Deep Neural Networks for Linear Sum Assignment Problems

Mengyuan Lee, Yuanhao Xiong, Guanding Yu, and Geoffrey Ye Li

***Abstract*—Many resource allocation issues in wireless communications can be modeled as assignment problems and can be solved online with global information. However, traditional methods for assignment problems take a lot of time to find the optimal solutions. In this letter, we solve the assignment problem using machine learning approach. Specifically, the linear sum assignment problems (LSAPs) are solved by the deep neural networks (DNNs). Since LSAP is a combinatorial optimization problem, it is first decomposed into several sub-assignment problems. Each of them is a classification problem and can be solved effectively with DNNs. Two kinds of DNNs, feed-forward neural network and convolutional neural network, are implemented to deal with the sub-assignment problems, respectively. Based on computer simulation, DNNs can effectively solve LSAPs with great time efficiency and only slight loss of accuracy.**

***Index Terms*—Linear sum assignment problem, deep neural network, machine learning, resource allocation.**

I. Introduction

MANY resource allocation issues in wireless communications can be regarded as assignment problems, which allocate a number of items (e.g., jobs) to a number of machines (or workers) in the best possible way to minimize or maximize certain utility functions. As a special case of assignment problems, the linear sum assignment problem (LSAP) is a classical combinatorial optimization problem and can be widely found in many wireless communication systems, such as mode selection for device-to-device communications [1], joint resource allocation in multiple-input multiple-output (MIMO) systems [2], and unlicensed channel allocation for LTE systems [3].

There have already existed many algorithms for LSAPs. In 1950s, Kuhn has proposed the Hungarian algorithm by combining the graph theory and the duality of linear programming [4], which is one of the first algorithms for LSAPs. Bertsekas has proposed a massively parallelizable algorithm called auction algorithm [5], which is one of the fastest algorithms to find the optimal solution to LSAPs. Ramakrishnan *et al*. [6] have modified the Karmarkar interior point algorithm for the linear programming problem and developed the approximate dual projective algorithm. Due to the dynamic nature and the expansionary size of many applications, heuristic algorithms that can achieve near-optimal solutions under tight time constraints have also been developed, such as the greedy randomized adaptive algorithm [7] and the deep greedy switching algorithm [8].

On the other hand, machine learning (ML) has currently emerged as a promising technique for different fields with striking performance and success. Several recent studies have applied the ML approach to solve mathematical optimization problems. In [9], deep neural networks (DNNs) are trained to learn the input/output relations of a wireless resource management problem with continuous variables. In [10], collaborative DNNs (C-DNNs) have been developed to deal with the link scheduling optimization problem, which can be recast as decentralized classification problem.

Inspired by that, we apply the ML technique to solve LSAPs in this letter. Specifically, we try to utilize DNNs to derive a simpler but general learning based optimization approach for LSAPs. We fist decompose LSAP into several sub-assignment problems. With this decomposition process, LSAP can be transformed into several classification subproblems. Then we adopt supervised learning scheme to work out each sub-assignment problem by treating the nonlinear mapping of the input/output relation as a black box. We make use of two different network architectures: feed-forward neural network (FNN) and convolutional neural network (CNN). Finally, we implement a low-complexity heuristic algorithm to combine the outputs of each sub-assignment problem together. Furthermore, we apply our approach to some instances to evaluate the performance. The results suggest that DNNs are able to solve LSAPs with great time efficiency and only slight loss of accuracy. The ML approach can help solve LSAPs online without incurring much complexity since the time-consuming training process can be implemented offline.

We organize the rest of this letter as follows. We will introduce LSAP and the decomposition process in Section II. The learning based approach will be developed in Section III. Section IV presents instance test and performance analysis. Finally, we conclude this letter in Section V.

II. LSAP and Decomposition Process

In this section, we will first introduce LSAP and then propose the decomposition procedure.

