Deep Learning-based Optimal Placement of a Mobile HAP for Wireless Powered Communication Networks

Hong-Sik Kim and Inwhee Joe, Hanyang University, Department of Computer Software

***Abstract*—HAP (Hybrid Access Point) is a node in Wireless Powered Communication Networks (WPCN) that can distribute energy to each wireless device and also can receive information from these devices. Recently, mobile HAPs have emerged for efficient network use and the throughput of the network depends on their location. There are two kinds of metrics for throughput, that is, sum throughput and common throughput, each is the sum and minimum value of throughput between a HAP and each wireless device, respectively. Likewise, two types of throughput maximization problems can be considered, sum throughput maximization and common throughput maximization. In this paper, we discuss about the latter. We propose deep learning-based methodology to maximize common throughput by optimally placing a mobile HAP for WPCN. Our study implies that deep learning can be applied to optimize a simple function of common throughput maximization, which is a convex function or a combination of a few convex functions. The experimental results show better performance than mathematical methods for smaller maps.**

# INTRODUCTION

I

N WPCN, there is Access Point (AP) mechanism [1] which contains energy nodes (EN), wireless devices (WD) and access points (AP). First, energy nodes send energy to each wireless device. When ENs receive the energy, it sends information to APs using the energy. That is, ENs send energy to WDs and WDs send information to APs. We can encapsulate AP and EN into Hybrid Access Point (HAP) and so can describe HAP mechanism. In this mechanism, the HAP sends energy to each WD, and each WD sends information to the HAP. The HAP allocates time for sending energy to each WD and itself, and for sending information to each WD, so time allocation for itself and each WD is also an important issue.

Because the distance between the HAP and each WD is different among each WD, there is an energy efficiency gap between the WDs caused by the difference of throughput for each WD. That is, a WD near to the HAP receives more energy and uses less energy to transmit information, and another WD far from the HAP receives less energy but uses more energy to transmit information. To solve this unfairness problem, the worst case, a WD which receives the least energy and uses the most energy, is very important. In this case, we use the concept of common throughput which is the minimum value of throughput among the throughput values of each WD, and we concentrate on maximizing the common throughput value in the WPCN environment.

This paper introduces a methodology to place HAPs in WPCN environment to maximize common throughput when time allocation is optimized, by using deep learning, and shows that this methodology is meaningful to solve this problem and shows better performance than the mathematical methodology already studied.

In [2], Suzhi Bi and Rui Zhang researched about the placement optimization of Energy and Information Access Point in WPCN using bisection search method, Greedy algorithm, Trial-and-error method, and alternating method for joint AP-EN placement. There can be more than 1 HAPs WDs in the supposed environment of this paper. Its methodology repeatedly adds HAPs and check if each WD satisfies conditions in the environment.

# HAP PLACEMENT MODEL

## Overview

**Fig. 1.** is the flow chart of the HAP placement model. The model is composed of three phases. First, making data is to create training and test data. Next, training using data is to process the data to convert to training and test data for the deep learning model, and train using the model. Last, finding the best point is to find the best HAP placement point using the throughput map derived from this model.

**Fig. 2.** describes the system architecture of the model. Mobile HAP can be placed at any place in the environment and can move to any other place in the environment. The goal is to maximize common throughput that is defined as the minimum throughput

|  |
| --- |
|  |
| Fig. 1. Flow chart of HAP placement model. Note that the number after the description of each phase means the order of the stage whose data is used in this stage. |
|  |
| Fig. 2. The system architecture of our model. Wireless devices are placed in the environment and we can represent the location of WDs as a WDs placement map. A mobile HAP is placed in the environment and the throughput value of the environment is calculated and represented as a throughput map. |

between the HAP and each WD by optimizing the HAP placement. **WDs placement map** means the square map with rows and columns, and K blocks in the map, randomly set at the training stage, contains a WD. There are no WDs with the same position in the map. From now on, we will call this . **Throughput map** means the square map with rows and columns, and each block contains the throughput value where the HAP is located in this block of . From now on, we will call this . **Best throughput point** means the position of HAP that maximizes throughput value in , derived from our model, so it could be not a real position that maximizes the throughput value. We will call this .

