УДК 004.021

APPLICATION OF REINFORCEMENT LEARNING TO REVENUE MANAGEMENT

Y. Balasanov
Professor of Universiry of Chicago, PhD

M. Tselishchev
Professor of Universiry of Chicago, PhD

University of Chicago, USA

Y. Balasanov

Yuri Balasanov is a lecturer at the University of Chicago since 1997. He teaches at Graduate Program on Financial Mathematics (MSFM) and Graduate Program on Analytics (MScA). He is also founder and President of Research Software International, Inc. since 1991 and iLykei Teaching Tech Corp since 2015. Dr. Balasanov earned his Master's degree in Applied Mathematics and Ph.D. in Probability Theory and Mathematical Statistics from the Lomonosov Moscow State University, Russia, where he studied under Andrey Kolmogorov and leading members of his school. His primary expertise and research interests are in the area of stochastic modeling, machine learning and artificial intelligence with applications in various fields including trading, risk management, finance and economics, business analytics, marketing, biology, medical studies. Dr. Balasanov has been a financial industry practitioner for more than 20 years, working at leading financial institutions as head quant, quantitative trader and risk manager. He has lead research teams working on analytical and data driven projects as well as development of software for analytics.

M. Tselishchev

2012 –Degree Lomonosov Moscow State University. 2016 – Candidate of Sciences (PhD) in Probability Theory and Mathematical Statistics Lomonosov Moscow State University.

Abstract. Problem of revenue management.Every seller of a product or service has to make some fundamental decisions:a child making a lemonade booth: when to have the sale, how much to ask for each cup and when to drop the price to finish the sale at the end of the day;a homeowner selling a house: when to list it, what price to ask, which offer to accept, when to lower the price or unlist the house if necessary; an eBay seller: what is the duration of the auction, what is the starting price, all these are variations of famous optimization problem known under names like: the secretary problem, the marriage problem, the sultan's dowry problem, the fussy suitor problem, etc. The key issue is optimization under uncertainty of the future.

Keywords: reinforcement learning problem, revenue management, Q-learning

Revenue management decisions

Allocation of existing capacity depending on estimated and predicted demand is the field of revenue management, also known as demand management, sales decisions, yield management.

Types of revenue management decisions:

– Structural decisions: choice of selling format (e.g., posted price, negotiations, auction); choice of segmentation mechanisms; choice of offered trade terms (volume discounts, cancellation

or refund options) - Price decisions: setting posted prices, individual-offer prices, reserve prices (in auctions), pricing across product categories, across time, marking down over the product life-time, etc.

– Quantity decisions: accepting or rejecting an offer to buy, allocating capacity to different classes, segments, products or channels; when to withhold a product from the market.

Examples of applications of dynamic pricing strategies cover a broad range of manufactured goods and services, such as, to name a few, food, electronic goods and garments, airline tickets, hotel reservations, tickets for theaters, concerts or sports arenas.

Historical example

With Airline Deregulation Act of 1978 the U.S. Civil Aviation Board (CAB) loosened control of prices; airlines did not need approval from CAB before changing prices, schedules, services, they started developing computerized reservation systems (CRSs) and global distribution systems (GDSs); significant price elasticity that appeared as a result, allowed many people switching from driving to flying; the market was revolutionized.

Many new low-costers entered the market which forced larger companies to innovate. American Airlines Marketing vice president Robert Crandall realized:

– His company was able to produce near zero marginal cost of a seat and compete with low-cost startups using surplus seats

– But how to identify the surplus seats without displacing high-paying business travelers? And how to not allow traditional business customers switching to cheaper class of leisure travelers?

The problem was solved by American using a combination of purchase restrictions and capacity-controlled fares:

– Discounted tickets had to be purchased 30 days in advance, were not refundable, required minimum 7 days stay

– The number of discounted tickets on each flight was limited

The resulting American Super-Saver Fares launched in 1978 and its successor Dynamic Inventory Allocation and Maintenance Optimizer (DYNAMO) launced in 1985 where tremendously successful. DYNAMO became the first large scale revenue management system in the industry. The effect was dramatic: by aggressively matching any special deals in the market AA drove many competitors out of business.

