# FINAL DOCUMENTATION FOR WPCN-UAV PAPER

## 1.   newGeneticDeviceVisitAI.py

1-1.    **Goal:** For each cluster, to determine whether the brute-force search of the optimal path for the movement of UAV

1-2.    **Brute-force search of optimal path:**



The picture on the left describes the start location of UAV **(S)** and the fixed locations of each device **(A, B and C).**

When doing Brute-force search, we check the total length of path when the UAV moves **"S-A-B-C", "S-A-C-B", ..., "S-C-B-A"**.

**S-A-B-C** : distance **4.6**

**S-A-C-B** : distance **5.6**

**S-B-A-C** : distance **4.0** **(OPTIMAL)**

**S-B-C-A** : distance **5.2**

**S-C-A-B** : distance **4.4**

**S-C-B-A** : distance **4.6**

The figure above is the result of our Brute-force search. For **N** devices, we should search for **N!** cases. In the situation above, we searched for **3! = 6** cases and we got the result that **the optimal path** with **the shortest total Euclidean distance** is **S-B-A-C** whose **distance is 4.0.**

## 1-3.    Swap-basic search:

Swap-basic search is the method to find the shortest total Euclidean distance with swapping the device visit sequence. **(suppose that the number of device is 8)**

With start location of UAV **S**, device **d1, d2, ..., d8** where **CURRENT (updated)** visit sequence is **S->d1->d2->...->d8**

While True: // until convergence
    For each **case** in **{d1, d2, d3}, {d1, d2, d3, d4}, ..., {d4, d5, d6, d7}, {d5, d6, d7, d8}, {d6, d7, d8}** // total (8 – 1 = 7) cases
        If **d1 in case** and **len(case) == 3:** // include start
          swapCondition = (S->**d2->d1**->d3) is shorter than (S->**d1->d2**->d3)
          if swapCondition is True:
            **(d1, d2) = (d2, d1)**
        else if **d8 in case** and **len(case) == 3:** // include end
          swapCondition = (d6->**d8->d7**) is shorter than (d6->**d7->d8**)
          if swapCondition is True:
            **(d7, d8) = (d8, d7)**
        **else:** // other cases
          swapCondition = (case[0]->**case[2]->case[1]**->case[3]) is shorter than (case[0]->**case[1]->case[2]**->case[3])
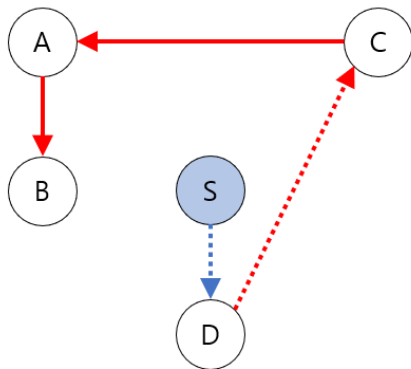          if swapCondition is True:
            **(case[1], case[2]) = (case[2], case[1])**
    **Terminate** if **do not updated** // terminate when reached convergence