A. LSAP

Generally, LSAP deals with the problem of how to assign n jobs to n people in an optimal way. It can be formulated as a 0-1 linear program, as following

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{1}$$

Manuscript received March 24, 2018; revised May 2, 2018; accepted May 29, 2018. Date of publication June 4, 2018; date of current version December 14, 2018. This work was supported by the Natural Science Foundation of China under Grant 61671407 and Grant 61725104. The associate editor coordinating the review of this paper and approving it for publication was X. Chu. *(Corresponding author: Guanding Yu.)*

M. Lee, Y. Xiong, and G. Yu are with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: mengyuan_lee@zju.edu.cn; xiongyh@zju.edu.cn; yuguanding@zju.edu.cn).

G. Y. Li is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250 USA (e-mail: liye@ece.gatech.edu).

Digital Object Identifier 10.1109/LWC.2018.2843359

subject to

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n, \tag{1a}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n, \tag{1b}$$

$$x_{ij} \in \{0, 1\}, \quad i, j = 1, \ldots, n, \tag{1c}$$

where c_{ij} is the cost of assigning job *i* to person *j*, and x_{ij} is the decision indicator, i.e., $x_{ij} = 1$ means job *i* is assigned to person *j*, and $x_{ij} = 0$ otherwise. Many resource allocation problems in wireless communications are similar or equivalent to problem (1) [1]–[3].

Using cost matrix $C = \{c_{ij}\}$ as input and decision matrix $X = \{x_{ij}\}$ as output is the most natural way to implement DNNs for LSAPs. In this way, we need to construct a DNN with n^2 input nodes and n^2 output nodes, which leads to a very huge network with the growth of *n*. In addition, LSAP is neither classification nor regression problem, and no existing loss functions can be directly applied to it. Therefore, we propose to decompose the problem into several sub-assignment problems in the following, which can recast problem (1) to classification sub-problems and scale down the size of the DNNs at the same time.

B. Problem Decomposition

The basic idea of the decomposition is described as follows. The original problem is a global decision making process with the cost matrix *C* as the input and the decision matrix *X* as the output. We decompose the problem into *n* sub-assignment problems. The *j*-th one solves an assignment problem on how to assign one of *n* jobs to people *j*, i.e., the cost matrix *C* is known and the decision vector for person *j*, X_j, is to be worked out. Constraints (1a) and (1c) remain the same in each sub-assignment problem, guaranteeing that X_j is a vector with 1 one and $n - 1$ zeros. Therefore, X_j is a one-hot vector, which is the same as the output of the classification problems. By this decomposition process, we transform the original problem in (1) into *n* classification sub-problems, which can be solved consequently by DNNs.

The decomposition process mentioned above only concerns with constraints (1a) and (1c). However, constraint (1b) is not taken into consideration, which guarantees that one job can only be assigned to one people. When we deal with the *n* sub-assignment problems separately, there may exist some collisions that one job may be assigned to different people simultaneously. Fortunately, we adopt a supervised learning approach where the training samples do not have any collision. Hence, by reproducing these solutions, the collision probability is very low and will eventually disappear when the accuracy approaches 100%. This motivates us to develop a low-complexity heuristic algorithm instead of dedicating to find the optimal algorithm for such a low collision probability. In particular, we use the greedy collision-avoidance rule. If job *i* is assigned to persons j_1 and j_2 simultaneously, we assign job *i* to person j_1 when $c_{ij_1} < c_{ij_2}$. More simulation verifications on the effectiveness of the proposed greedy collision-avoidance rule will be presented in Section IV.

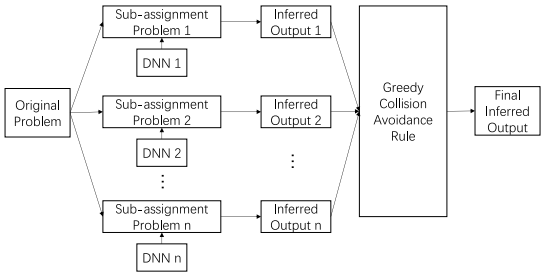

Fig. 1. System model.

III. Deep Learning Based Approach

In the above, we have introduced LSAP and its decomposition that recasts original LSAP into several classification sub-problems. In this section, we will introduce the deep learning based method for LSAP.

A. System Architecture

The architecture of the learning based approach for LSAP is illustrated in Fig. 1. For each LSAP with size *n*, we first decompose it into *n* sub-assignment problems as mentioned in Section II. Then, we use learning models which are trained offline to solve each sub-assignment problem. Finally, we implement the greedy collision-avoidance rule to get the inferred output for LSAP.

