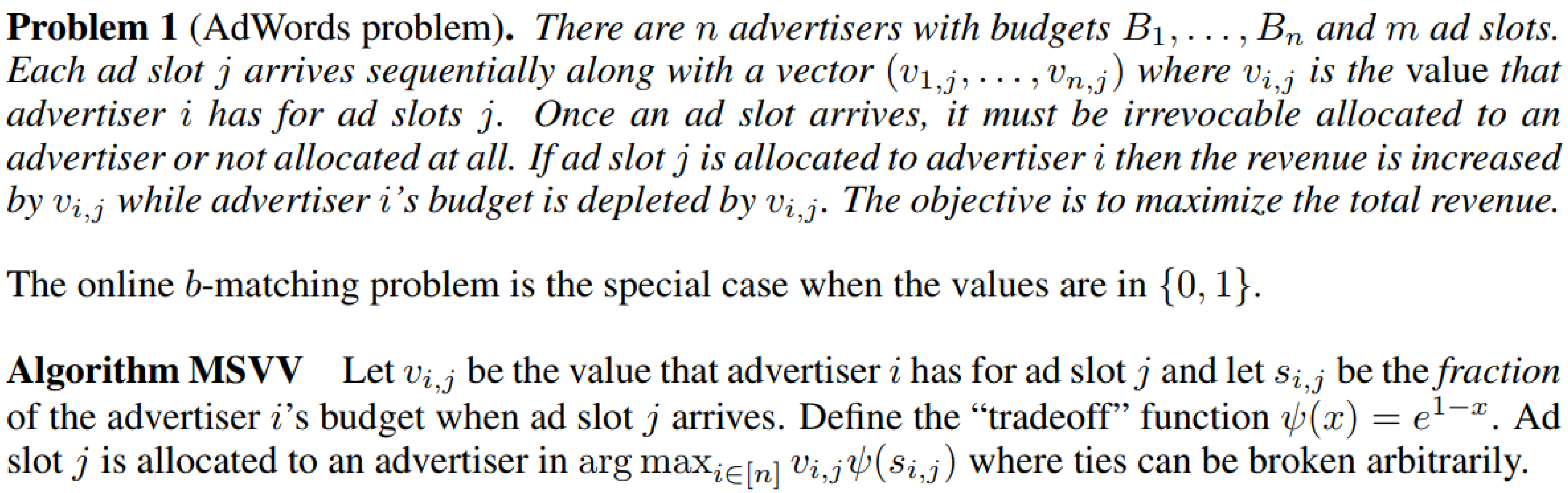
Problem 2: input distribitions

- ML approach: score is averaged over data distribution



- TCS approach: score in worst case

E1: AdWords problem

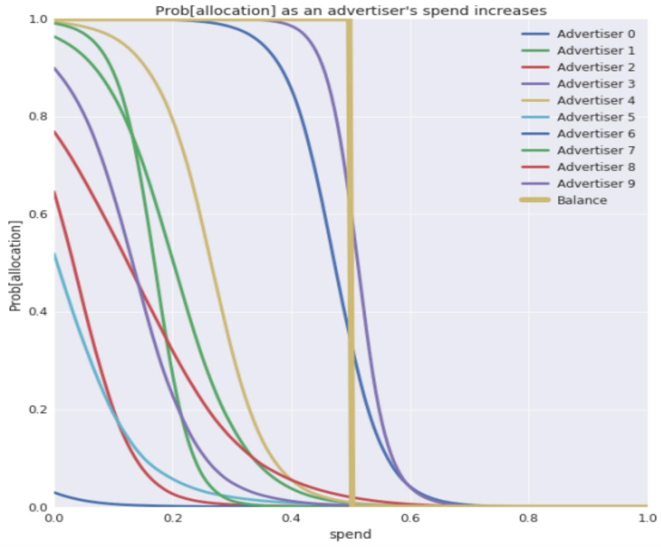


Problem 1 (AdWords problem)**.** *There are n advertisers with budgets $B_1, \ldots, B_n$ and m ad slots. Each ad slot j arrives sequentially along with a vector $(v_{1,j}, \ldots, v_{n,j})$ where $v_{i,j}$ is the* value *that advertiser i has for ad slots j. Once an ad slot arrives, it must be irrevocable allocated to an advertiser or not allocated at all. If ad slot j is allocated to advertiser i then the revenue is increased by $v_{i,j}$ while advertiser i's budget is depleted by $v_{i,j}$. The objective is to maximize the total revenue.*

