TRAFFIC BALANCING WHILE MAINTAINING OTHER KPIs: AN OPEN-ENDED INTELLIGENT SEARCH APPROACH.

By: Tirthankar Mittra

Contents:

Problem Description & Proposed Solution.

Load/traffic balancing among radio Base Stations while maintaining different system KPIs (like optimal throughput, transmission of data using less power, etc.) is one of the major challenges facing mobile operators due to the uneven distribution of user equipment (UEs), dynamic nature of channel conditions, etc. For example, in 4G systems 15% of network cells carry 50% of mobile traffic[4][5].

Here we discuss, how traffic balancing can be performed for Aerial base stations (**ABS, also known as unmanned aerial vehicle (UAV)-mounted base station (BS)**) using a form of deep reinforcement learning i.e. open-ended search[1][2][3] . We start training agents on simpler environments with lesser UEs and lesser obstructions and gradually increase the complexity & come up with novel environments. There is also a cross pollination step where niche champions from different environment are tested on other environments to check their performance.

To better understand this work I will briefly explain the basics of reinforcement learning, it's limitations in solving real life problems and how we used modifications on various forms of reinforcement learning to solve the problem of traffic balancing while maintaining various KPIs in case of UAV-mounted Base Stations.

Basics Of Reinforcement Learning.(1/2)

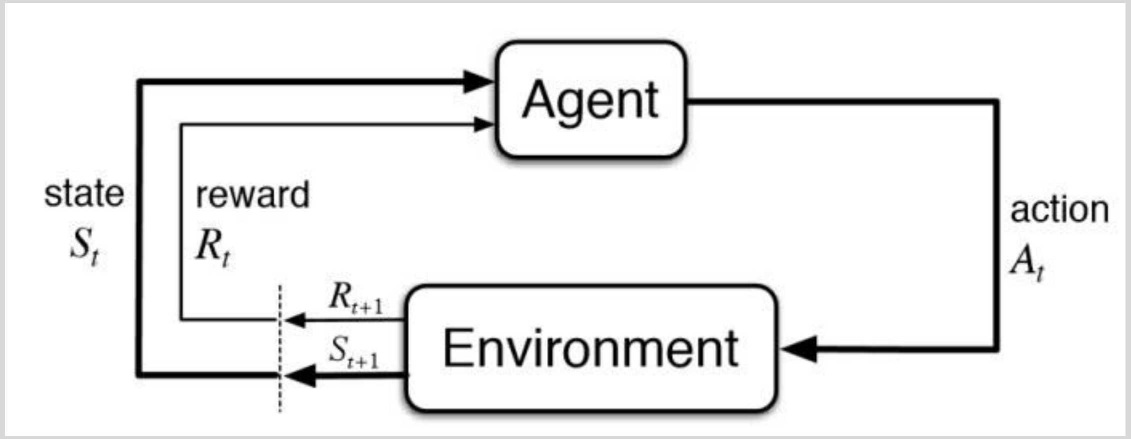


Fig. 1

The diagram shown above gives a general idea of Reinforcement learning.
- The **Agent** chooses a valid action **At** for a given state **St**.
- Based on the action the **Environment** transitions from the current state **St** to the next state **St+1**.
- The **Environment** also decides the reward **Rt+1** for the next state **St+1**.
- Based on **St+1** the **Agent** again chooses valid action **At+1**.

The job of the **Agent** is to choose actions which will lead to states providing the highest cumulative rewards in the future, the idea of future rewards is captured by **value function.**

The main objective of reinforcement learning is to a find optimal policy i.e. mapping between states of the environment and actions such that maximum total reward is generated over the entire duration of time.

Basics Of Reinforcement Learning.(2/2)

Elements of Reinforcement Learning that needs to be determined for successful implementation :-

1. Policy.

A *policy* defines the learning agent’s way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states by the agent.

2. Reward Signal.

A *reward signal* defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number, a *reward*. The aim to an agent in reinforcement learning is to maximize the future rewards.

3. Model Of Environment.

This is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave. The environment generates a reward and the next state when it receives an action from the agent in the current state.