Single-resource capacity control

Revenue management of single-resource capacity control finds optimal quantity-based revenue by allocating capacity of a single resource to different classes of demand.

Example: selling economy class cabin on a single leg flight at different prices.

In contrast, a multiple-resource or network control would deal with selling tickets with connecting flights or a sequence of nights at a hotel.

In simplest version of the problem statement demand for different classes is distinct and mutually exclusive.

The central problem is allocation of fixed capacity between classes either statically or dynamically; in the latter case demand for each class is observed in real time and is stochastic.

Types of control:

– Booking limits or protection levels

– Standard or theft nesting

– Bid prices

Booking limit is the maximum amount of capacity that can be sold to any particular class at a given point in time.

For example, discount offers sent to customers are available to the first 50 customers making purchase. After the first 50 sales the second class is closed and the rest 50 seats can be purchased at full price.

Littlewood's two-class method

A simple approach to revenue management of single-resource capacity can be based on the so-called Littlewood's rule.

Let $P_1 > P_2$ be prices of 2 classes. The demands for these clases are random:

$$D1 \sim F1(x),\ D2 \sim F2(x).$$

The main question is how many customers from lower class demand D_2 to accept before opening class 1 with demand D_1 ? It is also important to remember that in revenue management all sales typically have to end by a hard deadline after which the value of unsold units drops to zero.

Let remaining capacity be z. A request for price P_2 from demand D_2 arrives.

Possible decisions:

−Accept it and make P_2

−Reject it. Then entire capacity z can be sold for P_1 if and only if $D_1 \geq z$. The expected marginal value of reserving capacity z for class P_1 is then $P_1 P(D_1 \geq z)$

Then the optimal solution is: accept a class 2 request while $P_2 \geq P_1 P(D_1 \geq z)$, then switch to class 1.

The most critical piece of information required for the strategy is distributions of demands for all classes.

Revenue management as reinforcement learning problem

Dynamic pricing problem fits well within problems based on discrete finate Markov decision processes because pricing is a real time decision process in a stochastic environment with finite number of states (remaining capacity and remaining units of time before the deadline) and decisions depend only on the current, but not previous states.

The biggest advantage of using reinforcement learning method is: explicit knowledge of demands distributions is not necessary.

In order to avoid complexity of continuous price updates assume that price can only be reviewed and changed periodically at equal time steps.

Describe more formally the MDP for yield management problem:

−Let n be the total capacity (i.e. total number of units to sell) and m be the total number of times when the price can change

−Variable xt represents remaining capacity at time t and can take values $\{1,2,\ldots,n\}$, $t=\{0,1,\ldots,m\}$

−Remaining time to the deadline is $\tau t = m - t$

−$A(xt)$ is the set of allowed prices when remaining capacity at time t is xt. Prices a

−$t \in A(xt)$ are the actions available to the agent

−State of the Markov process describing the environment is then $St = <xt, \tau t>$

−Transition probabilities $p(S_{t+1}|S_t,a_t)$ that at time $t+1$ Markov process state is $S_{t+1}=<x_{t+1},\tau_{t+1}>$, given that at time t it was $S_t=<x_t,\tau_t>$ and the action (price) selected at time t was a_t

−Expected immediate revenue (reward) gained as a result of selecting price a_t when remaining capacity was x_t at time t is $r(S_t,a_t)$

− Policy of agent (pricing strategy) is formalized as discrete distribution on the set of available actions or probabilities $\pi(a|s)$ of selecting action a in state s

The objective is maximization of total expected revenue. Optimization is achieved by solving the action-value Bellman optimality equation using method of dynamic programing through iterations:

$$Q_{k+1}(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} Q_k(s',a')\right],$$

where γ is the discount factor for rewards and k is index of iterations of adjustments of the action-value function.

For more information about reinforcement learning see **[SuttonBarto]**

Dynamic pricing problem as Q-learning

Note that knowing transition probabilities $p(S_{t+1}|S_t,a_t)$ is in fact equivalent to knowing distribution of the demand.