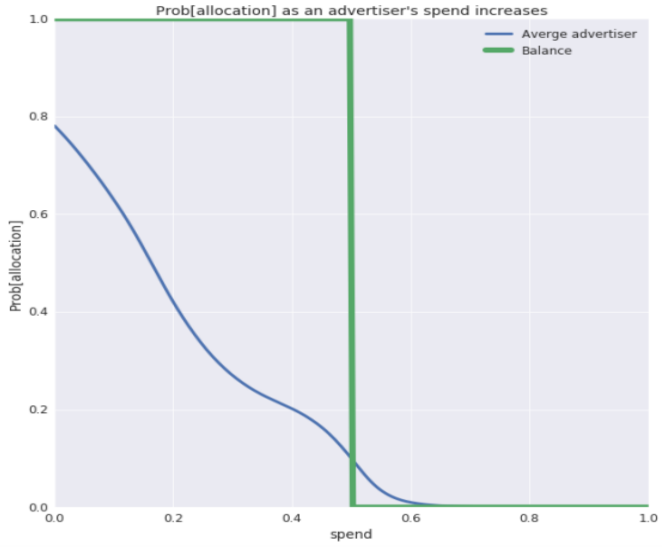
The online b-matching problem is the special case when the values are in $\{0, 1\}$.

Algorithm MSVV Let $v_{i,j}$ be the value that advertiser i has for ad slot j and let $s_{i,j}$ be the *fraction* of the advertiser i's budget when ad slot j arrives. Define the "tradeoff" function $\psi(x) = e^{1-x}$. Ad slot j is allocated to an advertiser in $\arg\max_{i \in [n]} v_{i,j}\psi(s_{i,j})$ where ties can be broken arbitrarily.

E1: AdWords problem



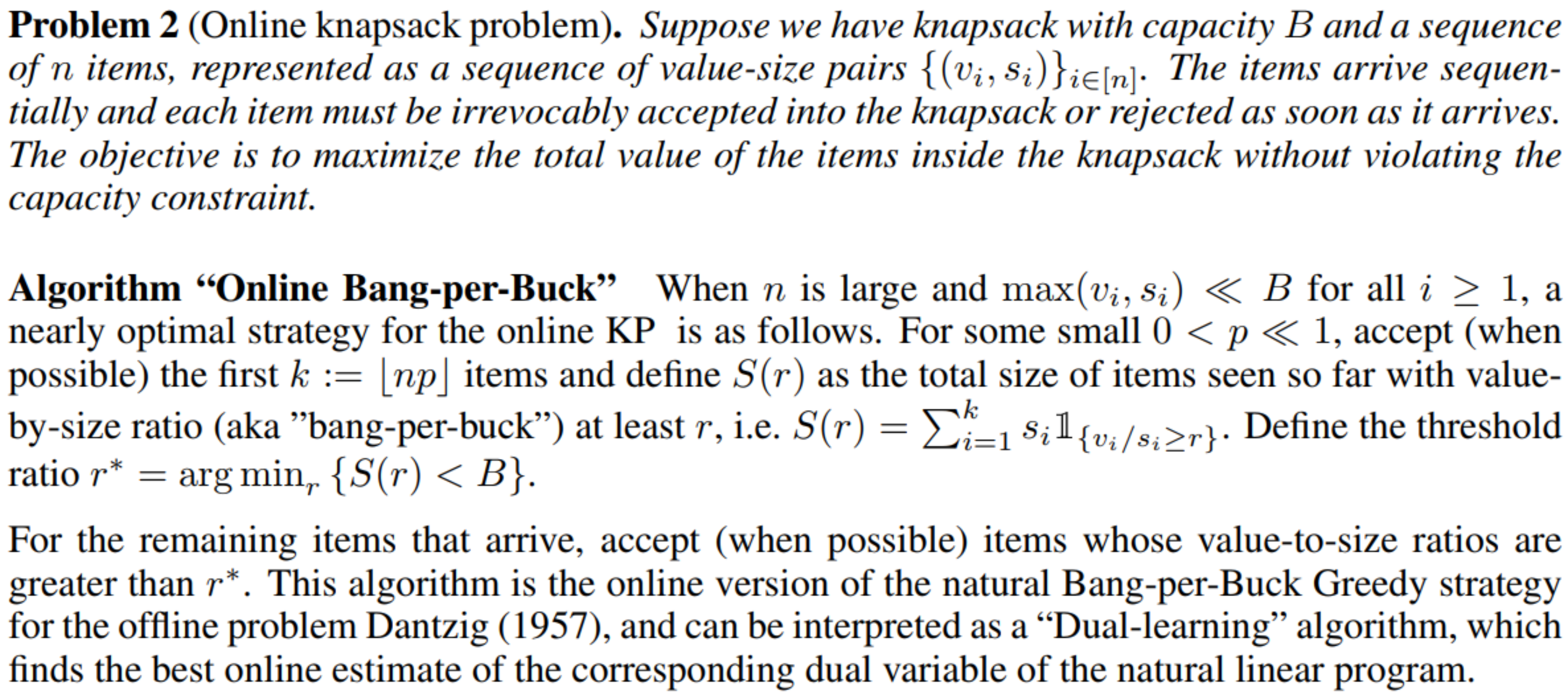
(a)