In the following, we will focus on how to get good trained models for each sub-assignment problem.

B. Data Generation

As we adopt a supervised learning scheme, the first thing for model training is to get the labels or solutions of the training data. First, we generate the cost matrix $C^{(m)}$ following a specific distribution, where *m* is the index of training samples. Then we use the Hungarian algorithm to get the optimal decision matrix $X^{(m)}$ for each $C^{(m)}$. For the *j*-th sub-assignment problem, we use the tuple $\{C^{(m)}, X_j^{(m)}\}$ as the *m*-th training sample. By repeating the above process, we can generate the entire data set. Furthermore, all tuples in the data set can be split randomly into the training and the validation sets. The validation set plays an important role in model selection and over-fitting avoidance. Generally, 70-90% of the tuples are assigned into the training set. In the same way, we can generate testing samples as well.

C. Network Architecture

After the dataset is prepared well, we need to choose specific network architectures of DNNs. Generally, deep FNNs and deep CNNs are two widely used DNN architectures for time-independent problems. FNNs are the baseline architectures and can be employed to all kinds of problems. CNNs, on the other hand, are more complicated and have good performance in computer vision and natural language processing. In our problem, the cost matrix *C* is an $n \times n$ matrix, which resembles an image with $n \times n$ pixels. The CNN proves to be efficient to deal with the image classification problem. Therefore, we use it in our work to extract and learn the features of the cost matrix *C* for classification.

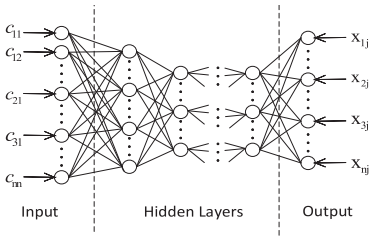

Fig. 2. DNN structure for *j*-th sub-assignment problem.

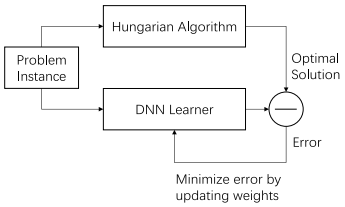

Fig. 3. Data generation and training process.

As shown in Fig. 2, for the *j*-th sub-assignment problem, the input of the DNNs is *vec*(*C*), i.e., the vectorization of *C*, and the output of the network is the decision vector X_j. And the numbers of layers and neurons in each layer need to be chosen according to values of *n*.

D. Training Stage

We use the entire training set to update the weights of DNNs as depicted in Fig. 3. As mentioned above, X_j is one-hot and we can utilize the cross entropy as the loss function of the proposed DNNs. To further improve the training performance, we use L2 regularization to limit the complexity of the network and avoid over-fitting as well. Hence the loss function for the *j*-th sub-assignment problem can be expressed as

$$L_j = -\frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{n} X_{ij}^{(m)} \log y_{ij}^{(m)} + \frac{\lambda}{2M}\sum_{w\in\Omega_j} w^2, \forall j, \quad (2)$$

where $y_{ij}^{(m)}$ is the inferred output for the *j*-th sub-assignment problem, *M* is the batch size, Ω_j is the set of all weights in DNNs for the *j*-th sub-assignment problem, and λ is the regularization parameter, which is set to be 0.01%. Meanwhile, we adopt the adaptive moment estimation (Adam) algorithm as the optimization algorithm, which is based on adaptive estimates of lower-order moments [11]. Furthermore, we implement the moving average technique to get more robust models and set the decay rate to be 0.99. Finally, learning rate and batch size are another two critical parameters for training stage. On the one hand, large batch size decelerates the convergence, whereas small one may lead to unstable convergence behavior. On the other hand, large learning rate may incur a high validation error, whereas low one slows down convergence speed. To strike a balance between convergence speed and accuracy, we find out how the training loss defined in (2) in the validation set changes with different batch sizes and learning rates during the first 500 epochs and choose the appropriate parameters according to the results.