## Making Data

To make s where is the sum of the number of training and test maps, first define a square map with rows and columns. Then repeat placing a WD at the randomly selected point times. To make s using these s, place HAP at each point in s and compute throughput for the location of HAP and each WD using **Algorithm 1** because the throughput is computed using **Formula (6)**

|  |
| --- |
| **Algorithm 1.** Finding throughput for a HAP location and each WD’s location |
| **input:** list of the location of each WD *wdList*, location (y-axis and x-axis) of HAP *HAPpoint*  **output:** throughput value *finalThrput*  *lr* 5.0;  *timeList* [1.0, 1.0, …, 1.0];  **for** *i* = 1 **To** 1000 **do**  *tpChange* [];  *thrput* *getThrput*(*wdList*, *HAPpoint*, *timeList*);  **for** *each* *x* **in** {*HAP*, *wdList*} **do**  *timeListCopy* *timeList*;  *timeListCopy*[*x*] *timeListCopy*[*x*] + 1;  *newThrput* *getThrput*(*wdList*, *HAPpoint*, *timeListCopy*);  *difThrput* ;  append max(0.01\*, ) to *tpChange*;  *timeList*[*x*] *timeList*[*x*] \* ;  **end**  **end**  *finalThrput* *getThrput*(*wdList*, *HAPpoint*, *timeList*);  **Return** *finalThrput;* |
| ***getThrput*(*wdList*, *HAPpoint*, *timeList*):**  (, ) (Y axis of *HAPpoint*, X axis of *HAPpoint*);  *sumOfTime* sum(*timeList*);  *HAPtime* *timeList*[0]/*sumOfTime*;  *result* *INFINITE*;  **for** each *WD i* **do**  (, ) (Y axis of *wdList*[*i*], X axis of *wdList*[*i*]);  *dist* ;  *chargeTime* *timeList*[1+*i*]/*sumOfTime*;  *throughput* *chargeTime* \*;  *result* min(*result*, *throughput*);  **end**  **Return** *result*; |
|  |

in [3]. getThrput function finds optimal time allocation given . Because we supposed that where , and where is the distance from the HAP and each WD, this formula can be converted into (1). To prevent divide by 0 error and consider the limit of throughput, we supposed that distance is 1.0 when actual distance is less than 1.0.

|  |  |
| --- | --- |
|  | (1) |

Then, because s and s are saved as text files, the model must read them before using them.

## Training

First, make input data for training and testing based on , supposing that the number of training and testing data is and each. The model considers first maps as training data and next maps as test data. The input data is an map whose value at each block is -1 when a WD is on this block and 0 otherwise. Then make output data for training based on s corresponding to s.

The output data is an map whose value at each block is processed version of the common throughput value derived by **Algorithm 1** where the HAP is on this block. We define as the value on the block of -th row and -th column of the training output map corresponding to . The following is the processing procedure. First, find maximum throughput value for each training output map , and then divide each value by . Last, process each value at each block using (2).

|  |  |
| --- | --- |
|  | (2) |

Then train using input data s and corresponding output data s using the deep learning model described in **Fig. 3.** with Adam optimizer[4] with learning rate 0.0001 and 1,000 epochs.

## Finding the Best Points

Using test input data, the model finds best point for HAP placement. For each test input data created based on , input these data into the model trained in **2.3.** and get output maps corresponding to . For each value at each block in each output map is converted by (3) using the inverse function of the sigmoid function, to convert them from from into form, where is the estimated common throughput value.

|  |  |
| --- | --- |
|  | (3) |

Then, for each output map, the model finds the maximum value among values in blocks of this map. Let’s call row and column axis of this value in the map and each, and call the maximum value . Then the row axis and column axis of optimal HAP location are computed

|  |
| --- |
|  |
| Fig. 3. Architecture of Deep Learning Model for Common Throughput Maximization: We use Convolutional Neural Network (CNN) [5] for our methodology. |
|  |
| Fig. 4. Decision algorithm for and . For each picture on the left and right, , , , , , and means the y and x axis of the environment, respectively. Each rectangle with the value means the common throughput value at the point of these y and x axis. |

by (4) and (5) each, and is computed by (6), described in **Fig. 4**.

|  |  |
| --- | --- |
|  | (4) |
|  | (5) |
|  | (6) |

If is greater than , moves down from original position and otherwise it moves up. Similarly, if is greater than , moves right and otherwise it moves left. Because original common throughput and can be converted into each other by just a linear transmission, there is no difference of and between when converted into and do not convert into any other form.