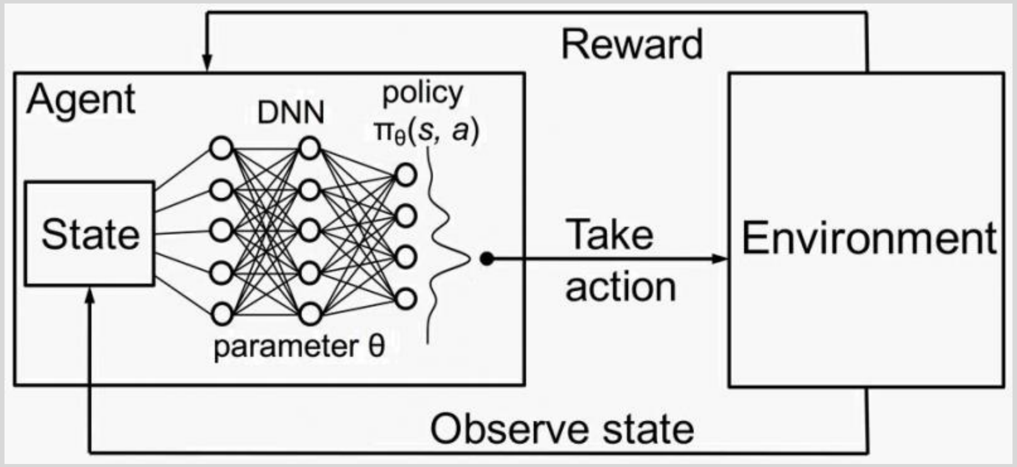
1

Limitations Of Reinforcement Learning.

In real life problems number of States(St) and Actions(At) become too large.

A famous off-policy Reinforcement Learning is Q-Learning. If the number of **States(St)** and **Actions(At)** are too large like in most real life problems, then the **[S X A]** matrix becomes intractable and the problem cannot be solved. A solution to the above problem is to used Deep Learning which approximates a function **f(St)** such that **At=f(St).** This variant is know as Deep Reinforcement Learning, here the policy is represented by matrix [**θ**].

~ Here **θ** is the weights of the different layers of the Deep Neural Network.



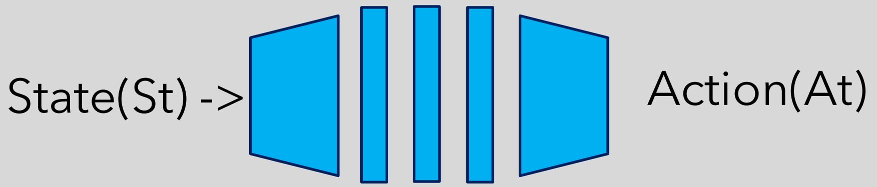


Fig. 2

The above diagram shows how the traditional Reinforcement Learning is modified (Fig. 1) when a deep neural network is used to figure out the policy i.e. State to Action mapping.

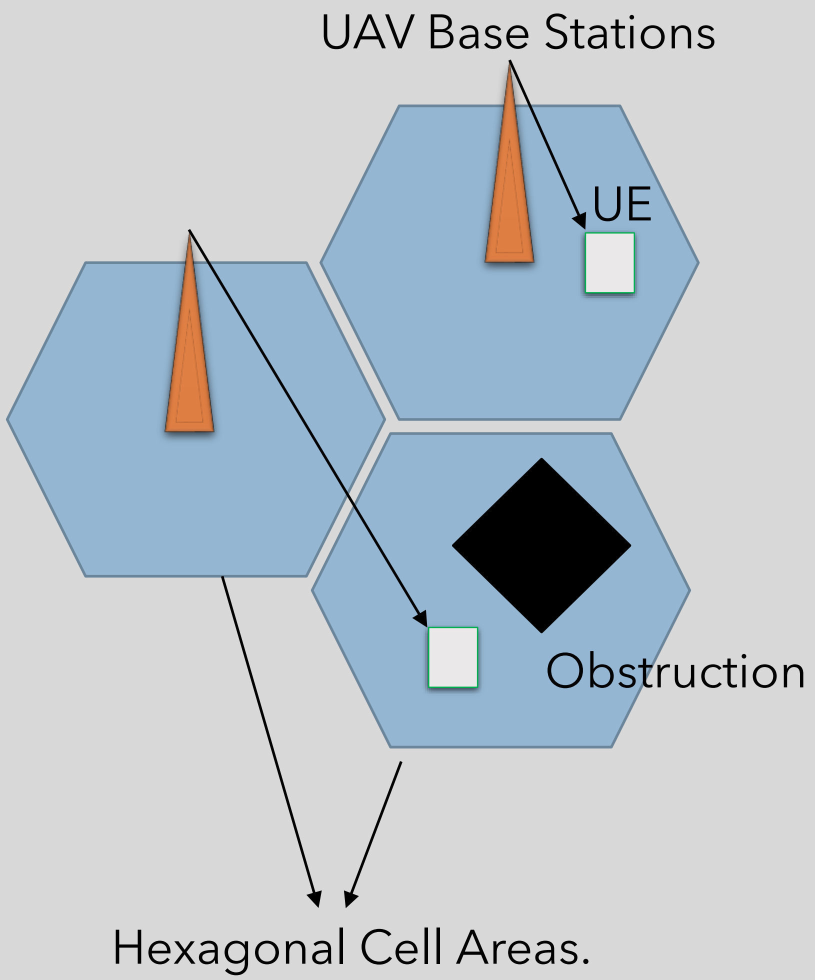
2 Limitations Of Reinforcement Learning.

