Accurate Prediction of Network Distance via Federated Deep Reinforcement Learning

Haojun Huang, Yiming Cai, Geyong Min, *Member, IEEE*, Haozhe Wang, Gaoyang Liu, *Member, IEEE*, and Dapeng Oliver Wu, *Fellow, IEEE*

***Abstract*— A large number of distributed applications necessitate accurate network distance, for example, in the form of delay or latency, to ensure the Quality of Service (QoS). Due to high network measurement overhead and severe traffic congestion, network distance prediction has been introduced, instead of direct network measurements, to infer the unknown network distance with the partial measurements. However, most existing efforts neglect to fully capitalize on the potential latent factors, such as spatial correlations, long-existing temporal results and multi-rule exploration fusion, to achieve better accuracies with quicker convergence. To fill this gap, in this paper, we propose an Accurate Prediction of Network Distance (APND) solution via Federated Deep Reinforcement Learning (FDRL), which has four novel features distinguishing from the previous work. Firstly, a local feature-based matrix with low rank is established in each network cluster, referring to a set of neighbor nodes, to represent the potential spatial correlations among reachable node-pairs. Secondly, the parallel FDRL-based matrix factorization with multi-rule exploration fusion is introduced into APND and executed in all local clusters to minimize prediction errors and accelerate learning convergence. Thirdly, the long-existing learning experience is designed for local model training via Deep Reinforcement Learning (DRL) with rapid convergence. Fourthly, following the real-world routing paths, the cross-domain network nodes are simultaneously classified into adjacent clusters, built on the spatial correlations among them, and their coordinates will be further refined with error-based and average-based policies. Extensive experiments built on available real-world datasets illustrate that APND can accurately predict network distance compared with state-of-the-art approaches at the moderate computing cost.**

Manuscript received 24 May 2023; revised 7 March 2024; accepted 24 March 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Kompella. Date of publication 2 April 2024; date of current version 20 August 2024. This work was supported in part by the Natural Science Foundation of China under Grant 62372192, in part by Hong Kong Research Grant Council-General Research Fund under Grant 11203523, and in part by the European Union's Horizon 2020 Research and Innovation Program under the Marie Skłodowska-Curie Grant under Agreement 101030505. *(Corresponding author: Gaoyang Liu.)*

Haojun Huang is with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China, and also with the Department of Computer Science, University of Exeter, EX4 4QF Exeter, U.K. (e-mail: hjhuang@hust.edu.cn).

Yiming Cai is with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: ymcai@hust.edu.cn).

Geyong Min and Haozhe Wang are with the Department of Computer Science, University of Exeter, EX4 4QF Exeter, U.K. (e-mail: g.min@exeter.ac.uk; h.wang3@exeter.ac.uk).

Gaoyang Liu is with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: gaoyangliu2020@gmail.com).

Dapeng Oliver Wu is with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: dpwu@ieee.org).

Digital Object Identifier 10.1109/TNET.2024.3383479

***Index Terms*— Network distance prediction, federated deep reinforcement learning, matrix factorization.**

I. INTRODUCTION

THE past years have witnessed the emergence of a number of distributed network-aware applications like real-time video communications, multiplayer online games and peer-to-peer file sharing [1] to provide convenience and low-latency services. Generally, these applications can benefit from accurately knowing the network distance, in the form of either one-way latency or more often Round-Trip Time (RTT), between each node-pair of networks to ensure the Quality of Service (QoS). For instance, the user experience with real-time online gaming services will be greatly enhanced when the network distances to all potential gaming platforms are known in advance.

Obviously, it is infeasible to actively probe global network distances among all reachable nodes as the network scale gradually increases due to the significant measurement overhead and severe traffic congestion. A promising idea is to implement network distance prediction, also known as network latency/delay prediction, which infers all unknown network distance from a small set of partially observed measurements [2], [3]. This understanding has motivated numerous research on Euclidean/non-Euclidean network embedding, which embeds nodes into Euclidean/non-Euclidean spaces to learn their coordinates and then, built on these, to predict all missing distance among nodes. The approaches based on Euclidean embedding have been widely investigated and achieved good QoS in interesting scenarios [4]. Because non-Euclidean embedding, typically referring to matrix factorization, has more accurate performance on network distance estimation than Euclidean embedding [5], [6], we focus on investigating the network-matrix-based non-Euclidean network distance prediction, where the network distances among nodes are represented by the product of two smaller matrices.

Although remarkable efforts have been made [4], [5], [6], [7], [8], [9], [10], [11], there still exist at least two problems to be mitigated for non-Euclidean approaches to achieve better prediction accuracies. First, how to exploit the potential spatial correlations of network distances among reachable node-pairs to establish low-rank feature-based network matrices. The efficient network distance prediction is heavily dependent on the low-rank features of the network matrices with the spatial

correlations of network measurements. However, the current network distance matrices always include weak-relevant distance measurements, which are characterized by the high or full rank due to no reservation of cross-domain network distances or no utilization of the spatial correlations of network distances among multiple local matrices, resulting in low accurate prediction. It is required to exploit the spatial correlations of network distance and cross-domain network distance to establish low-rank feature-based matrices. Second, how to intelligently achieve more accurate network distance prediction through multi-rule cooperation to dig out the hidden rules, referring to the specific iteration manners, and experience in coordinate iterations. Previous approaches like DMFSGD [5] and DISCS [6] always introduce typical machine learning or optimization strategies, which iterate coordinates in a settled single-rule manner, to explore and exploit the hidden knowledge included in network distance. Most of them fail to infer the missing network distance intelligently and cooperatively with multi-rule policies to obtain higher prediction rewards. Furthermore, the potential latent factors, such as spatial correlations and the long-existing intermediate experience, are often ignored to obtain more accurate results with quick convergence. As a result, the prediction performance cannot satisfy the QoS requirements of distributed applications with the ever-increasing network scales. This will be mitigated with the emerging Artificial Intelligence like Federated Learning (FL) and Deep Reinforcement Learning (DRL), which have great learning ability to explore and exploit the hidden knowledge and potential latent factors in model aggregation and strategy optimization in an alternative multi-rule manner. With the learned knowledge and experience, DRL can well dynamically select a better matrix-factorization action with more prediction gains, following the given iterative multi-rule policies, at each network state for accurate prediction of network distance. In order to accelerate the convergence of model training, FL can be integrated with DRL to coordinate multiple agents across different clusters in a distributed manner for joint network distance prediction as the network scale increases, without additional data exchange among them. These observations inspire us to design new non-Euclidean approaches, built on DRL and FL, to implement network distance prediction in an efficient and rapid manner.

In this paper, we propose Accurate Prediction of Network Distance (APND) to alleviate the above-mentioned issues via Federated Deep Reinforcement Learning (FDRL), by introducing FL and DRL into the global model aggregation and local matrix factorization, respectively, for accurate network distance prediction in a distributed manner. APND works as follows. Firstly, it divides the whole network into a number of clusters with the spatial correlations of node-pairs, and then builds the feature-based matrices, which include a number of cross-domain network distances, in all clusters with the partial measurements. On these bases, the available matrix factorization rules like SGD, AdaGrad [12], RMSProp and AdaDelta [13] are integrated into FDRL to parallelly factorize such matrices with the optimal multi-rule exploration fusion and Long-Short-Term-Memory (LSTM) exploitation. Within APND, FL is used as the learning architecture, while DRL acts as the efficient feature extractor to build the relationships among high-dimensional matrix states, matrix factorization actions and rewards, related to network coordinates for accurate network distance prediction. Due to the introduction of FDRL, APND will consume much more computational resources in network distance prediction, but can keep a balance in computing consumption among network clusters and obtain more accurate prediction results with quick convergence of model training. In this paper, we make the distinct contributions as follows:

- By analyzing extensive real-world network data, we identify that the network distances are characterized by low rank in local and global networks. Considering that different local networks have various amounts of network distances to be calculated with different convergence, we divide the networks into several clusters built on their spatial correlations. Based upon these investigations, we formulate the accurate network distance prediction as joint intelligent local matrix factorization with the partial observed measurements in networks.
- A novel FDRL-based decentralized matrix factorization approach is proposed to accurately predict unknown network distance. The multiple rules of matrix factorization and the LSTM-based learning experience with distinct achievements are parallelly exploited in all clusters to train the local models with the fewer prediction errors and quicker convergence. Moreover, two benefit-based aggregation policies have been proposed to further enhance local prediction models and accelerate global convergence in an efficient manner.
- Notice that a number of nodes will communicate with other nodes located in different clusters in real-world networks, thus we classify such nodes into adjacent clusters based on the spatial correlations among them, meaning that the cross-domain network distances among adjacent clusters have been reserved following the real routing paths. To refine the coordinates of such nodes, both error-based and average-based policies are designed.
- Extensive experiment simulations are performed in various scenarios built on the real-world data to assess the performance of APND. Simulation results indicate that APND outperforms previous solutions in the accuracies of distance prediction at moderate computing cost.

The rest of the paper is organized as follows. Section II presents the previous related work on network distance prediction. Section III describes the vital background knowledge, including network matrix, matrix factorization and problem formulations, to facilitate understanding of APND. Section IV describes the details of APND. Section V elaborates and analyses the simulations results to validate the performance of APND. Finally, Section VI concludes this paper.

II. Related Work

In this section, we mainly present significant efforts related to network distance prediction [3], [4], [8], [9], [14], [15], [16], [17]. In accordance with the embedding network spaces, these approaches can be divided into two categories: Euclidean embedding and non-Euclidean embedding.

TABLE I
THE IMPORTANT PARAMETERS AND NOTATIONS

Symbol	Meaning
$d_{ij}/\hat{d}_{ij}$	The measured/estimated distance from u_i to u_j
$D/\hat{D}$	$n\times n$ measured/estimated matrix
X/Y	$n\times k$ factorized matrix
x_i/y_i	The outgoing/incoming vector of u_i
$r_{sc}(u_i,u_j)$	The spatial correlation between u_i and u_j
S/S_t	State space/state at time slot t
A/A_t	Action space/action at time slot t
R/R_t	Reward space/reward at time slot t
$a/\hat{a}$	Predictive/target actor networks
$Q/\hat{Q}$	Predictive/target critic networks
(μ,θ)	Local model of each cluster
(M,Θ)	Global model aggregated by cloud server

Euclidean embedding is a straightforward approach, which embeds network nodes into a metric space to predict network distance. In this way, some nodes are assigned as landmarks with the known coordinates to infer the locations of ordinary nodes when the distance measurements to and from them are implemented. As a result, the distances between all node-pairs can be obtained. Examples of such approaches include GNP [4], Virtual Landmarks [14], PIC [8], NPS [9], ICS [16], BBS [17]. However, due to asymmetric routing and traffic congestion, the real network distances in the form of Euclidean probably have the feature of violations of triangle inequality (TIVs) [18], which cannot legitimately represent Euclidean distance.