# EXPERIMENTS AND RESULTS

## Experiment Design and Test Metrics

**Fig. 5.** is the flow chart for the experiment. For each optimal HAP location for each test map derived by **2.4** corresponding to s, first compute common throughput value using this point. Because we use s only for computing the difference when testing, the throughput maps as created as the output of the model, called s in this section, are not equal to corresponding s. Then compare the throughput value with , the maximum common throughput value among all points in corresponding . Then the test metrics are defined as and computed using (7), (8) and (9).

|  |
| --- |
|  |
| Fig. 5. Flow chart of design of testing. Note that 1-1 after the description of stage 1-2 means the data of stage 1-1 is used in stage 1-2. |

|  |
| --- |
| **Algorithm 2.** Solving (20) in the original paper |
| **input:** and , as defined in the original paper  **output:** optimal value of (by Gradient Descent Method)  *lr* ;  **for** *i* = 1 **To** 7000 **do**  min, *k* = 1,…,*K*;  ;  ;  min, *k* = 1,…,*K*;  min, *k* = 1,…,*K*;  + *lr*\*;  + *lr*\*;  **end**  **Return** ; |

|  |  |
| --- | --- |
|  | (7) |
|  | (8) |
|  | (9) |

means average common throughput for each test map with corresponding , and means maximum common throughput value for each throughput map corresponding to each test map, and means the rate between the sum of and the sum of for all test maps. It also means the rate between and . We also define performance rate as (10) meaning how well our methodology is compared to the methodology used in the original paper, and the original paper in (10)means [2].

|  |  |
| --- | --- |
|  | (10) |

can be larger than 1.0 because means the average of largest value among the value at discrete blocks from corresponding , but means the average of common throughput value with non-discrete HAP location.

## Experimental Environment

The computer system information for our experiment is as the following. The operating system is Window 10 Pro 64bit (10.0, build 18363), system manufacturer is LG Electronics, the system model is 17ZD90N-VX5BK, the BIOS is C2ZE0160 X64, the processor is Intel® Core™ i5-1035G7 CPU @ 1.20GHz (8 CPUs), ~1.5GHz, and the memory is 16384MB RAM. The programming language is Python 3.7.4, and used NumPy [6], Tensorflow [7] and Keras as libraries. You can download the experiment code from <https://github.com/WannaBeSuperteur/2020/tree/master/WPCN>.

## Experimental Results

**Table I** describes (%) and values for our methodology and the methodology in the original paper. We used and with for the methodology in [2], and the algorithm to solve (20) in [2]is described in **Algorithm 2**. For our methodology, value increases when the number of WDs increases and decreases when the size of maps increases, and decreases when both the number of WDs and the size of maps increases. For the methodology in the original paper, increases when the size of maps increases, but has no significant correlation with the number of WDs, and as of our methodology, decreases when both the number of WDs and the size of maps increases. **Table II** shows the values of

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TABLE I  CT.RATE AND CT.AVERAGE VALUES OF OUR METHODOLOGY AND THE METHODOLOGY IN THE ORIGINAL PAPER | | | | | | | | | |
| **Our Methodology** | | CT.RATE (%) | | | | CT.AVERAGE | | | |
| Size / WDs | | 6 WDs | | 10 WDs | | 6 WDs | | 10 WDs | |
| 8x8 | | 93.8477 | | 98.6283 | | 0.010175 | | 0.006757 | |
| 12x12 | | 88.9782 | | 91.1735 | | 0.002626 | | 0.001631 | |
| 16x16 | | 80.0318 | | 89.6754 | | 0.000796 | | 0.000525 | |
| **Methodology in the Original Paper** | | CT.RATE (%) | | | | CT.AVERAGE | | | |
| Size / WDs | | 6 WDs | | 10 WDs | | 6 WDs | | 10 WDs | |
| 8x8 | | 65.5779 | | 71.9627 | | 0.007110 | | 0.004930 | |
| 12x12 | | 89.1676 | | 90.3931 | | 0.002631 | | 0.001617 | |
| 16x16 | | 100.6565 | | 96.8606 | | 0.001001 | | 0.000568 | |
| The upper table is about our methodology and lower table represents CT.RATE and CT.AVERAGE about the methodology in the original paper. Size means the size of the board representing the environment of WPCN. | | | | | | | | | |
| TABLE II  THE VALUE OF CT.AVGMAX AND PR | | | | | | | | | |
| **CT.AVGMAX** | | | | | **PR** | | | | |
| Size / WDs | 6 WDs | | 10 WDs | | Size / WDs | | 6 WDs | | 10 WDs |
| 8x8 | 0.010842 | | 0.006851 | | 8x8 | | 1.431087 | | 1.370548 |
| 12x12 | 0.002951 | | 0.001789 | | 12x12 | | 0.997876 | | 1.008633 |
| 16x16 | 0.000994 | | 0.000586 | | 16x16 | | 0.795098 | | 0.925819 |
| The table on the left and on the right represents CT.AVGMAX and PR respectively, for each option (Size and the number of WDs). There can be some errors for PR values because the values of CT.AVERAGE have less significant figures than 6. | | | | | | | | | |
|  | | | | | | | | | | |
| Fig. 6. Line Chart Vestion of Table I and Table II. For the 4 tables in the left, upper 2 table shows the result of CT.RATE and CT.AVERAGE of our methodology and lower 2 table shows the result of them of the methodology in the original paper. The 2 tables in the right show the values of CT.AVGMAX and PR, respectively. | | | | | | | | | | |
|  | | | | | | | | | | |
| Fig. 7. Comparison of CT.RATE(%) of our methodology and the methodology in the original paper. Our methodology shows better performance than the methodology in the original paper for size 8x8, but the two methods show nearly the same performance for size 12x12, and our methodology shows worse performance for size 16x16. | | | | | | | | | | |