Traditional Reinforcement Learning can produce solutions which get stuck in local optima.

An important motivating hypothesis in paper[1] (POET algorithm) is that the stepping stones that lead to solutions to very challenging environments are more likely to be found through a divergent, open-ended process than through a direct attempt to optimize in the challenging environment. Indeed, if we take a particular environment evolved in POET algorithm and attempt to run ES/direct optimization (the same optimization algorithm used in POET) on that specific environment from scratch, the result is often premature convergence to degenerate behavior. Direct optimization will not work in many scenarios because the solution will get stuck in some local optima.

The algorithm discussed here will generate new environments and then solve these environments. The solution to these environments will be called niche champions. We will start from a simple environment and slowly move to more complicated environments. At certain intervals of time the algorithm will do a cross pollination step and check whether niche champions from other environments are performing better in the current environment because niche champions from other environments might have learnt some useful policies which can produce better result in the current environment.

Schematic Diagram.



The diagram shows how using the POET algorithm, aerial base stations(ABS) will independently without communicating with each other occupy hexagonal cells such that it can serve the maximum number of UEs with optimal load balancing, with optimal throughput and QoS and without consuming excess energy(Tx and movement of the aerial BS) . The ABSs are expected to also take into account the obstructions(reflective surfaces shown in black) while placing themselves in strategic positions.

Our Model Description.(The Environment)

Initially our environment is simple with no obstructions

A unique environment is represented by the vector shown below:-

$\{ N_{\mathbf{UEs}}$, $\{(x_{UE0},y_{UE0}),(x_{UE1},y_{UE2}),\ldots\}$, $N_{\mathbf{obs}}$, $\{(x_{obs0},y_{obs0},l_{obs0},w_{obs0},ht_{obs0}),(x_{obs1},y_{obs1},l_{obs1},w_{obs1},ht_{obs1}),\ldots\}$ $\}$

Where,

- N_{UEs} = # of UEs.
- N_{obs} = # of Obstructions. (The surface of the obstructions are reflective & for simplicity we have assumed that the shape of the obstructions are rectangular).
- l_{obsi} , w_{obsi}, ht_{obsi} = Are dimensions (length, width and height) of the obstructions respectively.
- x,y = Represent the co-ordinate of either the UEs or the Obstructions.

To create new environments we start from a very simple environment and then mutate it to make the environment more complicated. The initial environment will have $N_{\mathbf{obs}} = 0$ and few number of UEs that need to be served using some pre-defined(fixed) number of aerial base stations(**N**) for a given geographical location.

Our Model Description.(The Reward Signal)

$$R_t = \alpha \sum_{i=1}^{N} T_i + \beta \sum_{i=1}^{N} \left| R_i - \frac{1}{N} \sum_{i=1}^{N} R_i \right| + \gamma \sum_{i=1}^{N} \sum_{j=1}^{UE_i} E_{ij} + \delta \sum_{i=1}^{N} |\Delta x_i| + \lambda \sum_{i=1}^{N} Tx_i$$

$\beta, \gamma, \delta, \lambda \leq 0$
$\alpha \geq 0$

$$E_{ij} = \begin{cases} 1, & if\ UE_j\ is\ not\ served\ with\ minimum\ throughput\ r_{ij}\ by\ the\ ith\ BS\ to\ which\ it's\ attached. \\ 0, & otherwise \end{cases}$$

Based on the importance one wants to give to the 5 components of the reward signal their values are decided.

At time 't' we get a total reward 'R_t' for all the Base Stations as shown in the above equation, for the i-th Base Station we drop the summation and use it in our algorithm. We have considered that there are **N** Base Stations in our model.