In order to achieve efficient network distance prediction, more and more researchers are dedicated to the non-Euclidean embedding. The basic idea of it is to map nodes into a non-Euclidean space, mainly referring to matrix factorization [5], [10] and tensor completion [11] to infer the locations of nodes, in the form of non-Euclidean coordinates. Because TIVs and asymmetry, widely-existed in real-world network environments, are considered for the recovery of network distance, most of such solutions achieve more accurate performance than Euclidean embedding. Typical examples of non-Euclidean embedding involve IDES [10], Phoenix [19], DMFSGD [5], RMF [20], DISCS [6], RRDL [21], ANLP [22], TNDP [11], and NMMF-S [23]. However, most of them only achieve sub-optimal results or slow convergence because the hidden rules and experience have not been taken into consideration, which cannot satisfy the QoS requirements of different users. To tackle this issue, APND has introduced novel strategies distinguishing from the previous efforts, including low-rank feature-based matrix establishment, parallel FDRL-based matrix factorization, long-existing experience learning and benefit-based model aggregation to accurately infer unknown network distances.

III. PROBLEM STATEMENT

This section describes the primary knowledge essential to understand APND. We first present the network matrix, and then elaborate matrix factorization. For readability, we list the main parameters and notations used in the paper in TABLE I.

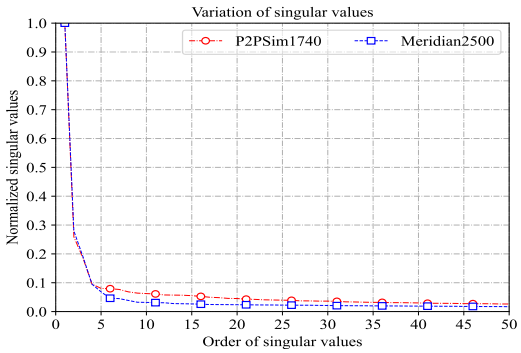


Fig. 1. Singular values of RTT matrices of 1740 × 1740 and 2500 × 2500, extracted from the Meridian2500 and P2PSim1740 datasets, respectively. The normalization of singular values indicates that both maximal singular values are equal to 1.

A. Network Matrix

Suppose that a network consists of a large number of nodes, where most routing paths among nodes are measured as the network distances in the form of RTT [5], [6], [10]. To algebraically describe network distance, we define an $n\times n$ distance matrix $D\in\mathbb{R}^{n\times n}$ filled with element d_{ij}, which stands for the actual distance from node u_i to node u_j, where $1\leq i,j\leq n$ and $d_{ii}=0(1\leq i\leq n)$. Due to the high network measurement overhead and severe traffic congestion [18], it is difficult to obtain all network distances. Therefore, similar to [5], [11], and [24], the missing network distance d_{ij} in such a matrix is presented in -1, which requires to be estimated. In this way, the reachable distances from node u_i to all other nodes are recorded in the i-th row of D, while the reachable distances from all other nodes to node u_j are included in its j-th column.

B. Matrix Factorization

Notice that a node-pair close to each other have approximate distances to other nodes in networks, so their corresponding row vectors in distance matrix are linearly correlated, which induces that the distance matrix is approximately low-rank [5], [11], [25]. To exploit matrix factorization, the low-rank feature of matrix ought to be satisfied. Therefore, we have empirically verified this property in two distance matrices, using real datasets P2PSim1740 and Meridian2500, via SVD to obtain non-zero singular values, the number of which equals the rank of matrices. Fig. 1 illustrates that the normalized singular values of both matrices reduce quickly, with 96.82% and 95.03% reductions in P2PSim1740 from the 10th value and Meridian2500 from the 17th one, respectively. As a result, the ratio of effective ranks in distance matrices is quite small, about 10 for P2PSim1740 and 17 for Meridian2500, indicating network distance matrices satisfy the low-rank feature.

Built upon this observation, the low-rank $n\times n$ distance matrix D can be factorized into a product of two m ($m\ll n$) rank factor matrices X and Y, denoted as

$$D\approx\hat{D}=X\times Y^T,$$
$$=\begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}\begin{bmatrix} y_{11} & \cdots & y_{1m} \\ \vdots & \ddots & \vdots \\ y_{n1} & \cdots & y_{nm} \end{bmatrix}^T, \quad (1)$$

where $\hat{D}$ denotes the predicted distance matrix. The outgoing vector x_i=$[x_{i1}, x_{i2}, \cdots, x_{in}]$ and incoming vector y_i=$[y_{i1}, y_{i2}, \cdots, y_{in}]^T$ represent the coordinates of node u_i. Thus, the actual distance d_{ij} between two arbitrary reachable nodes u_i and u_j can be estimated as the predicted distance $\hat{d}_{ij}$, given by

$$d_{ij} \approx \hat{d}_{ij} = x_i {y_j}^T = \sum_{k=1}^{m} x_{ik} y_{kj}. \tag{2}$$

In order to obtain accurate prediction, i.e., to make Eq. (2) true, we introduce a loss function $L(D, X, Y, W, \lambda)$ to evaluate the prediction accuracy, expressed as

$$L(D, X, Y, W, \lambda) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(d_{ij} - x_i {y_j}^T)^2 + \lambda \sum_{i=1}^{n} x_i {x_i}^T + \lambda \sum_{i=1}^{n} y_i {y_i}^T, \tag{3}$$

where W is a weight matrix whose value of w_{ij} is 1 when d_{ij} is measured or 0 otherwise. The parameter λ stands for the regularization coefficient to not only control the extent of regularization but also avoid the numerical instability due to the nonuniqueness of factorization.

C. Problem Formulation

With the given above-mentioned network matrix and matrix factorization, we can formulate the accurate network distance prediction as the problem of distributed matrix factorization via FDRL. Specifically, all nodes collaborate with their k-hop neighbor nodes to measure the network distances between them and compute their spatial correlations. With the learned spatial correlations, the networks are divided into several clusters and a cluster head is assigned for each cluster. Then, each node reports its measurement results to the cluster head for feature-based matrix establishment. Built on this basis, the agents in all clusters start to parallelly factorize the feature-based matrices and send the learned models to the cloud server for model aggregation. In this way, each node will retrieve its coordinates, i.e., outgoing vector x and incoming vector y. Built on Eq. (3), the outgoing vector x_i and incoming vector y_i of node u_i can be represented as

$$\begin{cases} x_i = \arg\min \sum_{j=1}^{n} w_{ij}(d_{ij} - x_i {y_j}^T)^2 + \lambda x_i {x_i}^T, \\ y_i = \arg\min \sum_{j=1}^{n} w_{ji}(d_{ji} - x_j {y_i}^T)^2 + \lambda y_i {y_i}^T. \end{cases} \tag{4}$$

To solve Eq. (4), we adopt the idea of gradient descent (GD) [5] to obtain the optimal coordinates x and y, in the form of $x \leftarrow x - \eta \cdot g^x$ and $y \leftarrow y - \eta \cdot g^y$, where η indicates the learning rate, g^x and g^y represent the gradient obtained by differentials $\frac{\partial L}{\partial x_i}$ and $\frac{\partial L}{\partial y_i}$ of Eq. (3), respectively, given by

$$\begin{cases} g^x = \dfrac{\partial L}{\partial x_i} = -\sum_{j=1}^{n} w_j^i (d_{ij} - x_i y_j^T) y_j + \lambda x_i, \\ g^y = \dfrac{\partial L}{\partial y_i} = -\sum_{j=1}^{n} w_i^j (d_{ji} - x_j y_i^T) x_j + \lambda y_i. \end{cases} \tag{5}$$

To achieve more accurate and quicker computation, the improved rules of GD like Momentum [26], AdaGrad [12], RMSProp, AdaDelta [13], and Adam [27] are jointly considered, all of which use gradient operator $G(g)$ for coordinate update step by step, denoted as $x \leftarrow x - \eta \cdot G(g)$ and $y \leftarrow y - \eta \cdot G(g)$ with different convergence and accuracies. In order to balance the precision and update rate, we design a novel rule that gradient operators of multiple rules are blent together proportionally via DRL-based matrix factorization to minimize Eq. (4) for accurate network distance prediction, expressed as

$$\begin{cases} x \leftarrow x - \eta[p(\alpha_{t,x}^{r_1}) \cdot G_x^1(g^x) + p(\alpha_{t,x}^{r_2}) \cdot G_x^2(g^x) + \cdots], \\ y \leftarrow y - \eta[p(\alpha_{t,y}^{r_1}) \cdot G_y^1(g^y) + p(\alpha_{t,y}^{r_2}) \cdot G_y^2(g^y) + \cdots], \end{cases} \tag{6}$$

where $G_x^i(g^x)$ and $G_y^i(g^y)$ represent the i-th rule operators referring to gradients obtained in Eq. (5), while $p(\alpha_{t,x}^{r_i})$ and $p(\alpha_{t,y}^{r_i})$ denote the proportions of i-th outgoing and incoming gradient operators, respectively. Built on the learned coordinates, the unknown network distances can be obtained in the form of coordinate products defined in Eq. (2).

IV. APND: FDRL-Based Network Distance Prediction

This section presents APND in detail for accurate network distance prediction. We firstly introduce the architecture of APND and introduce the essential 3-tuple design for its operations. Then, we present how to establish decentralized feature-based matrices and train local model in each agent. Finally, we illustrate FDRL-based model aggregation of APND.

A. Architecture Overview

The central concept behind APND is to predict the unknown network distances with FDRL-based matrix factorization. With the partial measured distance, a set of decentralized feature-based matrices, which include the unknown network distances, have been established and intelligently factorized by introducing FDRL in an alternative multi-rule factorization manner.