Reinforcement learning problem can still be solved in case when MDP is not fully defined: transition probabilities and rewards are not initially known. In such case solution cannot be found by dynamic programming. The method that has to be used instead of dynamic programming is Q-learning. Agent will acquire knowledge about environment transitions indirectly (not explicitly in the form of transition probabilities matrix) and gradually in the process of interaction with that environment. Besides the advantage of being model-free the Q-learning approach is adaptive: agent continues learning and adapting to the changing environment.

Q-learning is based on the temporal difference version of action-value Bellman optimality equation

$$\begin{aligned} Q_{t+1}(s_t,a_t) &= (1-\alpha)Q_t(s_t,a_t) + \alpha\left(r_{t+1} + \gamma \cdot \max_{a_{t+1}} Q_t(s_{t+1},a_{t+1})\right), \\ &= Q_t(s_t,a_t) + \alpha\left[\left(r_{t+1} + \gamma \cdot \max_{a_{t+1}} Q_t(s_{t+1},a_{t+1})\right) - Q_k(s_t,a_t)\right], \end{aligned}$$

where α is learning rate and s_{t+1}, r_{t+1} are next step $t+1$ observations of state and immediate reward after at state s_t action a_t was selected.

Term

$$\delta_t = \left[\left(r_{t+1} + \gamma \cdot \max_{a_{t+1}} Q_t(s_{t+1},a_{t+1})\right) - Q_t(s_t,a_t)\right]$$

is TD error, it is the difference between the target $(r_{t+1}+\gamma\cdot\max_{a_{t+1}}Q_t(s_{t+1},a_{t+1}))$ and the current value of the action-value function $Q_t(s_t,a_t)$. Then $\alpha\delta_t$ is the adjustment to the state-value function $Q_t(s_t,a_t)$ at the next time $t+1$ with learning rate α

Further information on Q-learning can be found in **[SuttonBarto]**. For information about application of reinforcement learning to revenue management see, for example, **[GosaviBandla-Das]**, **[RanaOliveira]** and further references there.

Description of the results

Reinforcement learning approach

Below we illustrate the results using example of selling 100 tickets for a one-leg flight by a deadline using reinforcement learning policy trained using Q-learning.

Then we compare the results with a more traditional strategy based on Littlewood's rule that requires knowledge of mean value and variance of Gaussian demands for each class.

The results are obtained in communication with a remote server that plays role of environment simulator necessary for training the selling agent. The policy is a real time illustration when local agent sets the price level and sends it to the remote server. Then remote server sends back to the agent a flow of customers buying tickets for the suggested price.

The real time demo will be shown during the conference.

```
In [1]: %matplotlib notebook

In [2]:!protoc --python_out=./ revenue.proto
```

Initiate the environment. Show the total capacity, number of time units when the price can be changed and the available prices of different classes.

```
In [3]:from remote_revenue_env import RemoteRevenueEnv
      env = RemoteRevenueEnv(episode_duration=100, plotting=True, redraw_sec-
onds=0.05)

In [4]:print('Total capacity:', env.total_capacity)
       print('Subepisodes:', env.time_horizon)
       print('Ticket Fares:', env.action_space)
       Total capacity: 100
       Subepisodes: 10
       Ticket Fares: [300, 400, 500, 600, 700, 800, 900, 1000]
```

Define the class for interaction with the environment.

```
In [5]:class QLearn:
          def __init__(self, gamma=0.95, alpha=0.05):
              from collections import defaultdict
              self.gamma = gamma
              self.alpha = alpha
              self.qmap = defaultdict(int)

          def iteration(self, old_state, old_action, reward, new_state,
new_possible_actions):
        # Produce iteration step (update Q-Value estimates)
```

```
        old_stateaction = tuple(old_state) + (old_action,)
        max_q = max([self.qmap[tuple(new_state) + (a,)] for a in new_possi-
ble_actions])
        self.qmap[old_stateaction] = (1-self.alpha)*self.qmap[old_state-
action] + self.alpha*(reward+self.gamma*max_q)
        return

       def best_action(self, state, possible_actions):
           # Get the action with highest Q-Value estimate for specific state
           a, q = max([(a, self.qmap[tuple(state) + (a,)]) for a in possi-
ble_actions], key=lambda x: x[1])
           return a
```

Define $\epsilon -$ greedy strategy which selects the best available action using the current state-action value function with probability $1-\epsilon$ and selects action randomly with probability ϵ .

```
In [6]:import random

      def egreedy_strategy(ql, state, possible_actions, eps=0.0):
          # eps-greedy strategy
          # ql is a QLearn object
          if random.random() < eps:
              # select random action
              action = random.choice(possible_actions)
          else:
              # select action with max Q-Value estimate
              action = ql.best_action(state, possible_actions)
          return action
```

Train the state-action value function.