(b)

Figure 1: The algorithm learned by the agent. Each curve in Figure 1a plots the probability that advertiser i (as seen by the network) is allocated as a function of their spend when all other advertiser have spend 0.5 and all advertiser have value 1. plots the following curves. Figure 1b is obtained by averaging the curves in Figure 1a.

E2: Knapsack problem



Problem 2 (Online knapsack problem)**.** *Suppose we have knapsack with capacity B and a sequence of n items, represented as a sequence of value-size pairs $\{(v_i, s_i)\}_{i\in[n]}$. The items arrive sequentially and each item must be irrevocably accepted into the knapsack or rejected as soon as it arrives. The objective is to maximize the total value of the items inside the knapsack without violating the capacity constraint.*

Algorithm "Online Bang-per-Buck" When n is large and $\max(v_i, s_i) \ll B$ for all $i \geq 1$, a nearly optimal strategy for the online KP is as follows. For some small $0 < p \ll 1$, accept (when possible) the first $k := \lfloor np \rfloor$ items and define $S(r)$ as the total size of items seen so far with value-by-size ratio (aka "bang-per-buck") at least r, i.e. $S(r) = \sum_{i=1}^{k} s_i \mathbb{1}_{\{v_i/s_i \geq r\}}$. Define the threshold ratio $r^* = \arg\min_r \{S(r) < B\}$.

For the remaining items that arrive, accept (when possible) items whose value-to-size ratios are greater than r^*. This algorithm is the online version of the natural Bang-per-Buck Greedy strategy for the offline problem Dantzig (1957), and can be interpreted as a "Dual-learning" algorithm, which finds the best online estimate of the corresponding dual variable of the natural linear program.