The architecture of APND is demonstrated in Fig. 2, which mainly refers to local training and global model learning. There are two kinds of nodes: agents and cloud server, which implement local training and global model learning, respectively. All nodes in networks have been classified into m clusters, depending on their spatial correlations, defined in Eq. (19), each of which has a cluster head acting as a function node to interact with the cloud server. There exist m agents located in clusters $C_1, C_2, \ldots, C_m$ to execute local training for regional feature-based matrices factorization. Each agent only communicates with the cloud server. The local training absorbs Q-learning and neural networks to update parameters of local model $[\theta_t^i, \mu_t^i](1 \le i \le m)$, which affects the choices of factorization and policy. The model aggregation is responsible for the collection of local models $\theta_t^1, \theta_t^2, \ldots, \theta_t^m$ and $\mu_t^1, \mu_t^2, \ldots, \mu_t^m$ to build a global model $[\Theta(t+1), \mathrm{M}(t+1)]$, and then download it to all agents for further matrix factorization with intelligent multi-rule policy and LSTM-based experience.

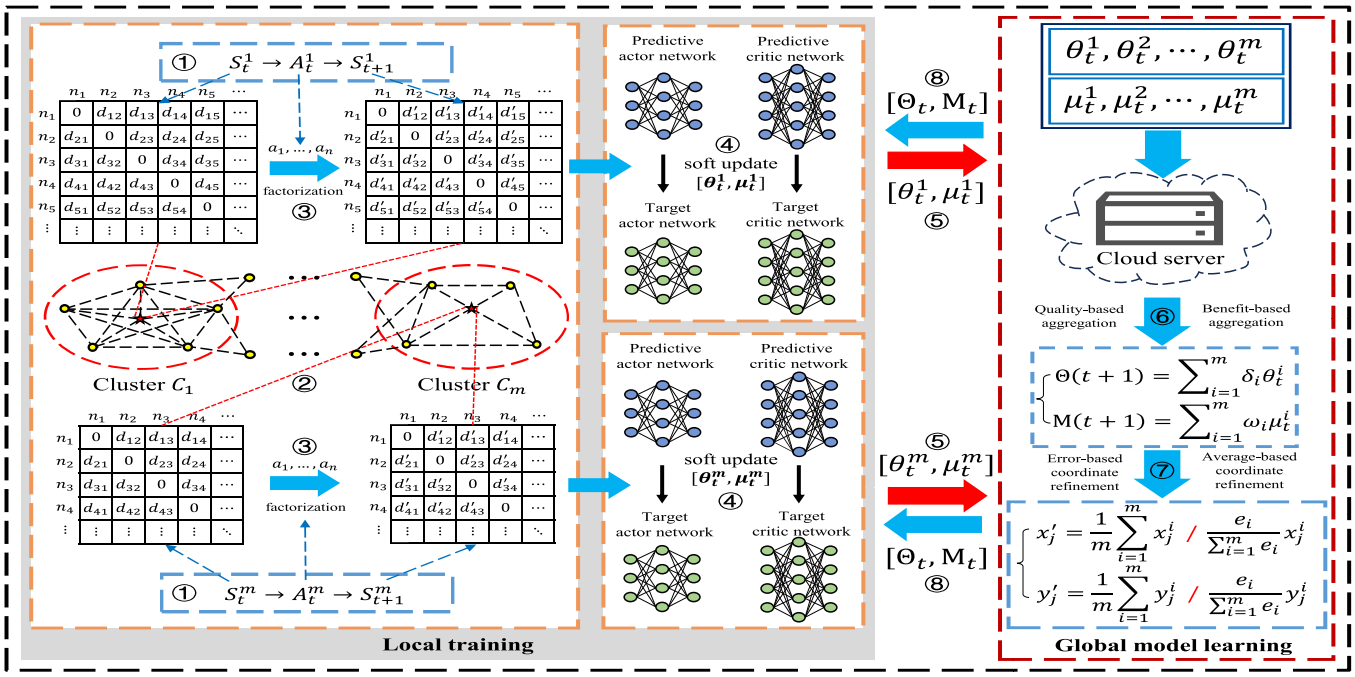


Fig. 2. The architecture of APND to realize accurate network distance prediction which includes local training and global model learning.

In our architecture, there are two different timescales: global episodes and local epochs. A global episode with the duration of T includes a series of epochs $\tau_1, \tau_2, \ldots, \tau_k$ equal to τ, used to mark the time when the cloud server updates $[\Theta(t), \mathrm{M}(t)]$, and is defined as $T{=}\{\tau_1, \tau_2, \ldots, \tau_k\}$, $|\tau_i|{=}|\tau_{i+1}|{=}\tau$. On the contrary, the epoch is proposed for cluster node of each cluster C_i to train local models θ_t^i and μ_t^i, which are reported to the cloud server for the update of global model $[\Theta(t), \mathrm{M}(t)]$. To avoid high communication cost and model overfitting, the local and global parameters communications will happen only one time in each episode, referring to global model download in the first epoch τ_1 and local model upload in the last epoch τ_k.

There will be several nodes divided into adjacent clusters simultaneously due to the cross-domain network distances, which means that multiple coordinates are assigned to them. To tackle this issue, the error-based and average-based refined policies are designed. The error-based policy takes the ratio of the local training error e_i in cluster C_i to the sum of all local training errors as vital coefficients to refine the coordinates x_j^i and y_j^i of node u_j obtained from local training, defined as

$$\begin{cases} x'_j = \sum_{i=1}^{m} \dfrac{e_i}{\sum_{i=1}^{m} e_i} x_j^i, \\ y'_j = \sum_{i=1}^{m} \dfrac{e_i}{\sum_{i=1}^{m} e_i} y_j^i. \end{cases} \tag{7}$$

where m is the total number of clusters which includes node u_j, while (x'_j,y'_j) denote its refined coordinates. The average-based policy considers the average of coordinates (x_j^1,y_j^1), $(x_j^2,y_j^2),\cdots,(x_j^m,y_j^m)$ obtained from clusters $C_1,C_2,\cdots,C_m$, which simultaneously include node u_j, as its refined coordinates (x'_j,y'_j), given by

$$\begin{cases} x'_j = \dfrac{1}{m} \sum_{i=1}^{m} x_j^i, \\ y'_j = \dfrac{1}{m} \sum_{i=1}^{m} y_j^i. \end{cases} \tag{8}$$

The operations of APND include the following eight steps: ① 3-tuple design, ② network nodes division, ③ decentralized feature-based matrix establishment, ④ local model training, ⑤ model upload, ⑥ model aggregation, ⑦ coordinate refinement, ⑧ model download. In step ①, the essential 3-tuple of Markov Decision Process (MDP), in the form of $<S,A,R>$, which refers to the action, state and reward, respectively, is designed for APND operations. In step ②, the nodes are divided into a set of clusters in accordance with their spatial correlations. On this basis, the available distances among node-pairs in each cluster are exploited to build local feature-based network distance matrix in step ③. In step ④, the local agent in each cluster implements local training to learn the optimized strategy for network matrix factorization. After a periodic training, the local training model of each cluster will be uploaded to the cloud server for aggregation in step ⑤. In step ⑥, the models learned by all agents will be aggregated with benefit/quality-based aggregation policies, along with the coordinates refinement for some nodes which simultaneously are included in adjacent clusters in step ⑦. In step ⑧, the cloud server downloads the aggregated models to all agents for their local model update and further learning.

B. 3-Tuple Design

The specific FDRL-based learning in APND is mainly dependent on the closed cooperations of neural networks

built-in DDPG, including the actor networks and critic networks, to implement iterative updates of a 3-tuple $<S,A,R>$, which refers to state, action and reward, respectively. To be more specific, S_t and A_t indicate the state of matrices in each step of matrix factorization and the taken action to factorize matrix at time slot t, respectively, while R_t represents the reward at time slot t. The agents act as the policy management, which mainly interactively makes decisions to interact with the environments at each epoch, for matrix factorization. Once executing action A_t at state S_t, the agent should step into a new state S_{t+1} along with an updated reward R_t, which coincides with the optimization objective of APND, defined in Eq. (4). The details of MDP-based 3-tuple associated with accurate network distance prediction are presented in the following.

1) State space: S is referred to as the distance-aware information included in m clusters, and can be denoted as

$$S =< S_t^1, S_t^2, \ldots, S_t^m > . \tag{9}$$

Here, S_t^i stands for the state of cluster C_i at time slot t, and can be defined as

$$S_t^i =< D_i, X_t^i, Y_t^i, \Phi >, \tag{10}$$

where D_i represents the original network distance matrix of cluster i, while X_t^i and Y_t^i denote the factor matrices of cluster C_i at time slot t. The iteration rules set Φ can be defined as

$$\Phi =< r_1, r_2, \ldots, r_k >, \tag{11}$$

which includes k alternative rules $r_1, r_2, \cdots, r_k$, specified by the important factorization strategies like SGD, AdaGrad and RMSProp. In order to facilitate understanding, we give three basis rules (k=3), built on SGD, AdaGrad and RMSProp, to compute the outgoing vector x_i and incoming vector y_i of X_t^j and Y_t^j, described as follows.

SGD-based rule. The rule used to obtain the variables x_i and y_i, built on the gradients g^x and g^y, defined in Eq. (5), can be represented as

$$\begin{cases} x_i \leftarrow (1-\eta\lambda)x_i + \eta \sum_{j=1}^{n} w_j^i (d_{ij} - x_i y_j^T) y_j, \\ y_i \leftarrow (1-\eta\lambda)y_i + \eta \sum_{j=1}^{n} w_i^j (d_{ji} - x_j y_i^T) x_j, \end{cases} \tag{12}$$

where parameter η controls the speed of the updates.

AdaGrad-based rule. This rule is built on SGD by introducing the cumulative variable s_t, which considers previous gradient g_k at descent time k to constrain the process of iteration, denoted as s_t=$\sum_{k=1}^{t} g_k^2$. The update rule of the variables x_i and y_i is given by

$$\begin{cases} x_i \leftarrow x_i - \eta \dfrac{g_t^x}{\sqrt{\sum_{k=1}^{t} (g_k^x)^2 + \epsilon}}, \\ y_i \leftarrow y_i - \eta \dfrac{g_t^y}{\sqrt{\sum_{k=1}^{t} (g_k^y)^2 + \epsilon}}, \end{cases} \tag{13}$$

where s_t^x=$\sum_{k=1}^{t} (g_k^x)^2$, s_t^y=$\sum_{k=1}^{t} (g_k^y)^2$, while g_k^x and g_k^y are the gradients in the k-th iterations of outgoing and incoming vectors, respectively.