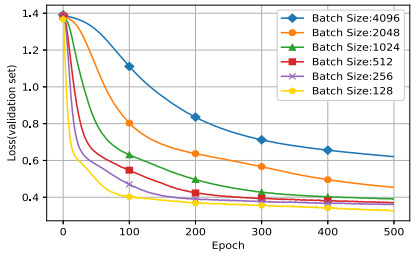

Fig. 4. Batch size selection.

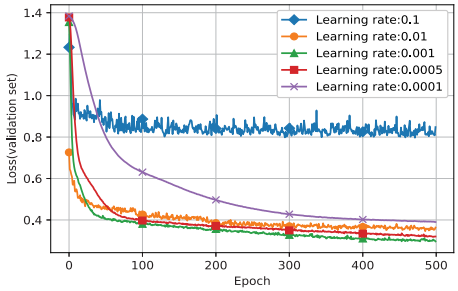

Fig. 5. Learning rate selection.

IV. Test Results

In this section, we first test a small-scale scenario where $n = 4$ following the procedures proposed in Section III, and then extend it into larger systems. All the codes for DNNs are implemented in python 3.6 with TensorFlow 1.0.0. To compare the performance of different methods fairly, the Hungarian algorithm is also implemented in python. Using the data generation method mentioned above, we first get 50,000 training samples obeying discrete uniform distribution over [1, 100). Then we split 90% of these data into the training data set and the rest 10% into the validation one.

A. Network Structures and Parameters for n = 4 System

When the dataset is prepared, we first choose specific numbers of layers and neurons for FNN and CNN, respectively. Then we implement the parameter selection procedure to choose appropriate parameters for the following training stage.

Network 1 (FNN): We construct a fully-connected FNN with five layers: one input layer, three hidden layers with 32, 64, and 256 nodes, respectively, and one output layer, and use Sigmoid function (eg. logistics function) as the activation function. Fig. 4 shows the influence of batch size on the training loss value. According to the figure, we select 1024 as the batch size since it can assure low loss value with a small sacrifice for convergence speed. On the other hand, Fig. 5 shows how loss value changes with different learning rates. According to the figure, we choose 0.001 as the learning rate to guarantee the lowest loss value while with relatively high convergence speed.

Network 2 (CNN): We construct a CNN with five layers: one input layer, two convolutional layers with 2 and 4 different kernels, one fully connected layer with 256 nodes, and one output layer. We set the stride to be 1 and the convolutional kernel size to be 1×1 in each convolutional layer due to the small size of input data. We change our activation function into ReLu since it performs better than Sigmoid in CNN [12].

TABLE I
PERFORMANCE COMPARISON FOR DIFFERENT METHODS

	Hungarian Algorithm	CNN	FNN
Time(s)	0.5916	0.0120	0.0040
Accuracy	100%	92.76%	90.80%

TABLE II
COMPARISON OF CNN WITH RANDOM ASSIGNMENT

	CNN	Random	Accuracy gain
$n = 4$	92.76%	25.00%	3.71
$n = 8$	77.80%	12.50%	6.22
$n = 16$	65.70%	6.25%	10.512

The selection of parameters is almost the same as FNN but we make a slight modification of learning rate. We decrease it to 0.0001 to ensure CNNs' convergence and stability.

B. Results

After network structures and parameters are settled, we do the training stage to get the learned models. Then, we evaluate the accuracy and time-efficiency of DNNs with 5,000 testing samples, and the results are summarized in Table I.

From Table I, DNNs outperform the Hungarian algorithm in terms of time efficiency with a slight loss of accuracy. According to [13], the Hungarian algorithm has the computational complexity of $O(n^3)$, which can be considered to be impractical for many applications. As for the testing stage of DNNs, the computational complexity decreases to $O(n^2)$ and $O(n)$ for CNN and FNN, respectively. Because we can solve n sub-assignment problems in parallel, the computational complexity of our proposed method solving the LSAP is the same as that of solving each sub-assignment problem. Furthermore, DNNs have only a slight decrement of accuracy. In particular, 9% loss in accuracy can lead to 49 times increase in time efficiency by using CNN, which suggests the obvious advantages of our proposed methods.