For example,



Sequence: **S-A-D-C-B**

Step 1.
S-A-D-C : distance **5.8**
S-D-A-C : distance **5.2 (shorter)**

**S-D-A-C is shorter, so SWAP (A, D), (D, A)**

Sequence: **S-D-A-C-B (updated)**

Step 2.
D-A-C-B : distance **6.4**
D-C-A-B : distance **5.2 (shorter)**

**D-C-A-B is shorter, so SWAP (A, C), (C, A)**
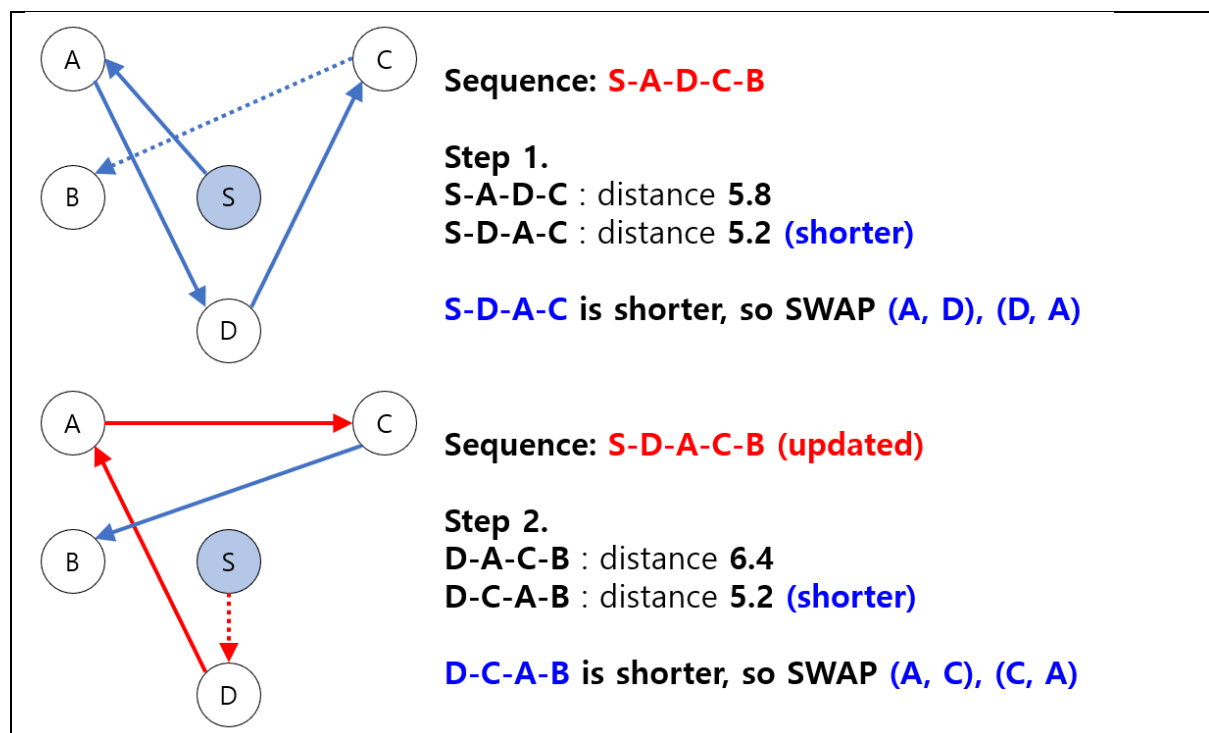
**Sequence: S-D-C-A-B (updated)**

**Step 3.**
**C-A-B** : distance **3.0** **(shorter)**
**C-B-A** : distance **3.2**

**C-A-B** is shorter, so **DO NOT SWAP**
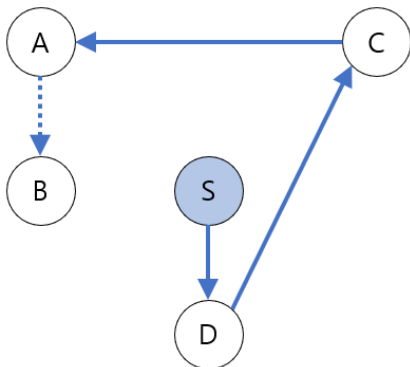
**Sequence: S-D-C-A-B**

**Step 4.**
**S-D-C-A** : distance **5.2** **(shorter)**
**S-C-D-A** : distance **5.8**

**S-D-C-A** is shorter, so **DO NOT SWAP**
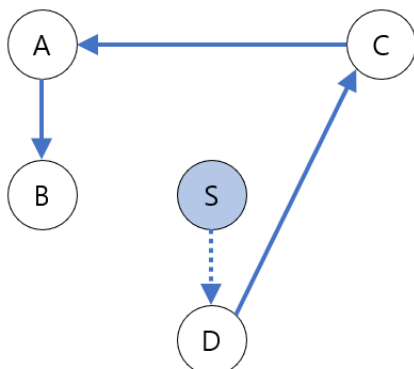
**Sequence: S-D-C-A-B**

**Step 5.**
**D-C-A-B** : distance **5.2** **(shorter)**
**D-A-C-B** : distance **6.4**

**D-C-A-B** is shorter, so **DO NOT SWAP**
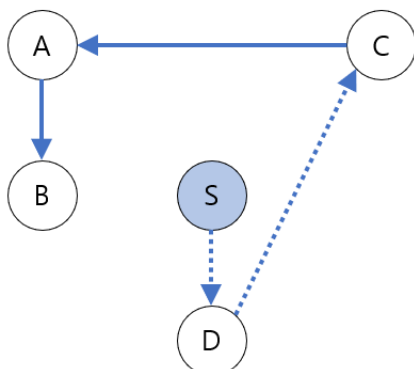
**Sequence: S-D-C-A-B**

**Step 6.**
**C-A-B** : distance **3.0** **(shorter)**
**C-B-A** : distance **3.2**

**C-A-B** is shorter, so **DO NOT SWAP**
**Terminate** because **NO UPDATE**

**1st** iteration : **Step 1 ~ Step 3**
**2nd** iteration : **Step 4 ~ Step 6**

## 1-4.    Deep Learning Approach

Use Deep Learning to **determine whether the Brute-force search is needed**:

| Original data | Movement **S->d1->d2->...->d8** |
|---|---|
| Input data | For each segment of the movement **S->d1, d1->d2, ..., d7->d8,**<br>    1.  **x distance** of the movement<br>    2.  **y distance** of the movement<br>For each segment of the movement **S->d1, d1->d2, ..., d7->d8,**<br>    3.  **sin(A)** where A is the angle of the movement<br>    4.  **cos(A)** where A is the angle of the movement<br>    5.  **Euclidean distance** of the movement |
| Output data | $$\mathbf{min}\left(\mathbf{1.0}, \mathbf{log_{1.75}}\frac{D_{SB}}{D_{BF}}\right) \rightarrow always\ between\ 0.0\ and\ 1.0$$ Where $D_{SB}$ is the total length of the movement path **after applying Swap-Basic** method, and $D_{BF}$ is the total length of the movement path of **the optimal movement** derived by **Brute-force search** (NOTE: $D_{SB} \geq D_{BF}$ is always true because $D_{BF}$ **is the optimal distance**)<br><br><br><br>**Sequence: S-D-C-A-B**<br><br>**Step 6.**<br>**C-A-B** : distance **3.0 (shorter)**<br>**C-B-A** : distance **3.2**<br><br>**C-A-B is shorter, so DO NOT SWAP**<br>**Terminate because NO UPDATE**<br><br>In the situation of **1-3** (the picture above), the movement path of **S->D->C->A->B** is derived by **Swap-Basic** method and whose **total distance is 6.2**, so $D_{SB} = \mathbf{6.2}$<br><br><br><br>In the situation of **1-2**, (the picture on the left side) the movement path of **S->B->A->C** is **optimal (derived by Brute-force method)** and whose **total distance is 4.0**, so $D_{BF} = \mathbf{4.0}$<br><br>**S-B-A-C** : distance **4.0 (OPTIMAL)**<br><br>**NOTE: The two cases above are DIFFERENT (so, DO NOT think of 6.2/4.0)** |

Suppose that **the start location of the UAV is the CENTROID of all the devices** and **the number of devices is 8**

**Main idea (important): Brute-force search is needed if $D_{SB}/D_{BF}$ value is large enough**

**Example about the input data:**
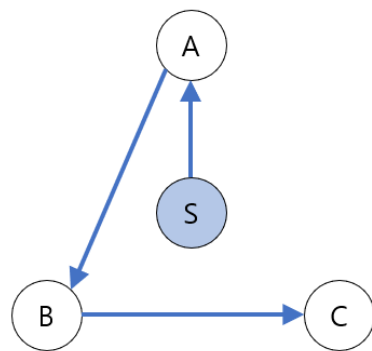


movement path: **S->A->B->C**

location of the start location of UAV **(S)** and each device **(A, B and C)** is,

➔ S (x=**0.0**, y=**0.0**)

➔ A (x=**0.0**, y=**+1.0**)

➔ B (x=**-0.866**, y=**-0.5**)

➔ C (x=**+0.866**, y=**-0.5**)

For each segment **S->A, A->B and B->C**,

|  | S->A | A->B | B->C |
|---|---|---|---|
| X distance | X1 = 0.0 | X2 = -0.866 | X3 = +1.732 |
| Y distance | Y1 = +1.0 | Y2 = -1.5 | Y3 = 0.0 |
| Angle A | 90 degree | 240 degree | 0 degree |
| Sin(A) | Sin1 = +1.0 | Sin2 = -0.866 | Sin3 = 0.0 |
| Cos(A) | Cos1 = 0.0 | Cos2 = -0.5 | Cos3 = +1.0 |
| Euclidean distance | Dist1 = 1.0 | Dist2 = 1.732 | Dist3 = 1.732 |

The form of the input data is,

(**X1, Y1, X2, Y2, X3, Y3**, **sin1, cos1, dist1, sin2, cos2, dist2, sin3, cos3, dist3**)

So, the input data for this case is,

(**0.0, +1.0, -0.866, -1.5, +1.732, 0.0**, **+1.0, 0.0, +1.0, -0.866, -0.5, +1.732, 0.0, +1.0, +1.732**)

## 1-5.    Model Genetation using the idea of 1-4

1. Create **16,000** cases of the clusters (refer to variable **'times'** of **newGeneticDeviceVisitAI.py**)
➔ The number of devices in the cluster is as the table below

```
if __name__ == '__main__':

    # numpy setting
    np.set_printoptions(edgeitems=20, linewidth=200)

    input_data      = []
    output_data     = []
    times           = 16000
    deviceCountList = []
```

2. Do Deep Learning for these cases using **12,960 (81%) training data, 1,440 (9%) validation data and 1,600 (10%) test data**
3. The model is generated based on the deep learning result.