RMSProp-based rule. This rule is built on AdaGrad by introducing a decay factor β to eliminate the increasing trend of the cumulative variable v, defined as v_t=$(1-\beta)$ $\sum_{k=0}^{t} \beta^{t-k} (g_k)^2$. The update rule of the variables x_i and y_i can be denoted as

$$\begin{cases} x_i \leftarrow x_i - \dfrac{\eta g_t}{\sqrt{(1-\beta)\sum_{k=1}^{t} \beta^{t-k} (g_k^x)^2 + \epsilon}}, \\ y_i \leftarrow y_i - \dfrac{\eta g_t}{\sqrt{(1-\beta)\sum_{k=1}^{t} \beta^{t-k} (g_k^y)^2 + \epsilon}}. \end{cases} \tag{14}$$

2) Action space: A denotes the action taken by agents for matrix factorization, and can be expressed as

$$A =< A_t^1, A_t^2, \ldots, A_t^m >, \quad A_t^i \cap A_t^j \neq \emptyset, \tag{15}$$

where A_t^i=$<A_{t,x}^i, A_{t,y}^i>$ represents the action in cluster C_i for the update of outgoing vector x and incoming vector y, respectively, at time slot t. $A_{t,x}^i$ and $A_{t,y}^i$ can be further represented as

$$\begin{cases} A_{t,x}^i =< \alpha_{t,x}^{r_1}, \alpha_{t,x}^{r_2}, \cdots, \alpha_{t,x}^{r_k} >, \\ A_{t,y}^i =< \alpha_{t,y}^{r_1}, \alpha_{t,y}^{r_2}, \cdots, \alpha_{t,y}^{r_k} >, \end{cases} \tag{16}$$

where $\alpha_{t,x}^{r_j}$ and $\alpha_{t,y}^{r_j}$ denote the taken actions with the factorization rule r_j to obtain the coordinates x and y at time slot t, respectively, while the proportions $p(\alpha_{t,x}^{r_j})$ and $p(\alpha_{t,y}^{r_j})$, learned by neural networks included in DDPG, satisfy $p(\alpha_{t,x}^{r_1})$+$\cdots$+$p(\alpha_{t,x}^{r_k})$=1 and $p(\alpha_{t,y}^{r_1})$+$\cdots$ +$p(\alpha_{t,y}^{r_k})$=1.

With three above-mentioned rules, the update of variables x_i and y_i associated with actions defined in Eq. (16) can be represented as

$$\begin{cases} x_i \leftarrow x_i - \eta[p(\alpha_{t,x}^{r_1}) g_t^x + \dfrac{p(\alpha_{t,x}^{r_2}) g_t^x}{\sqrt{s_t^x + \epsilon}} + \dfrac{p(\alpha_{t,x}^{r_3}) g_t^x}{\sqrt{v_t^x + \epsilon}}], \\ y_i \leftarrow y_i - \eta[p(\alpha_{t,y}^{r_1}) g_t^y + \dfrac{p(\alpha_{t,y}^{r_2}) g_t^y}{\sqrt{s_t^y + \epsilon}} + \dfrac{p(\alpha_{t,y}^{r_3}) g_t^y}{\sqrt{v_t^y + \epsilon}}]. \end{cases} \tag{17}$$

Following this updated gradient operator, the matrix factorization becomes continuous taken actions with more integrated existing rules, and thus can be executed with DDPG, which works well in continuous action spaces.

3) Reward: R is referred to as our optimization goal, defined in Eq. (4). The reward R_t at time slot t is yielded after the action A_t having been taken by an agent at state S_t. In order to minimize the prediction error, we denote R_t as

$$R_t = \begin{cases} 10^{\frac{e_{th}}{e_s}}, & e_s < 1, \\ 1 - e_s, & e_s \geq 1, \end{cases} \tag{18}$$

where e_s represents the stress error of prediction at time slot t defined in Eq. (25), and e_{th} is a constant to restrain the value of reward in a reasonable interval. As a result, reward R_t can be regarded as the weighted cumulative residual from original matrix D to estimated matrix $\hat{D}$ as Eq. (3).

C. Decentralized Feature-Based Matrix Establishment

The decentralized feature-based distance matrix establishment includes neighbor and reference sets establishment, partial network measurement and spatial correlation calculation, and local matrix establishment, as illustrated in Fig. 3.

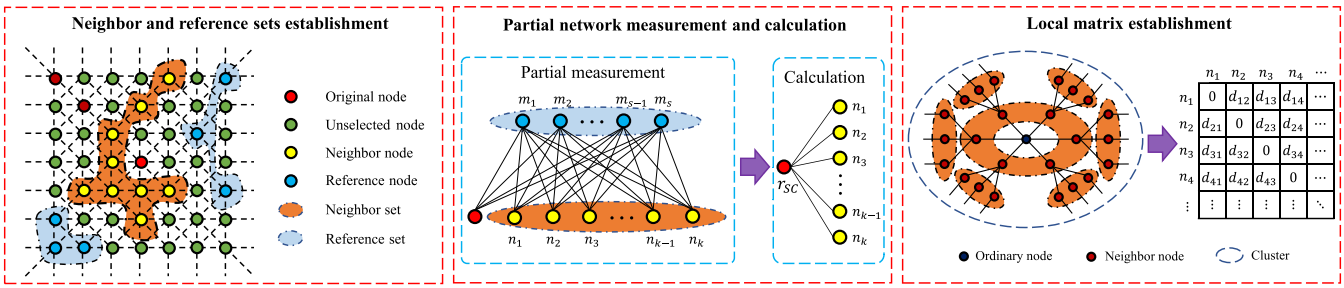


Fig. 3. Distributed feature-based matrix establishment, which includes neighbor and reference set establishment, measurement and calculation, and local matrix establishment.

Firstly, each node u_i sends a broadcast packet to its k-hop neighbor nodes through its 1-hop neighbor node forwarding. Once receiving this packet, each forwarder will add the receiving time stamp into it and broadcast it in the networks. Each k-hop neighbor node will send a reply packet, which includes the RTT between them, to node u_i. Then, node u_i probes m 1-hop nodes, which act as its forwarders for network measurement, to build a neighbor set U, denoted as $U=\{u_0, u_1, \ldots, u_m\}$, where u_0 stands for the initiating node and $u_j(1\leq j\leq k)$ represents its neighbor nodes. Meanwhile, node u_i will select n k-hop nodes at random, excluding its neighbor nodes, as its reference nodes to establish a reference set V, denoted as $V=\{v_1, v_2, \ldots, v_n\}$, where $v_j(1\leq j\leq n)$ expresses a reference node.

Then, with the known neighbor set U and reference set V, node u_i can obtain a distance vector $d_i=\{d_{i1}, d_{i2}, \ldots, d_{in}\}$, filled with the measurement distance $d_{ij}(1\leq j\leq n)$ from u_i to v_j. Built on this vector, node u_i starts to compute its spatial correlation $r_{sc}(u_i, u_j)$, which refers to the similarity of distances from a 1-hop neighbor-pair to k-hop neighbor nodes, defined as

$$r_{sc}(u_i,u_j)=\frac{\sum_{n=1}^{s}\left(\frac{|d_{in}-\bar{d}_i|}{\max\limits_{1\leq n\leq s}|d_{in}-\bar{d}_i|}\times\frac{|d_{jn}-\bar{d}_j|}{\max\limits_{1\leq n\leq s}|d_{jn}-\bar{d}_j|}\right)}{\sqrt{\sum_{n=1}^{s}\frac{(d_{in}-\bar{d}_i)^2}{\max\limits_{1\leq n\leq s}(d_{in}-\bar{d}_i)^2}}\sqrt{\sum_{n=1}^{s}\frac{(d_{jn}-\bar{d}_j)^2}{\max\limits_{1\leq n\leq s}(d_{jn}-\bar{d}_j)^2}}}, \quad (19)$$

where $1\leq i,j\leq m$, $\bar{d}_i=\frac{1}{s}\sum_{n=1}^{s}d_{in}$, and $\bar{d}_j=\frac{1}{s}\sum_{n=1}^{s}d_{jn}$. In this way, a spatial correlation vector r_{sc} can be built for each node as $r_{sc}=\{r_{sc}(u_0,u_1), r_{sc}(u_0,u_2), \ldots\}$. Because the spatial correlations will be different with the different distances among nodes, there exists disparity in them among different neighbor-pairs.

Leveraging the spatial correlations, the whole network can be divided into several clusters. Specifically, each node selects $m-1$ relevant neighbor nodes from its neighbor set to initialize a cluster, in the form of a node list, which will be transported to another neighbor node, excluding in its neighbor set, with the highest spatial correlation. When receiving the list, the node will join this cluster and repeat this process. To avoid too many nodes being pushed into a cluster, we set a threshold r_{th}, which is referred to as the width and length of clusters, to control the scale of each cluster. If the spatial correlation of a node-pair satisfies $r_{sc}>r_{th}$, the two nodes are considered in an identical cluster. In this way, each node will be classified into one local cluster. Once several local clusters have been established, each cluster will assign a node, which has powerful computing ability among all nodes, as a cluster head. Then, this cluster head broadcasts its role in the whole cluster, and gathers measurement results from its cluster members. With the known distances between strong-spatial-correlation nodes, a local feature-based network distance matrix with low rank can be established for it. In this way, a set of feature-based network distance matrices with different rows and columns have been built in all clusters.

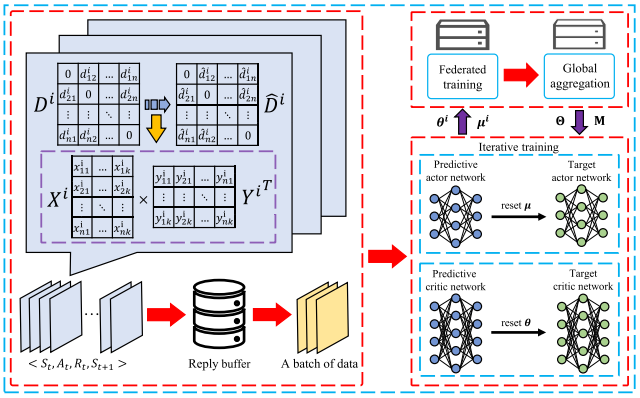


Fig. 4. Local predictive and target neural networks training.

