Passive Reinforcement Learning Model

Formally, a model [1] consists of

- Set of all possible states ***S*** consisting of a start state and multiple goal state.
- Set of possible actions ***A***.
- Reinforcement signals or rewards denoted by ***r***, It maybe unknown to agent.
- Probability based model $\pi = P(\acute{s} \mid a, s)$ where $\acute{s}$,s∈ S,a∈ A.

The agent follows a fixed policy π (pi) which refers to a mapping that assigns some probability distribution over the actions to all possible histories.

The agents task is to find the optimal policy π that maximizes the long-run measure of reinforcement learning.

A passive learner agent uses Utility or usefulness of states. It is a metric used to determine how useful a state is; the higher the utility of a state is the more the chance of reaching the goal state.

Computationally utility of state s is defined as [1]:

$$U[s] = U[s] + \alpha(N(s)) * (r(s) + \gamma * U[\acute{s}] - U[s]))$$

Where the parameters are:

- U[s] is the utility measure of state s.
- N(s) is the number of trial or iteration.
- α is the learning rate with N(s) as input and is calculated as 1/ (N(s) +1).
- r(s) is the reward function, the reward achieved for reaching state s. It determine how strongly future values of U[s] influence current predictions.
- γ is the discount rate which is 1.

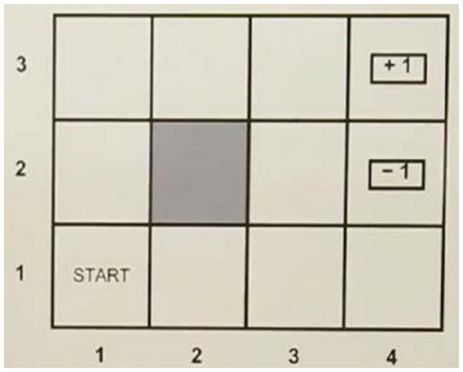
Roughly speaking we run the policy have a trial that goes through the state; when it gets to a terminal state, we start it over again at the start and run it again; and we keep track of how many times we visited each state, we update the utilities, and we get a better and better estimate for the utility.

What we expect from above is that after sufficient number of trials or episodes, the utilities of the states will converge to that of the optimal solution. Thereafter, we can construct the required optimal policy that maximizes the total utility measure of the problem.

Before we move to a sample problem which we solve by Reinforcement Learning, it is necessary to state some assumptions.

- Our agent is a passive learner, i.e. it computes utilities but does not learn and exploit its environment based on the learned elements.
- The policy remains fixed; after every iteration we do not modify our policy. In real world, an agent updates the probability of moving from one state to other. Based on its experiences.
- The stopping criteria we choose for our problem is fixed number of iterations (500). However, it can also be the how significant change in utilities is observed after some equations.

A 3 x 4 Gridworld is matrix with 3 rows and 4 columns depicted as G



There is one start state at G (1, 1). There are 2 goal states G (3, 4) with reward +1 and G (2, 4) with -1 reward.

Notice, no other state has reward. State G (2, 2) is wall and agent cant reach there.

The agent starts at G (1, 1) and its goal is to reach G (3, 4) and gain +1 reward and at the same time minimizing risk to fall into G (2, 4) where reward is -1.

The environment is unknown to agent. It does not know where +1 or -1 reward is.