- R_i is the resource utilization of the i-th Base Station.
- T_i is the throughput of the i-th Base Station.
- Tx_i is the Transmit power of the i-th Base Station.
- $|\Delta_{x_i}|$ is the distance moved by the i-th Base Station.

All the above values R_i, T_i,… are measured at each time stamp 't'.

Our Model Description.(State S_t)

For the i-th Aerial Base Station we construct the state Si_t as described in this slide.
~ Each aerial BS will be attached with LIDAR sensors to detect obstructions.

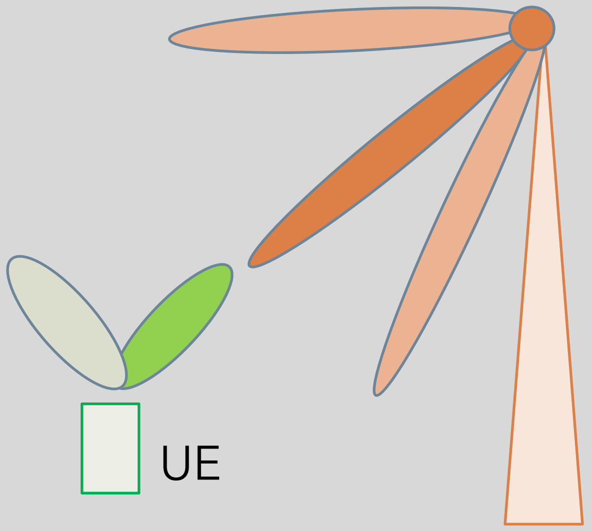
$$Si_t = \{ (x_i, y_i), beamId_i[\quad], LIDAR_i[\quad], MCS_i[\quad] \}$$

Coordinates of the i-th Base Station

$$beamId_i[j] = \frac{n_{ij}}{UE_i}$$

# of UEs attached to the j-th beam in the i-th Base Station

Total # of UEs attached to the i-th Base Station



$MCS_i[j]$ = fraction of UEs with the j-th MCS in the i-th Base Station

OUR MODEL.(Action A_t)

OPTION -1

If Aerial Base Stations can only be placed in certain coordinates in a hexagonal cell then the action for the i-th Base Station is given by the following equation.

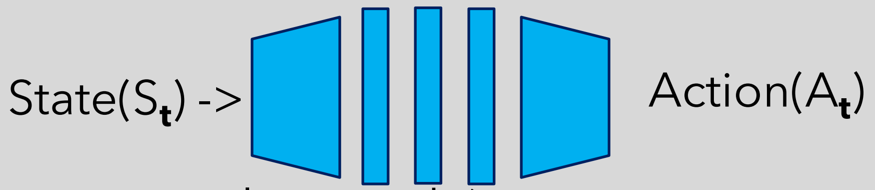
$$Ai_t = \begin{cases} A' = (xi, yi), & with\ probability\ 1 - \epsilon(t) \\ A\ , & with\ probabilty\ \epsilon(t) \end{cases}$$

The value of $\epsilon(t)$ decreases over time. This helps us explore different actions earlier in the algorithm

A random action chosen from the set of all possible actions in state S_t

The best possible coordinate as determined by the policy at time 't' for the i-th Base Station.

$(x_i, y_i) \rightarrow Is\ the\ coordinate\ where\ the\ ith\ Aerial\ Base\ Station\ must\ be\ placed.$



Deep Neural Network (Logistic Regression)

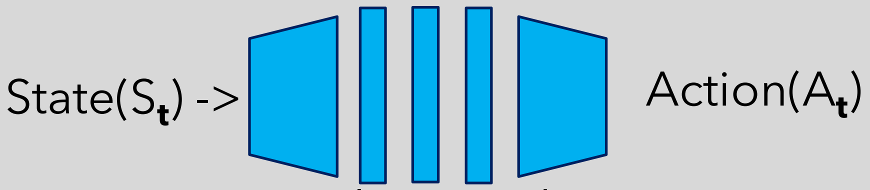
OUR MODEL.(Action A_t)

OPTION -2

If Aerial Base Stations can be placed in any coordinates in a hexagonal cell.

$$A_{i_t} = (x_i, y_i) \longrightarrow \boxed{\text{The best possible coordinate as determined by the policy at time 't' for the i-th Base Station.}}$$