The number of devices: **(probability)**

| 4 devices | 5 | 6 | 7 | 8 devices |
|---|---|---|---|---|
| 15% | 22% | 26% | 22% | 15% |

### 2-1. Configuration

In **settings.txt**, you can modify the variables below:

|  | description | Default value |
|---|---|---|
| width | Width in meter | 50 |
| height | Height in meter | 50 |
| L | The number of clusters | 5 |
| devices | The total number of devices in the all clusters | 30 |
| N | Flight period (the number of time slots) | 30 |
| iters | The number of iterations | 1,200 |
| clusteringAtLeast | cluster contains devices **at least int(this number * average)** | 0.67 |
| clusteringAtMost | cluster contains devices **at most int(this number * average)** | 1.34 |

The total number of cases (clusters) to test: **L * iters (default 5 * 1,200 = 6,000)**

In the setting above, the average number of device for each cluster is **(device / L) = 30 / 5 = 6**, so the cluster contains at least **int(0.67 * 6) = 4 devices**, and at most **int(1.34 * 6) = 8 devices**. It is similar to the distribution of the number of the devices in **1-5.**

### 2-2. Mode Settings

| Static (static_swapOnly) | Same as the **static strategy** from Tang et al<br>➔ Tang J, Song J, Ou Jea. Minimum Throughput Maximization for Multi-UAV Enabled WPCN: A Deep Reinforcement Learning Method. IEEE Access 2020. |
|---|---|
| **Train swap only** (train_swapOnly) | Find the **device-visit sequence (movement path)** using **Swap-basic search**, and then visit devices using this sequence. Compute the common throughput based on the sequence. |
| **Train gen visit** (train_genVisit) | Find the **device-visit sequence (movement path)** using **Swap-basic search**, and then input the movement path to **the model generated in 1-5**.<br>If the **output value** of the model is **at least (threshold = 0.5)**, then find the device-visit sequence using **Brute-force search** and then visit devices using this sequence.<br>Otherwise, visit devices using the sequence derived by **Swap-basic search**.<br>Compute the common throughput based on the **sequence of device-visit**. |

In () is **the prefix of the trajectory image file name**. (**static_swapOnly**_trajectory_iter0000.png for example)

## 2-3. The number of iterations for each mode

| mode | The number of iterations | Total generated clusters when iters=1,200, L=5 |
| --- | --- | --- |
| Static | iters // 2 | (1,200 // 2) * 5 = 3,000 |
| Train swap only | iters | 1,200 * 5 = 6,000 |
| Train gen visit | iters | 1,200 * 5 = 6,000 |

## 2-4. interpretation of the result

**Static mode**

minThroughputList_static2208010048_swapOnly_iter_1200_L_0005_devs_0030_N_0030.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

mean: 0.01836036916346323, std: 0.007857989538148612, nonzero: 3000

Time: **2022.08.01 00:48**

Mode: **Static** (refer to **static**2208010048_swapOnly)

Configurations: iters=**1200**, L=**5**, devices=**30**, N=**30**

Average value of the common throughput: **0.018360**

Standard deviation of the common throughput: **0.007858**

The total number of clusters whose common throughput is non-zero: **3,000** / 3,000 (100.0%)

**Train swap only**

minThroughputList_train2208010048_swapOnly_iter_1200_L_0005_devs_0030_N_0030.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

mean: 0.02500243346203403, std: 0.011188922783291363, nonzero: 5994|

Mode: **Train swap only** (refer to **train**2208010048_**swapOnly**)

Average value, standard deviation of the common throughput: **0.025002, 0.011189**

The total number of clusters whose common throughput is non-zero: **5,994** / 6,000 (99.9%)

**Train gen visit**

minThroughputList_train2208010048_genVisit_iter_1200_L_0005_devs_0030_N_0030.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

mean: 0.026219649894172997, std: 0.010857653190466882, nonzero: 5999

Mode: **Train gen visit** (refer to **train**2208010048_**genVisit**)

Average value, standard deviation of the common throughput: **0.026220, 0.010858**

The total number of clusters whose common throughput is non-zero: **5,999** / 6,000 (99.98%)

## 3. Running

### 3-1. run newGeneticDeviceVisitAI.py first

| | | |
|---|---|---|
| 📁 Genetic_Visit_AI_model | 2022-08-01 오전 12:47 | 파일 폴더 |

If you already have Genetic_Visit_AI_model directory with the model, you can skip this stage.



Otherwise, run **python newGeneticDeviceVisitAI.py**.

### 3-2. run throughputTest_NewGenetic.py



Run the command **python throughputTest_NewGenetic.py.**