Table I also indicates that CNN outperforms FNN. The reason can be explained as follows. By using CNN, we can regard the cost matrix as an image. Each convolutional layer of the CNN can extract different features of the input cost matrices before the fully-connected layers. Instead of directly learning the input/output relations by the fully-connected layers in FNN, CNN first analyzes the inputs by digging out more hidden features, which contributes to the increase of accuracy.

Furthermore, we use 1,000 testing samples to verify the effectiveness of the greedy collision-avoidance rule. When the accuracy of solving each sub-assignment problem is about 90.8%, the probability of collisions is 7.6% for each person. In this case, the accuracy of solving the problem in (1) without and with the greedy collision-avoidance rule is 69.4% and 82.9%, respectively, which demonstrates the effectiveness of the low-complexity heuristic algorithm.

C. Scalability

To examine the scalability of our method, we use CNN to do more test work with $n = 8$ and 16. As the size of the problem increases, we set the number of training samples up to one million and modify our network structure at the same time. We add one more convolutional layer to the network and also increase the node number of the fully-connected layer. The results are summarized in Table II.

Since the difficulty of the problem increases, the accuracy will drop to some degree. However, as shown in the Table II, the accuracy of our model is quite satisfactory as compared with the random assignment.

V. CONCLUSION AND FUTURE WORK

In this letter, we have implemented the DNN technique to solve LSAPs. We first decompose LSAPs into several classification sub-problems, and then train DNNs, specifically FNNs and CNNs, to solve each sub-assignment problem. Our test results show that DNNs can be used to obtain the real-time solution to LSAPs with a slight loss of accuracy. This initial study suggests that ML approaches have great potential in solving combinatorial optimization problems. Further attempts to solve more complicated combinatorial optimization problems using DNNs are important future directions. Also, improving the training of the DNNs, finding the optimal DNN architecture, and designing better collision avoidance rules are very interesting issues for fine-tuning our proposal.

REFERENCES

[1] G. Yu *et al.*, "Joint mode selection and resource allocation for device-to-device communications," *IEEE Trans. Wireless Commun.*, vol. 62, no. 11, pp. 3814–3824, Nov. 2014.
[2] X. Lu, Q. Ni, W. Li, and H. Zhang, "Dynamic user grouping and joint resource allocation with multi-cell cooperation for uplink virtual MIMO systems," *IEEE Trans. Wireless Commun.*, vol. 16, no. 6, pp. 3854–3869, Jun. 2017.
[3] H. Song, X. Fang, and Y. Fang, "Unlicensed spectra fusion and interference coordination for LTE systems," *IEEE Trans. Mobile Comput.*, vol. 15, no. 12, pp. 3171–3184, Dec. 2016.
[4] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logist.*, vol. 2, nos. 1–2, pp. 83–97, Mar. 1955.
[5] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Ann. Oper. Res.*, vol. 14, no. 1, pp. 105–123, Dec. 1988.
[6] K. Ramakrishnan, N. Karmarkar, and A. Kamath, "An approximate dual projective algorithm for solving assignment problems," *Network Flows and Matching: First DIMACS Implementation Challenge*, vol. 12. Providence, RI, USA: Amer. Math. Soc., May 1993, pp. 431–451.
[7] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *J. Glob. Optim.*, vol. 6, no. 2, pp. 109–133, Mar. 1995.
[8] A. Naiem and M. El-Beltagy, "Deep greedy switching: A fast and simple approach for linear assignment problems," in *Proc. 7th Int. Conf. Numerical Anal. Appl. Math.*, Jan. 2009, pp. 1–6.
[9] H. Sun *et al.*, "Learning to optimize: Training deep neural networks for wireless resource management," in *Proc. IEEE 18th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Jul. 2017, pp. 1–6.
[10] P. de Kerret, D. Gesbert, and M. Filippone, "Decentralized deep scheduling for interference channels," *arXiv preprint arXiv:1711.00625*, 2017. [Online]. Available: https://arxiv.org/abs/1711.00625
[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, May 2014, pp. 1–6.
[12] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Stat.*, Jun. 2011, pp. 315–323.
[13] G. A. Mills-Tettey, A. Stentz, and M. B. Dias, "The dynamic Hungarian algorithm for the assignment problem with changing costs," Robot. Inst., Pittsburgh, PA, USA, Rep. CMU-RI-TR-07-27, Jul. 2007.