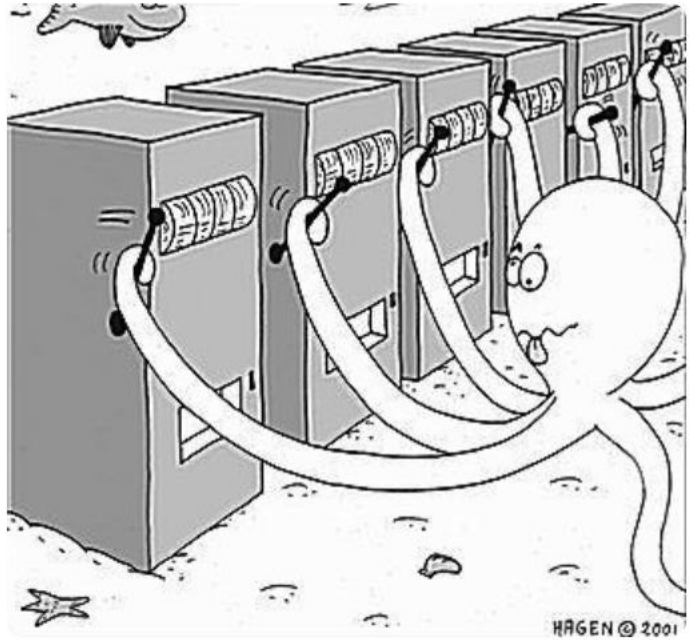
AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Bandit Problem

Multi-arm Bandit Problem (MABP)

- A bandit is defined as someone who steals your money.
- A **one-armed bandit** is a simple slot machine wherein you insert a coin into the machine, pull a lever, and get an immediate reward.
- But why is it called a bandit? It turns out all casinos configure these slot machines in such a way that all gamblers end up losing money!



Multi-armed bandit is a complicated slot machine wherein instead of 1, there are several levers which a gambler can pull, with each lever giving a different return. The probability distribution for the reward corresponding to each lever is different and is unknown to the gambler

The task is to identify which lever to pull in order to get maximum reward after a given set of trials Each arm chosen is equivalent to an action, which then leads to an immediate reward.

Exploration Exploitation in the context of Bernoulli MABP

The table shows the sample results for a 5-armed Bernoulli bandit with arms labelled as 1, 2, 3, 4 and 5:

Arm	Reward
1	0
2	0
3	1
4	1
5	0
3	1
3	1
2	0
1	1
4	0
2	0

This is called Bernoulli, as the reward returned is either 1 or 0.

Exploitation: In this example, it looks like the arm number 3 gives the maximum return and hence one idea is to keep playing this arm in order to obtain the maximum reward

Just based on the knowledge from the given sample, 5 might look like a bad arm to play, but we need to keep in mind that we have played this arm only once and maybe we should play it a few more times (exploration) to be more confident. Only then should we decide which arm to play (exploitation).

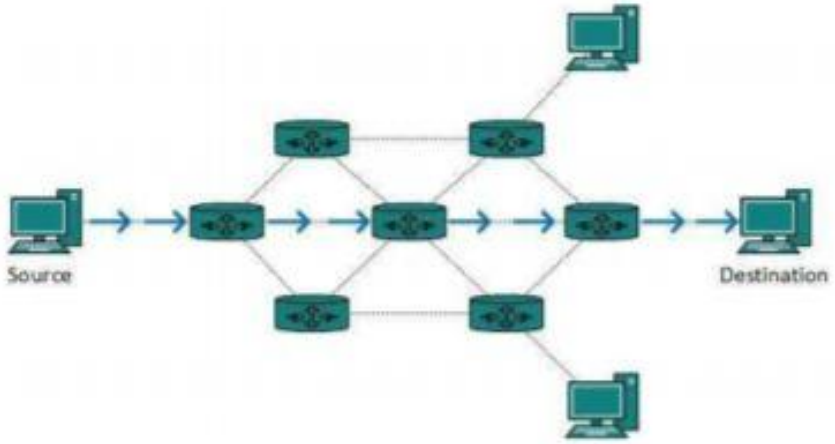
Use Cases

Clinical Trials

The well being of patients during clinical trials is as important as the actual results of the study. Here, exploration is equivalent to identifying the best treatment, and exploitation is treating patients as effectively as possible during the trial.