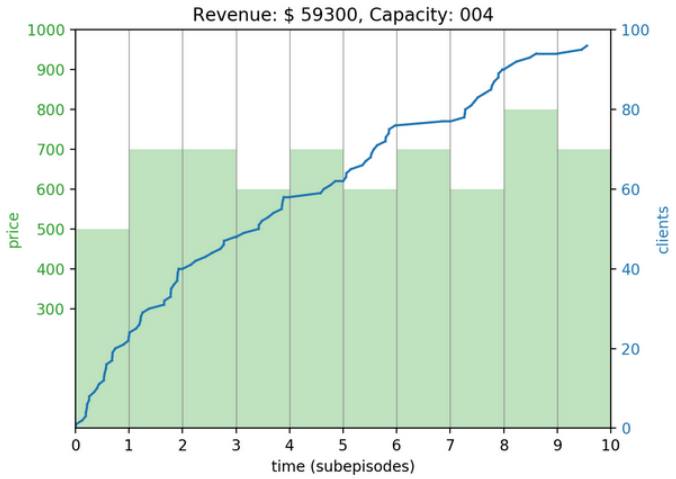
```
In [7]:def play_and_train(env, eps, n_games, ql, training):
            scores = []
            for _ in range(n_games):
                obs = env.reset()
                done = False
                score = 0
                while not done:
                    # select next action using eps-greedy strategy
                                action = egreedy_strategy(ql, obs, env.ac-
                    tion_space, eps=eps)
                    new_obs, reward, done, info = env.step(action)
                    score += reward
                    if training:
                        # update Q-Value estimates
                        ql.iteration(obs, action, reward,
                                     new_obs, env.action_space)
                    obs = new_obs
                scores.append(score)
            return scores

In [8]: from remote_revenue_env import RemoteRevenueTrainEnv
```

```
env = RemoteRevenueTrainEnv()

ql = QLearn(gamma=0.95, alpha=0.05)
train_scores = []  # container for train results
n_games = 100000  # number of episodes per eps for training
eps_list = [0.9, 0.7, 0.5, 0.3, 0.2, 0.1, 0.05, 0.0]
for eps in eps_list:
    print('Training with eps = {} ...'.format(eps))
    train_scores += play_and_train(env, eps, n_games, ql, train-
ing=True)
print('Done.')

Training with eps = 0.9 ...
Training with eps = 0.7 ...
Training with eps = 0.5 ...
Training with eps = 0.3 ...
Training with eps = 0.2 ...
Training with eps = 0.1 ...
Training with eps = 0.05 ...
Training with eps = 0.0 ...
Done.
```

Let the agent use the learned policy.

The curve shows capacity sold by the current moment (right axis). The bars show the changing price level: decisions by the agent (left axis).

```
In [9]: env = RemoteRevenueEnv()
        n_games = 1
        for i in range(n_games):
            obs = env.reset()
            done = False
            while not done:
                action = egreedy_strategy(ql, obs, env.action_space, eps=0)
                obs, reward, done, info = env.step(action)
            print(f'Episode # {i+1}, revenue: {info["revenue"]}, capacity
      left: {obs[0]}')
```



```
Episode # 1, revenue: 59300, capacity left: 4
```

EMSRb algorithm

In this section we apply EMSRb algorithm to the same problem.

Since this method requires knowledge of distributions of demands, collect, collect data from remote server imitating the environment and estimate expectation and standard deviation of customer demand for every possible price.

