In order not to overburden participants with too much technicality, in this article, we entirely focus on the value-based approach of reinforcement learning. For more advanced techniques, such as policy gradient methods, participants in keen interest can consult chapter 13 in Sutton's book(Sutton, Barto: *Reinforcement Learning: An Introduction, Second Edition*, MIT Press, Cambridge, MA, 2018).

In the value-based approach, one seeks a mapping from the state space to a real value which indicates how desirable a state is. If a good value function is found, then it is almost to trivial to construct a good strategy: once you are in state S_t, among all its possible succeeding states, choose the one with the highest value.

Note that this simple idea works due to the deterministic nature of the bin packing problem. There is no need to resort to the more involved action value function which maps a state-action pair to a real value, because the action of placing a box in some free space in a vehicle completely specifies the next state.

Although this determinism simplifies the learning mechanism considerably, one still needs to take care of the various subtleties that are unique in the bin packing procedure.

1 Vectorizing the states

Since eventually a state will be fed into a parametrized value function, e.g. a neural network, one must find a structured representation of any state, i.e. a vector. Among various choices of representation, one possibility is to discretize the space in the vehicle to obtain a matrix whose size is vehicle length by vehicle width, and whose entry is either 0 or 1, indicating the space is free or occupied, respectively(think about the game of tetris).

For example, if the vehicle is of size 5-by-3, and there is only one box of size 3-by-2 to be packed, then initially, the matrix is $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ indicating that the space is completely free. If we decide to place the box of size 3-by-2 on the upper left corner, then the resulting matrix is

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

depending on the orientation of the box.

A vector encoding the status of the boxes is also needed but relatively straightforward: each box is encoded by three entries, the box length, the box width, and a boolian value indicating whether the box is packed or not.

2 Reasonable actions

Although we are exempted from the burden of dealing with the action value function($q(s, a)$ from which the standard Q-learning technique derives its name), the obligation of specifying what counts as a valid action given a state doesn't dissolve. It is certainly true that you can place a box in any unoccupied space in the vehicle, but any sensible human being would try to have the boxes placed as compactly as possible. By compactness, we mean at least the edge of a newly packed box should align with either the edge of another already packed box, or the edge of the vehicle.

This is much easier said than done. According to the author's own experience, although this is more of a modeling problem than a machine learning problem, it is of paramount importance to devise an efficient algorithm that is able to return a complete set of legal AND reasonable positions of placing the next box given any intermediate state.

3 Reward

In principle, how to design the reward associated with each state is up to personal taste. However, here we make a word of caution.

It may be tempting to use the volume(or area if you are pedantic) of the newly packed box as the reward. However, depending on how one incorporates the routing part with the bin packing problem, it is possible that one demands the vehicle to successively pick up goods in various stations and thus she wants the packing to be as efficiently as possible in each station. Unfortunately, simply using the total volume of packed boxes as the final return would exhibit no difference as long as the boxes are packed into the vehicle, which maybe trivial in the first few stations.

One way to characterize how good a way of packing is when there is no diffculty of loading all boxes into the truck is to require that the boxes to be placed as close to the rear side of the vehicle as possible. If the encoding scheme in section 1 is adopted, this amounts to use as few rows in the matrix as possible. In fact, one can do better by using the following formula:

$$\frac{\text{packed volume}}{(\text{vehicle width}) \times (\text{used length})}$$

where vehicle width is the number of columns in the matrix, and the used length is the maximum row index among rows that contain at least one occupied entry if all of the boxes can be packed or simply the number of rows otherwise. This fomulation unifies the two possibilities, whether packing completely is possible or not. However, this formula is not monotonically increasing as the boxes are being packed, so the performance of Q-learning or temporal-difference learning can be severely hindered. If this formula is to be adopted, we recommend participants to set the reward of all intermediate states to be 0 and that of the final state to be this effective-loading-rate-ish metric, and to use Monte Carlo method to update the parameters.

4 Pseudo code

Reinforcement learning is a highly modularized technique. Once the building blocks mentioned in the previous sections are ready, one is free to use 1) whatever learning models, from simple linear models to complex neural networks, 2) whatever degree of bootstrapping(Monte Carlo, n-step Sarsa), and 3) whatever updaing scheme(gradient descent, RMSProp, Adam).

Here, for the sake of completion, we provide a sample pseudo code(for more examples, see Chapter 10 in Sutton's book), using one step Sarsa and gradient descent.

Let $v(s)$ be the **state value function**, choose a parametrized family of functions $\hat{v}(s; w)$ which serves as an approximate of $v(s)$. Initialize the weights w, and pick learning rate α.

```
Loop for each episode:
  S ← initial state
  Loop for each step of episode:
    Choose the succeeding state S' by ε-greedy algorithm,
    and observe reward R
    If S' is terminal:
      w ← w + α[R − v̂(S, w)]∇v̂(S, w)
      Break, go to next episode
    Else
      w ← w + α[R + v̂(S', w) − v̂(S, w)]∇v̂(S, w)
      S ← S'
```