Network Routing

Routing is the process of selecting a path for traffic in a network, such as telephone networks or computer networks (internet). Allocation of channels to the right users, such that the overall throughput is maximised, can be formulated as a MABP.



Purely evaluative feedback indicates how good the action taken is but not whether it is the best or the worst action possible. Uses training information that evaluates the actions taken rather than instructs by giving correct actions

Solution Strategies

Action-Value Function

The expected payoff or expected reward can also be called an action-value function. It is represented by q(a) and defines the average reward for each action at a time t.

$$Q(a) = \mathbb{E}[r|a]$$

Suppose the reward probabilities for a K-armed bandit are given by {P1, P2, P3 …… Pk}. If the *ith* arm is selected at time t, then Qt(a) = Pi.

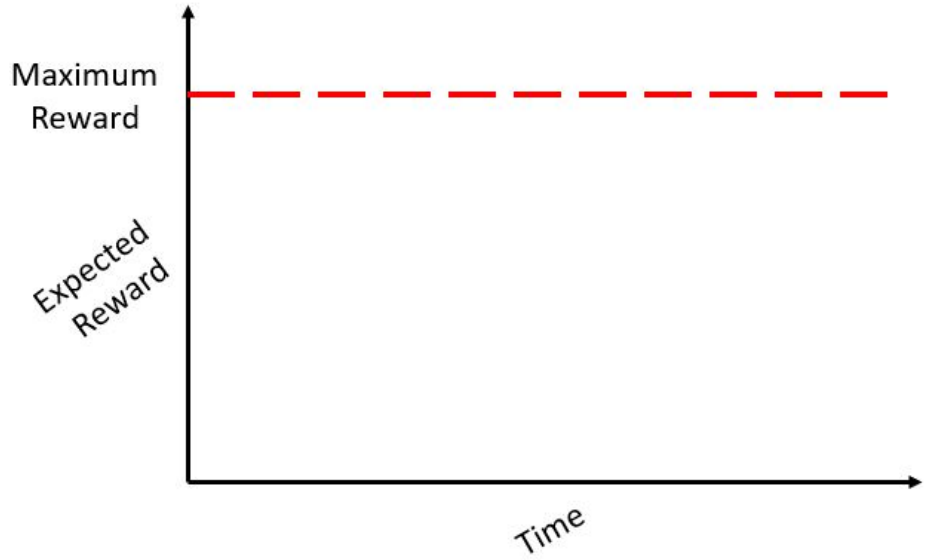
The question is, how do we decide whether a given strategy is better than the rest?

One direct way is to compare the total or average reward which we get for each strategy after *n* trials.

-concept of **regret**.

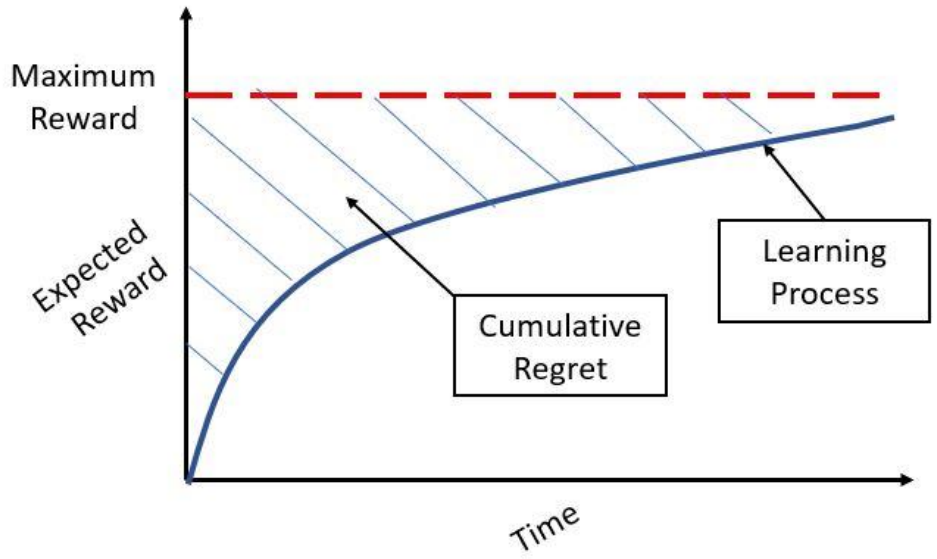
Regret

Let's say that we are already aware of the best arm to pull for the given bandit problem. If we keep pulling this arm repeatedly, we will get a maximum expected reward which can be represented as a horizontal line (as shown in the figure below):