E2: Knapsack problem

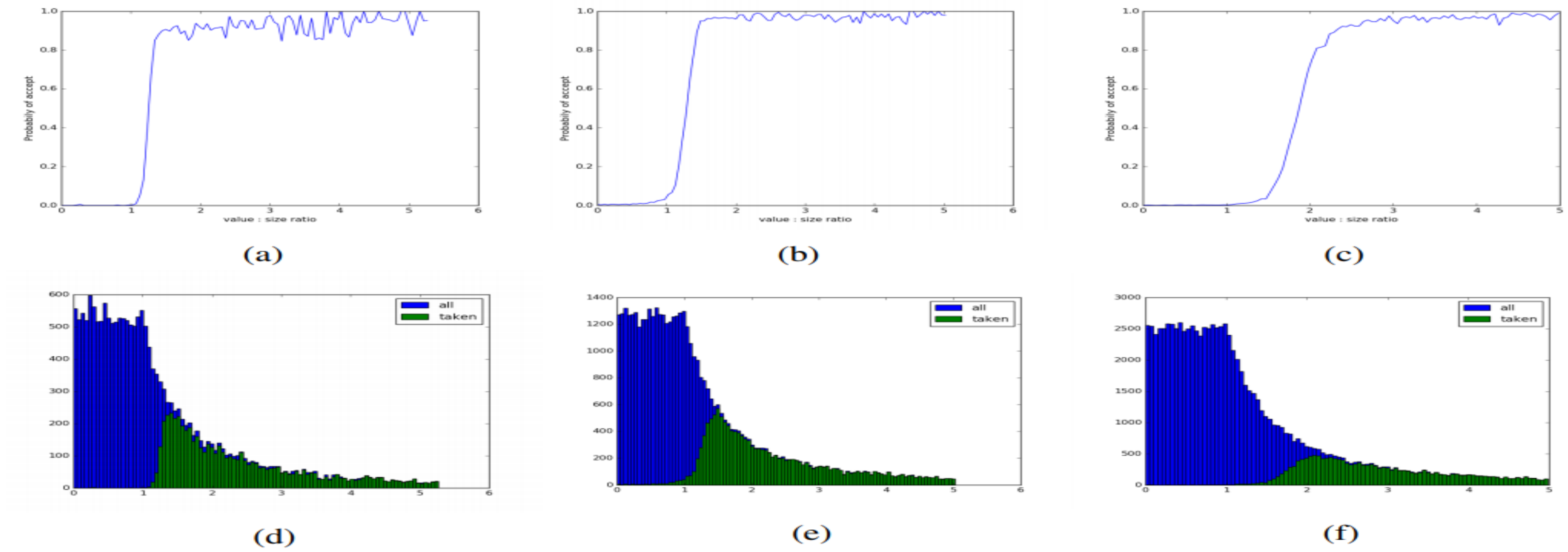
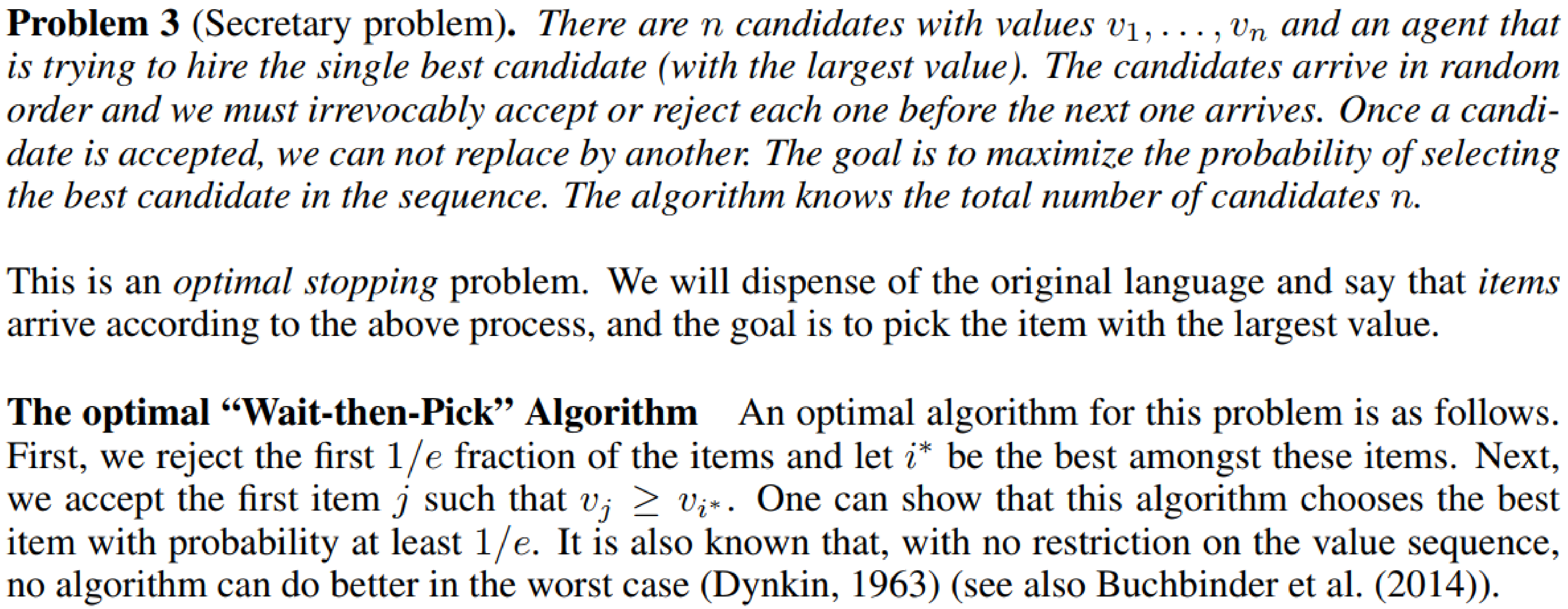


Figure 2: The agent's learned algorithm for the online Knapsack problem where the values and sizes are picked i.i.d. from $\mathcal{U}^2[0,1]$. The budget and length of sequence vary in the three figures: $(B,n) = (20, 200)$ (left), $(B,n) = (50, 500)$ (center), and $(B,n) = (50, 1000)$ (right). The top row depicts the probability that the agent will accept an item as a function of its value-by-size ratio. The bottom row depicts the histogram of items as a function of their value-by-size ratio ("all" is over all items in the sequence, and "taken" is over only the items that the agent accepts into the knapsack).

