Dynamic Service Knowledge Base Construction at WeChat

Haoyang LI[1], Ziyuan ZHAO[2], Xueling LIN[1], Qiang YAN[2], Tiezheng MAO[2], Zijian LI[1], Hao XIN[1], Lei CHEN[1]

[1]The Hong Kong University of Science and Technology
[2]Search Algorithm Group, WeChat, Tencent
{hlicg,xlinai,zlicb,hxinaa,leichen}@cse.ust.hk
{joshuazhao,rolanyan,woodtmao}@tencent.com

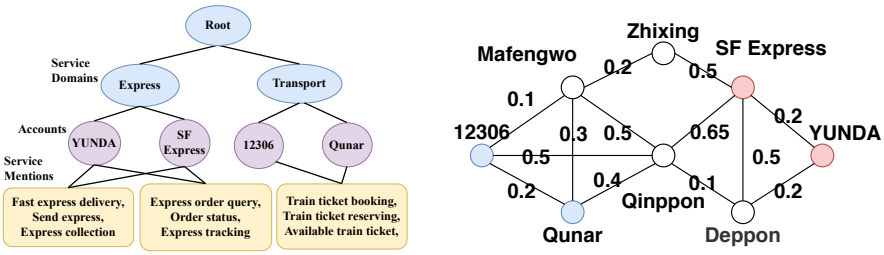
Abstract. Service-oriented knowledge base (KB) has been applied to a variety of applications. In online platforms, the official accounts registered by companies display service mentions (e.g., *train ticket booking*) to provide services for users. To facilitate the downstream tasks, we propose to construct a dynamic service KB. Specifically, the service KB contains accounts in different service domains. Moreover, since these service mentions for the same service have different names, the service KB should store the canonicalized service mentions for each account. Despite the fruitiness, existing KB construction and NLP approaches rely on abundant annotated labels. However, in real-world applications, it is impractical and expensive to annotate abundant account and service mention labels, due to a large number of accounts and services. In this paper, we propose an end-to-end dynamic service KB construction system. First, we infer the service domain of each account under limited labeled accounts. Second, without service mention labels, we propose to adaptively separate service mentions into disjoint partitions, where service mentions inside the same partition provide the same service. Third, our system can be updated with dynamic scenarios naturally, including service changes, new accounts, and new service domains. The experiments on real-world datasets and a large-scale online A/B testing demonstrate the effectiveness and high practicality of our system.

Keywords: Knowledge base · Account acquisition · Canonicalization

1 Introduction

Knowledge Base (KB), such as Freebase [3] and DBpedia [14], has been successfully applied on many real-world applications such as question answering [17] and online recommendation [7]. Recently, professional KBs [1, 6, 18, 24] are constructed to improve tasks in professional domains. For example, PubMed [6] facilitates medical literature searching by providing medical facts.

In real-world online platforms, such as WeChat, Taobao, and Amazon, many official accounts are registered by companies that provide services for users. For example, the account *12306* provides a *train ticket booking* service. Specifically, when users query some services such as *train ticket booking*, the platforms



(a) Service KB example. (b) Account graph example.

Fig. 1: Service KB example and account graph example

should return accounts providing this service, such as *12306*. However, without knowledge about services that an account can provide, the platforms can hardly capture the relatedness between the account and the user's requirement. To facilitate the service query, we propose to construct a professional KB based on the registered accounts, which we refer to as a service knowledge base (service KB). Fig. 1(a) shows an example of the service KB, which contains three levels. The first level is service domains, such as *express* and *transport*. The second level consists of accounts in each service domain. The third level contains the service mention partitions, where service mentions inside the same partition provide the same service. The edge between an account in the second level and a service partition in the third level indicates that the account provides this service. However, there are three challenges to build such a service KB.

First, we need to infer the domain of accounts under short account descriptions and limited labels. Specifically, the account descriptions are short, i.e., most of them are less than 20 words. Besides, accounts of a new domain (e.g., COVID testing) always arrive in a continuous manner. To meet the real-time requirement, it is challenging to label abundant accounts for new domains. Consequently, existing classification approaches [4, 11, 13] that rely on a large number of labels or long texts cannot adapt to infer the account domain effectively.

Second, the service mentions displayed by accounts are short and ambiguous, such as *Flight ticket booking* and *Snap up ticket*, which is not friendly for service queries. Hence, we need to canonicalize service mentions, i.e., we group service mentions that provide the same service into the same cluster. Existing effective approaches [19, 20, 22] rely on a large number of labeled service mention synonyms for each service. However, it is costly to label synonyms for services, and it is unrealistic to know the exact number of services in real-world applications.

Third, to keep the freshness of service KB is another challenge. There are three general dynamic scenarios in real-word: (1) service changes: accounts may add new services or delete old services; (2) new accounts: many new accounts are emerging, and we need to distinguish new accounts within our target domains and put them into our service KB. (3) new service domains: the service domains always come, such as COVID domain. However, the current approaches [3, 4, 11, 13, 19, 20, 22] cannot handle these scenarios because they are designed for static scenarios, which hurts the effectiveness of the service KB construction system.

In this paper, to solve the above challenges, we propose an end-to-end service KB construction system, which takes a set of unlabelled accounts and limited

seed accounts as inputs, and outputs a canonicalized service KB. Specifically, with limited seed accounts, we first construct an account graph based on their meta information, such as their descriptions and names. We then formulate the account acquisition problem to infer the domains for unlabeled accounts based on the distance between them and seed accounts. We then prove this problem is NP-hard and propose an efficient greedy algorithm. Second, To canonicalize service mentions without labels, we first build the service mention graph. We then formulate the service mention partition problem to adaptively separate the service mention graph into a set of disjoint partitions by maximizing the global and local qualities of partitions. We prove this partition problem is NP-hard as well and cannot be approximated within a constant factor unless P=NP. Then, we propose a greedy algorithm. Last but not least, our system can build a service KB from scratch and update the service KB naturally, which can handle the dynamic scenarios. We summarize the contributions of this paper as follows.

- We present an end-to-end service KB construction system. Moreover, it can handle dynamic scenarios naturally in an effective and efficient manner, including service changes, new accounts, and new service domains.
- We formulate the account acquisition problem to infer the domains for unlabeled accounts based on the distance between them and limited seed accounts. We prove this problem is NP-hard and propose an efficient algorithm.
- Without service mention labels and the exact service number, we formulate the service mention partition problem to adaptively canonicalize service mentions. We prove this problem is also NP-hard and cannot be approximated in a constant factor unless P=NP. Then, we propose a greedy algorithm.
- Experimental results demonstrate the effectiveness and high practicality of our system. The constructed service KB at WeChat has been used to facilitate the daily lives of billions of users.

2 Framework Overview

At WeChat, each account has meta information, such as account description, account name, and services mentions. We denote each account as $a_i = \{n_i, d_i, S_i\}$, where $n_i = \{w_1, \cdots, w_{|n_i|}\}$ is the account name of a_i, $d_i = \{w_1, \cdots, w_{|d_i|}\}$ is the account description of a_i, and $S_i = \{s_1, \cdots, s_{|S_i|}\}$ is a set of service mentions displayed by a_i. Moreover, a service mention $s_i = \{w_1, \cdots, w_{|s_i|}\}$ consists of a set of words, which denotes a real-world service, such as *train ticket reserving*. Additionally, a query $q = \{w_i\}_{i=1}^{|q|}$ consists of words. We use $Q_i = \{q_j\}_{j=1}^{|Q_i|}$ to denote the query set of account a_i where users input queries in Q_i and click a_i.

Given a set of unlabeled accounts $A = \{a_i\}_{i=1}^{n}$, the query set $Q = \{Q_i\}_{i=1}^{n}$, a set of target service domains $T = \{t_i\}_{i=1}^{m}$, and a set seeds of account $SA = \{SA_{t_j}\}_{j=1}^{m}$ where $SA_{t_j} = \{a_k\}_{k=1}^{|SA_{t_j}|}$ consists of the accounts of the service domain t_j, the target is to construct a service KB in the target domains T based on the set of accounts A in an effective and efficient manner. Specifically, as shown in Fig. 1(a), the service KB contains accounts in different domains and store the canonicalized service mentions provided by each account. The service KB construction system works in three stages as illustrated in Fig. 2.

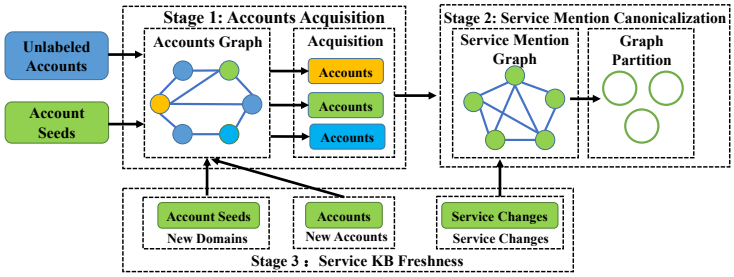


Fig. 2: the framework of the Service KB construction system

Stage 1: Accounts Acquisition. Given unlabeled accounts A, the query Q, the target domains T, and limited seed accounts SA, the task is to obtain accounts in the target domains $A_T = \{A_t\}_{t=1}^{|T|}$, where $A_t = \{a_i\}_{i=1}^{|A_t|}$ are in domain t.

Stage 2: Service Mention Canonicalization. Given service domain t, the service mentions $M_t = \bigcup_{j=1}^{|A_t|} S_j$ is obtained by gathering all service mentions from accounts A_t. We canonicalize mentions in M_t into disjoint partitions $DP_t = \{P_j\}_{j=1}^{|P_t|}$, while mentions inside the same partition provide the same service.

Stage 3: Service KB Freshness. We update the service KB in three general scenarios: (1) service changes, (2) new accounts, (3) new service domains.

3 Account Acquisition

In this section, we first construct an account graph among unlabeled and labeled accounts from different views. We then infer the domain of unlabeled accounts.

3.1 Account Graph Construction

We first construct an un-directed weighted account graph. Specifically, we consider whether two accounts are similar from two complementary views: (1) account view: the account information indicates its service domain, such as its description, name, and service mentions. Intuitively, if the information of two accounts are similar, the domains of these two accounts tend to be the same; (2) user view: users always input queries and retrieve the related accounts, and hence two accounts that are more similar will share more common queries.

Account Description. Given the description d_i, d_j of the account a_i and a_j, the similarity between both descriptions $DS(a_i, a_j)$ is defined as : $DS(a_i, a_j) = \frac{\sum_{w_k \in d_i \cap d_j} tf(w_k)}{\sum_{w_k \in d_i \cup d_j} tf(w_k)}$, where $tf(w_k)$ is the tf-idf scores of word w_k.

Account Name. Formally, given the names of two accounts n_i and n_j, the name similarity $NS(a_i, a_j)$ is computed as: $NS(a_i, a_j) = |n_{a_i} \cap n_j|/|n_i \cup n_j|$.

Account Service Mentions. Given service mentions S_i and S_j provided by two accounts, the mention similarity $MS(a_i, a_j)$ is $MS(a_i, a_j) = |S_i \cap S_i|/|S_j \cup S_j|$.

Account Query. Formally, given two query sets of accounts Q_i and Q_j, the query similarity is computed as: $QS(a_i, a_j) = |Q_i \cap Q_j|/|Q_i \cup Q_j|$.

Edge Weight. We argue that if any pair of these aspects are similar in two accounts, the domains of these two accounts tend to be the same. Hence, we define the similarity $SIM(a_i, a_j) = 1 - (1 - DS(a_i, a_j))(1 - NS(a_i, a_j))(1 -$

$MS(a_i, a_j))(1-QS(a_i, a_j))$, where $SIM_a(a_i, a_j) \in [0, 1]$ and larger $SIM_a(a_i, a_j)$ indicates higher similarity between accounts a_i and a_j. We then define the distance between two accounts as follows: $d(a_i, a_j) = -ln\ SIM(a_i, a_j)$, where $d(a_i, a_j) \in [0, +\infty]$ denotes the distance between accounts a_i and a_j, and smaller $d(a_i, a_j)$ indicates two accounts are similar.

3.2 Account Acquisition

The target is to obtain accounts in the target service domains $A_T = \{A_t\}_{t=1}^{|T|}$ from unlabeled accounts A. Intuitively, the accounts in the same A_t can be regarded as a cluster. It implies the distance among accounts in the same domain should be smaller. Also, the similarity among $a_i \in A_t$ should have a smaller distance with seed accounts SA_t in the same domain than the other domains. Formally, we denote the average distance of each account set A_t as $ADIS(t) = \sum_{a_i, a_j \in A_t} d(a_i, a_j)/|A_t|$. Then, we define a transitive distance to measure the similarity among each unlabeled account and seed account. Given an unlabeled account a_i and a seed account a_j, the transitive distance between a_i and a_j is the shortest path value $TDIS(a_i, a_j) = min_{p_i \in Path(a_i, a_j)} \sum_{e(a_m, a_n) \in p_i} d(a_m, a_n)$ where $Path(a_i, a_j)$ is all paths between a_i and a_j, $e(a_m, a_n)$ is the edge. For example, in Fig. 1(b), $TDIS(12306, Zhixing) = 0.1 + 0.2 = 0.3$. Furthermore, we define the minimum transitive distance between account partition A_t and the domain t as $TRAN(t) = \sum_{a_i \in A_t} \min_{a_j \in SA_t} TDIS(a_i, a_j)$.

Definition 1 (Account Acquisition Problem). *Given a set of unlabeled accounts A, seed accounts SA, the task is to obtain accounts in the target service domains $A_T = \{A_t\}_{t=1}^{|T|}$, where $A_t = \{a_i\}_{i=1}^{|A_t|}$ is a set of accounts with service domain t, which minimizes $\sum_{t \in T} (1-\delta)TRAN(t) + \delta ADIS(t)$.*

Theorem 1. *The account acquisition problem is NP-hard.*

Proof. We prove that the account acquisition problem is NP-hard from the reduction from the k-way Maximum Sum of Densities (K-MSD) problem [10]. The K-MSD is defined as follows: given a graph $G = \{V, E\}$, the target is to separate G into K partitions $P_i = \{V_i, E_i\}$, such that the sum density of these K partitions is maximized, where the density of each partition P_i is defined as $\sum_{e \in E_i} w(e)/|V_i|$, where $w(e)$ is the weight of the edge $e \in E_i$.

Given an instance $\mathcal{I}$ of K-MSD as follows: with a graph $G = \{V, E\}$ with all positive edge weight, we find the k partitions by maximizing the sum of the density of partitions. we transform it into an instance $\mathcal{J}$ of account acquisition problem as follows: we add k anchor node as the seed of k domains into the graph G by adding edge with the same weight between each seed and each node $u \in V$, and transform the edge weight to the negative. We argue that we can solve the K-MSD problem from the graph G if we can solve the account acquisition problem. We assume that $\{P_i\}_{i=1}^{k}$ is the solution of the account acquisition problem. Since no matter how we divide G into K partitions, $TRAN(P_i)$ is the same. Hence, the sum of density of partitions is minimized. $\{P_i\}_{i=1}^{k}$ is also the solution of K-MSD. Therefore, we can solve K-MSD problem if we can solve the account acquisition problem. Hence, the account acquisition problem is NP-hard. □

Algorithm 1: Account Acquisition

Input: Acounts $A = \{a_i\}_{i=1}^{n}$, target domains $T = \{t_i\}_{i=1}^{m}$ and seeds SA_a
Output: $A_T = \{A_t\}_{t=1}^{|T|}$ where $A_t = \{a_j\}_{j=1}^{|A_t|}$ includes accounts with domain t

1 $G_a \leftarrow$ construct the account graph, $A_t \leftarrow \emptyset \quad \forall t \in T$
2 **foreach** *unlabeled account* $a_i \in A$ **do**
3 $\quad p = \min_{a_j \in SA} TDIS(a_i, a_j)$
4 $\quad$ **if** $p \leq \theta_1$ **then** $A \leftarrow A \setminus \{a_i\}$ `// Remove accounts not in target domains`
5 **foreach** *unlabeled account* $a_i \in A$ **do**
6 $\quad$ **foreach** $t \in T$ **do** $\triangle G(a_i|t) = (1-\delta)\triangle TRAN(a_i|t) + \delta\triangle ADIS(a_i|t)$
7 $\quad t^* = \arg\min_{t \in T} \triangle G(a_i|t)$, $A_{t^*} \leftarrow A_{t^*} \cup \{a_i\}$
8 **Return** $A_T \leftarrow \bigcup_{t \in T}\{A_t\}$

Marginal Gain. Given an account a_i, and a service domain t, we define the marginal gain if we assign the domain t to the account a_i as $\triangle G(a_i|t) = (1-\delta)\triangle TRAN(a_i|t)+\delta\triangle ADIS(a_i|t)$, where $\triangle TRAN(a_i|t) = \min_{a_j \in SA_t} TDIS(a_i, a_j)$ and $\triangle ADIS(a_i|t) = \sum_{a_j \in A_t} d(a_i, a_j)/|A_t|$.

Algorithm. Since the account acquisition problem is NP-hard, we propose an efficient greedy algorithm. The basic idea is to assign the accounts to the service domains by minimizing the above marginal gain. Specifically, we first construct an account graph (line 1). We then compute the transitive distance between unlabeled accounts and seed accounts. And, we only keep these unlabeled accounts whose minimum transitive distance with seed accounts is smaller than the threshold θ_1 (lines 2-4), to guarantee the precision. We then assign each filtered unlabelled account into the domain with the minimum marginal gain $\triangle G(a_i|t_k)$ (lines 5-7).

Time Complexity. Suppose there are n accounts, n_s seed accounts. The time complexity of account graph construction is $O(|E|)$ where $|E| \ll n^2$, since we only compute the weight of accounts that share common words. We then compute the transitive distance between unlabeled accounts and seed accounts is $O(n_s|E|)$. The time complexity of assigning each account to the domain is $O(n_s n + n_s|E|)$ (lines 6-10). Hence the total complexity is $O(n_s n + n_s|E|)$.

4 Service Canonicalization

In this section, for each service domain t, we can gather all service mentions $M_t = \bigcup_{j=1}^{|A_t|} S_j$ from accounts A_t. We then construct a service mention graph $G_{s,t} = \{V_{s,t}, E_{s,t}\}$ for each service domain t and adaptively separate the graph into different partitions. Since we canonicalize the service mention for each domain t independently, for simplicity, we do not use the domain subscript t in notations.

4.1 Service Graph Construction

Given a domain t, we consider the similarity among service mentions in M_t from three aspects: (1) words in mentions, (2) word embeddings, (3) co-occur times. The first two aspects provide positive evidence to show whether two mentions

are similar. The last one provides the negative evidence, i.e., the frequently co-occur mentions tend to provide different services, because accounts are likely to provide one unique service by displaying a service mention for simplicity.

Words. given two mentions s_i, s_j, the word similarity is $WS(s_i, s_j) = \frac{|s_i \cup s_j|}{\min(|s_i|,|s_j|)}$.

Word Embedding Value. Embedding captures the semantic meaning of words. Given mention $s_i = \{w_i\}_{i=0}^{|s_i|}$, the embedding value of s_i is the average value $\boldsymbol{s_i} = \frac{1}{|s_i|}\sum_{i=0}^{|s_i|} \boldsymbol{w_i}$. Then, the embedding similarity $ES(s_i, s_j) = cosine(\boldsymbol{s_i}, \boldsymbol{s_j})$.

Co-occur Times. We define the conflict score $CS(s_i, s_j)$ between two co-occurred service mentions s_i and s_j as follows: $CS(s_i, s_j) = 1 - e^{-co(s_i,s_j)}$, where $co(s_i, s_j)$ is the co-occur times of s_i and s_j in the same accounts.

Aggregated Similarity. Two service mentions are similar if they have a higher word score or higher embedding score. On the other hand, a higher conflict score indicates that these two service mentions provide different services. Formally, given two service mentions s_i and s_j, we define the aggregated similarity $AS(s_i, s_j)$ as $AS(s_i, s_j) = \max(WS(s_i, s_j), ES(s_i, s_j)) - CS(s_i, s_j)$, where $AS(s_i, s_j) \in [-1, 1]$. We regard the aggregated similarity of two service mentions as their edge and obtain the service mention graph $G_s = \{V_s, E_s\}$.

4.2 Service Partition Identification

After obtaining the service graph $G_s = \{V_s, E_s\}$, the target is to separate this graph into disjoint partitions where each partition contains the service mentions of a unique service. We consider the quality of a service mention partition from two metrics. (1) global quality: if the sum weights of edges that in the partitions are higher, the global quality of these partitions are better. This is because wrong service mention assignment will decrease the global quality. Formally, given a set of partitions $MP = \{P_i\}_{i=1}^{n_p}$ and $P_i = \{s_j\}_{j=1}^{|P_i|}$, we define the the global quality of partitions as $\sum_{P_i \in MP} GQ(P_i)$, where $GQ(P_i) = \frac{\sum_{s_i, s_j \in P_i \wedge s_i \neq s_j} AS(s_i, s_j)}{2}$, and $AS(s_i, s_j)$ is the aggregate similarity between s_i and s_j. (2) local quality: since service mentions inside the same partition provide the same service, each service mention should have higher similarity with other service mentions. Formally, we define the local quality as $LQ(P_i) = \min_{s_i \in P_i} \frac{\sum_{s_j \in P_i \setminus s_i} AS(s_i, s_j)}{|P_i| - 1}$. To guarantee the local quality of partition, we let $LQ(P_i) \geq \theta_2$ where θ_2 is a threshold. Based on two metrics, we formulate the *service mention partition problem* as follows:

Definition 2 (Service Mention Partition Problem). *Given the service mention graph $G_s = \{V_s, E_s\}$, the target is to separate the graph into the disjoint partitions $MP = \{P_i\}_{i=1}^{n_p}$ by maximizing the total global quality $\sum_{P_i \in MP} GQ(P_i)$, which satisfies three constraints: (1) the local quality of each partition is not less than a threshold θ_2, i.e, $LQ(P_i) \geq \theta_2$; (2) $P_i \cap P_j = \emptyset, \forall P_i, P_j \in P$; (3) $\bigcup_{i=1}^{n} P_i = V_s$. Note that the number of partitions n_p is not predefined.*

Theorem 2. *The problem is NP-hard and cannot be approximated within a factor of $|V_s|^{1-\epsilon}$ for a fixed ϵ unless P=NP.*

Algorithm 2: Service Mention Partition

Input: The service mention graph $G_s = \{V_s, E_s\}$
Output: The service mention partitions $MP = \{P_i\}_{i=1}^{n}$ where $P_i = \{s_j\}_{j=1}^{|P_i|}$

1 $MP \leftarrow \emptyset$
2 **foreach** *service mention* $s_j \in V_s$ **do**
3 **if** $MP == \emptyset$ **then** $P_0 = \{s_j\}$, $MP \leftarrow MP \cup P_0$;
4 **else**
5 Gain_List = []
6 **foreach** *Partition* $P_i \in MP$ **do**
7 $\triangle GQ(s_j|P_i) = GQ(P_i \cup \{s_j\}) - GQ(P_i)$
8 **if** $LQ(P_i \cup \{s_j\}) \geq \theta_2$ **then** $Gain\_List.append(\triangle GQ(s_j|P_i))$;
9 **if** $len(Gain\_List)! = 0$ **then**
10 $P^* \leftarrow \arg\max_{P_i \in Gain\_List} \triangle GQ(s_j|P_i)$, $P^* \leftarrow P^* \cup \{s_j\}$
11 **else** $P_{|MP|} = \{s_j\}$, $MP \leftarrow MP \cup P_{|MP|}$;
12 **Return** MP

Proof. We assume that each edge in the service mention graph $G_s = \{V_s, E_s\}$ is weighted as 1 or 0 and the threshold θ_2 is 1. We need to separate G_s into a set of partitions $MP = \{P_i\}_{i=1}^{n_p}$ while maximizing $\sum_{P_i \in MP} GQ(P_i)$ and satisfying $LQ(P_i) \geq 1$. As a result, each partition in P_i is a clique.

We argue that the exact solution is to find a set of maximum cliques in a greedy manner. Specifically, we firstly find the maximum clique P_1 in G_s. We then get a new graph G_s^1 by deleting nodes and edges from G_s, where these nodes are in P_1 and edges are adjoint with nodes in P_1. Following this way, we can find the maximum clique P_i from graph G_s^{i-1} until the graph G_s^{i+1} is empty. Finally, we get a set of partitions $MP = \{P_i\}_{i=1}^{n_p}$. We prove MP is the optimal result as follows. We assume MP is not the optimal result. It means we can remove some nodes $V^{'}$ from a partition P_i to another partition P_j, where $|P_i| \geq |P_j|$. However, the total global quality does not become larger because $\sum_{s_i \in V'} \sum_{s_j \in P_i \setminus V'} AS(s_i, s_j) \geq \sum_{s_i \in V'} \sum_{s_j \in P_j \bigcup V'} AS(s_i, s_j)$ where $|P_i| \geq |P_j|$. Hence the result of the exact solution is the optimal result.We find that the key of this exact solution is to find the maximum clique from the graph, which is NP-hard and cannot be approximated within a factor of $|V_s|^{1-\epsilon}$ for a fixed constant ϵ [8]. □

Algorithm. Due to this partition problem is NP-hard, we propose a greedy algorithm. The basic idea is to greedily assign the service mention to the partition by maximizing the global score and satisfying the local constrain. Firstly, we select a mention s_j to form the initial partition. Then for each unassigned service mention s_j in V_s, we compute the global quality gain $\triangle GQ(s_j|P_i) = GQ(P_i \cup \{s_j\}) - GQ(P_i)$ between s_j and each partition $P_i \in MP$, and the local quality $LQ(P_i \cup \{s_j\})$ (lines 6-8). We then choose the partition P^*, which gets the most the global quality gain as the corresponding partition for this service mention

(lines 9-10). Otherwise, we assign s_j as a new partition $P_{|MP|}$ (line 11). We repeat this process until we assign all service mentions to a partition.

Time Complexity. The service mention graph construction takes $O(|V_s|^2)$ where $|V_s|$ is the number of service mentions in each service domain. Algorithm 2 assign service mention to partitions where it computes the global quality gain by comparing the service mention with the service mentions in partitions, leading to $O(|V_s|^2)$ time. Hence, the total time complexity is $O(|V_s|^2)$.

5 Service KB Freshness

In this section, we show that our system can handle these scenarios naturally.

New Account. Given new accounts, we first add new accounts into the account graph G_a, and add edges between these new accounts and accounts existing in G_a. Then we can infer its domain following Alg. 1. Secondly, we can assign the service mentions similarly to the below service change part.

Service Change. When the account a_i deletes one service mention s_j, we delete the edge between a_i and s_j in Service KB. When the account adds one service mention, we assign it to the suitable service partition or assign it to a new service partition following lines 2-11 in Alg. 2.

New Service Domain. Our system can involve a new service domain naturally. For example, our service KB only has *Housework and Express* domains, and we want to involve the new *Medical* domain . Specifically, we first label seed accounts in *Medical* domain in G_a. We then find the accounts with medical type following Alg. 1 and canonicalize service mentions following Alg. 2.

6 Experiments

We first introduce the datasets in Sec 6.1. Then, we evaluate the account acquisition in Sec. 6.2 and service canonicalization in Sec. 6.3. Also, we explore the parameter sensitivity evaluations (i.e., δ in Sec. 3.2, θ_1 in Alg 1, and θ_2 in Alg 2) in Sec. 6.4 and put dynamic updating evaluations on three scenarios (i.e., new account, service change,and new service domain) in Sec. 6.5. Finally, we demonstrate the high utility of our service KB that facilitates billions of users by an A/B online test at WeChat in Sec. 6.6.

6.1 Experiment Environment Setting

In this paper, the service KB construction system is implemented in Python 3.6 on a CPU (2.6 GHz) in the Ubuntu 18.04.1 LTS platform. The baseline TextCNN is trained on a GPU server with one NVIDIA Tesla P100 GPU(16G) and six CPU core (2.6 GHz) in the Ubuntu 18.04.1 LTS platform. The parameters of TextCNN used in our experiment is the suggested default setting in [11].

Dataset We use 11000 high-quality official accounts at Wechat to evaluate our service KB construction system. Moreover, we pre-define five service domains, i.e., *Medical*, *Express*, *Transport*, *Estate*, and *Housework* as the target service domains. In particular, to evaluate the influence of the number of active users

Table 1: Results of accounts acquisition

	WeChat-Low			*WeChat-High*			*WeChat-All*		
Method	***P***	***R***	***F1***	***P***	***R***	***F1***	***P***	***R***	***F1***
TextCNN	0.354	0.791	0.490	0.592	0.711	0.646	0.535	0.747	0.623
Nearest	0.453	0.870	0.595	0.716	0.825	0.767	0.640	0.867	0.738
KNN	0.475	0.885	0.615	0.735	0.856	0.791	0.665	0.887	0.760
Bert	0.701	0.902	0.789	0.825	0.876	0.850	0.791	0.890	0.834
Our	**0.837**	**0.913**	**0.873**	**0.918**	**0.900**	**0.910**	**0.910**	**0.894**	**0.885**

of accounts, we split the 11000 official accounts into two datasets regarding the number of active users in each account (1) WeChat-Low: there are 3371 accounts with less than 2000 active users. We randomly label 1000 accounts as test dataset. (2) WeChat-High: there are 7629 accounts with at least 2000 active users. We randomly label 2000 accounts as the test dataset. (3) WeChat-All: the number of accounts is 11000. The test dataset is 3000. Specifically, for each dataset, we label 30 seed accounts for each service domain that does not exist in test dataset.

Data Processing We segment the words in the descriptions, names, and service mentions of the accounts by the jieba segmentation tool[1]. We collect the menu mentions of each account as the service mentions for it. In particular, we delete menus that do not contain any service, such as *My pocket, My center* and *Daily attendance*. Also, the queries of accounts are selected from one week. Moreover, we use the pre-trained Tencent embedding matrix [21] to obtain the embedding value of service mention in Section 4, which is used in many tasks and get good performance.

6.2 Evaluation of Account Acquisition

We evaluate the performance of account acquisition with SOTA and representative baselines that are frequently used in industry. we select four service domains, i.e., *Medical*, *Express*, *Transport*, and *Estate*. Particularly, *Housework* are used to evaluate the dynamic updating in Section **??**.

Baselines. (1) **TextCNN** [11] and **BERT** [4]: for each account, we concatenate its name, its description, its mentions and its queries as the input to predict its domain. (2) **Nearest**: It finds the most similar seed account by pair comparisons for each unlabeled account. (3) **KNN**: It gets the top-K similar seed accounts for each unlabeled account by pair comparisons We then assign the major service domain of the K neighbors for this unlabeled account. We set K=10. (4) **Our** in Alg. 1: we set the maximum distance θ_1 as 1. In particular, we use the precision (P), recall (R) and F1 score to evaluate the performance.

Evaluation Analysis. Results in Tab. 1 demonstrate that our approach outperforms other baselines in all metrics. It is because we consider the correlations among all accounts from a global view. TextCNN and BERT cannot capture

[1] https://github.com/fxsjy/jieba

Table 2: Results of service mention partition

	Housework			*Express*			*Transport*			*Estate*			*Medical*		
Method	*Ma-P*	*Mi-P*	*Coh*	*Ma-P*	*Mi-P*	*Coh*	*Ma-P*	*Mi-P*	*Coh*	*Ma-P*	*Mi-P*	*Coh*	*Ma-P*	*Mi-P*	*Coh*
K-means	0.15	0.34	0.07	0.14	0.31	0.08	0.12	0.27	0.04	0.13	0.22	0.06	0.16	0.32	0.14
HAC	0.79	0.87	0.52	0.72	0.85	0.43	0.66	0.80	0.38	0.65	0.77	0.32	0.82	0.84	0.56
Our-Pos.	0.82	0.88	0.60	0.79	0.87	0.52	0.74	0.84	0.47	0.72	0.83	0.51	0.84	0.88	0.62
Our-all	**0.87**	**0.92**	**0.70**	**0.85**	**0.91**	**0.66**	**0.81**	**0.89**	**0.62**	**0.79**	**0.87**	**0.56**	**0.85**	**0.88**	**0.66**

the correlation between the account semantic meaning and domains with a few labeled accounts. Moreover, KNN and Nearest get unsatisfactory performance because they only consider pair comparisons between unlabeled accounts and labeled seed accounts in a local view. In particular, the performance of all approaches on WeChat-High is better than the other datasets, which shows that accounts with more active users are easy to be distinguished.

6.3 Evaluation of Service Canonicalization

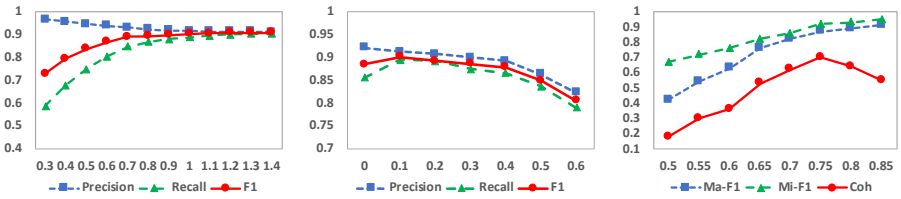
The evaluation of service mention partitions is a challenging task since there is no ground truth. We consider the following aspects for the evaluation.

Service Partition Precision. Given a set of partitions $MP = \{P_i\}_{i=1}^{|P|}$, the macro precision is defined as: **Macro-P** $= \sum_{i=1}^{|MP|} \mathbb{I}(P_i)/|MP|$, where $\mathbb{I}(P_i) = 1$ if all service mentions in partition P_i provide the same service. Moreover, we define Micro precision as **Micro-P** $= \sum_{P_i \in MP} N(P_i) / \sum_{P_i \in MP} |P_i|$, where $N(P_i)$ is the number of service mentions which provide the service.

Service Partition Coherence. Service partition coherence (**Coh**) aims to qualify the recall of these partitions. Following [23], we perform an intrusion study. Specifically, given a partition P_i, we inject this partition a service mention s_j where $s_j = \arg\max_{s_j \in P \setminus P_i} \sum_{s_k \in P_i} AS(s_j, s_k)$, i.e., s_j selected from other partitions is the most similar with service mentions in P_i. We then show this partition with the injected mention to evaluators and ask which one is the injected one. We compute the correct ratio as the service partition coherence score.

Baselines. We evaluate our approach with the following baselines: (1) **K-means**: we set the distance between two mentions s_i and s_j as $-ln(max(0, AS(s_i, s_j)))$, and K=200. (2) **HAC**: It is the agglomerative hierarchical clustering. The distance between mentions is $-ln(max(0, AS(s_i, s_j)))$, and the distance between partitions utilizes the average metric. (3) **Our-positive**: it only utilizes words and embedding to compute the similarity. (4) **Our-all**: it utilizes words, embedding value, and co-occur times to compute the similarity between mentions.

Evaluation Analysis. Results in Tab. 2 show that our approach outperforms baselines on both metrics. The reason is that we consider the quality of partitions in a global view and can guarantee the lower similarity of service mentions inside the same partition. Also, our-all is better than our-positive because introducing negative evidence prevents a service mention from being assigned to a wrong partition. Furthermore, K-means does not get good performance since it needs to predefine the number of partitions while the number of service partitions is uncertain. Our approaches outperform HAC since we consider the partition from the global optimization view.



(a) θ_1 on WeChat-All. (b) δ on WeChat-All (c) θ_2 on Housework.

Fig. 3: Performance of different hyper parameters

6.4 Parameter Sensitivity

We discuses the parameter setting of δ in Sec. 3.2, θ_1 in Alg 1, and θ_2 in Alg 2.

The maximum distance θ_1. We analyze the performance sensitivity of the account acquisition algorithm in terms of the maximum distance θ_1, on WeChat-All, while $\delta = 0.1$ for default. Results in Fig. 3 (a) shows that as the distance threshold becomes large, the precision decreases while the recall score and F1 score increase. Moreover, performance becomes insensitive when θ_1 is more than 1. Hence, we set $\theta_1 = 1$ in our experiments.

The parameter δ. We analyze the performance sensitivity of the account acquisition algorithm in terms of the parameter δ on WeChat-All. Results in Fig. 3(b) show that, as the δ becomes larger, the F1 score first increases and then decreases. That indicates it is helpful if we consider the distance of each service domain, but emphasizing more on it will damage the performance. Hence, we set $\delta = 0.1$ in our experiments

The local quality threshold θ_2. We test the sensitivity of the local quality threshold on the *Housework* domain on WeChat-All. Results in Fig. 3(c) show that as the local threshold becomes larger, the **Ma-P** and **Mi-P** become higher. That is because high local threshold guarantees service mentions in the same partition are similar. Moreover, the coherence score increases when θ_2 is less than 0.75, and decreases when θ_2 is more than 0.75. That is because larger local quality makes two similar service mentions be assigned into two different partitions. Hence, we set the local quality threshold as 0.75 in our experiments.

6.5 Service KB Freshness

We discuss the scenarios of the service KB freshness as follows:

Evaluation Analysis of Service Change. We utilize the service mentions of the *Medical* domain of Wechat-All. We split the service mentions as the old and the new service mention in the proportion 70%:30%. We first separate 70% old mentions and then assign these 30% new service mentions to the existing partitions or assign them to a new partition. Finally, we select 100 new service mentions and its assigned partition to label whether it is right. We do not compare HAC since HAC need to re-partition from scratch, which is not practical. Specifically, the precision of K-means, our-positive, our-all is 0.12, 0.78 and 0.85

Table 3: Results of dynamic scenarios

	New Accounts			*New Domain*		
Method	***P***	***R***	***F1***	***P***	***R***	***F1***
TextCNN	0.558	0.669	0.608	0.611	0.733	0.667
Nearest	0.692	0.811	0.747	0.663	0.972	0.788
KNN	0.726	0.832	0.775	0.722	0.985	0.833
Bert	0.847	0.892	0.869	0.820	0.985	0.895
Ours	**0.924**	**0.913**	**0.918**	**0.938**	**0.987**	**0.962**

respectively. K-means get low performance since it cannot assign the new service mention as a new partition. Also, the performance of our-all is better than our-positive shows that introducing a negative score can improve the precision.

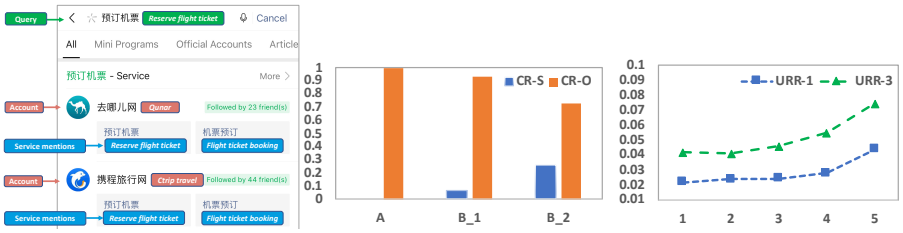
Evaluation Analysis of New Accounts. We obtain another 500 accounts in WeChat which do not exist in the 11000 mentioned in Sec. 6.1. Results in Tab. 3 show that our approach outperforms when obtaining new accounts within the target domains, which indicates the effectiveness of our system to handle the new accounts scenario.

Evaluation Analysis of New Service Domain. We use the *Housework* domain in WeChat-All as the new service domain. Then, we first obtain the accounts in the new domain and then canonicalize the service mention in this new domain. First, we show the evaluation of obtaining the accounts in the new service domain in Tab. 3. Specifically, Tab. 3 shows that our approach outperform other baselines, which indicates the excellent flexibility of our service KB construction system. Moreover, our approach can obtain the accounts of new domains without retraining the model.

Second, since canonicalizing service mentions in a new domain is independently with other domains, thus the evaluation process is the same as the experiment in Sec. 6.3. The observations in new new domains are similar to results in Sec. 6.3, therefore we do not elaborate it in details.

6.6 Online A/B Testing for Service Searching

Our service KB has been deployed in WeChat to support various of downstream applications for billions of users, such as the service searching application. Fig. 3(a) shows an example of it. Specifically, when the user searches the service *Reserve flight ticker*, the search engine of WeChat will retrieve a set of official accounts from our service KB that provides this service for users, such as *Qunar* and *Ctrip travel*. We perform a large-scale online A/B testing on one million users to show how service KB facilitates the users. For the online A/B testing, we split the users into three groups in 2:4:4 proportion: (1) A group: they cannot access the service searching, which means the search engine will not retrieve the result supported by our service KB for these users. (2) B_1 group: They can only access the *Transport* service domain. (3) B_2 group: they can access the service searching of the *Housework, Transport*, and *Medical* service domains. We record the activities of these users in 8 days based on the following metrics:



(a) Application Example. (b) The click rate (CR). (c) the URR of B_2 group.

Fig. 4: Performance of the online A/B testing experiment

- **Click Rate of Service (CR-S)**: the proportion of users who input queries and click the service supported by our service KB. Also we use **CR-O** to denote the click rate of the other information.
- **User Retention Rate of 1 day (URR-1)**: The proportion of the users who use the application today and use it again in the next day.
- **User Retention Rate of 3 day (URR-3)**: The proportion of the users who use the application today and use it again in next three days.

The Click Rate (CR) and User Retention Rate (URR) are the two most critical metrics in real-world industry applications. CR measures the influence of applications since the users are likely to click the most useful one. URR measures whether one application attracts and can retain users. Specifically, URR-3 can reflect long term retention tendency of the users compared with URR-1. Fig. 4(b) shows the click rate distribution of three groups, which is obtained by averaging the click rate of the eight days. The high CR-S of B_1 and B_2 group indicates the great influence of the service searching application, which can support the requirements of the users. Since the computation of the URR-3 need the result at least three days, we report the results from the first day to the fifth day. Fig. 4(c) shows the URR-1 and URR-3 in these five days of B_2 group. We observe that both URR-1 and URR-3 become larger as time goes, which indicates users tend to use the service. Furthermore, the increasing trend of the URR-3 is faster than the URR-1, which shows that the users are really attracted by service searching and tend to use them in the future.

6.7 Deployment at WeChat

We have integrated our service KB into the WeChat search engine to facilitate various downstream applications. Specifically, our service KB is deployed on the Tencent svrkit server, which is a Remote Procedure Call (RPC) framework designed by WeChat. It is a high concurrency framework that is used as the fundamental backend system of WeChat application. It has developed for 9 years and can support 1 billion people's daily use of WeChat, including query searching, articles reading. Furthermore, our service KB construction system is implemented by Python 3.6 and running on 20 dockers. Each docker is configured with Intel(R) Xeon(R) Gold 6133 CPU @ 2.50GHz and 6 GB memory.

7 Related Work

Knowledge Base Construction. Existing knowledge base construction (KBC) approaches [1, 3, 14, 15, 25, 16] propose to extract triples knowledge from long texts and tables. Specifically, they utilize open information extraction (OpenIE) tools to extract triples from data and fuse triples with the same meaning. However, they are not generalizable enough to be applied to construct the service KB for many real-world applications, such as Amazon and WeChat. Since account descriptions and service mentions are short and ambiguous, these KBC approaches based on long texts and OpenIE are incapable of distinguishing accounts for the target domains and fusing short and ambiguous service mentions.

Text Classification and Synonym Discovery. We introduce the related work on the text classification and synonym discovery problems. First, existing text classification approaches [4, 11, 13] build on the supervised learning manner. They rely on many human labels to train models, while these accounts are numerous in real applications, and it is costly to annotate numerous labels. Besides, these approaches are designed for static scenarios, i.e., the class number is fixed. Consequently, they cannot adapt to our dynamic scenario in an effective and efficient way. Second, most existing synonym discovery approaches [5, 20] build on the supervised manner as well, failing to handle unsupervised scenarios (i.e., no labels). Also, existing unsupervised synonym discovery approaches [2, 9, 12] need to predefine the number of synonym clusters and cannot work effectively under the uncertain number of synonym clusters.

8 Conclusion

In this paper, we propose an end-to-end dynamic service KB construction system at WeChat, which facilitates various downstream applications, such as service searching. It takes a set of unlabelled accounts, a set of target service domains, limited seed accounts, and the queries of these accounts as inputs, and outputs a canonicalized service KB. This service KB contains accounts in different domains and stores the canonicalized service mentions for each account. The extensive experiments demonstrate the effectiveness and high practicality of our system.

References

1. Abu-Salih, B., et al.: Healthcare knowledge graph construction: State-of-the-art, open issues, and opportunities. arXiv preprint arXiv:2207.03771 (2022)
2. Alsabti, K., Ranka, S., Singh, V.: An efficient k-means clustering algorithm (1997)
3. Bollacker, K., et al.: Freebase: a collaboratively created graph database for structuring human knowledge. In: SIGMOD. pp. 1247–1250 (2008)
4. Devlin, J., et al.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
5. Fei, H., et al.: Hierarchical multi-task word embedding learning for synonym prediction. In: SIGKDD. pp. 834–842 (2019)
6. Goh, K.I., et al.: The human disease network. Proceedings of the National Academy of Sciences **104**(21), 8685–8690 (2007)

7. Guo, Q., et al.: A survey on knowledge graph recommender systems. TKDE (2020)
8. Hastad, J.: Clique is hard to approximate $n^{1-\epsilon}$. In: Proceedings of 37th Conference on Foundations of Computer Science. pp. 627–636. IEEE (1996)
9. He, Y., et al.: Automatic discovery of attribute synonyms using query logs and table corpora. In: WWW. pp. 1429–1439 (2016)
10. Huang, D.H., Kahng, A.B.: When clusters meet partitions: new density-based methods for circuit decomposition. In: Proceedings the European Design and Test Conference. ED&TC 1995. pp. 60–64. IEEE (1995)
11. Kim, Y.: Convolutional neural networks for sentence classification. In: EMNLP. pp. 1746–1751 (2014)
12. Kisilevich, S., Mansmann, F., Nanni, M., Rinzivillo, S., Clustering, S.T.: Data mining and knowledge discovery handbook (2010)
13. Lavanya, P., Sasikala, E.: Deep learning techniques on text classification using natural language processing (nlp) in social healthcare network: A comprehensive survey. In: ICPSC. pp. 603–609. IEEE (2021)
14. Lehmann, J., et al.: Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. Semantic web pp. 167–195 (2015)
15. Lin, X., Chen, L., Zhang, C.: Tenet: Joint entity and relation linking with coherence relaxation. In: SIGMOD. pp. 1142–1155 (2021)
16. Lin, X., Li, H., Xin, H., Li, Z., Chen, L.: Kbpearl: a knowledge base population system supported by joint entity and relation linking. Proceedings of the VLDB Endowment **13**(7), 1035–1049 (2020)
17. Orogat, A., et al.: Smartbench: demonstrating automatic generation of comprehensive benchmarks for question answering over knowledge graphs. VLDB **15** (2022)
18. Peng, C., Xia, F., Naseriparsa, M., Osborne, F.: Knowledge graphs: Opportunities and challenges. Artificial Intelligence Review pp. 1–32 (2023)
19. Ren, W., Lian, X., Ghazinour, K.: Online topic-aware entity resolution over incomplete data streams. In: SIGMOD. pp. 1478–1490 (2021)
20. Shen, J., et al.: Mining entity synonyms with efficient neural set generation. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33 (2019)
21. Song, Y., et al.: Directional skip-gram: Explicitly distinguishing left and right context for word embeddings. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers). pp. 175–180 (2018)
22. Wang, P., Zheng, W., Wang, J., Pei, J.: Automating entity matching model development. In: ICDE. IEEE (2021)
23. Zhang, C., et al.: Taxogen: Unsupervised topic taxonomy construction by adaptive term embedding and clustering. In: SIGKDD. pp. 2701–2709 (2018)
24. Zhu, X., Wang, L., Xin, H., Wang, X., Jia, Z., Wang, J., Ma, C., Zengt, Y.: T-finkb: A platform of temporal financial knowledge base construction. In: 2023 IEEE 39th International Conference on Data Engineering (ICDE). pp. 3671–3674. IEEE (2023)
25. Zhu, X., Xin, H., Shen, Y., Chen, L.: Hit-an effective approach to build a dynamic financial knowledge base. In: International Conference on Database Systems for Advanced Applications. pp. 716–731. Springer (2023)