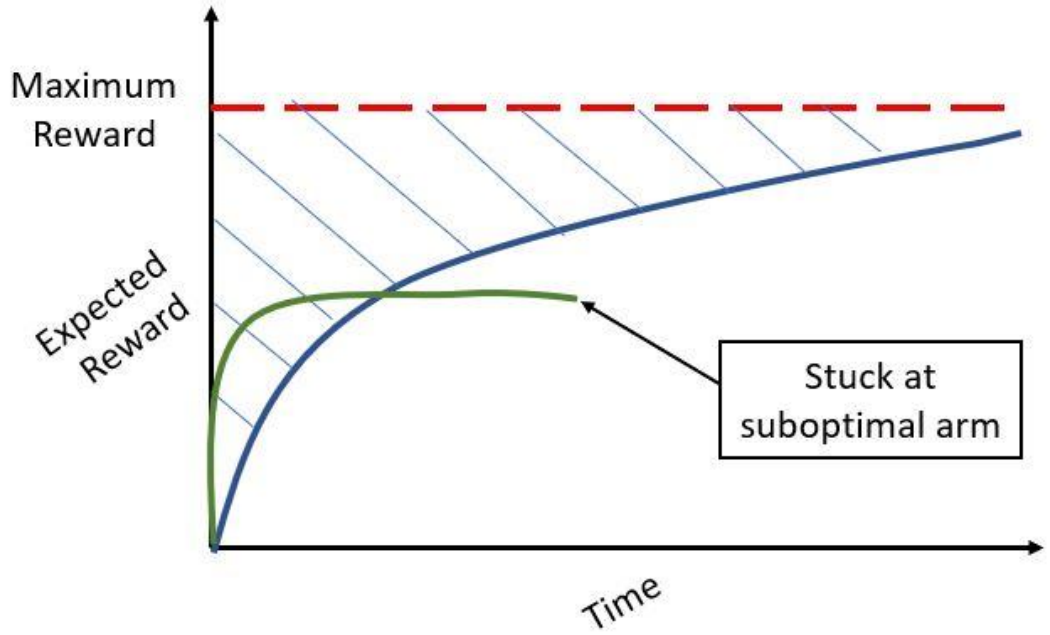
But in a real problem statement, we need to make repeated trials by pulling different arms till we are approximately sure of the arm to pull for maximum average return at a time t.

The loss that we incur due to time/rounds spent due to the learning is called regret. In other words, **we want to maximise my reward even during the learning phase**. Regret quantifies exactly how much you regret not picking the optim



Regret change: when no exploration

Now, one might be curious as to how does the regret change if we are following an approach that does not do enough exploration and ends exploiting a suboptimal arm. Initially there might be low regret but overall we are far lower than the maximum achievable reward for the given problem as shown by the green curve in the following figure.



No Exploration (Greedy Approach)

Action-value function $Q_t(a)$

$Q_t(a)$ is our **estimate** of how good action a is, **at time** t.

The formula given is:

$$Q_t(a) = \frac{\sum_{i=1}^{t} 1(a_t = a) \cdot R_i}{\sum_{i=1}^{t} 1(a_t = a)}$$

Where:

- $Q_t(a)$: Estimated value of action a at time t.
- $1(a_t = a)$: Indicator function = 1 if action a was taken at time t, else 0.
- R_i: Reward received at time i.
- Numerator: Sum of all rewards obtained when action a was chosen.
- Denominator: Number of times action a was chosen.

So:

- $Q_t(a)$ = **Average reward** for action a so far.