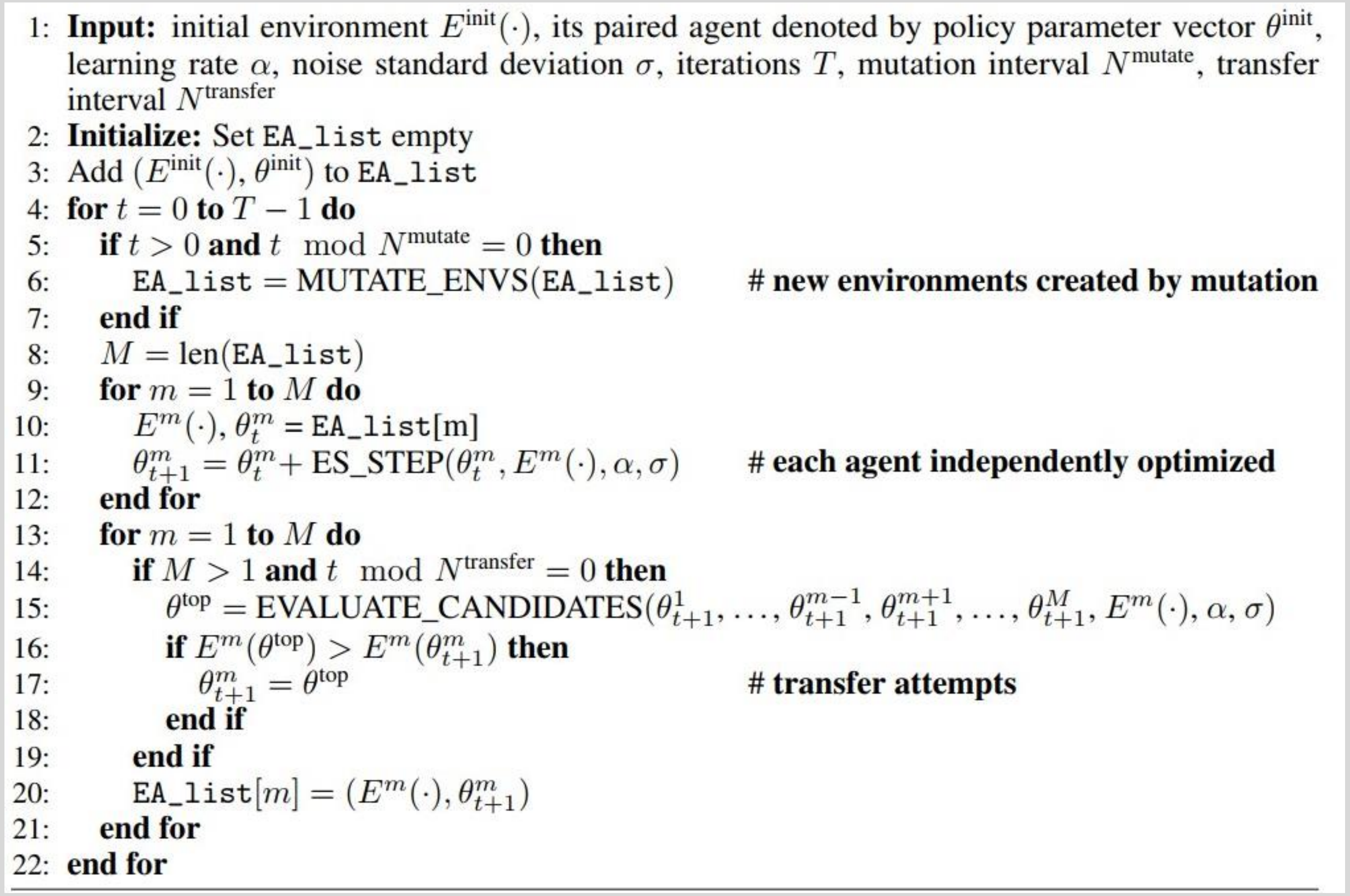
(x_i, y_i) $\rightarrow$ *Is the coordinate where the ith Aerial Base Station must be placed.*



Deep Neural Network (Regression)

Algorithm Used.

Main Code:



```
Input: initial environment $E^{\text{init}}(\cdot)$, its paired agent denoted by policy parameter vector $\theta^{\text{init}}$, learning rate $\alpha$, noise standard deviation $\sigma$, iterations $T$, mutation interval $N^{\text{mutate}}$, transfer interval $N^{\text{transfer}}$
Initialize: Set EA_list empty
Add $(E^{\text{init}}(\cdot), \theta^{\text{init}})$ to EA_list
for $t = 0$ to $T - 1$ do
  if $t > 0$ and $t \mod N^{\text{mutate}} = 0$ then
    EA_list = MUTATE_ENVS(EA_list)        # new environments created by mutation
  end if
  $M$ = len(EA_list)
  for $m = 1$ to $M$ do
    $E^m(\cdot), \theta_t^m$ = EA_list[m]
    $\theta_{t+1}^m = \theta_t^m +$ ES_STEP$(\theta_t^m, E^m(\cdot), \alpha, \sigma)$        # each agent independently optimized
  end for
  for $m = 1$ to $M$ do
    if $M > 1$ and $t \mod N^{\text{transfer}} = 0$ then
      $\theta^{\text{top}}$ = EVALUATE_CANDIDATES$(\theta_{t+1}^1, \ldots, \theta_{t+1}^{m-1}, \theta_{t+1}^{m+1}, \ldots, \theta_{t+1}^M, E^m(\cdot), \alpha, \sigma)$
      if $E^m(\theta^{\text{top}}) > E^m(\theta_{t+1}^m)$ then
        $\theta_{t+1}^m = \theta^{\text{top}}$        # transfer attempts
      end if
    end if
    EA_list$[m] = (E^m(\cdot), \theta_{t+1}^m)$
  end for
end for
```

This is the main code of our algorithm. **N**

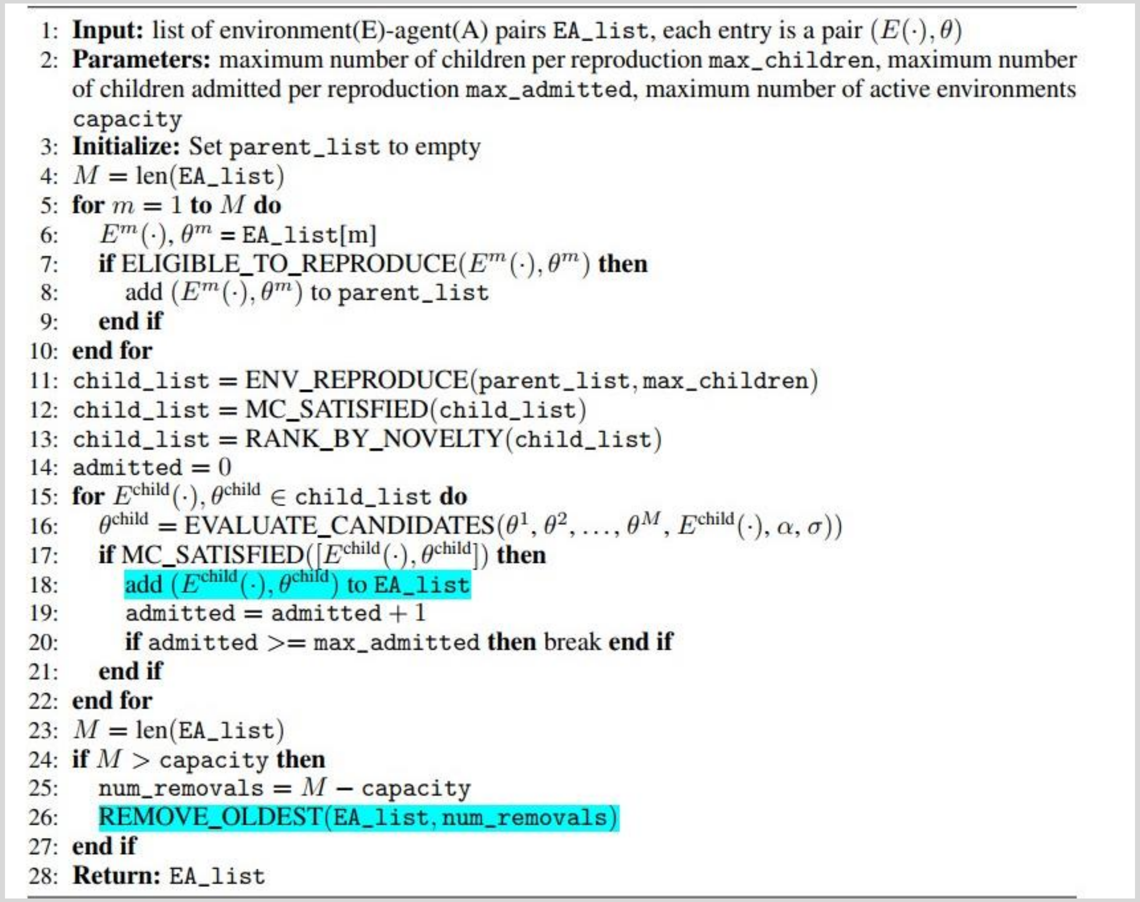
This is the main code of our algorithm. **N^{mutate}** is a constant time interval after which existing environments are mutated. **N^{transfer}** is a constant time interval after which niche champions of other environments are evaluated of an environment. The entire simulation takes **T** time steps/iterations.

Implementation details of the following function are present in the next slides.

- MUTATE_ENVS(…)
- EVALUATE_CANDIDATES(…)
- ES_STEP(…)

Algorithm Used.

MUTATE_ENVS(…)



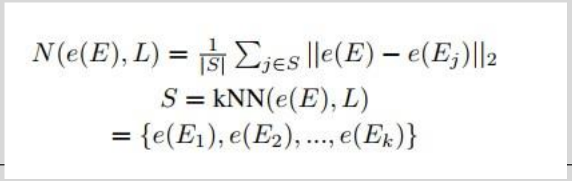
```
Input: list of environment(E)-agent(A) pairs EA_list, each entry is a pair $(E(\cdot), \theta)$
Parameters: maximum number of children per reproduction max_children, maximum number of children admitted per reproduction max_admitted, maximum number of active environments capacity
Initialize: Set parent_list to empty
$M$ = len(EA_list)
for $m = 1$ to $M$ do
    $E^m(\cdot), \theta^m$ = EA_list[m]
    if ELIGIBLE_TO_REPRODUCE($E^m(\cdot), \theta^m$) then
        add ($E^m(\cdot), \theta^m$) to parent_list
    end if
end for
child_list = ENV_REPRODUCE(parent_list, max_children)
child_list = MC_SATISFIED(child_list)
child_list = RANK_BY_NOVELTY(child_list)
admitted = 0
for $E^{\text{child}}(\cdot), \theta^{\text{child}} \in$ child_list do
    $\theta^{\text{child}}$ = EVALUATE_CANDIDATES($\theta^1, \theta^2, \ldots, \theta^M, E^{\text{child}}(\cdot), \alpha, \sigma$)
    if MC_SATISFIED([$E^{\text{child}}(\cdot), \theta^{\text{child}}$]) then
        add ($E^{\text{child}}(\cdot), \theta^{\text{child}}$) to EA_list
        admitted = admitted + 1
        if admitted >= max_admitted then break end if
    end if
end for
$M$ = len(EA_list)
if $M >$ capacity then
    num_removals = $M$ − capacity
    REMOVE_OLDEST(EA_list, num_removals)
end if
Return: EA_list
```

EA_list is a queue of size **capacity** which holds active environment and niche champion pair. If the queue size exceeds the oldest entry(most likely solved) is archived into the database.

max_children -> # of mutated children per parent environment.

The new $E^{child}(.)$,θ^{child} is passed through MC_SATISFIED(.) to eliminate trivial or too difficult environments.

The generated **child_list** is also ranked/sorted by novelty when passed to RANK_BY_NOVELTY(.) using the following algorithm(L2 norm + kNN).



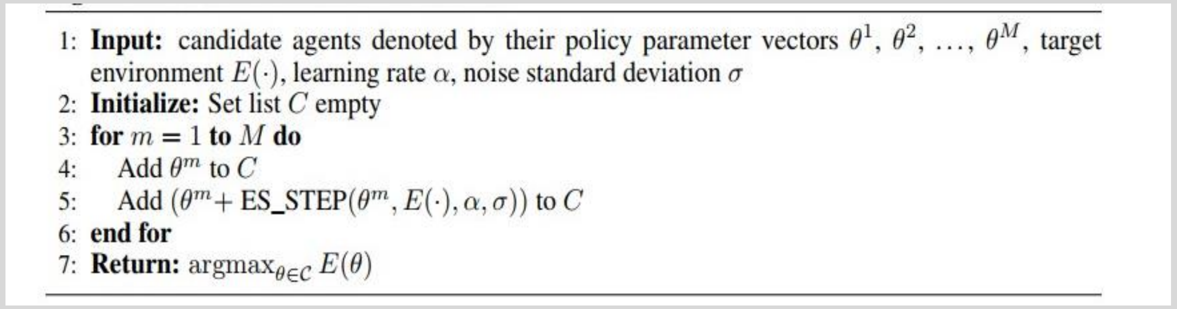
$$N(e(E), L) = \frac{1}{|S|} \sum_{j \in S} ||e(E) - e(E_j)||_2$$
$$S = \text{kNN}(e(E), L)$$
$$= \{e(E_1), e(E_2), \ldots, e(E_k)\}$$

Algorithm Used.

The function implementation of **EVALUATE_CANDIDATES(…)** from the previous slide is shown here. This function finds the best candidate (at time 't') for an environment by checking all niche champions and niche champions with one **ES_STEP(…)** in the given environment **E(.)** .

Implementation details of **ES_STEP(…)** is presented in the next slide.

EVALUATE_CANDIDATES(…)

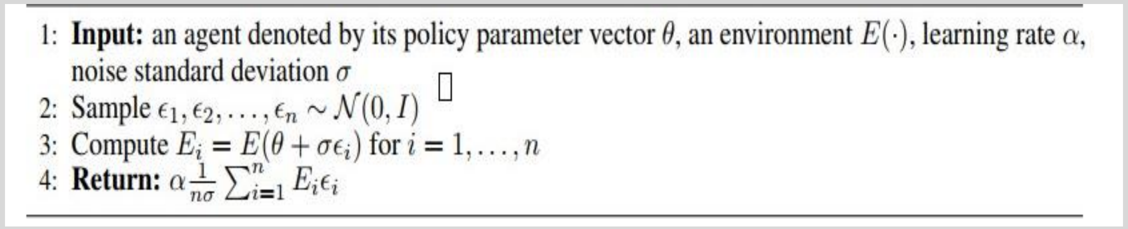


```
1: Input: candidate agents denoted by their policy parameter vectors $\theta^1$, $\theta^2$, $\ldots$, $\theta^M$, target
   environment $E(\cdot)$, learning rate $\alpha$, noise standard deviation $\sigma$
2: Initialize: Set list $C$ empty
3: for $m = 1$ to $M$ do
4:     Add $\theta^m$ to $C$
5:     Add ($\theta^m$+ ES_STEP($\theta^m, E(\cdot), \alpha, \sigma$)) to $C$
6: end for
7: Return: $\text{argmax}_{\theta \in C}\, E(\theta)$
```

Algorithm Used.

ES_STEP(…) function updates the policy θ according to normal evolutionary strategy E(.) . The policy θ is a higher dimensional vector on which an isotropic gaussian distribution is placed from which 'n' samples are taken. E(θ) gives the reward in the environment E(.) for the given policy so in "step 4" of the algorithm we are basically moving the policy θ in the direction which will yield the highest reward in the environment E(.) .

ES_STEP(…)



```
1: **Input:** an agent denoted by its policy parameter vector $\theta$, an environment $E(\cdot)$, learning rate $\alpha$,
   noise standard deviation $\sigma$
2: Sample $\epsilon_1, \epsilon_2, \ldots, \epsilon_n \sim \mathcal{N}(0, I)$
3: Compute $E_i = E(\theta + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
4: **Return:** $\alpha \frac{1}{n\sigma} \sum_{i=1}^{n} E_i \epsilon_i$
```

Conclusion:

Open-Ended Search produces complicated environments i.e. traffic balancing with lots of UEs and obstructions with ‘N’ aerial Base Stations . The algorithm also produces diverse and novel solutions (niche champions) to these environments which cannot be solved using normal Reinforcement Learning or Evolutionary Search algorithms.

Note: In our model we have chosen specific types/values of R_t, A_t & S_t to demonstrate open-ended search, these parameters of our model can be modified as per different requirements.

References:

[1] Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions Rui Wang Joel Lehman Jeff Clune* Kenneth O. Stanley* Uber AI Labs San Francisco, CA 94103 ruiwang, joel.lehman, jeffclune, kstanley@uber.com *co-senior authors

[2] Joel Lehman and Kenneth O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation, pages 211–218, 2011.

[3] Jonathan C. Brant and Kenneth O. Stanley. Minimal criterion coevolution: A new approach to open-ended search. In *Proceedings of the 2017 on Genetic and Evolutionary Computation Conference (GECCO)*, pages 67–74, 2017 .

[4] K. Attiah *et al.*, "Load Balancing in Cellular Networks: A Reinforcement Learning Approach," *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, 2020, pp. 1-6, doi: 10.1109/CCNC46108.2020.9045699.

[5] Harri Holma and Anti Toskala, LTE advanced; 3GPP solutions for IMT-Advanced. John Wiley & Sons. 2012.

Thank You

By: Tirthankar Mittra