and for each size and number of WDs. The unit for size is one block, as mentioned in section II. For example, the size of 12x12 means that the environment contains 12 rows and each row contains 12 blocks. decreases when both the number of WDs and the size of maps increases and decreases when the size of maps increases, but has no significant correlation with the number of WDs. For smaller sizes, our methodology shows significantly better performance () than the methodology in the original paper, but for 12x12 size, these two methods show almost the same performance. (), and for 16x16 size, our methodology shows worse performance. () **Fig. 6.** is the line chart representation of **Table I** and **Table II**, and **Fig. 7.** is the bar chart for comparison of our methodology and the methodology in the original paper.

Our method shows higher for smaller maps and the methodology in the original paper shows higher for larger maps. The reason for the former is that common throughput is usually depending on the WDs near the boundary of the environment, and the influence on the learning of them decreases for larger maps because the number of blocks influencing the learning is larger. These WDs enlarge the minimum value of maximum possible distance between the HAP and each WD. The reason for the latter is that the location of WDs can be ‘skewed’ for smaller maps, and the discreteness of their location has more influence than larger maps, so the methodology in the original paper is not so accurate.

# CONCLUSION

We showed that our deep learning-based method shows better performance than the mathematical method in the original paper [2] when the size is smaller than 12x12. Although our method may show worse performance if the size is larger than 12x12, our approach to find the optimal placement and time allocation for HAP using deep-learning is meaningful because there is no attempt to apply deep-learning to this problem yet. There are some limits for our study. First, our study has an advantageous point for our method that it uses only 1 HAP which is fitted to the experimental environment, but the method in the original paper may and commonly uses more than 1 HAPs. Second, we studied with just a few conditions, 3 options for map size and 2 options for the number of WDs. So some future research should be done for many options in terms of the map size and the number of WDs, and the number of HAPs.

References

1. Suzhi Bi, Yong Zeng, and Rui Zhang, “Wireless Powered Communication Networks: An Overview”, IEEE, available online at <https://arXiv:1508.06366>.
2. Suzhi Bi, Member, IEEE, and Rui Zhang, “Placement Optimization of Energy and Information Access Points in Wireless Powered Communication Networks”, IEEE Transactions on wireless communications, VOL. 15, NO. 3, MARCH 2016.
3. Hyungsik Ju and Rui Zhang, “Throughput Maximization in Wireless Powered Communication Networks”, available online at <https://arXiv:1304.7886v4>.
4. Diederik P. Kingma, Jimmy Lei Ba, “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION”, ICLR 2015, available online at <https://arXiv:1412.6980>.
5. Saad ALBA WI, and Saad AL-ZA WI, “Understanding of a Convolutional Neural Network”, ICET 2017, available online at <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8308186>.
6. IEEE, “The NumPy Array: A Structure for Efficient Numerical Computation”, Scientific Python, available online at <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5725236>.
7. Mart´ın Abadi, Paul Barham, Jianmin Chen et al, “TensorFlow: A system for large-scale machine learning”, Google Brain, available online at <https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf>.