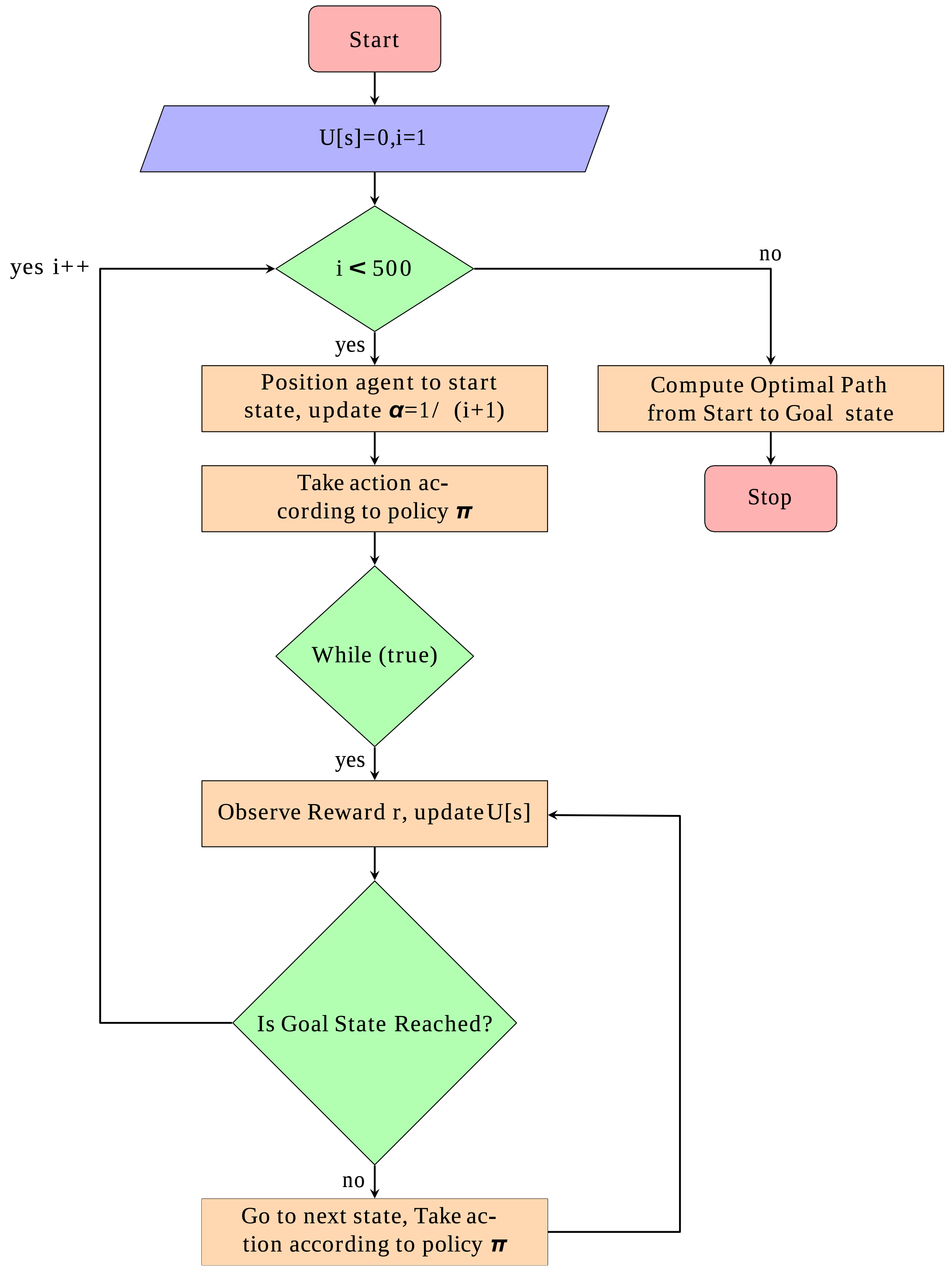
It may also be caught in loop between 2 or more states.

Now we state the formal algorithm we use to solve the Gridworld problem.

III.1 Algorithm

```
Data: :Probability -based policy pi of 3x4 Gridworld that decide where
        an agent can go from each state s in set S.Discount rate gamma
        that determine how strongly future values of U[s] influence
        current predictions.
Result: :Estimated Utility Of each state oF 3x4 Gridworld based on
        policy $\pi$,
Initialization: utilities U[s] of all non-goal states s to zero $\acute{s}$ is current
  state , s is previous state, set i = 0 (iteration counter).
while i in range(1,500) do
    position agent to start state;
    update learning rate $\alpha$ to 1/(i+1);
    take action according to policy $\pi$ move agent;
    while true do
        observe reward ras;
        $U[s] = U[s] + \alpha(N(s)) * (r(s) + \gamma * U[\acute{s}] - U[s]))$;
        if $\acute{s}$ is goal state then
            terminate as goal state is reached;
        else
            make s=$\acute{s}$as previous state;
            go to next state $\acute{s}$;
            take action according to policy ;
        end
    end
end
```