E3: Secretary problem

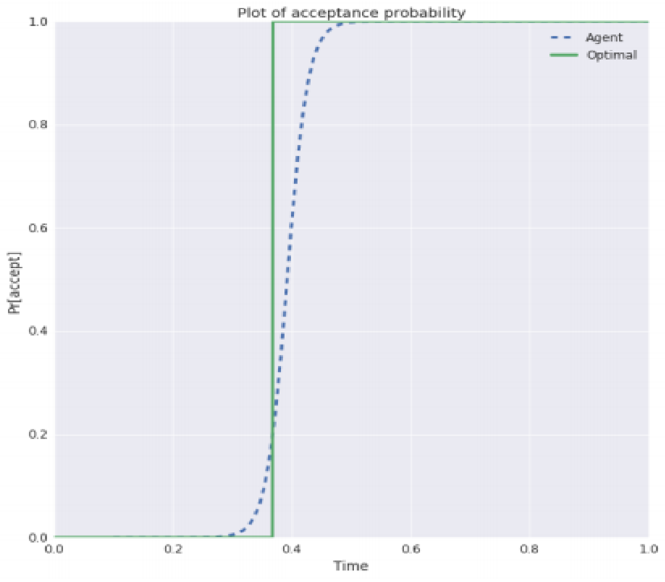


Problem 3 (Secretary problem)**.** *There are n candidates with values $v_1, \ldots, v_n$ and an agent that is trying to hire the single best candidate (with the largest value). The candidates arrive in random order and we must irrevocably accept or reject each one before the next one arrives. Once a candidate is accepted, we can not replace by another. The goal is to maximize the probability of selecting the best candidate in the sequence. The algorithm knows the total number of candidates n.*

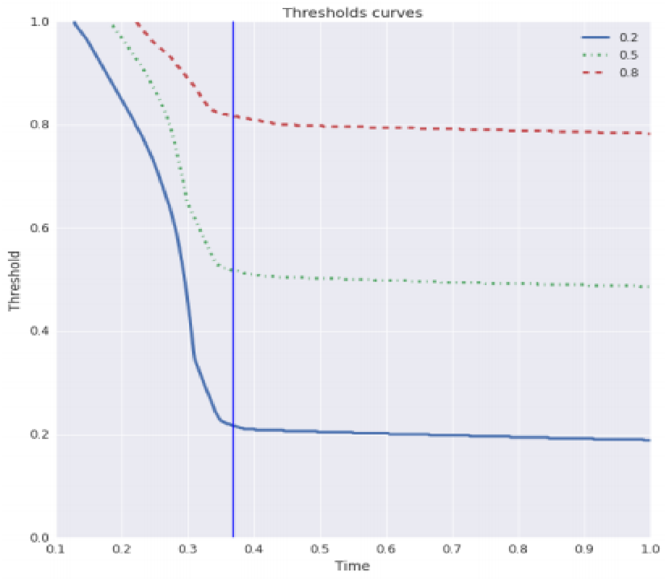
This is an *optimal stopping* problem. We will dispense of the original language and say that *items* arrive according to the above process, and the goal is to pick the item with the largest value.

The optimal "Wait-then-Pick" Algorithm An optimal algorithm for this problem is as follows. First, we reject the first $1/e$ fraction of the items and let i^* be the best amongst these items. Next, we accept the first item j such that $v_j \geq v_{i^*}$. One can show that this algorithm chooses the best item with probability at least $1/e$. It is also known that, with no restriction on the value sequence, no algorithm can do better in the worst case (Dynkin, 1963) (see also Buchbinder et al. (2014)).

E3: Secretary problem



(a)



(b)

Figure 3: Figure 3a compares the agent's learned algorithm with the optimal algorithm in the binary setting. Figure 3b plots the threshold for the agent's learned algorithm in the value setting with changing distributions. Observe that both have learned a threshold at around $1/e$.

Thank you!

Original paper: bit.ly/2TMP9cC

Slides:

Digest: