Knowledge is at the Edge! How to Search in Distributed Machine Learning Models

Thomas Bach, Muhammad Adnan Tariq, Ruben Mayer, and Kurt Rothermel

Institute of Parallel and Distributed Systems, University of Stuttgart, Germany,
⟨firstname.lastname⟩@ipvs.uni-stuttgart.de,
http://www.uni-stuttgart.de

Abstract. With the advent of the internet of things and industry 4.0 an enormous amount of data is produced at the edge of the network. Due to a lack of computing power, this data is currently send to the cloud where centralized machine learning models are trained to derive higher level knowledge. With the recent development of specialized machine learning hardware for mobile devices, a new era of distributed learning is about to begin that raises a new research question: How can we search in distributed machine learning models? Machine learning at the edge of the network has many benefits, such as low-latency inference and increased privacy. Such distributed machine learning models can also learn personalized for a human user, a specific context, or application scenario. As training data stays on the devices, control over possibly sensitive data is preserved as it is not shared with a third party. This new form of distributed learning leads to the partitioning of knowledge between many devices which makes access difficult. In this paper we tackle the problem of finding specific knowledge by forwarding a search request (query) to a device that can answer it best. To that end, we use a entropy based quality metric that takes the context of a query and the learning quality of a device into account. We show that our forwarding strategy can achieve over 95 % accuracy in a urban mobility scenario where we use data from 30 000 people commuting in the city of Trento, Italy.

Keywords: Knowledge retrieval, Distributed knowledge, Query routing

1 Introduction

In many areas such as stock trading, drug design, manufacturing, and urban mobility [9, 16] machine learning is the key enabler of optimization and driver of performance [9, 21, 15]. Besides choosing the right machine learning algorithm and applying it right, the amount of training data is key to success [9]. While the selection and application of machine learning algorithms is a research field of its own, enough training data is needed to calibrate machine learning models, such that they can make correct predictions.

With the advent of paradigms like the Internet of Things, smart city, and Industry 4.0, data will be abundantly available [21]. Cisco, for example, estimates that the I.o.T. alone will generate over 400 ZB of data annually, by 2020 [1]. In particular, the proliferation of smart phones made training data from different sensors, such as accelerometers, cameras, microphones, and GPS units widely available [15]. Google reported that by centralizing a great amount of training data for speech recognition from Google voice search [32], it became possible to train high-quality feature-rich machine learning models for voice recognition[12].

The current approach to share such information is massive centralization. In many application scenarios, however, centralization of possibly sensitive data is not desirable as centralized data is regularly subject to breaches [14]. Today, it is well known that it is possible to derive knowledge of a user's habits, such as his home and work location from his GPS traces [4]. Many human users are thus unwilling, at least uncomfortable sharing such private information [39].

The common approach to tackle this issue is to distribute the computing infrastructure [24], and even push computing towards the edge of the network [11, 26]. The upcoming trend of *fog computing*[13, 23] supports this by providing computational resources close to the edge, creating a computational continuum that spans from the edge devices to the centralized cloud data centers. Sensitive data can then be processed directly on devices that are under control of the user or on fog nodes very close to them. In this respect, mobile device manufactures are building specialized machine learning hardware that enables machine learning at the edge[1]. Machine learning at the edge has many additional advantages, it allows for example to keep the user in the loop, learn personalized, and offer low latency feedback [11, 26]. Google for example has recognized this trend and made approaches, where personalized learning is done directly on smart phones [25]; however, the generated local machine learning models are synchronized with a central server. Google argues, that by processing the data locally, privacy is increased compared to an entirely centralized approach.

Completely decentralized learning also holds great challenges. As training data is not centralized, each machine learning model is only trained with respect to its local experiences. In particular, such models may become local experts that are very good in predicting local phenomena. In a medical scenario this might be an advantage, as a model could learn the peculiarities of one specific patient and enable a detailed analysis. For other use cases, this is not enough. In an urban mobility scenario, for example, users are usually more interested, in traffic conditions in another part of the city which they have never seen before. Distributed learning holds the opportunity to learn about local phenomena in great detail on the one hand, on the other hand it creates the problem of locating specific knowledge.

To address this problem, in this paper, we present methods to route a query for specific knowledge through a network of nodes (local experts) that each

[1] Mark Gurman; BloombergTechnology, Apple Is Working on a Dedicated Chip to Power AI on Devices: https://www.bloomberg.com/news/articles/2017-05-26/apple-said-to-plan-dedicated-chip-to-power-ai-on-devices

train a local machine learning model. Our goal is to forward such a query to the node that can answer it best. In particular, we look at scenarios where knowledge in the form of machine learning models is fully distributed. Such a fully decentralized approach holds three mayor difficulties: First, we cannot assume a central index of all available knowledge. Second, the different devices (nodes) might learn based on different local observations and contexts. Third, parts of the knowledge changes or becomes outdated over time.

To this end, our contributions are: (1) We propose a decentralized routing strategy that forwards queries for specific knowledge towards nodes that can answer them best. (2) We propose methods to maintain routing tables that guide the forwarding of such a query. (3) We use entropy to evaluate how good a given query can be answered based on its context and the local machine learning model of a node. (4) We develop a modified form of the Barabasi Albert model [2] to generate a scale free topology that clusters network nodes with similar knowledge close together to deal with heterogeneous knowledge. With its scale free properties such a topology provides short paths between any two network nodes and is robust against node failures. (5) We show that we can achieve over 95 % accuracy when using synthetic data and data generated by a mobility simulator where 30 000 people commute in the city of Trento, Italy, in the context of different weather conditions, times of the day, and traffic conditions.

2 System Model and Problem Formulation

We assume a distributed system of fog [13, 23] nodes that each train a machine learning model based on local observations. These nodes can join and leave the system at any time and range from user managed devices such as smart phones, laptops, and desktop computers, to infrastructure based services located in data centers, such as private clouds. All nodes communicate directly over a undirected, scale free topology, i.e. power law distributed node degree and short paths. These properties make scale free networks particularly well suited for our problem as they connect two arbitrary nodes (e.g. the source and optimal destination of a query) with a small number of hops. Furthermore, many existing networks such as social networks or the internet already show scale free properties [3]. Maintaining such a topology is also a well studied research problem [17, 2].

In order to learn, all nodes maintain a graphical machine learning model as shown in Fig. 1. Graphical models (Probabilistic Graphical Models, PGM) such as Bayesian networks or conditional random fields have a wide range of machine learning applications in computer vision, natural language processing, and bioinformatics [36]. In a PGM, random variables are represented as nodes and dependencies between them as edges of a graph. This gives them great flexibility in modeling complex dependencies.

We assume, that all network nodes maintain structural identical PGMs that are continuously evolving based on individual training data (observations). This training data can be generated either by the nodes themselves, e.g. a smart phone generates GPS traces from its internal sensors, or can be received from other sen-

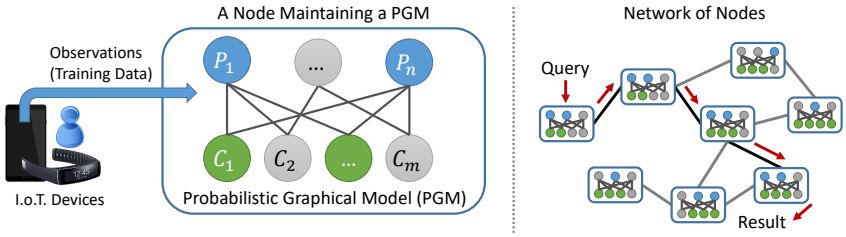

Fig. 1. System overview.

sors, such as a wristband sensor that collects cardiovascular data. Furthermore, the different nodes learn about different phenomena in different contexts, leading to individual expertise of the different nodes. In an urban mobility scenario, for example, two nodes could learn about traffic conditions in different parts of the city at different times of the day. In particular, this means that the different nodes train different subsets of random variables. In consequence, not all nodes can predict all random variables equally well. A reliable prediction about the outcome of a specific random variable thus requires to search for (i.e. query) the network node that has best training.

Given any PGM, we categorize the random variables of the PGM into two groups: predicting variables and context variables. Predicting variables are the subset of random variables that we want to predict based on a certain context, modeled as context variables. In an urban mobility scenario, where we want to predict the travel time for the streets in a city with a PGM, the travel time for each street would be represented as a predicting variable. Factors that influence this travel time, such as weather or time of the day would be represented as context variables. In Fig. 1 this categorization is reflected by the color of the random variable. Predicting variables P_n in blue, context variables C_n in green and untrained random variables of both types in grey.

In this context, we define a query as a request to predict the outcome for a specific predicting variable in a certain context. In this respect, we define context as given assignments for a set of context variables. Our goal is to forward a query to a node that can answer it with the highest possible quality (we introduce quality in Sec: 3).

2.1 Formal Model and Problem Statement

More formally, we assume a set of network nodes $N = \{N_1, ..., N_n\}$ that are connected over a scale free topology. Each node N_x holds a PGM that consists of a graph $G = (V, E)$ of discrete random variables V where dependencies between random variables are modeled by the edges E. We classify the random variables into predicting variables $P_n = \{p_n^1, ..., p_n^m\}$ and context variables $C_n = \{c_n^1, ..., c_n^{m'}\}$, where p_n and c_n are possible assignments. Each random variable must be classified either as predicting variable or as context variable ($V = C \cup P$ and $C \cap P = \emptyset$).

We define a query $\overrightarrow{q} = (P_x, \{c_x^y, ..., c_{x'}^{y'}\}, H, R, Q, \overrightarrow{N})$, where P_x is the random variable that needs to be predicted in context of a given set of assignments ($\{c_x^y, ..., c_{x'}^{y'}\}$) for a subset of context variables and a limited number of hops H (number of times a query can be forwarded). Furthermore, the query contains a field to hold the prediction result R, the estimated quality of this result Q and a vector of visited nodes $\overrightarrow{N}$.

We can now define the concrete knowledge retrieval problem. Given i) a set of nodes $\{N_1, ..., N_n\}$ holding ii) continuously evolving, heterogeneously trained PGMs and iii) a query $\overrightarrow{q}$ for specific knowledge, our goal is to maximize the retrieval quality of a query while forwarding it only H times.

In the following, we first establish a notion of knowledge quality in the context of PGMs and describe how to measure the quality with that a node can answer a query. (cf. Sec. 3). Based on this quality metric, we present methods to route a query towards the node that can answer it with highest quality in Sec. 4.

3 Entropy, a Measure of Training Quality

In this section we discuss how we measure the training quality of a PGM. Based on this quality, we describe how to estimate the quality with that a PGM can answer a query for specific knowledge.

As stated above, Probabilistic Graphical Models (PGM) consist of interdependent random variables. Such models are usually designed by an expert who puts his domain knowledge in the structure of the model, e.g. chooses the random variables and their conditional dependencies such that the model reflects the dependencies in the real world. Training data is then used to converge the probability distributions of the random variables from a uniform distribution to the distributions of the real world. In other words, if an increasing amount of training data is fed into the machine learning model, the uncertainty of the model decreases. In machine learning, this uncertainty (or often called surprise) of a model is measured by calculating the entropy of its random variables [20].

If, for example, we want to learn the probability of a coin flip being "Heads Up", we could use a very simplistic model that only consists of one random variable X with possible outcomes $\{0\%, ..., 100\%\}$. We now flip a fair coin several times and use the results to train the random variable as shown in Fig. 2. With an increasing number of observations (or coin tosses) the "true" probability distribution establishes and the entropy decreases.

Given a random variable X with possible assignments $\{x_1, ..., x_n\}$ we can calculate the entropy ($H(X)$) as the average surprise (or uncertainty) of the random variable (cf. Eq 1). The logarithm of the probability of an assignment $log(P(x_n))$ represents the amount of surprise we perceive for the specific outcome [20]. The "surprises" of all possible outcomes are then weighted by their probability $P(x_n)$ and summed up to one entropy value often also called self-information [20].

$$H(X) = -\sum_{k=1}^{n} P(x_k) log_2 P(x_k) \tag{1}$$

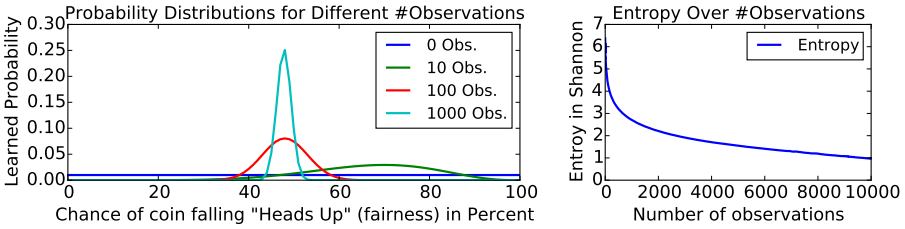

Fig. 2. Probability distribution (l) and entropy (r) of a coin flip for different number of observations.

In complex machine learning models, we usually want to predict the outcome of multiple random variables. In these cases we calculate the joint entropy $H(X_0, ..., X_n)$ in order to describe their "joint uncertainty", e.g. $\{X_0, ..., X_n\}$ (cf. Eq. 2). Similar to entropy for one random variable, the idea is to calculate the uncertainty for each combination of random variables involved and weight these combinations w.r.t. their probabilities.

$$H(X_0, ..., X_n) = -\sum_{x_0 \in X_0} \cdots \sum_{x_n \in X_n} P(x_0, ..., x_n) log_2 P(x_0, ..., x_n) \tag{2}$$

The joint entropy describes the "total uncertainty" of multiple random variables. Given a random variable X_0 that is dependent on the outcome of other random variables $\{X_1, ..., X_n\}$, the entropy $H(X_0, ..., X_n)$ denotes the uncertainty of the outcome given that we don't know anything about the outcome of $\{X_0, ..., X_n\}$. If we now gain information about the outcome of one of the variables (e.g. X_0 is a context variable and its outcome is given by a query), we can derive the remaining entropy (uncertainty) according to the chain rule of conditional entropy by subtracting the entropy $H(X_0)$ from the total entropy $H(X_0, ..., X_n)$, cf. Eq. 3. In the following, we use this chain rule to calculate the uncertainty which the PGM of a specific network node has to answer a query.

$$H(X_1, ..., X_n | X_0) = H(X_0, ..., X_n) - H(X_0) \tag{3}$$

In Sec. 2 we divided the random variables of a PGM in two categories, predicting variables and context variables, where each predicting variable is dependent on the outcomes of a number of independent context variables. For a given predicting variable P_0 that is dependent on the outcome of context variables $\{C_0, C_1, C_2\}$ we can calculate the joint entropy $H(P_0, C_0, C_1, C_2)$ and individual entropies for the context variables $H(C_0), H(C_1), H(C_2)$. Given a query $\overrightarrow{q} = (P_0, \{c_0^1, c_1^3\}, ...)$[2] for P_0 with observed outcomes $c_0^1 \in C_0$ and $c_1^3 \in C_1$ we can calculate the remaining uncertainty of the PGM to answer the query by subtracting the entropy of the context variables from the joint entropy of the

[2] For better readability we do not state all fields of the query here (i.e. H, R, Q, $\overrightarrow{N}$).

predicting variable (cf. Eq. 4). This results in the remaining uncertainty of the PGM to answer the query.

$$H(P_0, C_3|C_0, C_1) = H(P_0, C_0, C_1, C_2) - H(C_0) - H(C_1) \quad (4)$$

For the rest of this paper we will also refer to entropy as the learning or training quality of a PGM.

4 Routing

Now that we have established how we can measure the training quality of the PGMs of each node, we describe how we build routing models and use them to forward queries towards the node that can answer them best. In contrast to a classic routing table, where a network address is associated with a specific port (outgoing link), each node N_i maintains a routing model $RM_{N_n}^{N_i}$ for each neighbor N_n. Each $RM_{N_n}^{N_i}$ serves as a descriptive model that cumulatively represents the knowledge available over the respective outgoing link. Keeping link-individual routing models is necessary, because we need to calculate the estimated answering quality of a query with respect to its context (cf. Sec. 3) as storing all possible combinations of contexts easily becomes too much overhead.

In order to process a query, a node first tries to improve the prediction R of a query based on its local PGM. In the next step, the node uses its routing models to determine to which neighbor the query should be forwarded. As this is an approximate routing process, we limit the number of hops (H) that a query $\overrightarrow{q}$ is forwarded before the result is returned to the sender.

In order to improve the retrieval quality we use a network topology that clusters nodes with similar knowledge (nodes that have learned about a similar set of predicting nodes). We can then optimize our routing models by maintaining context information of predicting variables only for predicting variables that have been learned by the cluster. This leads to a double-staged routing approach, where a query for a predicting variable P_n is first forwarded to a cluster of nodes that have learned about P_n. In the second stage, the query is then forwarded within a cluster to a node that has learned it in the requested context. In the following we describe how we build the routing tables, forward a query, maintain the network topology and deal with loops in the topology in detail.

4.1 Building the Routing Tables

The routing models $RM_{N_n}^{N_i}$ that each node N_i maintains for every neighbor N_n store entropy values of random variables (predicting variables and context variables) and represent the knowledge available to the respective neighbor N_n. In order to maintain them in a proactive fashion, each node sends summaries of its entropy values stored in its local PGM and its routing models to its neighbors, whenever they have changed above a certain threshold and a minimum amount of time has passed since the last update. This makes sure that all neighbors have

up-to-date information about their neighbors and at the same time avoids that the network is flooded with updates.

In the following, we explain this forwarding process with respect to a set of nodes $\{N_1, N_2, N_3\}$ in detail. For better readability and without loss of generality this example is with respect to one predicting variable P_0 and context variables $\{C_1, C_2\}$. A further simplification is the use of a flat topology, i.e. all nodes are connected in a line (cf. Fig. 3). In the given example, N_3 has only one neighbor (N_2) and therefore can directly forward its set of entropy values $\{H(P_0, C_1, C_2)_{PGM},\ H(C_1)_{PGM},\ H(C_2)_{PGM}\}$ from its PGM to its neighbor N_2, where they are used as entries in the routing table $RM_{N_3}^{N_2}$ (cf. Fig. 3 A). These entropy values represent the learning quality for P_0 available over the edge $N_2 \rightarrow N_3$. As described in Sec. 3 this values can be used to calculate the quality with which a query in any context (i.e. $\{\{C_0\}, \{C_1\}, \{C_0, C_1\}\}$ can be answered.

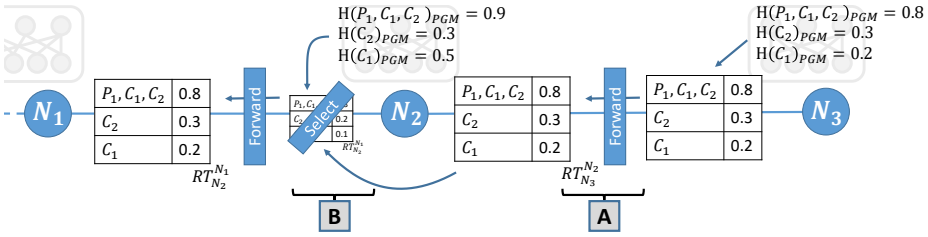

Fig. 3. Forwarding entropy values in same context.

When N_2 has received the set of entropy values from N_3, it decides to update the entropy values send to N_1. In contrast to N_3, N_2 cannot send the entropy values from its PGM directly as it has to consider the entropy received from N_3. Node N_2 needs to select which set of entropy values ($\{H(P_0, C_1, C_2),\ H(C_1),\ H(C_2)\}$) is forwarded. In order to determine this, it compares the joint entropy value of its local PGM ($H(P_0, C_1, C_2)_{PGM} = 0.9$) with the joint entropy value of its routing table $RM_{N_2}^{N_1}$ ($H(P_0, C_1, C_2) = 0.8$) As the local joint entropy is higher ($0.9 > 0.8$) it forwards the complete entropy set for P_0 received from N_3 ($\{H(P_0, C_1, C_2) = 0.8, H(C_1) = 0.2, H(C_2) = 0.3\}$) to N_1 (cf. Fig. 3 B).

Generalization: If, in contrast to our example, a node has multiple neighbors, it stores the entropy value sets it receives from them in a separate routing table for each neighbor. In order to decide which entropy set (e.g. $\{H(P_0, C_1, C_2),\ H(C_1),\ H(C_2)\}$) should be forwarded, we compare all the entropy sets, including the entropy of the local PGM by their joint entropy values (i.e. $H(P_0, C_1, C_2)$) and forward the set with the lowest joint entropy. This forwarding approach makes sure that for each predicting variable (P), the lowest joint entropy (e.g. $H(P_0, C_1, C_2) = 0.8$) and the entropy values of its corresponding context variables (e.g $H(C_1) = 0.2, H(C_2) = 0.3$) are propagated.

In cases where predicting variables have been trained with respect to different context variables by different nodes, entropy values cannot simply be merged.

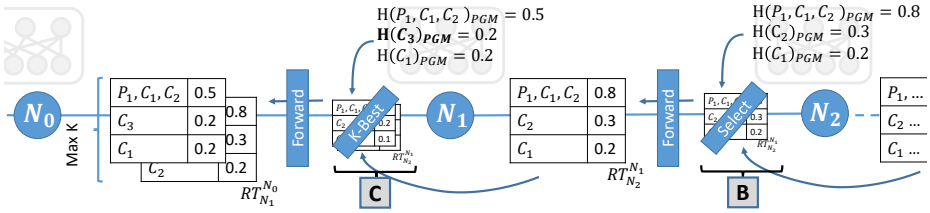

Fig. 4. Forwarding entropy values in different context.

For example, N_2 has trained P_1 with respect to context variables $\{C_1, C_2\}$ and N_1 has trained P_1 with respect to $\{C_1, C_3\}$ (cf. Fig. 4). In such cases, we store and forward up to $K \in \mathbb{N}$ different context combinations for each predicting variable P (cf. Fig. 4 C). In this respect, K is a design parameter that determines how many context combinations for one predicting variable are stored in the routing tables. In general K is dependent on the number of relevant contexts in a concrete scenario. In cases where we have to limit them we can use existing dimension reduction algorithms to select the most important context variables. We will discuss the influence of K in our evaluations in Sec. 5.

4.2 Forwarding of a Query

As discussed in Sec. 2, a query $\overrightarrow{q}$ is a message issued by one node in the network, to retrieve a prediction for a specific predicting variable P_x in a given context, represented by a set of assignments $(\{c_x^y, ..., c_{x'}^{y'}\})$ for a subset of context variables $\{C_x, ..., C_{x'}\} \in C$. A query is forwarded from one node to another until the predefined number of hops, H, has been reached.

When a node N_i receives a query $\overrightarrow{q}$ it first decreases the hop counter H of the query and then determines if the query can be improved by the local PGM, by computing the entropy of answering the query (cf. Sec. 3). The resulting entropy value is then compared to the entropy value Q in the query. If the entropy value in the query is higher than the locally computed value (i.e. the node has less uncertainty cf. Sec. 3), the node predicts the outcome of the query with its PGM and updates the result field R and the quality field Q of the query $\overrightarrow{q}$ accordingly. If the hop counter H of the query is greater then zero ($H > 0$) the node uses its local routing models to select a neighbor to which the query is forwarded.

In order to select a neighbor to send the query to, node N_i compares all routing models $RM_{N_n}^{N_i}$ by computing the conditional entropy $H(P_x|C_x, ..., C_{x'})$. This is done by subtracting the entropy values of the context variables $(\{H(C_x), ..., H(C_{x'})\})$ from the joint entropy value $H(P_x, C_x, ..., C_{x'})$ stored in the routing tables $RM_{N_n}^{N_i}$ (cf. Sec. 3). The query is then forwarded to the neighbor with the smallest conditional entropy $H(P_x|C_x, ..., C_{x'})$.

Discussion: So far, we have described how routing models are built and how they are used to forward a query. The maintenance of entropy values in multiple routing models, especially for different context combinations, produces significant overhead. The number of possible context combination grows according to

the binomial coefficient. If, for example, the nodes learns w.r.t. 5 out of 10 possible context variables, there are already 252 possible combinations. In order to reduce this overhead, we cluster nodes that have learned about a similar set of predicting variables. Based on this clusters, our routing protocol uses two optimizations. First, nodes only forward entropy values from their PGM if they have a minimum level of quality (i.e. the entropy value is below a certain threshold). Second, if a node has not reached a minimum quality for a predicting variable P_n it only maintains a single joint entropy value for P_n (no entropy values for context) in its routing models. This single joint entropy value can then be used to forward a query to the next cluster that has learned about P_n, where context sensitive routing, as described above, is performed. In the following we describe how we maintain such a clustered network topology.

4.3 Topology Maintainance

As mentioned in previous sections, the topology of our network should exhibit scale free properties, such as a power law distributed node degree and short paths. Additionally we want to cluster nodes that have learned about a similar subset of predicting variables. In order to manage the overhead of maintaining routing models for each neighbor, we also need to give each node the option to limit the maximum number of neighbors. This limit can be determined node-individually, e.g., dependent on the amount of memory consumed by the routing models. In order to generate such a topology, we use a modified version of the Barabasi Albert model [2]. Our idea is to make the preferential attachment of the Barabasi Albert model dependent on node similarity. The original algorithm starts with an initial set of m_0 nodes and connects a new node N_n to an existing node N_e with a probability proportional to the edge degree of the existing nodes. This way, N_n can connect with up to $m < m_0$ existing nodes. In the original algorithm the probability of a node N_x connecting to an existing node N_y is given by $p_{x \to y} = \frac{k_y}{\sum_j k_j}$ where k_y is the degree of the existing node divided by sum of all edge degrees. We multiply this probability with a similarity factor $S(PGM_{N_x}, PGM_{N_y}) \to [0, 1]$ that describes the similarity between two PGM (e.g. PGM_{N_x} and PGM_{N_y}). If a node already has reached its individual maximum number of edges (*edgelimit*) we set the probability to zero as shown in Eq. 5.

$$p_{x \to y} = \begin{cases} \frac{k_y}{\sum_j k_j} \cdot Similarity(PGM_{N_x}, PGM_{N_y}) & \text{if } k_y \leq edgelimit \\ 0 & \text{else} \end{cases} \tag{5}$$

Let there be two nodes $\{N_1, N_2\}$ where N_1 has trained the set $A = \{P_1, P_2, P_3\}$ and N_2 the set $B = \{P_1, P_3, P_4, P_5\}$ of predicting variables of their PGM as shown in Fig. 5. We define the similarity between them as the size of the intersection between A and B divided by the minimum cardinality of A and B cf. Eq. 6.

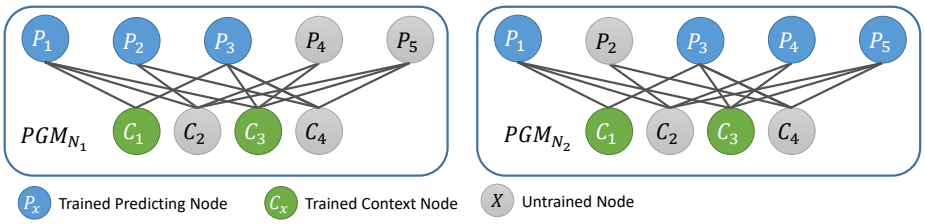

Fig. 5. Two PGMs with similar learning.

$$S(PGM_{N_1}, PGM_{N_2}) = \frac{|A \cap B|}{\min(|A|, |B|)} = \frac{2}{\min(3,4)} = \frac{2}{3} \tag{6}$$

To demonstrate that this modified algorithm still produces a topology with power law distributed node degree, we plotted the number of edges for a network of 600 nodes using the original Barabasi Albert Model and our modified version where we limited the number of edges to 60. The major difference is, that our modified version exhibits multiple nodes degree 60 instead of having several nodes with degree > 60.

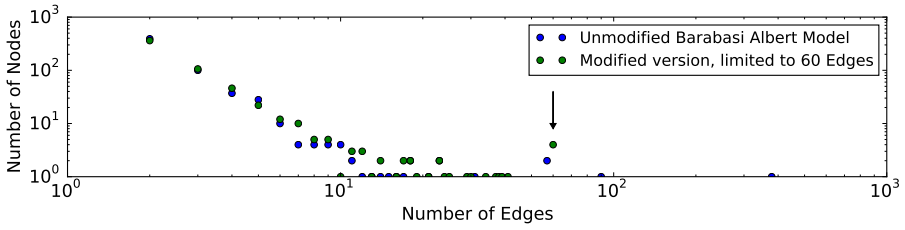

Fig. 6. Comparison of node degree between the modified and unmodified B. A. Model.

In order to deal with loops in the topology we slightly increase the forwarded entropy value with every hop. This way, propagated entropy values increase more over longer propagation paths than over shorter ones. Queries will then always be forwarded over shortest paths with lower entropy values.

5 Evaluation

In this section we evaluate the above presented aggregation based routing strategy (ABS) with respect to different network sizes, number of context nodes, context combinations, number of hops, and the context diversity parameter K (cf. Sec. 4) on synthetic training data and on data from an urban mobility scenario, in the following referred to as "Trento data". We compare our strategy to

a directed random walk approach commonly used in unstructured Peer to Peer networks cf. Sec. 6.

We implemented our routing strategy (cf. Sec. 4) in the Peer to Peer simulator PeerSim [27] and performed our evaluations on an Open Stack virtual machine with 64 cores and 256 GB RAM running Ubuntu 16.04. We used PeerSim to instantiate up to 32 768 (2^{15}) network nodes. To represent the Probabilistic Graphical Models (PGM), each node individually trained a Bayesian network consisting of an experiment dependent number of random variables. In the following we will state for each experiment how many predicting variables (P_n) and context variables (C_n) were used to form a PGM as described in Sec. 2. The Bayesian networks on the individual nodes were then trained either on synthetic training data (Gaussian distributed observations) or on data from the Trento data set respectively. The use of synthetic data gave us the ability to flexibly generate experiment setups with any number of predicting or context variables. Based on the trained nodes, the topology between the nodes was created as described in Sec. 4.

Based on this setup, we used the cycle based engine of PeerSim to perform our evaluations. In each cycle, each node first propagated its knowledge and then issues a query that is forwarded as described in Sec. 4. In order to determine the accuracy of the result, we compared the entropy of the result with the optimal entropy which was determined by an exhaustive search over all nodes.

The "Trento data" data originates from a real world simulator for collaborative and distributed learning [29] developed at the German center for artificial intelligence (DFKI) to generate large-scale, realistic data sets for machine learning. The simulator is based on the city map of Trento in Italy. It features genuine bus tables, weather, and commuting statistics of the city. Based on this data, we used the simulator to hosts 30 000 autonomous agents that emulate the behavior of citizens, even forming traffic jams that lead to different travel times at different times of the day, days of the week under different weather conditions for different road segments of the Trento street graph. In our experiments we used weather, time of the day, and day of the week as context nodes to predict travel times for different road segments that we used as predicting variables.

In our first evaluation (Fig. 7) we compare the retrieval accuracy of our aggregation based strategy (ABS) on synthetic data and the Trento data averaged over an increasing number of cycles with the random walk approach. In this evaluation we used 1024 nodes that we trained on 100 different predicting variables and 3 context variables (weather, time of the day, and day of the week in the Trento data). We can see, that in the first cycle, the accuracy is low, as most of the routing models are empty. As knowledge gets propagated through the network, the retrieval quality increases until it settles around 90%. This evaluation already indicates the good performance of our algorithm on synthetic data and on the Trento data.

In Fig. 8 we evaluate the performance of ABS at different network sizes (up to 32 768 nodes) with the random walk approach using synthetic and Trento data. Just like in the previous evaluation we used 100 predicting nodes and

3 context nodes according to the Trento data. We forwarded each query $2 \cdot log(network\,size)$ times. In comparison to the synthetic data, the standard deviation (indicated by the whiskers in the graph) is a bit higher for the Trento data. The reason for this is, that the knowledge about some predicting variables is scarce and thus harder to find.

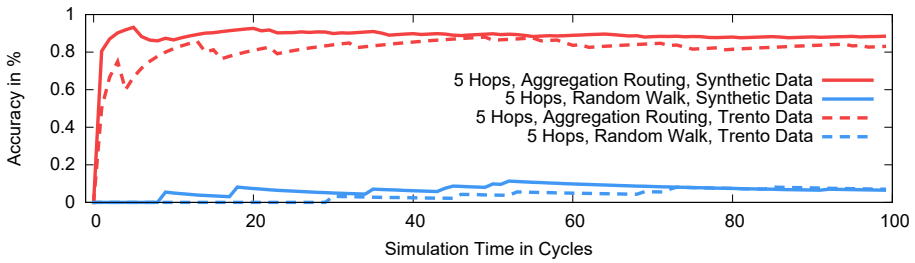

Fig. 7. Retrieval accuracy averaged over time (cycles) compared.

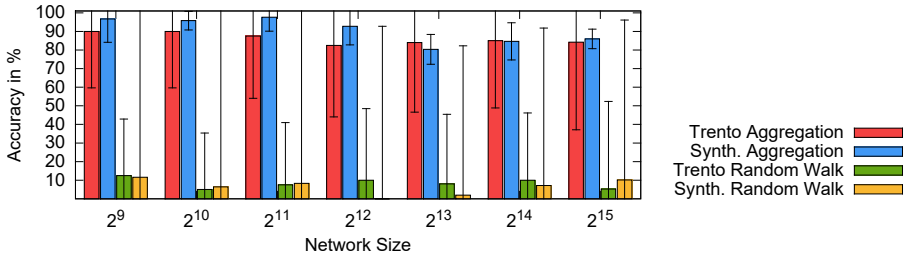

Fig. 8. Network size and retrieval quality.

Fig. 9 shows the influence of the number of hops. For this evaluation we used a network of 512 nodes, 10 predicting variables, and 3 context variables on synthetic data. We can see that with an increasing number of hops not only the accuracy increases, but also the standard deviation (whiskers) decreases. In general the number of required hops grows proportional to the network diameter. As we are using a free scale topology, this is approximately logarithmic to the number of nodes (cf. Sec. 4).

In the following, we will have a closer look at the influence of the number of possible combinations of context variables and their influence on routing accuracy. We introduced the problem of different context combinations in Sec. 4 and tackled it by introducing a parameter K that defines how many different context combinations are stored in the routing models. In Fig. 10 we can see that keeping about 50% of all possible context combinations already leads to a reasonably good retrieval accuracy.

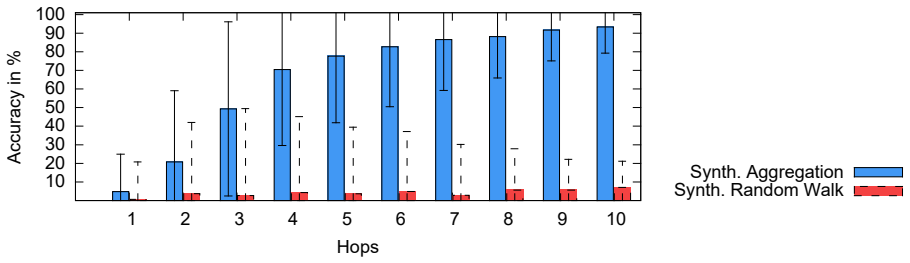

Fig. 9. Retrieval Quality w.r.t different number of hops.

Fig. 11 shows how the accuracy degrades with an increasing amount of context combinations for a network of 512 nodes, $K = 10$, one predicting variable, and 10 context variables of which up to 4 have been trained. According to the binomial coefficient this creates up to 252 possible context combinations that could have been learned. We can see that with an increasing amount of possible context, not only the accuracy decreases, but also the standard deviation of the accuracy increases. The surprisingly good performance and low standard deviation for the random strategy for 10 context combinations can be explained when realizing that there are potentially up to 51 nodes that have learned the respective contexts.

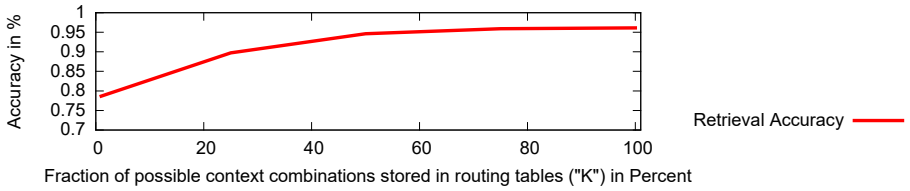

Fig. 10. Retrieval quality with respect to K.

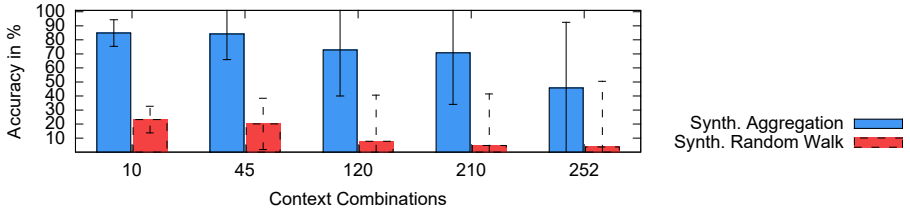

Fig. 11. Possible context combination for $K = 10$, 512 nodes, and 3 hops.

When K is chosen around 50% of the number of context combinations, retrieval accuracy is fairly independent form the number of context variables used, as shown in Fig. 12. For this experiment we used a network of 2048 nodes that

were trained on 500 predicting variables in up to 10 contexts, 25 context combinations and $k = 12$. As we highlighted in Sec. 4 we can use of the shelf methods for dimension reduction to determine most important context combinations w.r.t. the application scenario at hand.

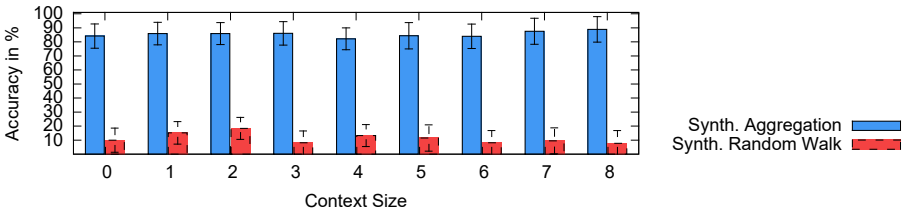

Fig. 12. Retrieval accuracy with respect to different number of context nodes

6 Related Work

Information retrieval from peer to peer (P2P) systems and machine learning are both well studied areas. Today, machine learning is often done in Big Data scenarios, where all training data is logically centralized. There exist many approaches to distribute the training data and machine learning models between several machines, for example on a cluster. These systems have the benefit of a centralized controller that actively manages how information is distributed between the different machines. In such scenarios, communication-efficient distribution of data between machines is a hard research problem of its own [22].

In this paper we argue that with the trend of decentralized computing, machine learning is coming to the edge of the network (cf. Sec. 1). On the one hand, this enables many benefits such as low latency access and the ability to maintain control over sensible information. On the other hand, without a central index structure, the problem of searching in distribute machine learning models is created. Therefore we focus on related work that tackles the problem of content sharing in P2P networks and discuss how fit these approaches are for knowledge retrieval.

First P2P systems such as Chord [35], CAN [30] and Pastry [31] tackled the problem of how to find specific data items in a distributed system. Except for CAN, most of these early work focuses on retrieval based on one unique key such as a hash value. CAN allows for multidimensional keys in euclidean space to locate data items. All approaches, however, share the draw back that they can only retrieve items that are identified by a unique index.

The second generation of P2P systems (e.g. Mercury [7], Squid [33], and Znet[34]) introduced the support for more complex, multidimensional, and range queries. This enabled searches like *Find persons age* $\geq$ *10 and age* $\leq$ *20 and gender = female*. These systems enable search in multidimensional space, where

Data locality is usually achieved by dimension reduction techniques, such as space filling curves (e.g. [10]). A general problem is that range queries might be too restricted in cases with sparse data. For example, if there are very few results for the above mentioned query, results for persons slightly older than 20 years would also be interesting for the user.

This gap was filled by research centered around nearest neighbor queries for P2P systems, like pSearch [37] and Semantic Small World [17]. The main idea is to provide nearest neighborhood search for multidimensional queries. Most work focuses on selecting important dimensions [28, 18] or methods to form an overlay network that connects nodes with similar information [38]. Just like in this paper, some of these approaches also form a small world topology [17] that has a small network diameter, which makes each node reachable with only a few hops and enables efficient routing. There also exists work that relies on a predefined similarity metric, e.g. the euclidean distance, and retrieves the k nearest data items in a large collection of high dimensional data [19, 8, 6, 18].

All these approaches have been designed to retrieve items that are explicitly defined by matching a specific identifier (id, hash value), fall in a specific range of a set of attributes, or are close to a given query. In order to retrieve knowledge this notion has to be extended by some sort of confidence metric that can take the quality of available knowledge (information) into account. Such a confidence metric needs to express the expertise of a node, reflecting for example that it holds a lot of similar information [5] or can do reliable prediction. In our previous work [5] we have tackled this issue for knowledge modeled as N-Dimensional point-clouds. We proposed a point-cluster-based confidence metric that took the variance and number of points in each cluster as an indicator of quality into account.

To the best of our knowledge, there is no peer to peer based approach that is specifically designed so search for knowledge in graphical machine learning models.

7 Conclusion and Future Work

In this paper we have stressed the importance of machine learning at the edge of the network. We argued that with an increasing amount of fog computing devices carrying specialized machine learning hardware knowledge becomes inherently distributed. In this setting we defined and tackled the problem of finding and retrieving specific knowledge. We showed that our aggregation based routing approach can retrieve specific knowledge with over 95% accuracy even if it was learned in many different contexts.

We think that the field of distributed knowledge management is in its infancy and will rapidly gain importance.With our generic notion of predicting variables and context variables our retrieval strategy is flexible and can be adapted to many future application scenarios in health care, manufacturing, and urban mobility.

Acknowledgment

The authors would like to thank the European Union's Seventh Framework Programme for partially funding this research through the ALLOW Ensembles project (project 600792).

References

1. Cisco global cloud index: Forecast and methodology, 2013–2018. Online, 2014.
2. R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.
3. L. A. N. Amaral, A. Scala, M. Barthelemy, and H. E. Stanley. Classes of small-world networks. *Proceedings of the national academy of sciences*, 2000.
4. D. Ashbrook and T. Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7(5), 2003.
5. T. Bach, M. A. Tariq, C. Mayer, and K. Rothermel. Utilizing the hive mind – how to manage knowledge in fully distributed environments. In *Proc. CoopIS*, 2015.
6. M. Batko, C. Gennaro, and P. Zezula. A scalable nearest neighbor search in p2p systems. In *Databases, information systems, and peer-to-peer computing*. 2005.
7. A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *ACM SIGCOMM Computer Comm. Review*, 2004.
8. D. Chen, J. Zhou, and J. Le. Reverse nearest neighbor search in peer-to-peer systems. In *Flexible Query Answering Systems*. Springer, 2006.
9. P. Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.
10. P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule them all: multi-dimensional queries in p2p systems. In *Proc. of the 7th Int. Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004*. ACM, 2004.
11. P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, 2015.
12. G. Heigold, V. Vanhoucke, A. Senior, P. Nguyen, M. Ranzato, M. Devin, and J. Dean. Multilingual acoustic models using distributed deep neural networks. In *IEEE ICASSP*, 2013.
13. K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pages 15–20. ACM, 2013.
14. informationisbeautiful.net. World's biggest data breaches. Online, 2015.
15. R. Khan, S. U. Khan, R. Zaheer, and S. Khan. Future internet: the internet of things architecture, possible applications and key challenges. In *Proc. FIT*, 2012.
16. M. G. Kienzle. Cognitive technologies for smarter cities. In *Proc. ICDCS*, 2016.
17. M. Li, W.-C. Lee, and A. Sivasubramaniam. Semantic small world: An overlay network for peer-to-peer search. In *Proc. ICNP*, 2004.
18. M. Li, W.-C. Lee, A. Sivasubramaniam, and J. Zhao. Supporting k nearest neighbors query on high-dimensional data in p2p systems. *FCS*, 2008.
19. Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 2014.

20. C. D. Manning, H. Schütze, et al. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.
21. J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. 2011.
22. C. Mayer, M. A. Tariq, C. Li, and K. Rothermel. GrapH: Heterogeneity-Aware Graph Computation with Adaptive Partitioning. In *Proc. ICDCS*, 2016.
23. R. Mayer, H. Gupta, E. Saurez, and U. Ramachandran. The fog makes sense: Enabling social sensing services with limited internet connectivity. In *Proc. 2nd International Workshop on Social Sensing*. ACM, 2017.
24. R. Mayer, B. Koldehofe, and K. Rothermel. Predictable low-latency event detection with parallel complex event processing. *IEEE Internet of Things Journal*, 2015.
25. B. McMahan and D. Ramage. Federated learning: Collaborative machine learning without centralized training data. Technical report, Google, 2017.
26. A. Montresor. Reflecting on the past, preparing for the future: From peer-to-peer to edge-centric computing. In *Proc. ICDCS*, 2016.
27. A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth Int. Conf. on*, pages 99–100. IEEE, 2009.
28. W. Müller and A. Henrich. Fast retrieval of high-dimensional feature vectors in p2p networks using compact peer data summaries. In *Proc. ACM SIGMM int. Ws. on Multimedia Information Retrieval*, pages 79–86. ACM, 2003.
29. A. Poxrucker, G. Bahle, and P. Lukowicz. Towards a real-world simulator for collaborative distributed learning in the scenario of urban mobility. In *Proc. SASOW, 2014*, 2014.
30. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *ACM SIGCOMM Computer Communication Review*, 2001.
31. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*. Springer, 2001.
32. J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope. your word is my command: Google search by voice: A case study. In *Advances in Speech Recognition*, pages 61–90. Springer, 2010.
33. C. Schmidt and M. Parashar. Squid: Enabling search in dht-based systems. *Journal of Parallel and Distributed Computing*, 68(7):962–975, 2008.
34. Y. Shu, B. C. Ooi, K.-L. Tan, and A. Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Proc. P2P*. IEEE, 2005.
35. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.
36. C. Sutton and A. McCallum. *An introduction to conditional random fields for relational learning*, volume 2. Introduction to statistical relational learning. MIT Press, 2006.
37. C. Tang, Z. Xu, and M. Mahalingam. psearch: Information retrieval in structured overlays. *ACM SIGCOMM Computer Communication Review*, 33(1):89–94, 2003.
38. H. F. Witschel. Content-oriented topology restructuring for search in p2p networks. Technical report, Technical report, University of Leipzig, Germany, 2005.
39. J. H. Ziegeldorf, O. G. Morchon, and K. Wehrle. Privacy in the internet of things: threats and challenges. *Security and Communication Networks*, 2014.