A naïve approach could be to calculate the q, or action value function, for all arms at each timestep. From that point onwards, select an action which gives the maximum q. The action values for each action will be stored at each timestep by the following function:

Action selection rule

The agent chooses:

$$\arg\max_a Q_t(a)$$

Which means:

- Look at the estimated value of each action.
- Pick the one with the **highest average reward so far**.

More compact form (mean update rule)

$$Q_t(a) = Q_{t-1}(a) + \frac{1}{N_t(a)}\left(R_t - Q_{t-1}(a)\right)$$

Interpretation:

- $R_t - Q_{t-1}(a)$ = **error term** = how far the new reward is from our current average.
- Multiply by $\frac{1}{N_t(a)}$ = step size for averaging.
- Add to the old $Q_{t-1}(a)$ to get the new estimate.

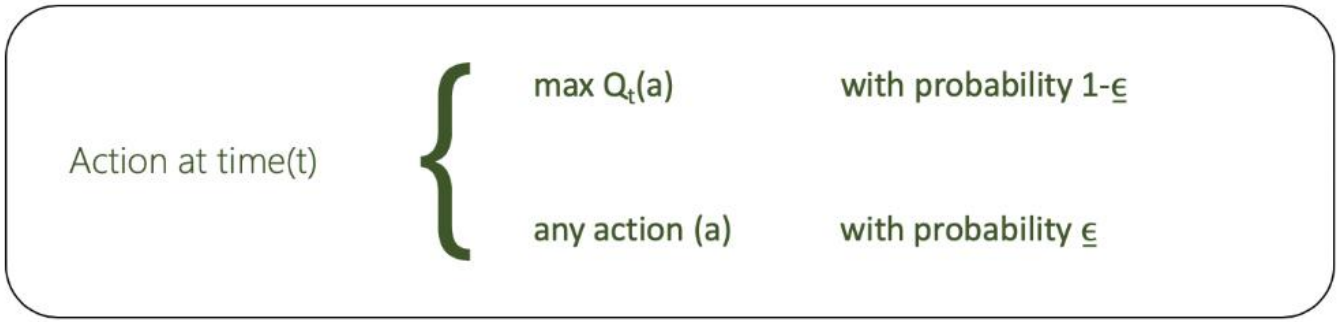
This is the **standard incremental average formula**.

$$Q_t(a) = \frac{\sum_{i=1}^{t} 1(a_t = a) \cdot R_i}{\sum_{i=1}^{t} 1(a_t = a)}$$

Epsilon Greedy Approach

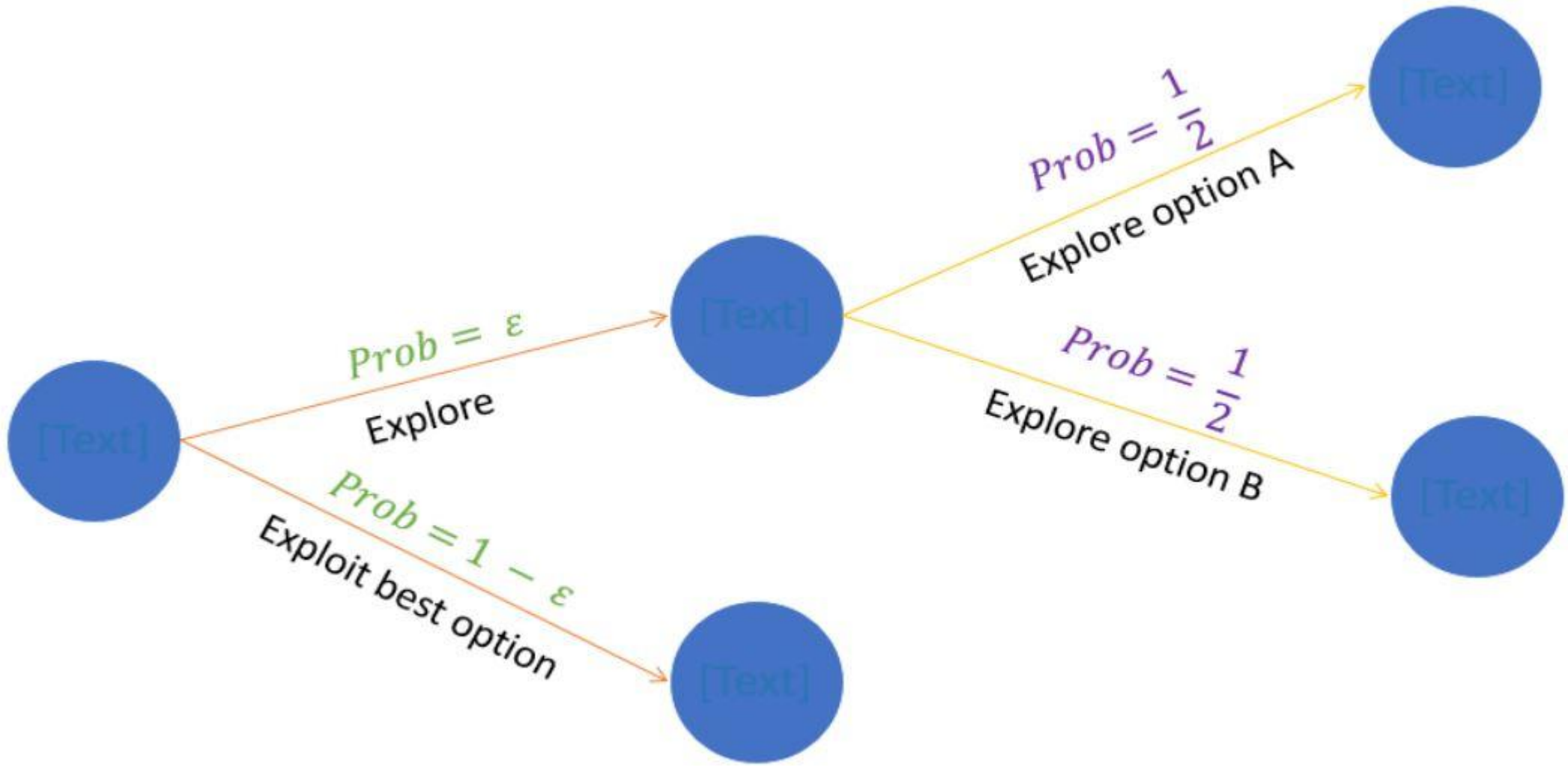
We need balance Exploration- Exploitation : One potential solution could be - we can explore new actions so that we ensure we are not missing out on a better choice of arm.

Epsilon-Greedy Action Selection Epsilon-Greedy is a simple method to balance exploration and exploitation by **choosing between exploration and exploitation randomly**. The epsilon-greedy, where epsilon refers to the probability of choosing to explore, exploits most of the time with a small chance of exploring.




```
p = random()

if p < ε:
   pull random action
else:
   pull current-best action
```



This is much better than the greedy approach as we have an element of exploration here. However,

Problem : if two actions have a very minute difference between their q values, then even this algorithm will choose only that action which has a probability higher than the others.--→ Solution- Softmax exploration

Softmax Exploration

The solution is to have the probability of choosing an action proportional to q. This can be done using the softmax function, where the probability of choosing action *a* at each step is given by the following expression:

$$P(a_t = a) = \frac{e^{Q_t(a)}}{\sum_1^n e^{Q_t(b)}}$$

- In **ε-greedy**, if two actions a_1, a_2 have very close Q-values, the algorithm:
  - Exploits → **always picks the one with the slightly higher Q** (deterministic).
  - Explores → picks uniformly at random.
- This means a tiny difference in estimated values can **lock the agent** into one action most of the time.

Decayed Epsilon Greedy

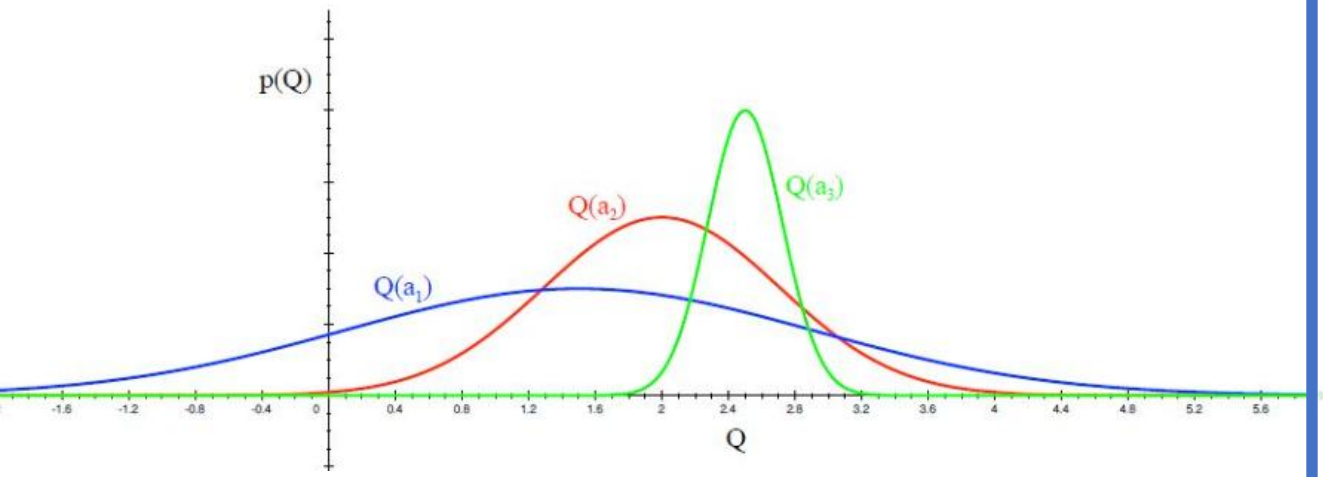
- The value of epsilon is very important in deciding how well the **epsilon** greedy works for a given problem.
- We can avoid setting this value by **keeping epsilon dependent on time**. For example, epsilon can be kept equal to ***1/log(t+0.00001)*.**
- It will keep reducing as time passes, to the point where we starting exploring less and less as we become more confident of the optimal action or arm.

- The problem with random selection of actions is that after sufficient timesteps even if we know that some arm is bad, this algorithm will keep choosing that with probability *epsilon/n*.
- Essentially, we are exploring a bad action which does not sound very efficient.
- The approach to get around this could be to favour exploration of arms with a strong potential in order to get an optimal value.

Upper Confidence Bound

Upper Confidence Bound (UCB) -most widely used solution method for multi-armed bandit problems.

This algorithm is based on the principle of optimism in the face of uncertainty.-In other words, the more uncertain we are about an arm, the more important it becomes to explore that arm.



•Distribution of action-value functions for 3 different arms a1, a2 and a3 after several trials is shown in the figure , This distribution shows that the action value for a1 has the highest variance and hence maximum uncertainty.

•UCB says **that we should choose the arm a_1 and receive a reward making us less uncertain about its action-value**. For the next trial/timestep, if we still are very uncertain about a_1, we will **choose it again until the uncertainty is reduced below a threshold**.

Steps involved in UCB1:

Steps involved in UCB1:

1. *Play each of the K actions once, giving initial values for mean rewards corresponding to each action*
2. *For each round t = K:Let $N_t(a)$ represent the number of times action a was played so far*

total score = **reward estimate + uncertainty bonus**
Play an action often → confidence grows → bonus shrinks.
Ignore an action while time passes → curiosity grows → bonus rises.

2.1Play the action at maximising the following expression

Exploration Bonus

exploitation

$$Q(a) + \sqrt{\frac{2 \log t}{N_t(a)}}$$

2.2. Observe the reward and update the mean reward or expected payoff for the chosen action

This dynamic forces **systematic exploration** without randomness.

Intuition- UCB1

•Each time *a* is selected, the uncertainty is presumably reduced: $N_t(a)$ increments, and, as it appears in the denominator, the uncertainty term decreases.

•On the other hand, each time an action other than *a* is selected, t increases, but $N_t(a)$ does not; because t appears in the numerator, the uncertainty estimate increases.

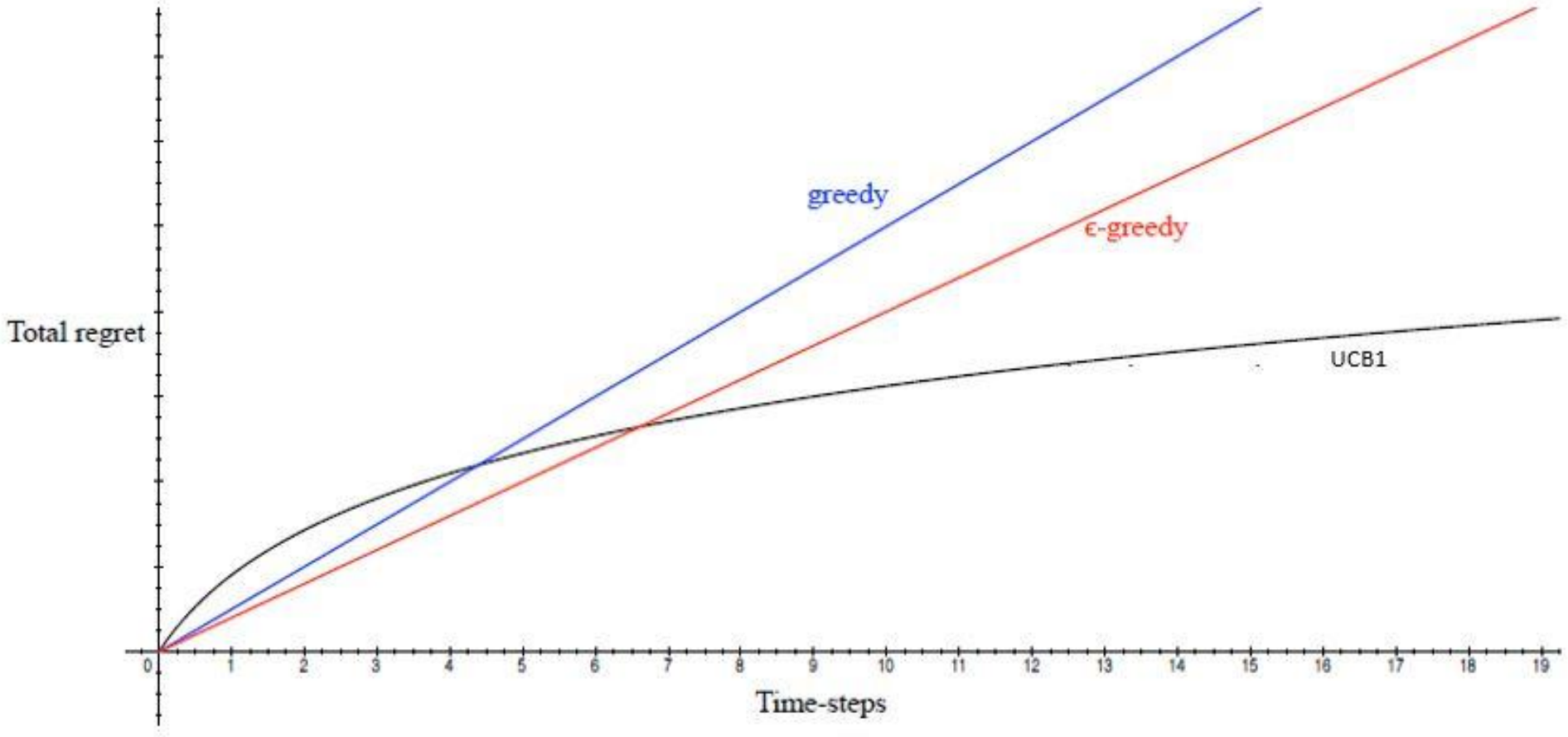
$N_t(a)$ ↑ $\sqrt{\frac{2 \log t}{N_t(a)}}$ ↓

t ↑ *With* $N_t(a)$ *constant* $\sqrt{\frac{2 \log t}{N_t(a)}}$ ↑

•The use of the natural logarithm means that the increases get smaller over time; all actions will eventually be selected, but actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time.

•This will ultimately lead to the optimal action being selected repeatedly in the end.

Regret Comparison



Among all the algorithms given in this article, only the UCB algorithm provides a strategy where the regret increases as log(t), while in the other algorithms we get linear regret with different slopes.

Namah Shivaya