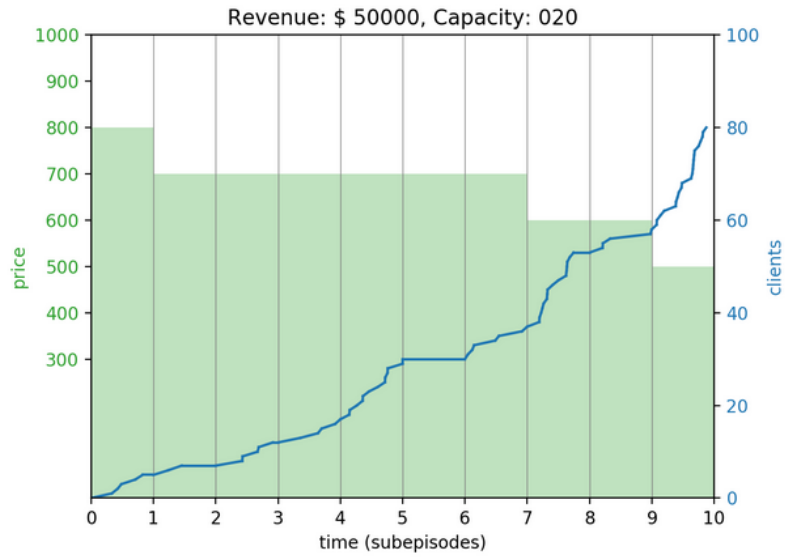
```
In [10]: # collect statistics for ESMR-b
         import numpy as np
         from remote_revenue_env import RemoteRevenueTrainEnv

         env = RemoteRevenueTrainEnv(prune_overbook=False)

         fares = sorted(env.action_space, reverse=True)
         ngames = 1000  # per price
         d_means = []
         d_sigmas = []

         for price in fares:
             demands = []
             for _ in range(ngames):
                 obs = env.reset()
                 done = False
                 d = 0  # demand in current episode
                 while not done:
                     obs, reward, done, info = env.step(price)
                     d += info['new_clients']
                 demands.append(d)
             d_means.append(np.mean(demands))
             d_sigmas.append(np.std(demands))

         fares = np.array(fares)
         d_means = np.array(d_means)
         d_sigmas = np.array(d_sigmas)

         print('Fares:', fares)
         print('Means:', d_means)
         print('Sigmas:', d_sigmas)
     Fares: [1000  900  800  700  600  500  400  300]
     Means: [  24.81    37.241   55.357   81.907  120.69   176.262  264.825
     389.253]
     Sigmas: [  6.0549071    8.49640624  11.56873161  15.30961629  22.23910745
       36.5007309   51.00417998  78.38981433]
```

Once the data are collected, implement the strategy, coded below.

```
In [11]: def emsrb_strategy(capacity_left, fares, protection_levels):
          for j in reversed(range(len(fares))):
              if capacity_left >= protection_levels[j]:
                  return fares[j]
```

Run the strategy and visualize the results.

```
In [16]: from remote_revenue_env import RemoteRevenueEnv
         from revpy.revpy import protection_levels

         env = RemoteRevenueEnv(episode_duration=100, plotting=True, re-
         draw_seconds=0.05)
         n_games = 1

         for i in range(n_games):
             obs = env.reset()
             done = False
             subepisodes_left = env.time_horizon
             while not done:
                 fractiontime_left = subepisodes_left / env.time_horizon
                 pl = protection_levels(fares, d_means*fractiontime_left,
                                        d_sigmas*np.sqrt(fractiontime_left),
         method='EMSRb')
                 action = emsrb_strategy(obs[0], fares, pl)
                 obs, reward, done, info = env.step(action)
                 subepisodes_left -= 1
             print(f'Episode # {i+1}, revenue: {info["revenue"]}, capacity
         left: {obs[0]}')
```




```
Episode # 1, revenue: 50000, capacity left: 20
```

References

[1] [SuttonBarto]: Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series), © 2018 Richard S. Sutton, Andrew G. Barto, The MIT Press

[2] [GosaviBandlaDas]: A reinforcement learning approach to a single leg airline revenue management problem with multiple fare classes and overbooking, Gosavi, A., Bandla, N. & Das, T.K. IIE Transactions (2002) 34: 729. https://doi.org/10.1023/A:1015583703449

[3] [RanaOliveira]: Real-time dynamic pricing in a non-stationary environment using model-free reinforcement learning, Rupal Rana, Fernando S. Oliveira, Omega, © 2013, Elsevier