Flowchart


Start
U[s]=0,i=1
i < 500
no
yes i++
yes
Position agent to start state, update α=1/ (i+1)
Compute Optimal Path from Start to Goal state
Stop
Take action according to policy π
While (true)
yes
Observe Reward r, update U[s]
Is Goal State Reached?
no
Go to next state, Take action according to policy π

Active Reinforcement Learning Model

Unlike passive learning, an active learning agent considers what actions to take, what their outcomes may be and how it will affect rewards.

Some changes in Active learning are [2]:

- Transition from one state to another now depends on action as well i.e. we look at a value Q_{ij} that is how much useful it is to reach state I by taking action j.
- Also, we compute rewards as $r + \gamma * \max_{\acute{a}} Q(\acute{i}, \acute{j}) - Q(I, j)$.

Another issue that comes in Active learning is EXPLORATION [2].

An active learning agent needs to decide whether it should go for to that next state which gives highest reward, hoping that it will be the optimal path, or it should explore other state, exploring alternate paths.

There is a trade-off between this greed and exploration. Agent uses an exploration function f(q,n)[2] where n is the count of how many times a state-action pair was reached.

$$f(q, n) = \begin{cases} R, & \text{if } n < Ne \\ q, & \text{otherwise} \end{cases}$$

where R is the optimistic estimate of the best possible reward obtainable in any state. Ne is the fixed parameter that decides that at least Ne times each state-action pair must be tried.

Q-learning

We consider a model-free algorithm of Q-learning. We use $Q(s, a)$ to denote the value of doing action a in state s.

Q is directly related to utility U in passive learning as:

$$U(i) = \max_a Q(s, a)$$

Thus we take action based on highest Q-value. Update equation of Q-learning is[2]:

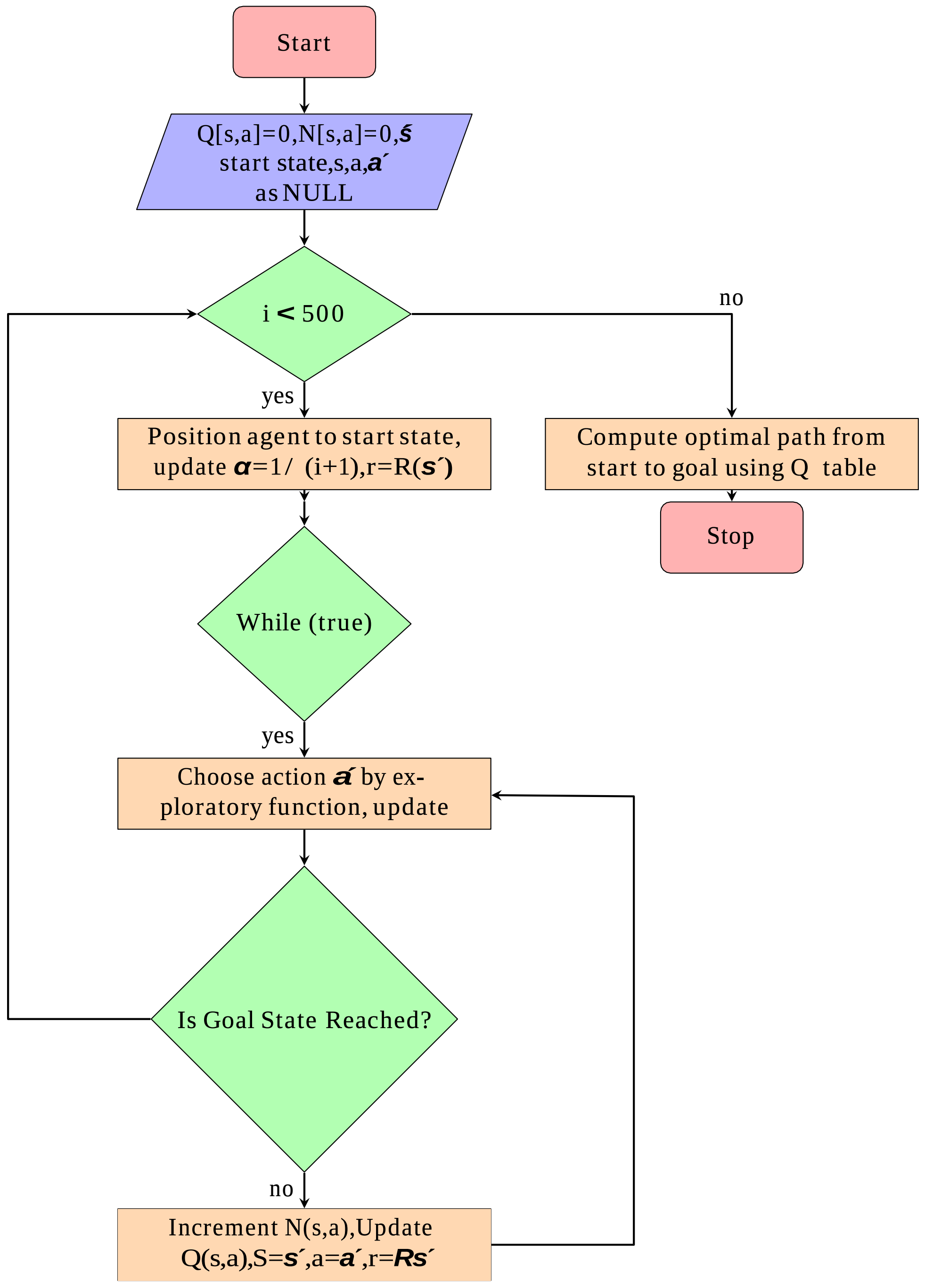
$$Q(s, a) = Q(s, a) + \alpha * N(s, a) * [r + \gamma * \max_{\acute{a}} Q(\acute{s}, \acute{a}) - Q(s, a)]$$

Where $N(s, a)$ is the number of times from state s , action a is taken.

Algorithm

```
Data: :A table Q for all possible states in rows and possible actions in
      columns, table N storing state action frequencies.Gamma as
      discount rate rewards at each state.
Result: :Update table Q and N and an optimal path from start state to
        goal. A graph depicting convergence achieved in Q-values.
Initializations: Set Q and N tables to zero, $\acute{s}$ as start state, current and
  previous actions $\acute{a}$ and a as none. Reward r as none, set i=0 (iteration
  counter).
while i in range(1,500) do
    position agent to start state;
    update learning rate $\alpha$ to 1/(i+1);
    Update reward r=$R(\acute{s})$;
    while true do
        Choose $\acute{a} = \arg\max_{a'} f(Q(s',i), N(\acute{s},i))$,where i is all possible
          actions;
        Set $\acute{s}$ based on action chosen.;
        if $\acute{s}$ is goal state then
            Terminate as goal state is reached;
        else
            Increment $N(s,a)$;
            Update $Q(s, a) =$
              $Q(s,as) + \alpha * N(s,a) * [r + \gamma * \max_{\acute{a}} Q(\acute{s},\acute{a}) - Q(s,a)]$;
        end
        Make s=$\acute{s}$ as previous state, a=$\acute{a}$ as previous action, r=$R(\acute{s})$;
    end
end
```

IV.1 Flowchart


Start
Q[s,a]=0,N[s,a]=0,ś start state,s,a,a´ as NULL
i < 500
no
yes
Position agent to start state, update α=1/ (i+1),r=R(s´)
Compute optimal path from start to goal using Q table
Stop
While (true)
yes
Choose action a´ by exploratory function, update
Is Goal State Reached?
no
Increment N(s,a),Update Q(s,a),S=s´,a=a´,r=Rs´