D. Local Model Training in Each Agent

The local training in m clusters shown in Fig. 4 is built on the actor function $a(S_t|\mu)$ and the critic function $Q(S_t, A_t|\theta)$, which jointly determine the mapping relation from state S_t to action A_t and evaluate the local training with DDPG, for accurate network distance prediction. During local model training, the critic function $Q(S_t, A_t|\theta)$ is created by the predictive critic network, of which θ is the current weight of local training model with state S_t and action A_t decided by neural network $a(S_t|\mu)$, and can be expressed as

$$Q(S_t, A_t|\theta)=\mathbb{E}[\sum_{k=t}^{\infty}\gamma^{k-t}R_t|S_t=s, A_t=a(S_t|\mu), \theta], \quad (20)$$

where $\gamma(0<\gamma<1)$ represents the discount factor. The actor function $a(S_t|\mu)$ can be expressed as $a(S_t|\mu)=\pi(S_t=s|\mu)+N_t$, where N_t stands for Gaussian noise interfering agents to explore diverse decisions. The weight μ of $a(S_t|\mu)$ of

predictive actor network is updated following policy gradient descent as

$$\mu = \mu + \beta[R_t + \gamma \max_{\hat{a}(S_{t+1}|\mu)} \hat{Q}(S_{t+1}, A_{t+1}|\theta') - Q(S_t, A_t, \theta)] \frac{\partial Q(S_t, A_t|\theta)}{\partial a} \frac{\partial a}{\partial \mu}. \quad (21)$$

There are four neural networks installed in each agent, including predictive/target actor network and predictive/target critic network. Both actor networks are designed to train $a(S_t|\mu)$ with the built-in parameters μ step by step, while two critic networks are introduced to obtain the optimal action function $Q^*(S_t, A_t|\theta)R_t + \gamma \max_{a(S_{t+1}|\mu)} Q^*(S_{t+1}, A_{t+1}|\theta)$, by soft update. The weight θ of $Q(S_t, A_t|\theta)$ of target critic network will update in each episode as

$$\theta = \theta + \alpha[R_t + \gamma \max_{\hat{a}(S_{t+1}|\mu)} \hat{Q}(S_{t+1}, A_{t+1}|\theta') - Q(S_t, A_t, \theta)] \frac{\partial Q(S_t, A_t|\theta)}{\partial \theta}. \quad (22)$$

The target actor network $\hat{a}(S_{t+1}|\mu')$ and target critic network $\hat{Q}(S_{t+1}, A_{t+1}|\theta')$ will work together to obtain target reward at state S_{t+1}, denoted as $G_t{=}R_t{+}\gamma \max_{\hat{a}(S_{t+1}|\mu')} \hat{Q}(S_{t+1}, A_{t+1}|\theta')$. To accurately obtain $Q^*(S_t, A_t)$, the loss function $L(\theta, \mu)$, used to represent the deviation of critic function $Q(S_t, A_t|\theta)$, has been introduced and defined as

$$L(\theta, \mu) = \mathbb{E}[R_t + \gamma \max_{\hat{a}(S_{t+1}|\mu)} \hat{Q}(S_{t+1}, A_{t+1}|\theta') - Q(S_t, A_t|\theta)]. \quad (23)$$

In order to learn more experience from previous transition samples during local training, the replay buffer $\mathcal{D}^i$ with capacity $|\mathcal{D}|$ has been introduced into our local model $[\theta_t^i, \mu_t^i]$, in which all transition samples are stored in the form of $e_t{=}<S_t^i, A_t^i, R_t^i, S_{t+1}^i>$. Usually, these transitions are iterated on the first-in-first-out basis, and randomly selected out $<S_t^i, A_t^i>$ and $<S_{t+1}^i>$, respectively, by action space, for predictive critic networks and target critic networks to train $Q(S_t^i, A_t^i)$. Such an exploration is strongly associated with the short-term-memory states and actions, causing low accuracies of factorization and slow convergence of learning [11]. To mitigate this problem, we introduce the LSTM-based experience samples to explore the replay buffer. A series of LSTM samples $(e_{t_1}, \ldots, e_{t_k})$ will be replaced by the incoming ones $(e_{t_{1+j}}, \ldots, e_{t_{k+j}})$ and exploited for local learning to achieve $Q(S_t^i, A_t^i)$ more accurately.

In order to facilitate understanding, the pseudocode of local training is demonstrated in Algorithm 1. Lines 4-5 show that an action A_t^i selected by actor network $a(S_t^i)$ with cumulative-error-based policy is taken in environment, which returns a reward R_t^i and S_{t+1}^i, and local agent stores the information in the form of $<S_t^i, A_t^i, R_t^i, S_{t+1}^i>$ into memory buffer $\mathcal{D}_i$ in Line 6. In Line 7, a batch of samples stored in buffer $\mathcal{D}_i$ is randomly chosen for participating local training. As a result, we can obtain the loss function and a local training model in Lines 8-9 according to Eqs. (22-21), and update the local models in all clusters in Lines 11-12 of Algorithm 1. Then the new state and ϵ parameter start to update in Lines 13-14.

Algorithm 1 Local Training via DDPG

Input: Initial model θ and μ
Output: Improved model θ' and μ'
1 **for** $t \in [(j-1)T, jT]$ **do**
2 **for** *Agent in cluster* C_i *in parallel* **do**
3 **for** $\tau \in [1, T]$ **do**
4 Select action $A_t^i = a^i(S_t|\mu) + N_t$;
5 Execute action A_t^i to obtain R_t^i and S_{t+1}^i;
6 Save $< S_t^i, A_t^i, R_t^i, S_{t+1}^i >$ in $\mathcal{D}_i$;
7 Sample a series of LSTM-based minibatch training data from $\mathcal{D}_i$;
8 $G_t^i = R_t^i + \gamma \max_{a^i(S_{t+1}|\mu)} \hat{Q}(S_{t+1}^i, A_{t+1}^i|\theta')$;
9 $\theta_t \leftarrow \theta_t + \alpha[G_t^i - Q(S_t^i, A_t^i|\theta_t)]\nabla_\theta Q$;
10 $\mu_t \leftarrow \mu_t + \beta[G_t^i - Q(S_t^i, A_t^i|\theta_t)]\nabla_\mu Q$;
11 $\mu' \leftarrow \delta\mu + (1-\delta)\mu'$;
12 $\theta' \leftarrow \delta\theta + (1-\delta)\theta'$;
13 $S_t \leftarrow S_{t+1}$;
14 $N_t = \epsilon N_t$;
15 **end**
16 **end**
17 **end**

E. FDRL-Based Model Aggregation

In our proposed APND, there is a cloud server arranged to aggregate all local models learned by all agents in the divided clusters. The agent in cluster C_i performs iterative training to obtain a local model $[\theta_t^i, \mu_t^i]$, and then sends it to the cloud server for model aggregation. The global model parameters update can be expressed as

$$\begin{cases} \Theta(t+1) = \sum_{i=1}^{m} \delta_i \theta_t^i, \\ M(t+1) = \sum_{i=1}^{m} \delta_i \mu_t^i, \end{cases} \quad (24)$$

where δ_i denotes the weight of local model learned from cluster C_i to aggregate the optimal global model, and $\sum_{i=1}^{m} \delta_i = 1$.

In order to aggregate all received local models in an efficient manner, the benefit-based and quality-based aggregation strategies have been designed to assign the weights of all local models. The benefit-based strategy exploits the state-action reward R_i obtained from local model training in cluster C_i to proportionally assign the weight δ_i for global model aggregation, defined as $\delta_i = \frac{R_i}{\sum_{i=1}^{m} R_i}$. Following this principle, the local models which gain higher rewards have larger weights while those have lower rewards obtain smaller ones. The quality-based policy considers the quality of learning, which refers to the performance gain of local model training during an episode, is proportional to the weight for model aggregation. The quality of learning q_i of cluster C_i in the federated episode T can be represented in the form of an exponential forgetting function, given by $q_i = \sum_{j=0}^{T} \rho^{T-j} q_{ij}$, where $\rho (0<\rho<1)$ denotes the forgetting factor and q_{ij} represents the learning quality, usually evaluated by state-action reward, of local cluster C_i in epoch τ_j. Built on this definition,

Algorithm 2 FDRL-Based Model Aggregation

Input: Initial model $[\Theta(t), \mathrm{M(t)}]$
Output: Improved model $[\Theta(t+1), \mathrm{M(t+1)}]$
1 **for** $t \in [(j-1)T, jT]$ **do**
2 **Cloud server does**
3 Send $[\Theta(t), \mathrm{M(t)}]$ to C_i;
4 **local agent does**
5 **for** *Agent in cluster* C_i *in parallel* **do**
6 **for** *each* τ **do**
7 **if** τ_1 **then**
8 $\theta_t^i \leftarrow \Theta(t)$;
9 $\mu_t^i \leftarrow \mathrm{M(t)}$;
10 **end**
11 $\nabla L(\theta, \mu)$;
12 $\theta_t^i \leftarrow \theta_t^i + \alpha \nabla_\theta L(\theta, \mu)$;
13 $\mu_t^i \leftarrow \mu_t^i + \beta \nabla_\mu L(\theta, \mu)$;
14 **if** τ_k **then**
15 Upload θ_t^i, μ_t^i;
16 **end**
17 **end**
18 **end**
19 **Cloud server does**
20 $i \in \{1, 2, \ldots, m\}$;
21 Gather $\delta_i, \theta_t^i, \mu_t^i$;
22 $\Theta(t+1) \leftarrow \sum_i \delta_i \theta_t^i$, $\mathrm{M(t+1)} \leftarrow \sum_\mathrm{i} \delta_\mathrm{i} \mu_\mathrm{t}^\mathrm{i}$;
23 **return** $[\Theta(t+1), \mathrm{M(t+1)}]$;
24 **end**

the weight δ_i can be computed as $\delta_i = \frac{\sum_{j=0}^{T} \rho^{T-j} q_{ij}}{\sum_{i=0}^{m} \sum_{j=0}^{T-j} \rho^{T-j} q_{ij}}$. This strategy can assign larger weights to better performance gain of model learning and smaller weights to the worse ones.

Taking all above-mentioned designs in mind, all parameters of local models, such as the size of replay buffer and the learning rate of factorization, can be exploited to obtain a better global aggregated model, which will be downloaded to each agent for further local training.

The pseudocode of FDRL-based model aggregation is illustrated in Algorithm 2. Lines 2-3 refer to the aggregation model downloading, Lines 4-18 describe the local training in each cluster and local model uploading, while Lines 19-24 indicate the global model aggregation in cloud server.

V. Experimental Evaluation and Results

In this section, we evaluate the performance of APND via comprehensive simulation experiments, running on a server equipped with an Intel(R) Xeon(R) CPU E5-2609v3@ 1.90GHz and an NVIDIA GeForce GTX GPU. First, we introduce the simulation environments, and present the evaluation metrics. Then, the experimental simulation comparisons among APND and nine well-known network distance prediction solutions and their traditional basics, including IDES [10], AMF [28], DMFSGD [5], DISCS [6], TNDP [11], NMMF-S [23], SGD, AdaGrad and RMSProp, are illustrated. Among them, IDES, AMF, DMFSGD, DISCS and TNDP are five well-known matrix-factorization network distance prediction, while NMMF is DL-based matrix factorization for data recovery. SGD, AdaGrad and RMSProp are three traditional matrix factorization approaches, which can be used for data recovery and distance prediction.

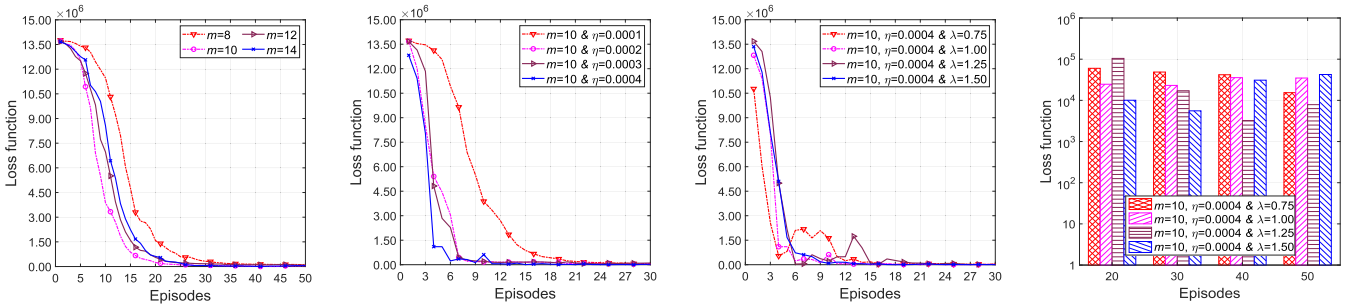
TABLE II
Parameter Settings in Simulations

Hyperparameters	Descriptions	Value
N	The number of nodes in networks	300
n	The number of established clusters	3
N_i	The number of nodes in cluster C_i	150
T	The training episodes	300
k	The training epochs in each episode	100
γ	The discount factor	0.998
ϵ	The decay coefficient of noise	0.995
$\mathcal{D}_i$	The capability of replay buffer	20
$-$	The capability of batch size	10

A. Simulation Setup

There are N nodes randomly selected to build a network distance matrix as a training dataset. Through distributed feature-based matrix establishment, we extracted sub-matrices from training matrix to represent several local clusters, each of which is labeled as C_i, with the size of $N_i \times N_i$. The DDPG model of cluster C_i includes two categories of neural networks: actor networks and critic networks. The actor networks consist of predictive actor network and target actor network, both of which are composed of three layers, i.e., a fully connected input layer, a LSTM hidden layer and a fully connected output layer with $(N_i \times 8 \times k)$, $((N_i \times (N_i \times 8 + N_i) + N_i) \times 4)$ and 3 neurons, respectively. The number of neurons in input layer refers to the dimensions of node features $x, y, g^x, g^y, m^x, m^y, v^x$ and v^y in cluster C_i, which represents the current state S_t, and the output layer indicates optional actions of all nodes in cluster C_i. Similarly, the critic networks consisting of predictive critic network and target critic network have three layers, i.e., a fully connected input layer, a LSTM hidden layer and a fully connected output layer with $(N_i \times 8 \times k)$, $((N_i \times (N_i \times 8 + N_i) + N_i) \times 4 + 3)$ and 1 neurons, respectively. The input layer is related to the dimensions of state S_t and action A_t obtained from actor network, and the output layer outputs an evaluation. Then, every cluster starts to train its local DDPG model for T episodes, each of which includes k epochs. In current episode, every cluster will upload its model for aggregation, and download aggregated model in the first epoch of next episode.

The simulation environment of network distance prediction is built on Python 3.7 and imports PyTorch 1.7.0 library to construct neural networks of learning models. The parameter settings include local replay buffer capability D_i, batch size of training samples, discount factor γ in Eq. (20), and decay coefficient ϵ of noise N_t. To enhance the readability, most parameters mentioned are listed in Table II. The experiment simulations were conducted on a recent widely-used dataset PlanetLab490 [29], which was obtained from PlanetLab [30]. This dataset has measured the RTTs between 490 desktops in a stable university network within 9 days, with the time slot



(a) The varied rank m (b) The varied learning rate η (c) The varied regularization coefficient λ (d) The optimal parameters

Fig. 5. Convergence of APND with the varied parameters of learning rate η, rank m and regularization coefficient λ in PlanetLab490.

collected in 14.7 hours, and thus included 18 latency matrices with size of 490×490.

B. Evaluation Metrics

To evaluate the effectiveness and convergence of our APND, four criteria are considered in our simulations as follows.

Loss function: The loss function is defined as Eq. (3) with the regularization coefficient λ between D and $\hat{D}$, and computed in each episode for obtaining $\hat{D}$, which indicates the convergence of APND and other approaches during the matrix factorization. If the loss function starts to maintain stable, the local models achieve convergence.

Stress Error: The stress e_s can be defined as

$$e_s = \sqrt{\sum_{i,j=1}^{n}(d_{ij}-\hat{d}_{ij})^2 / \sum_{i,j=1}^{n} d_{ij}^2}. \tag{25}$$

This metric is used to measure the global fitness between D and $\hat{D}$. Considering that the global networks are divided into several clusters, we use stress of local clusters to help training local models, given by Eq. (18), while stress of global networks is used to assess the prediction accuracies.

Relative Error: The relative error can be formally defined as

$$e_r = \frac{\|d_{ij}-\hat{d}_{ij}\|}{d_{ij}} \tag{26}$$

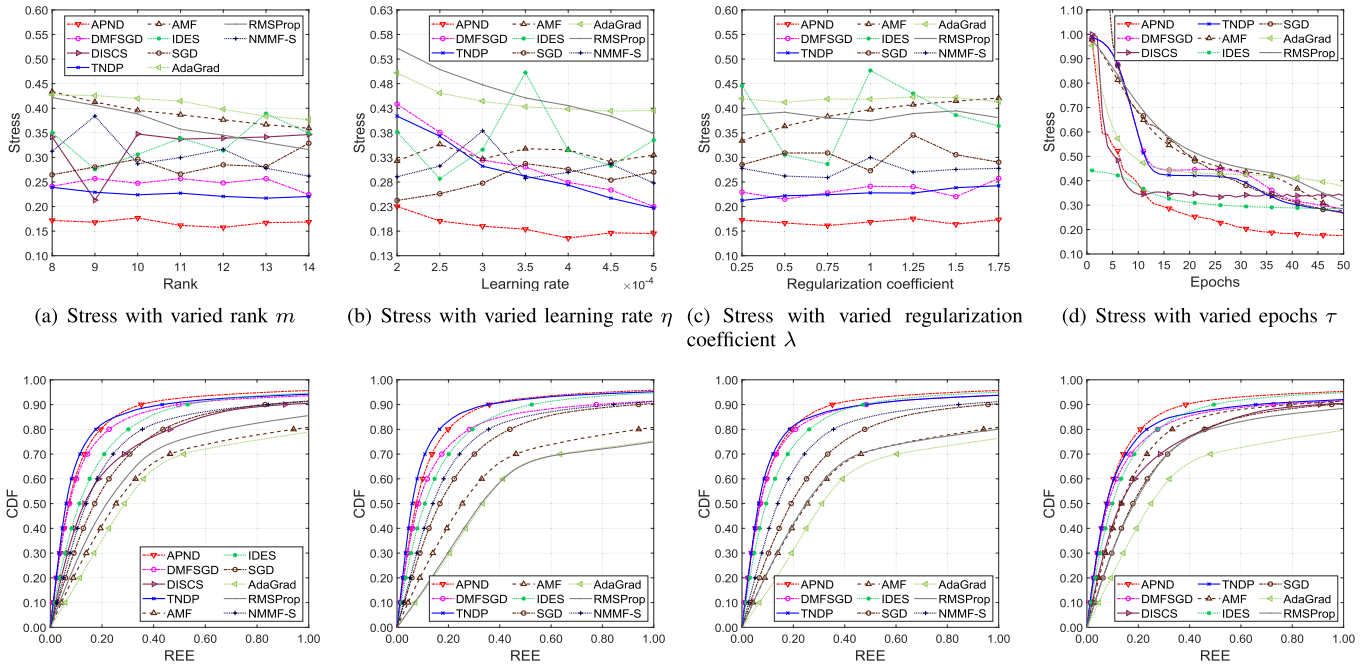
This metric is used to measure the relative errors between distance entries included in D and $\hat{D}$.

Computing cost efficiency: The metric is defined as the quotient of inverse prediction errors to the computing cost, which represents the prediction efficiency of matrix factorization. Since the computing resources are consumed in each epoch, its cost is proportional to the number of epochs. Lower prediction errors with fewer epochs represent higher efficiency.

C. Simulation Results

1) Convergence of APND: The convergence with the varied parameters of APND is illustrated in Fig. 5, evaluated in the form of loss functions, defined in Eq. (23), during 50 episodes in model training. In Fig. 5(a), the rank m of factorized matrices is set as 8, 10, 12 and 14, respectively. It is obvious to see that the loss functions gradually decrease to 0 with different values of rank m and global models converge the fastest when the rank m is set to be 10, which demonstrates the effectiveness of APND for accurate network distance prediction. In order to find the optimal parameters of APND for the fastest convergence, additional simulation experiments with the varied parameters of learning rate η and regularization coefficient λ have been implemented and listed in Figs. 5(b) and 5(c). In Fig. 5(b), the learning rate η of matrix factorization is varied from 1×10^{-4} to 4×10^{-4} while the optimal rank m is set as 10. It is clear to see that the loss function gradually decreases in first 6 episodes and maintains stability at the 12th episode when the learning rate η is set to be 4×10^{-4}, meaning that global models achieve convergence rapidly. Built on this, it is illustrated in Fig. 5(c) that the learning rate η is set as 4×10^{-4} and the regularization coefficient λ is varied from 0.75 to 1.25. All the loss functions start to remain stable at the 20th episode, which indicates the global training has achieved the convergence. By combining the advantages of SGD, AdaGrad and RMSProp, APND has learned an optimized factorization strategy to achieve quick convergence and high accuracies. Furthermore, as Fig. 5(d) illustrated, global models with λ=1.5 have quicker convergence than others in first 30 episodes, but in latter episodes its performance is not stable enough for accurate network distance prediction. When λ is equal to 1.25, the loss functions have a slower but more stable convergence than others, which indicates global models with this parameter can satisfy the requirements of accurate network distance prediction. In order to achieve accurate network distance prediction, we will set rank m=10, learning rate η=0.0004 and regularization coefficient λ=1.25 in our subsequent experimental simulations.

2) Comparisons of Prediction Accuracies: In this subsection, we compare the stress of APND with DMFSGD, DISCS, TNDP, AMF, IDES, SGD, AdaGrad, RMSProp and NMMF-S with the varied parameters of rank m, learning rate η and regularization coefficient λ. Because the network distance in TNDP is expressed with confidence intervals, we take the mean values of confidence intervals to represent the predicted network distances. The rank m is set in the range of [8,14], the learning rate η is in the range of [2×10^{-4},5×10^{-4}], and the regularization coefficient λ is varied from 0.25 to 1.75. The stress of these ten solutions is demonstrated in Fig. 6(a)



(a) Stress with varied rank m (b) Stress with varied learning rate η (c) Stress with varied regularization coefficient λ (d) Stress with varied epochs τ

(e) Relative error with rank $m = 12$ (f) Relative error with learning rate η=4×10^{-4} (g) Relative error with regularization coefficient $\lambda = 0.75$ (h) Relative error with optimal parameters

Fig. 6. Prediction accuracies of APND, DMFSGD, DISCS, TNDP, AMF, IDES, SGD, AdaGrad, RMSProp and NMMF-S with the varied parameters of learning rate η, rank m and regularization coefficient λ in PlanetLab490.

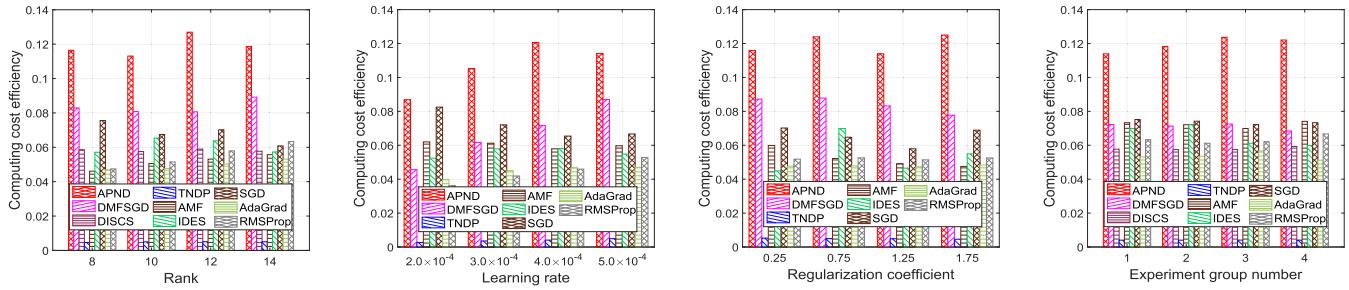
when the rank m changes from 8 to 14. Compared with DMFSGD, DISCS, TNDP, AMF, IDES, SGD, AdaGrad, RMSProp and NMMF-S, our proposed APND has obtained the least stress, with the reductions of 32.34%, 48.31%, 25.68%, 56.97%, 49.01%, 41.25%, 58.75%, 53.92% and 44.47% on average, respectively. In Fig. 6(e), the relative errors of these approaches are demonstrated as m=12. It is obvious to see that the relative errors of 85% network distance are lower than 0.23, which indicates that APND can achieve more accurate prediction than other approaches.

The stress of APND, DMFSGD, TNDP, AMF, IDES, SGD, AdaGrad, RMSProp and NMMF-S with the varied learning rate η is demonstrated in Fig. 6(b). Due to no learning rate η and regularization coefficient λ, DISCS is not considered in this group of simulations. It is illustrated that APND achieves the minimized stress with different learning rate η compared with other approaches. Specifically, APND outperforms DMFSGD, TNDP, AMF, IDES, SGD, AdaGrad, RMSProp and NMMF-S on optimizing the stress, with the reductions of 46.64%, 36.78%, 43.72%, 46.51%, 32.22%, 57.70%, 58.77% and 38.84% in Fig. 6(b), respectively. Such achievements further prove that the performance of APND is superior to other prediction schemes with different learning rates. Furthermore, when the learning rate η is equal to 4×10^{-4}, APND has achieved the minimum stress, where the proportion of relative errors (≤ 0.38), as shown in Fig. 6(f), is higher than others. This is mainly due to that the LSTM-based experience of learning rate η with multiple rules used for matrix factorization has been reserved and exploited.

The stress of APND, DMFSGD, TNDP, AMF, IDES, SGD, AdaGrad, RMSProp and NMMF-S with the varied regularization coefficient λ is illustrated in Fig. 6(c). Compared with them, APND achieves the minimum results again, with the reductions of 27.53%, 25.74%, 56.29%, 54.87%, 43.89%, 59.61%, 56.16% and 38.40%, respectively. When the regularization coefficient λ is set to be 0.75, APND obtains the optimal stress, which is equal to 0.1613, and 80% relative errors less than 0.20, as illustrated in Fig. 6(g), which is slightly lower than TNDP but observably higher than the others by introducing multiple rules and LSTM-based experience for matrix factorization.

Built on the previous simulation experiments, Fig. 6(d) demonstrates the varied stress of APND, DMFSGD, DISCS, TNDP, AMF, IDES, SGD, AdaGrad and RMSProp with the optimal parameters during the epochs. Due to no epoch of matrix factorization, NMMF-S is not taken into account in this comparison. As shown in Fig. 6(d), the stress of DISCS decreases the fastest, but stops decreasing at the 10th epoch, which is not low enough for accurate distance prediction. DMFSGD, TNDP, IDES and RMSProp finally achieve lower stress than DISCS, but their slow decrease of stress cannot satisfy the requirements of accurate distance prediction. However, APND acquires the lower stress e_s=0.2856 at the 16th epoch and achieves the most accurate result e_s=0.1764 with 90% relative errors less than 0.40, shown in Fig. 6(h), meaning that it is able to obtain the optimal result of stress with the least epochs. Such achievements indicate that APND has learned an optimal matrix factorization strategy for network distance prediction built on LSTM-based experience exploitation and multiple rules of factorization learning.

3) Comparisons of Computing Cost Efficiency: In this subsection, the computing cost efficiency of APND, DMFSGD, DISCS, TNDP, AMF, IDES, SGD, AdaGrad and RMSProp, is illustrated in Fig. 7. In order to accurately infer



(a) The varied rank m (b) The varied learning rate η (c) The regularization coefficient λ (d) The optimal parameter

Fig. 7. Computing cost efficiency of APND, DMFSGD, DISCS, TNDP, AMF, IDES, SGD, AdaGrad and RMSProp with the varied parameters of learning rate η, rank m and regularization coefficient λ in PlanetLab490.

the computing cost efficiency of these solutions, we set the times of epochs to be 50. The rank m is set as 8, 10, 12 and 14 in Fig. 7(a), the learning rate η is in the range of [2×10^{-4},5×10^{-4}] in Fig. 7(b), while the regularization coefficient λ is varied between 0.25 and 1.75 in Fig. 7(c). Their optimal values are demonstrated in Fig. 7(d). It is clear to see from Figs. 7(a)-7(d) that the computing cost efficiency of TNDP is distinctly lower than the others, because additional epochs are required to factorize the built tensor, which includes a number of matrices instead of a single one. On the whole, APND outperforms DMFSGD, DISCS, TNDP, AMF, IDES, SGD, AdaGrad and RMSProp on improving computing cost efficiency, with the average achievements of 47.64%, 88.39%, 2314.35%, 132.35%, 95.77%, 74.41%, 140.51% and 117.88% with varied rank m in Fig. 7(a), 63.27%, 2774.31%, 77.87%, 90.99%, 51.76%, 138.23% and 142.51% with different learning rate η in Fig. 7(b) as well as 33.02%, 2244.37%, 131.41%, 127.07%, 83.69%, 150.46% and 129.73% with the changed regularization coefficient λ in Fig. 7(c), compared with other schemes. This indicates that an optimized strategy for matrix factorization has been learned by APND to obtain accurate results with lowest computing cost efficiency. Built on these findings, the computing cost efficiency of them with optimal parameters is illustrated in Fig. 7(d), which shows that APND always obtains the optimal results in computing cost efficiency, with the achievements of 68.02%, 105.97%, 2853.11%, 65.29%, 83.18%, 62.19%, 123.15% and 88.86% compared with others on average.

4) Time Complexity Analysis: Let T and N denote the iterations of matrix factorization and the number of nodes in networks, respectively. Thus, the time complexity of APND, DMFSGD, DISCS, TNDP, AMF, IDES, SGD, AdaGrad, RMSProp and NMMF-S can be given in TABLE III.

APND is dependent on local model learning and global model aggregation to execute accurate network distance prediction. The time complexity of local model learning is mainly determined by the framework of predictive networks and target networks. Notice that the size of state space is related to N nodes and the rank R of $N\times N$ factorized matrices, therefore, the number of neurons in the input layer of predictive network and target network can be expressed as (NR). Because the output layer depends on the action space $\mathcal{A}$, then the number of its neurons can be denoted as $|\mathcal{A}|$. Let W and P denote the number of agents and hidden layers of LSTM in a DDPG model, respectively, and B stands for the times of global model aggregation. Thus, the time complexity of each local model training and the FDRL-based model aggregation can be expressed as $O(TNRP^2|\mathcal{A}|)$ and $O(BW)$, respectively. Therefore, the final time complexity of APND can be expressed as $O(TNRP^2|\mathcal{A}|+BW)$.

TABLE III
TIME COMPLEXITY

Approaches	Time Complexity
APND	$O(TNRP^2\|\mathcal{A}\|+BW)$
DMFSGD	$O(TNK)$
DISCS	$O(TNKFG)$
TNDP	$O(TNKI)$
SGD/AdaGrad RMSProp/AMF	$O(TN^2)$
IDES	$O(TNQ)$
NMMF-S	$O(IH^2P^2)$

DMFSGD allows N network nodes to exchange messages with each other and only requires each node to implement local measurements and coordinate epochs. Thus, the time complexity of it is determined by the number of nodes N and their K neighbor nodes, denoted as $O(TNK)$.

DISCS introduces the innovative robust non-negative matrix completion to improve prediction accuracies. The time complexity to build distance matrix is dependent on N nodes and K nodes located in each neighbor set, and can be represented as $O(KN)$. The time complexity to decompose matrix lies on the consecutive rounds F, G and T in 3-step-nested non-negative matrix completion, and can be given by $O(TFG)$. Thus, the total time complexity of it can be represented as $O(TNKFG)$.

TNDP makes use of the $N\times N\times I$ random distance tensor and executes SGD-based distributed matrix factorization. The time complexity can be represented as $O(TNKI)$, where I and K denote the dimension of random distance tensor and the number of neighbor nodes, respectively.

Being the classical matrix factorization, SGD decomposes $N\times N$ matrix through T iterations. Both AdaGrad and RMSProp are SGD-based variants. Therefore, the time complexity of them can be given by $O(TN^2)$. AMF decomposes

$N \times N$ matrix by exploiting Box-Cox-based data transformation and SGD-based matrix factorization with T iterations. Thus, the time complexity of it can be represented as $O(TN^2)$.

IDES takes advantage of SVD/NMF-based method to decompose $N \times N$ matrix, where there are Q inter-landmarks, to obtain the accurate network distances through T iterations. Thus, the time complexity of it can be given by $O(TNQ)$.

Being DL-based distance data recovery, NMMF-S makes use of the historical measurements in past I time slots to infer the current network distances, and works in context extraction and generation modules. The former module includes a fully connected layer and a GRU layer, while the latter module includes a fully connected layer. There exists H neurons in fully connected layer and P GRU hidden layers. Thus, the time complexity of NMMF-S can be given by $O(IH^2P^2)$.

VI. Conclusion

This paper has presented APND, a novel FDRL-based network distance prediction approach, for network-aware applications. By utilizing the partial network measurements among randomly-selected nodes, a set of distance matrices have been established in all clusters to be intelligently factorized to explore unknown distances. The multiple matrix-factorization rules and the LSTM-based learning experience with distinct achievements have been parallelly exploited to train the local models with fewer prediction errors and quicker convergence. Furthermore, two benefit-based aggregation policies have been proposed to further enhance local prediction models and accelerate global convergence in an efficient manner. Besides, the error-based and average-based refinement policies are proposed to adjust the coordinates of nodes which are simultaneously included in adjacent clusters. Extensive simulations over real data have shown that APND has outperformed the previous approaches on network distance prediction. In reality, APND can be used to accurately predict network distance for distributed applications if a number of FDRL-based agents have been deployed in networks and some network distances among nodes can be measured with the widely-used network monitor servers.

Acknowledgment

This article reflects only the authors' view. The European Union Commission is not responsible for any use that may be made of the information it contains.

References

[1] A. Thune, S.-A. Reinemo, T. Skeie, and X. Cai, "Detailed modeling of heterogeneous and contention-constrained point-to-point MPI communication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 5, pp. 1580–1593, May 2023.

[2] Z. Lai, W. Liu, Q. Wu, H. Li, J. Xu, and J. Wu, "SpaceRTC: Unleashing the low-latency potential of mega-constellations for real-time communications," in *Proc. IEEE Conf. Comput. Commun.*, May 2022, pp. 1339–1348.

[3] Z. Zhang, H. Wang, H. Lv, J. Sun, G. Li, and X. Han, "CHAT: Accurate network latency measurement for 5G E2E networks," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 2854–2869, 2023.

[4] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun. Societies*, Jun. 2002, pp. 170–179.

[5] Y. Liao, W. Du, P. Geurts, and G. Leduc, "DMFSGD: A decentralized matrix factorization algorithm for network distance prediction," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1511–1524, Oct. 2013.

[6] J. Cheng, Y. Liu, Q. Ye, H. Du, and A. V. Vasilakos, "DISCS: A distributed coordinate system based on robust nonnegative matrix completion," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 934–947, Apr. 2017.

[7] Q. Wang and S. Zhang, "DGL: Device generic latency model for neural architecture search on mobile devices," *IEEE Trans. Mobile Comput.*, vol. 23, no. 2, pp. 1–14, Feb. 2023.

[8] M. Costa, M. Castro, R. Rowstron, and P. Key, "PIC: Practical Internet coordinates for distance estimation," in *Proc. 24th Int. Conf. Distrib. Comput. Syst.*, Mar. 2004, pp. 178–187.

[9] T. E. Ng and H. Zhang, "A network positioning system for the internet," in *Proc. USENIX Annu. Tech. Conf.*, Jun. 2004, pp. 141–154.

[10] Y. Mao, L. K. Saul, and J. M. Smith, "IDES: An Internet distance estimation service for large networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2273–2284, Dec. 2006.

[11] H. Huang, L. Li, G. Min, W. Miao, Y. Zhu, and Y. Zhao, "TNDP: Tensor-based network distance prediction with confidence intervals," *IEEE Trans. Services Comput.*, vol. 15, no. 6, pp. 3554–3565, Nov. 2022.

[12] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011.

[13] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

[14] L. Tang and M. Crovella, "Virtual landmarks for the Internet," in *Proc. Conf. Internet Meas. Conf.*, 2003, pp. 143–152.

[15] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 15–26, Aug. 2004.

[16] H. Lim, J. C. Hou, and C.-H. Choi, "Constructing Internet coordinate system based on delay measurement," in *Proc. Conf. Internet Meas. Conf.*, 2003, pp. 129–142.

[17] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in Euclidean space," *IEEE/ACM Trans. Netw.*, vol. 12, no. 6, pp. 993–1006, Dec. 2004.

[18] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality variations in the Internet," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas.*, Nov. 2009, p. 177.

[19] Y. Chen et al., "Phoenix: A weight-based network coordinate system using matrix factorization," *IEEE Trans. Netw. Service Manage.*, vol. 8, no. 4, pp. 334–347, Dec. 2011.

[20] Y. Fu and X. Xiaoping, "Self-stabilized distributed network distance prediction," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 451–464, Feb. 2017.

[21] H. Huang, W. Miao, G. Min, C. Huang, X. Zhang, and C. Wang, "Resilient range-based d-dimensional localization for mobile sensor networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 5, pp. 2037–2050, Oct. 2020.

[22] R. Tripathi and K. Rajawat, "Adaptive network latency prediction from noisy measurements," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 807–821, Mar. 2021.

[23] K. Xie, R. Xie, X. Wang, G. Xie, D. Zhang, and J. Wen, "NMMF-Stream: A fast and accurate stream-processing scheme for network monitoring data recovery," in *Proc. IEEE Conf. Comput. Commun.*, May 2022, pp. 2218–2227.

[24] X. Li et al., "Tripartite graph aided tensor completion for sparse network measurement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 1, pp. 48–62, Jan. 2023.

[25] D. B. Chua, E. D. Kolaczyk, and M. Crovella, "Network kriging," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2263–2272, Nov. 2006.

[26] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Netw.*, vol. 12, no. 1, pp. 145–151, Jan. 1999.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[28] J. Zhu, P. He, Z. Zheng, and M. R. Lyu, "Online QoS prediction for runtime service adaptation via adaptive matrix factorization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2911–2924, Oct. 2017.

[29] (2016). *Network Distances Datasets*. [Online]. Available: https://github.com/uofa-rzhu3/NetLatency-Data

[30] J. Stribling. (2014). *All Pairs Ping Data for Planetlab*. [Online]. Available: http://www.pdos.lcs.mit.edu/~strib/pl_app

Haojun Huang received the B.S. degree in computer science from Wuhan University of Technology in 2005 and the Ph.D. degree in communication and information engineering from the University of Electronic Science and Technology of China in 2012. He is an Associate Professor with the School of Electronic Information and Communications, Huazhong University of Science and Technology, China, and the Marie Skłodowska-Curie Fellow with the Department of Computer Science, University of Exeter, Exeter, U.K. His current research interests include artificial intelligence, wireless communications, network function virtualization, software-defined networking, and the Internet of Things.

Yiming Cai received the B.S. degree in communication engineering from Huazhong University of Science and Technology, China, in 2021, where he is currently pursuing the master's degree in information and communication engineering. His research interests include computer networks and reinforcement learning.

Geyong Min (Member, IEEE) received the B.S. degree in computer science from Huazhong University of Science and Technology, China, in 1995, and the Ph.D. degree in computing science from the University of Glasgow, U.K., in 2003. He is a Professor of high performance computing and networking with the Department of Computer Science, College of Engineering, Mathematics and Physical Sciences, University of Exeter, U.K. His research interests include computer networks, wireless communications, parallel and distributed computing, ubiquitous computing, multimedia systems, and modeling and performance engineering.

Haozhe Wang received the B.S. and M.S. degrees from Dalian University of Technology, Dalian, China, in 2005 and 2012, respectively, and the Ph.D. degree from the University of Exeter, U.K., in 2017. He is currently a Research Associate with the Computer Science Department, University of Exeter. His research interests include AI-empowered network automation, future internet architecture, information-centric networking, and network analytical modeling.

Gaoyang Liu (Member, IEEE) received the B.S. and Ph.D. degrees from Huazhong University of Science and Technology, China, in 2015 and 2021, respectively. He is currently a Post-Doctoral Researcher with the School of Computing Science, Simon Fraser University, BC, Canada. His research interests include trustworthy machine learning, mobile sensing, and data privacy protection.

Dapeng Oliver Wu (Fellow, IEEE) received the B.E. degree in electrical engineering from Huazhong University of Science and Technology, Wuhan, China, in 1990, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2003. He is currently a Chair Professor with the Department of Computer Science, City University of Hong Kong. His research interests are in the areas of networking, communications, signal processing, computer vision, machine learning, smart grids, and information and network security. He has served as an Editor-in-Chief for IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING; an Associate Editor for IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY; and a Guest Editor for IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS.