The Journal of Systems and Software 163 (2020) 110540

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Diversified keyword search based web service composition

Huanyu Cheng[a], Ming Zhong[a,*], Jian Wang[a]

School of Computer Science, Wuhan University, 430072, China

ARTICLE INFO

Article history:
Received 8 March 2019
Revised 18 December 2019
Accepted 29 January 2020
Available online 30 January 2020

Keywords:
Diversification
Service composition
Keyword search
Web service
Quality of service

ABSTRACT

To assist system engineers in efficiently building service-based software systems, the keyword search based service composition approach on service connection graphs (scgraphs) has been proposed recently. However, due to the ambiguity of keywords, a keyword query may represent a bunch of different user requirements. Thus the current approach that only returns the composition with the optimal Quality of Service (QoS) cannot guarantee to hit the spot. In this paper, in order to satisfy the various possible requirements underlying a given keyword query, we formally introduce the top-*k* diverse service composition problem, and present a novel diversified keyword search approach on scgraphs to address it. Specifically, we firstly propose an All-Then-Diversify (ATD) algorithm that enumerates all potential compositions by searching a scgraph and then derives the top-*k* diverse subsets by deriving the maximal independent sets of a similarity graph. Then, due to the possibly large number of potential compositions, we present a Pop-And-Diversify (PAD) algorithm that only maintains a similarity graph of the top compositions that have been found so far during the search and computes its maximal independent sets incrementally until convergence, thereby reducing unnecessary computation overheads. Moreover, we propose two composition similarity measurements w.r.t. the categories or descriptions of services respectively. Lastly, the experimental results on ProgrammableWeb.com demonstrate that, our approach outperforms another state-of-the-art composition diversification approach on both metrics of density and redundancy, and meanwhile, improves the efficiency of diversification significantly.

1. Introduction

Web services are software systems that are designed to support interoperable machine-to-machine interaction over a network (Hugo Haas, 2004). They can be composed loosely for building complex distributed software systems under a framework called Service Oriented Architecture (SOA). In contrast to conventional software architectures primarily delineating the organization of a system in its (sub)systems and their interrelationships, SOA captures a logical way of designing a software system to provide services to either end-user applications or other services distributed in a network through published and discoverable interfaces (Psiuk and Zielinski, 2015; Alho and Mattila, 2015). The component services jointly offer the functionality of a Service-Based System (abbr. SBS) and collectively fulfil its users quality requirements (He et al., 2017).

Due to the advantage of SOA, there has been a rapid growth of demands of building SBSs. The traditional service composition approaches consist of three key tasks, namely, system planning, service discovery and service selection. System planning (e.g. Klusch et al., 2009; Pistore et al., 2005; Kster et al., 2007) is to determine the slots of services needed by a specific SBS to fulfill the functionality. Service discovery (e.g. Calinescu et al., 2011; Wang et al., 2017) is to find candidate services for each slot through service registries or service search engines. Lastly, service selection (e.g. Hoffmann et al., 2007; He et al., 2014) is to identify a service from each set of candidate services, so that these services can be composed together and meanwhile achieve the optimality and constraints in system qualities, such as cost, reliability, throughput, etc. Obviously, such approaches are too complicated and thereby not efficient enough for ordinary system engineers without comprehensive and in-depth knowledge of the three tasks.

To assist system engineers in efficiently building SBSs, He et al. (2017) propose KS3, a keyword search based service composition approach, which integrates and automates the procedures of system planning, service discovery and service selection. Given a service connection graph where the nodes represent web services and the edges between them represent their composability, the users can input a few keywords to describe the main functional services of an SBS with system quality constraints and optimization goals, and KS3 will return a subgraph of the service connection graph as the solution to the service composition for the SBS.

* Corresponding author.
E-mail addresses: chy@whu.edu.cn (H. Cheng), clock@whu.edu.cn (M. Zhong), jianwang@whu.edu.cn (J. Wang).

https://doi.org/10.1016/j.jss.2020.110540
0164-1212/

The subgraph contains not only the functional services matched by the given keywords but also bridging services that connect these functional services, even though they are not directly required by the users. Therefore, by leveraging the studies of keyword search over graphs (e.g. He et al., 2007; Kacholia et al., 2005; Golenberg et al., 2008), the keyword search based service composition approach can automatically produce prototype-like solution with respect to a given set of keywords describing the SBS.

However, the optimal solution may not satisfy the real user requirements underlying the given keywords, because a same set of keywords may represent different user requirements due to the inherent ambiguity of keywords. For example, a user requires a system that interacts with a map to find books in nearby public libraries as shown in Fig. 1. Thus, she inputs two keywords "book" and "map" to search for service compositions from a real-world web service database called ProgrammableWeb.com (abbr. PW)[1] The search results of the keyword "book" include nearly 600 services divided into a few of categories by PW. Most of them, such as "Booker Customer", "Viator", "Optimal Booking", etc., fall into the categories such as "Booking", "Travel", "Hotels", etc., and apparently, offer the functionality of booking something like hotels. In contrast, only several search results in the categories "Book" and "Reference" are really used for deriving the information of books. As a result, if we search for paths that connect the returned services of "book" and "map" only with respect to the overall QoS, we will only get many compositions with similar functionality like the mashup "Holidayen" in PW which includes "Viator" and "Google Maps" and helps travelers plan and book personalized holiday experiences, but not the mashup "CodexMap" which includes "LibraryThing" and "Goolge Maps" and lets the user find books graphically on a map. Consequently, the existing approaches like KS3 that ignore the ambiguity of user requests cannot satisfy the various requirements of different users.

Therefore, the diversity should be taken into account in keyword search based service composition. In a nutshell, the diversification is to find a set of results, which have as high objective scores (like quality in our case) as possible, and meanwhile, are dissimilar to each other. Since it can improve user satisfaction by expanding the user requirements covered by the returned results, it is widely used in many applications (Agrawal et al., 2009; Capannini et al., 2011; Vee et al., 2008; Yu et al., 2009a; Demidova et al., 2010; Fraternali et al., 2012). In this way, the search results like "CodexMap" can be returned in the results.

The existing diversification approaches for services are not devised to find a set of diverse service compositions. Mei et al. (2010) propose DivRank based on a reinforced random walk, which automatically balanced the prestige and the diversity. Kang et al. (2016) use expansion ratio to evaluate the diversity of web services. Besides, semantic divergence based evaluation of web service communities are proposed in Nam et al. (2016b). However, they can only diversify individual services but not compositions of services. Nam et al. (2016a) first achieve the diverse functional web services and then leveraged web service dependency network to diversify the compositions. Although they try to diversify service compositions, they actually have a totally different definition of "diversity". Their diversity is the scalability of compositions, which means measuring how many dependent services exist for each single composition. In contrast, we focus on find a set of service compositions that are dissimilar enough to each other.

The diversification of keyword search based service compositions is a nontrivial task. Because it can generate massive results by combining a number of search paths on a graph. This is similar to the *over-specialization* in recommender systems Yu et al. (2009b). Thus, it would be too time-consuming to directly adopt the traditional diversification approaches, which enumerate all possible service compositions and then diversify them separately. Instead, we hope to implement diversification simultaneously while generating the results.

In this paper, we propose a novel keyword search based approach that finds the top-k diverse service compositions with different optimization goals of QoS, such as cost, throughput and reliability. As shown in Fig. 2, our approach has two phases, namely, offline and online. In the offline phase, we build a service connection graph from a library of web services crawled from a given web service portal. Moreover, we build an inverted index from keywords to the nodes on the graph by parsing the service descriptions. In the online phase, the system engineers can input a few keywords describing their requirements or the main functionality of expected SBSs. The nodes/services matched by the keywords will be retrieved from the index and be passed to our algorithm. The All-Then-Diversify algorithm will firstly generate all top results of compositions. Then, it will measure the similarity and gradually build a *similarity graph* by adding the results as nodes in the descending order of QoS, and meanwhile, incrementally compute the *maximal independent sets* of the similarity graph in a baseline method until convergence, namely, a maximal independent set of k results have been found. The Pop-And-Diversify algorithm will generate the top results of compositions one by one, and only use the top results to build the similarity graph, until a maximal independent set of k results have been found in the runtime. Lastly, the top-k diverse results will be post-processed as in He et al. (2017) to generate the final prototype-like service compositions.

The main contributions of this paper are as follows.

- We formalize a diversified keyword search based service composition problem, which is to find the top-k compositions that are dissimilar to each other with respect to QoS on a specific service connection graph. For measuring the dissimilarity between compositions, two metrics are proposed.
- We develop two algorithms, namely, All-Then-Diversify and Pop-And-Diversify to address the problem. Both of them use graph search to enumerate compositions and find a diverse subset of compositions by deriving the maximal independent sets of a composition similarity graph. Differently, the latter one can eliminate unnecessary computations, thereby improving efficiency.
- We perform comprehensive experiments on a real world service connection graph, which is built from programmableweb.com. The experimental results show that our approach can reveal more diverse service compositions compared with other diversification method and the Pop-And-Diversify algorithm can significantly reduce extra overheads.

The rest of this paper is organized as follows. In Section 2 we describe our graph model for web service library and research problems. Section 3 describes in detail our similarity metrics. Section 4 states our search approach in detail. Section 5 evaluates the effectiveness and efficiency with experimental results. Section 6 reviews the related work. Section 7 shows the limitation and threats to validity. Section 8 concludes the paper.

2. Problem statement

2.1. Preliminaries

In the scenario of service composition, we can generally model a library of web services as a service connection graph (abbr. scgraph). A scgraph which is similar to service dependency network in Nam et al. (2016a) is a node-labeled directed simple graph,

[1] http://www.programmableweb.com/.

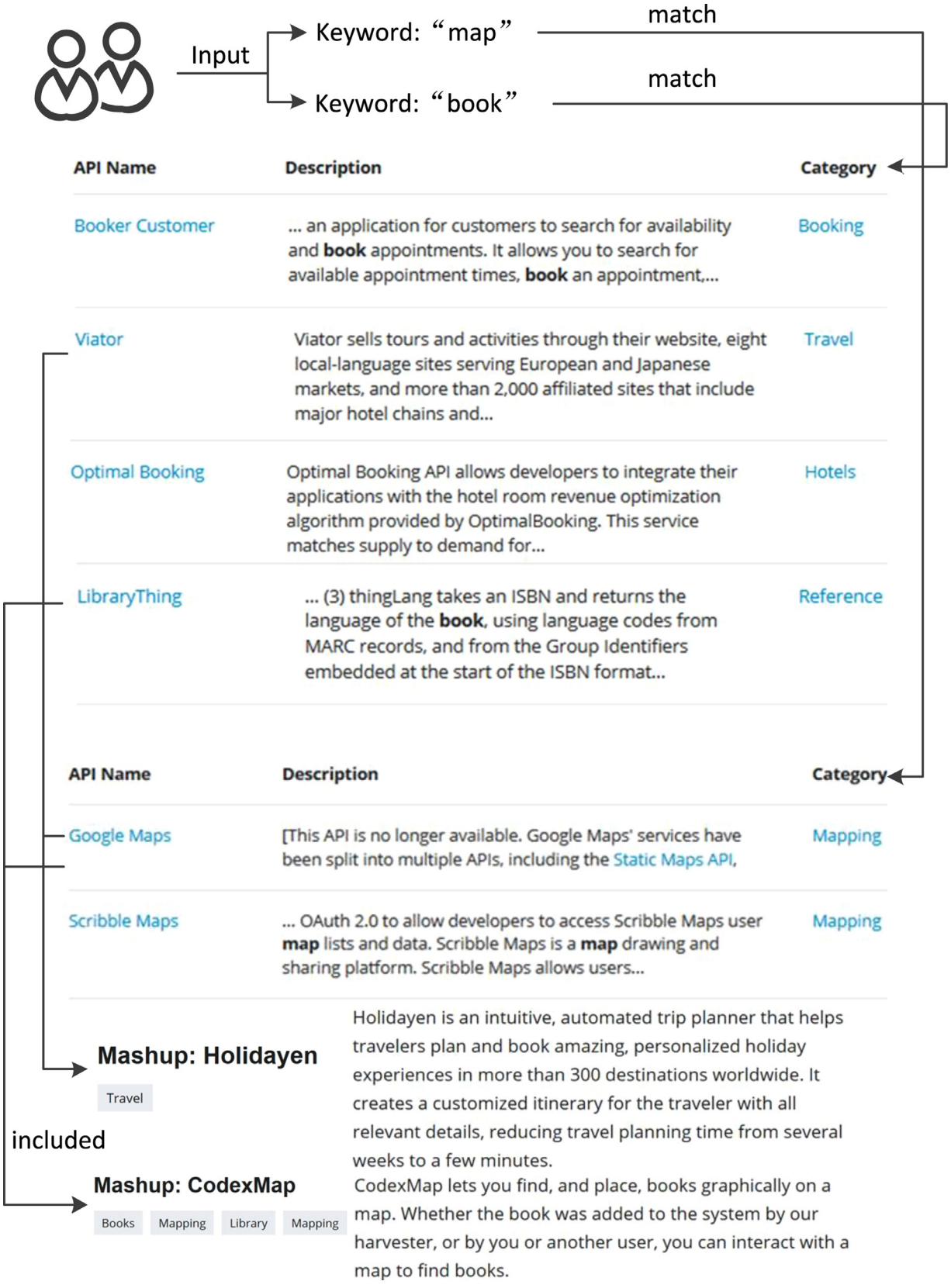


Fig. 1. An example from ProgrammableWeb.com.

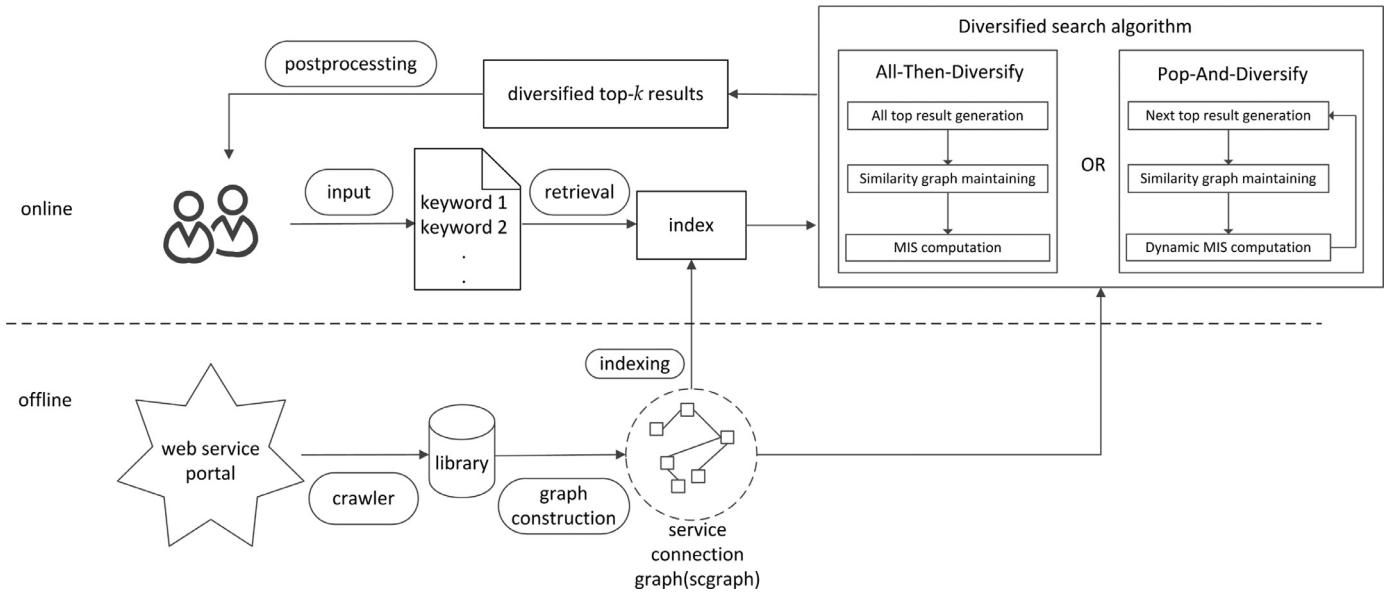


Fig. 2. An overview of our proposed approach.

where the nodes represent the web services and the edges between them represent the service connection, namely, whether the two corresponding web services can be composed in the order specified by the direction of the edge. The formal definition of scgraph is given as follows.

Definition 1 (SCGraph)**.** We use $G = (V, E)$ to denote a scgraph for a given web service library, where V is the set of nodes and E is the set of edges in G.

Definition 2 (Nodes)**.** For each web service in the library, the scgraph G has a corresponding node $v \in V$. Each node in G has a description that introduces its functionality and a few of quality values. Moreover, the nodes may have been categorized in some scgraphs. For example, the web services in programableweb.com have categories such as social, sports, education, enterprise, etc.

Definition 3 (Edges)**.** For each pair of composable web services v_1 and $v_2 \in V$, the scgraph contains an edge $e(v_1, v_2) \in E$ between v_1 and v_2. $e(v_1, v_2)$ is directed, pointing from v_1 to v_2 if v_2 can be the succeeding node of v_1 in the composition of v_1, v_2. Moreover, an edge e can be bidirectional if v_1 can also be the succeeding node of v_2 in the composition.

The edges in our service connection graphs represent sequentially and loosely coupling relationships between two web services, which could be simple I/O pipelines or complicated procedures implemented by system engineers.

The approaches for building an scgraph from a web service library can be grouped into two major categories: data mining based (Kil et al., 2009; Huang et al., 2012; Ni et al., 2016; Lyu et al., 2014) and semantics based Dojchinovski et al. (2012); Feng et al. (2014). The data mining based approaches mine the service composability information from their collaboration history. The semantics based approaches discover the service composability information by mining semantic associations and interactions between services according to well-defined ontologies. Once a scgraph has been built offline, it remains relatively stable and can still be updated with the minimum overheads upon certain events, e.g., adding new services or removing existing services.

2.2. Problems

To simplify the problem, we use subtrees but not subgraphs of scgraphs to represent composition results. Though the topology of real service compositions could be graphs and more complicated than trees, we can deal with it by postprocessing the result trees. In the postprocessing stage, we will elaborate the result trees by complementing the missing edges, identifying an entry node and an exit node, and so on like in (He et al., 2017). Given a scgraph G and a keyword query $Q = \{t_1, ..., t_l\}$ containing l ($l \geq 2$) keywords, we denote the set of result trees of Q on G as $Q(G)$, which are subtrees of G and contain all keywords in Q. To formalize this concept, let us consider the following definitions.

Definition 4 (Search Path)**.** Given a graph $G = (V, E)$ and a keyword t, a search path P in G is a sequence of nodes $v_1/v_2/\cdots$, where 1) v_1 contains t, 2) $v_i \in V$, 3) $e(v_i, v_{i+1}) \in E$,with $i \geq 1$.

Definition 5 (Result Tree)**.** Given a graph $G = (V, E)$ and a query Q, a result tree T is comprised of a set of search paths from each keyword in Q on G, whose last nodes are a same node, namely, the root of T.

For example, Fig. 3 presents a small example scgraph G, and given a keyword query $Q = \{$ticket booking, taxi calling$\}$, Fig. 4 shows five result trees, i.e., T_1, T_2, T_3, T_4 and $T_5 \in Q(G)$. For conciseness, only the service ID is shown in each node. Let us consider T_1. The node v_3 contains "ticket booking" and node v_5 contains "taxi calling" in their descriptions respectively. Although v_3 and v_5 are not directly connected, they can be connected via v_1. Thus, v_1 is included in T_1 as a bridging node (bridging service), indicating that v_1, v_3 and v_5 can be composed together to perform both ticket booking and taxi calling. Besides T_1, T_2, T_3, T_4 and T_5, other potential result trees not included in Fig. 4 can also integrate ticket booking and taxi calling functionalities.

In order to rank the result trees, we need to measure the goodness of service compositions represented by them. There could be various measurements like QoS, relevance to queries, etc. In this paper, we consider reliability, throughput and cost as the metrics of result trees respectively. These qualities named *structure-independent system qualities* He et al. (2017) are calculated independently of its structure and dynamics, based only on the quality of its component services. Due to the postprocessing where the

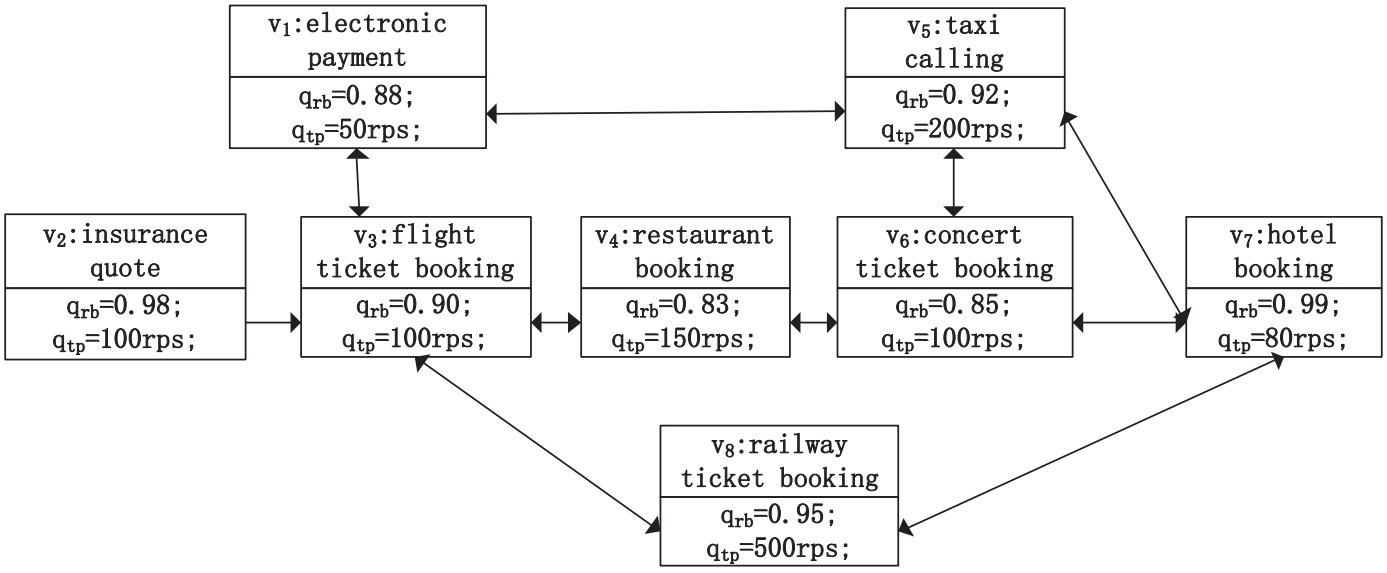


Fig. 3. An example scgraph.

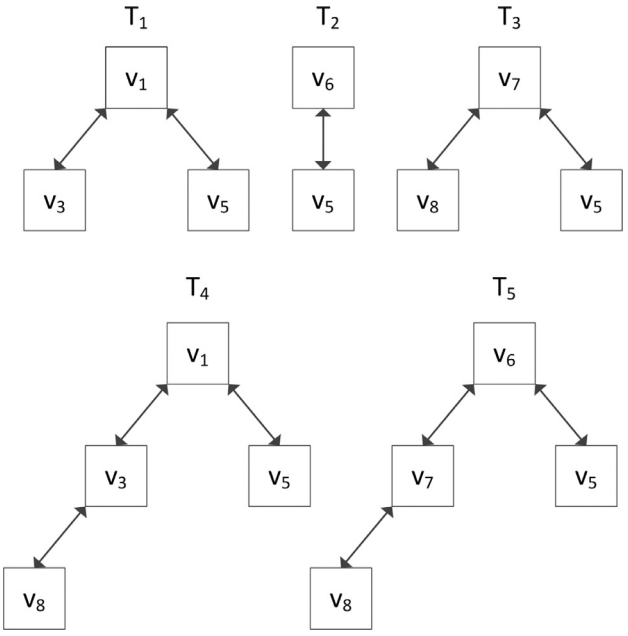


Fig. 4. The result trees to query $Q = \{\text{ticket booking, taxi calling}\}$.

topology of the result trees will change, it is difficult to consider the structure-dependent system qualities. Thus, we only consider the structure-independent system qualities.

Let the reliability $q_{rb}(T)$ of a result tree T be the product of reliability of its nodes. We have

$$q_{rb}(T) = \prod_{v \in V^T} q_{rb}(v) \tag{1}$$

where V^T is the set of nodes in T and $q_{rb}(v)$ is the reliability of service represented by node $v \in V^T$.

Let the throughput $q_{tp}(T)$ of a result tree T be the minimum throughput of nodes. We have

$$q_{tp}(T) = \min_{v \in V^T} q_{tp}(v) \tag{2}$$

where $q_{tp}(v)$ is the throughput of service represented by node $v \in V^T$.

Table 1
The metric values of example result trees.

Result Trees	Reliability	Throughput	Cost
T_1	0.73	50rps	3
T_2	0.78	100rps	2
T_3	0.87	80rps	3
T_4	0.69	50rps	4
T_5	0.74	80rps	4

Let the cost $q_c(T)$ of a result tree T be the sum of node cost. We have

$$q_c(T) = \sum_{v \in V^T} q_c(v) \tag{3}$$

where $q_c(v)$ is the cost of service represented by node $v \in V^T$. For conciseness, we assume the cost of all services is equal to 1.

Based on the above metrics of result trees, the top-k result trees with respect to a specific metric can be identified correspondingly. For example, the metric values of the result trees in Fig. 4 are given in Table 1. Thus, the top-k with $k = 1$ among these example result trees is T_3, T_2 or T_2 with respect to reliability, throughput or cost respectively.

Different from the previous work (He et al., 2017) that finds only the optimal (i.e., top-1) connected tree in scgraphs, we aim to find the top-k diverse result trees, so that the result set could satisfy different user demands. Formally, the problem to be addressed in this paper is as follows.

Definition 6 (Diversified Top-. *k*Service Composition) Given a scgraph G, a keyword query Q, an objective function F (e.g., q_{rb}, q_{tp} or q_c), a diversity function D and a positive real number $\theta \in [0, 1]$ as diversity threshold, find a set of result trees $R \subseteq Q(G)$ such that 1) $|R| = k$, 2) $\min_{T,T' \in R} D(T, T') \geqslant \theta$, and 3) $\min_{T \in R} F(T)$ is maximized.

3. Similarity metrics

Due to the difference of data, e.g. some short of category and some without description, we introduce the following similarity metrics which can be used to calculate the diversity function in Definition 6 by the following equation:

$$D(T, T') = 1 - sim(T, T') \tag{4}$$

where T and T' are two result trees and $sim(T, T')$ is their similarity function.

Jaccard Similarity based on Category (JSC). If the services have been categorized, we assume the more common service categories two result trees have, the more similar they are. Formally, given two result trees T and T', we compute their Jaccard similarity as

$$sim_{jsc}(T, T') = \frac{|C(T) \bigcap C(T')|}{|C(T) \bigcup C(T')|} \tag{5}$$

where $C(T)$ is the set of all distinct categories of services contained by T.

Cosine Similarity based on Topic Distribution (abbr. CSTD). Lastly, we apply Latent Dirichlet Allocation (abbr. LDA) on the description of services. Based on the generated topic distributions of services, each result tree T can be represented as a vector $\vec{T} = \{z_1, \cdots, z_n\}$, where n is the number of latent topics and each element z_i with $1 \leq i \leq n$ is the mean of the probabilities that the services contained by T relates to the corresponding topic. The Cosine similarity between two result trees T and T' is computed as

$$sim_{cstd}(\vec{T}, \vec{T'}) = \frac{\vec{T} \cdot \vec{T'}}{\|\vec{T}\| \cdot \|\vec{T'}\|} \tag{6}$$

In practice, we can choose the suitable metric with respect to whether service categorization exists, how much similarity computation overheads can be afforded, etc. Moreover, the terms with frequencies and latent topics of services can be generated offline, thereby reducing the costs of building the term sets, term frequency vectors and latent topic vectors of result trees effectively.

4. Diversified search

In this section, we address the problem of deriving the top-k diverse result trees on a scgraph. We first propose a All-Then-diversify algorithm that enumerates all candidate results and then derives the top-k diverse results in Section 4.1. Since keyword search on graphs usually generates massive candidate results, in order to improve search efficiency, we propose a Pop-And-Diversify algorithm in Section 4.2. It executes result generation and result diversification simultaneously, and can stop early when the top-k diverse results have been found.

4.1. All-Then-Diversify Algorithm

4.1.1. Algorithm overview

A straightforward way to find the top-k diverse result trees is to generate all results trees and then find a diverse k-size set of top results from them.

Algorithm 1 presents the pseudo-code of such an algorithm called All-Then-Diversify (ATD). We denote a set where the diversity between each pair of result trees is greater than θ and a list of top results respectively by I and R. In the followings, we will present the algorithms of result generation and result diversification, respectively.

4.1.2. Result generation

Following the line of *backward search* Bhalotia et al. (2002), our result generation algorithm decomposes the problem of deriving result trees of a keyword query Q into $|Q|$ independent subproblems, each of which is to iteratively enumerate search paths from one of the query keywords by heuristics. Once a set of search paths from all keywords meet at a same root node, a result tree is generated by joining the paths.

Algorithm 2 presents the pseudo-code of the result generation algorithm. Let R be a priority queue of candidate result trees in descending order of their objective values, PQ_t be a priority queue of search paths from $t \in Q$ that have not been traversed yet, and $H_{t,v}$ be a container that records the traversed search paths from t to a specific node $v \in V$. Firstly, the algorithm uses function INIT() to init the parameters. As shown in Algorithm 3, all PQ_t for each keyword in query will be initialized (line 1-4). Then, Algorithm 2 traverses the scgraph G iteratively, until all search path queues are empty (line 3-5).

Algorithm 4 shows the the pseudo-code of the function TRAVERSE(). A nonempty queue PQ_t is chosen in a round-robin way (for avoiding search stagnation), and a search path P_t whose end node is v is dequeued from PQ_t and is added into $H_{t,v}$ (line 1-4). Then, the newly found search paths from t through appending a neighbor edge of v at the end of P_t will be generated and

Algorithm 1: All-Then-Diversify Algorithm.

Input: a scgraph G, a keyword query Q, a positive integer k, and a diversity threshold $\theta \in [0, 1]$;
Output: the diverse result trees;
1 $I \leftarrow \emptyset$;
2 $R \leftarrow \emptyset$;
3 $R \leftarrow$ allResults(G, Q);
4 $I \leftarrow$ findMIS(R, k, θ);
// Find a k-size set where the diversity between each pair of result trees is greater than θ from R
5 **return** I;

Algorithm 2: allResults(G, Q).

Input: a scgraph G, and a keyword query Q;
Output: the candidate results;
1 $R \leftarrow \emptyset$;
2 INIT();
3 **while** $\exists t \in Q$, PQ_t *is not empty* **do**
4 TRAVERSE();
5 **end**
6 **return** the result tree set R;

Algorithm 3: INIT().

1 **foreach** *keyword* $t \in Q$ **do**
2 retrieve the services/nodes containing t from the index;
3 put the neighbor edges of each node containing t into PQ_t;
4 **end**

Algorithm 4: TRAVERSE().

1 choose a nonempty PQ_t in a round-robin way;
2 $P_t \leftarrow PQ_t$.dequeue();
3 $v \leftarrow$ the last node on P_t;
4 $H_{t,v}$.add(P_t);
5 **foreach** *neighbor edge* $e \in E$ *of* v **do**
6 generate a new search path P'_t by appending e at the end of P_t;
7 PQ_t.enqueue(P'_t);
8 **end**
9 **foreach** *combination of paths in* $H_{t',v}$ *with* $t' \neq t \in Q$ **do**
10 generate a result tree T comprised of the combination and P_t;
11 R.insert(T);
12 **end**

enqueued into PQ_t (line 5-8). Each combination of traversed search paths from other keywords $t' \in Q$ in $H_{t',v}$ will be joined with P_t to generate a new result tree, and all new result trees will be enqueued into R (line 5-8). At each iteration, the result trees set R will be updated.

In addition, we explain the sorting of search paths in PQ_t as follows. A search path can be considered as a special result tree with only one branch. Thus, we can use Eqs. (1)–(3) to calculate the objective value $F(P_t)$ of a search path P_t respectively. Heuristically, a search path P_t takes precedence over another search path P'_t in PQ_t if and only if $F(P_t)$ is better than $F(P'_t)$.

4.1.3. Result diversification

In order to find the sets of diverse top compositions, we firstly introduce the concept of *similarity graph* (abbr. SG).

Definition 7 (Similarity Graph)**.** Given a list of top result R and a dissimilarity threshold θ, the similarity graph of R, denoted as $G^s = (\mathbb{T}^s, E^s)$, is an undirected graph where 1) for each result $T \in R$, there is a corresponding node $T \in \mathbb{T}^s$, and 2) for any two results T, $T' \in R$, there is an edge $e(T, T') \in E^s$ if and only if T and T' are similar to each other, namely, $D(T, T') \leq \theta$ where $D(\cdot, \cdot)$ is a diversity function, namely, Eq. (4).

Note that the similarity graph is of web service compositions, different from the previous work (Kang et al., 2016) of which the similarity graph is of individual web services.

With the similarity graph, a set of top results that are dissimilar to each other is actually equal to a set of nodes that are not adjacent to each other, namely, an independent set on the similarity graph. Thus, our diversified search algorithm builds a similarity graph from scratch by gradually adding the next result with the highest objective value into the graph, and meanwhile, finds the *maximal independent sets* (MIS) on each version of similarity graph until there exists an MIS with k nodes. It is easy to prove that the nodes of the MIS are the top-k diverse results.

Definition 8 (Maximal Independent Set)**.** Given a similarity graph G^s, a set of nodes $I \subseteq \mathbb{T}^s$ is an independent set, if each node in I is not adjacent to the others. Let $M(G^s)$ be the set of all independent sets on G^s. If there is no other independent set $I' \in M(G^s)$ such that $I \subset I'$, I is a maximal independent set.

As shown in Algorithm 5, it incrementally finds the MISs of R until there exists a MIS of which the size is k (line 2-6). To find the MISs, we use a baseline method that leverages the computation of the minimal coverage sets, which are the complement of maximal independent sets. The computational complexity of this approach is $O(n^3)$, where n is the number of nodes on similarity graph.

4.2. Pop-And-Diversify algorithm

Since the backward search style algorithm does not generate the result trees in a strict descending order of objective value, it is actually not necessary to diversify all intermediate results generated during the search. The computational complexity of diversification is inherently very high, so that we should avoid to consider an intermediate result unless it has good enough QoS to be possibly one of the top-k diverse results.

Therefore, we propose a *Pop-And-Diversify (PAD)* algorithm to minimize the overheads of diversification. It is based on a fact that, the keyword search algorithms on graphs normally offer effective estimation of the upper bound of objective values for unknown results to support efficient top-k search. Thus, it can avoid unnecessary result diversification and improve the efficiency of deriving MISs from dynamic similarity graph.

Algorithm 5: findMIS(R, k, θ).

Input: a candidate result set R, a positive integer k, and a diversity threshold $\theta \in [0, 1]$;
Output: the top-k diverse results;
1 $M \leftarrow \emptyset$;
2 **do**
3 $T \leftarrow R$.pop();
4 add T to the similarity graph SG with θ;
5 $M \leftarrow$ the set of MISs retrieved from SG;
6 **while** $\forall I \in M$, $|I| < k$;
7 **return** the result set $I \in M$ with $|I| = k$;

Algorithm 6: Pop-And-Diversify algorithm.

Input: a scgraph G, a keyword query Q, a positive integer k, and a diversity threshold $\theta \in [0, 1]$;
Output: the top-k diverse result trees of Q on G;
1 $M \leftarrow \emptyset$;
2 INIT();
3 **do**
4 $T \leftarrow$ nextTop(G, Q);
5 $M \leftarrow$ dynamicMIS(M, T, θ);
6 **while** $\forall I \in M$, $|I| < k$;
7 **return** the result set $I \in M$ with $|I| = k$;

4.2.1. Algorithm overview

The pseudo-code of the Pop-And-Diversify algorithm is shown in Algorithm 6. Our Pop-And-Diversify algorithm uses a function nextTop() to enumerate the next top result T of the list R, namely, the one with the highest objective value among the remaining results (including unknown results) during the search (line 4). Once a result is popped, another function called dynamicMIS() is used to dynamically find the sets of diverse top results that satisfy the diversity constraint (line 5). Whenever a set of k diverse top results have been found, they are certainly the top-k diverse result trees (see proof in Zhong et al. (2018)), and thus the algorithm can stop early.

4.2.2. Result generation

Algorithm 7 presents the pseudo-code of the function nextTop(). The principle is to ensure that the results that have not yet been generated are worse than the results that have already been generated. This function traverses the scgraph G iteratively, until a result tree with the highest objective value higher than the upper bounds has been found or all search path queues are empty (line 1-4). At each iteration, it is the same as that of All-Then-Diversify algorithm except that the upper bounds of objective values of results that have not been generated yet are estimated in the end (line 3). Lastly, the result tree with the highest object value in R will be returned (line 5).

For each node $v \in V$, we denote by $\eta(v)$ the upper bound of objective values of result trees $\mathbb{T}_v$ rooted at v (including unknown

Algorithm 7: nextTop(G, Q).

Input: G, Q;
Output: the result tree with the highest objective value;
1 **while** $\exists v \in V$, $\eta(v) >$ *R.peak() and* $\exists t \in Q$, PQ_t *is not empty* **do**
2 TRAVERSE();
3 update $\eta(v)$ for each $v \in V$;
4 **end**
5 **return** *R.pop()*;

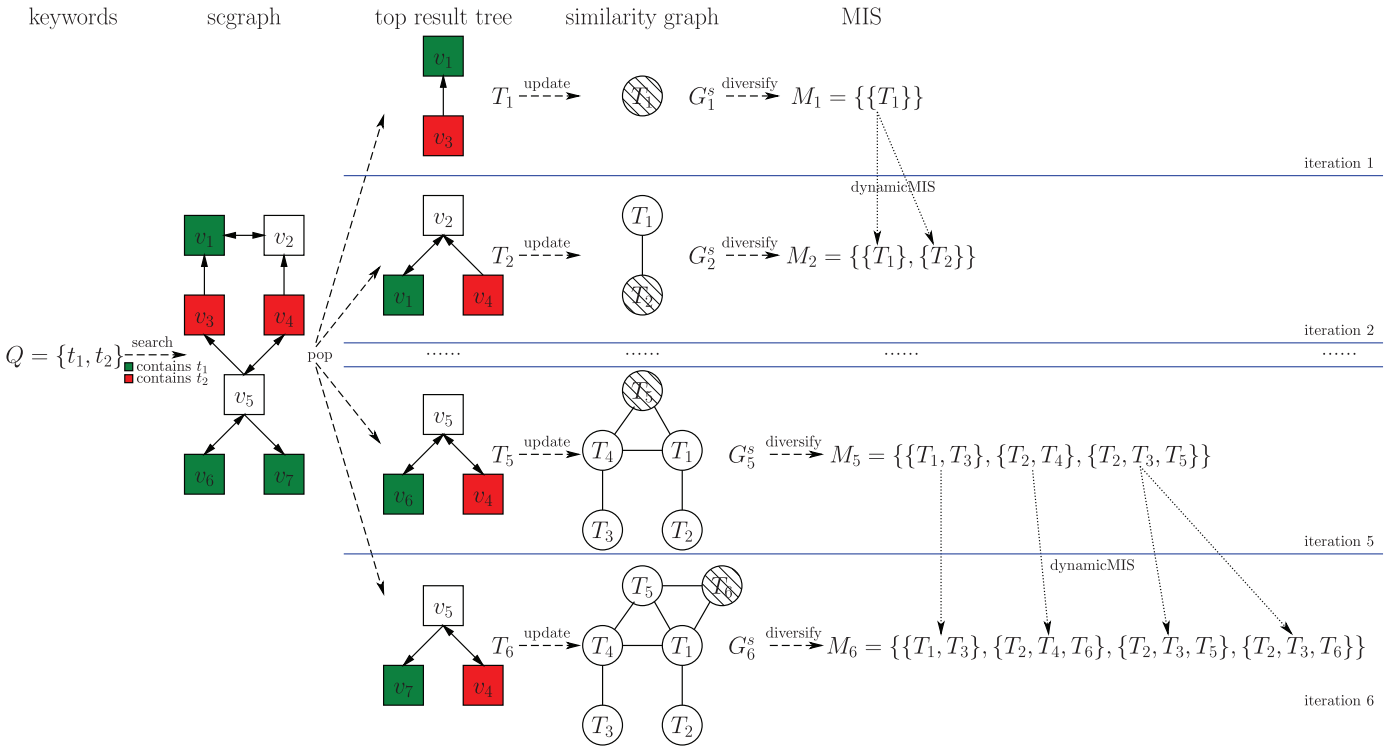


Fig. 5. The example of PAD algorithm.

results). Let $\eta(v)_c$, $\eta(v)_{rb}$ and $\eta(v)_{tp}$ be the upper bound of cost, reliability and throughput respectively. They can be estimated by using the following equations. It is easy to prove the correctness. The result tree with the best QoS in $\mathbb{T}_v$ is surely comprised of a new enumerated keyword path to v. For each keyword, we assume the next path dequeued from PQ_t is exactly the path, though it is unknown whether its last node is v. Thus, we generate a result tree by joining this path with the paths of best QoS from other keywords to v. As a result, the maximum or minimum QoS of these trees must be the final upper bound while QoS can be calculated by Eq. (1)–(3). Once the upper bounds $\eta(v)$ of all nodes $v \in V(G)$ are not better than the QoS of current k-th best result (namely, R), the algorithm can stop early.

$$\eta(v)_{rb} = \max_{t \in Q}\Big(\prod_{t' \in Q, t' \neq t} \max_{P_{t'} \in H_{t',v}} q_{rb}(P_{t'}) \cdot PQ_t.peak\Big) \tag{7}$$

$$\eta(v)_{tp} = \max_{t \in Q} \min_{t' \in Q, t' \neq t} \{\max_{P_{t'} \in H_{t',v}} q_{tp}(P_{t'}), PQ_t.peak\} \tag{8}$$

$$\eta(v)_c = \min_{t \in Q}\Big(\sum_{t' \in Q, t' \neq t} \min_{P_{t'} \in H_{t',v}} q_c(P_{t'}) + PQ_t.peak\Big) \tag{9}$$

where $PQ_t.peak$ is the peak value of $q_{rb}(P_t)$, $q_{tp}(P_t)$ or $q_c(P_t)$ for $P_t \in PQ_t$.

Fig. 5 shows an example of our algorithm. Firstly, the algorithm finds the matched nodes in scgraph for keywords t_1 and t_2. For each matched node, the algorithm searches paths to pop result trees (e.g. T_1, T_2 and so on). Then it updates the SG in order to find the maximal independent sets by function dynamicMIS() iteratively until the MISs meet the requirements.

4.2.3. Result diversification

During the search, a similarity graph grows constantly. After each call of nextTop(), a new node that represents the returned top result and the corresponding edges will be added into the similarity graph. Obviously, repeating the baseline method in each time when the similarity graph has been updated is too expensive due to its high complexity. Thus, we present a dynamic programming based algorithm corresponding to dynamicMIS(), to incrementally find the new maximal independent sets on a similarity graph that is built by adding a new node into the previous similarity graph. Our algorithm can avoid the redundant computation that occurs while deriving the maximal independent sets on a constantly evolving graph, thereby reducing the overall overheads.

Algorithm 8 presents the pseudo-code of the function dynamicMIS(), which can incrementally find the new MISs on a similarity graph that is updated constantly by adding a new node representing the popped top result into it.The details of function

Algorithm 8: dynamicMIS(M, T, θ).

Input: a set of diverse top result sets M, a new top result T and a diversity threshold θ;
Output: the MISs on the updated similarity graph;
update the similarity graph by adding a new node T, and for each other node $T' \in \mathbb{T}^s$ adding an edge $e(T, T')$ if $D(T, T') \leqslant \theta$ where T' is the result tree represented by T';
$M' \leftarrow \emptyset$;
foreach $I \in M$ **do**
  $I' = I \cup \{T\}$;
  if *there are nodes in* I *adjacent to* T **then**
    remove the nodes from I';
  else
    remove I from M;
  end
  add I' to M';
end
remove the non-maximal independent sets in M';
$M \leftarrow M \cup M'$;
return M;

dynamicMIS() is as follows. Let M be the set of MISs on the previous similarity graph G^s_n, and M' be the set of new MISs on the new similarity graph G^s_{n+1} to which a new node T representing the input top result has been added. There are two steps to find M'. Firstly, for each MIS $I \in M$, create a new set $I' = I \cup \{T\}$ (line 8). If there are nodes in I' adjacent to T, remove them from I' so that I' is an independent set (line 4). Otherwise, remove I from M (line 6), because I is a subset of I' and thereby is not maximal. Then, put I' into M' (line 8). Secondly, remove the independent sets in the M' that are non-maximal, namely, are the subsets of some other sets (line 10), and merge the previous set of MISs (with non-maximal sets removed) M and the new set of MISs M' (line 12). Finally, we can find all MISs on the new similarity graph. The proof of correctness of our algorithm can be found in Zhong et al. (2018).

For example, as shown in Fig. 5, the similarity graph G^s_1 is built by importing node T_1. At this moment, the set of MISs on G^s_1 $M_1 = \{\{T_1\}\}$. By adding T_2 to G^s_1, we have G^s_2 and the new node sets $M_2 = \{\{T_1\}, \{T_2\}\}$. The algorithm iterates until node T_6 added into G^s_5. The set of MISs on G^s_5 $M = \{\{T_1, T_3\}, \{T_2, T_4\}, \{T_2, T_3, T_5\}\}$. By adding T_6 to these sets respectively, we have the new node sets $\{T_1, T_3, T_6\}$, $\{T_2, T_4, T_6\}$ and $\{T_2, T_3, T_5, T_6\}$. By removing the adjacent nodes of T_6 from these sets, we have the new set of MISs $M' = \{\{T_3, T_6\}, \{T_2, T_4, T_6\}, \{T_2, T_3, T_6\}\}$ that contains T_6. Meanwhile, we remove $\{T_2, T_4\}$ from M_5 because it is a subset of $\{v_2, v_4, v_6\}$ in M'. Then, we have $M' = \{\{T_2, T_4, T_6\}, \{T_2, T_3, T_6\}\}$ by removing the non-maximal set $\{T_3, T_6\} \subset \{T_2, T_3, T_6\}\}$. Lastly, the new MISs M_6 on G^s_6 are $M_5 \cup M'$, including $\{T_1, T_3\}$, $\{T_2, T_3, T_5\}$, $\{T_2, T_4, T_6\}$ and $\{T_2, T_3, T_6\}$.

5. Experimental evaluation

In this section, we conducted a series of experiments to evaluate the efficiency and effectiveness of the proposed approach.

5.1. Experiment setup

The experiments are performed on a Windows 7 PC with 3.20 GHz CPU and 16GB memory. Our approach is implemented in Java 1.8. We have conducted a series on the Programmable Web dataset, which contains the functional information about real-word web services and SBSs crawled from programmableweb.com, a service portal that has been accumulating a variety of web services and SBSs since 2005.

5.1.1. Experiment series

Although there are tens of thousands of web services on PW, only a portion of them are really used in the existing SBSs. Thus, we only extract 1104 web services and 2429 SBSs composed of them to construct the PW dataset for experiments, because the composability of other services that have not been used in SBSs so far are unknown and they will be isolated nodes in the service connection graph. The web services in PW dataset are divided into 118 categories according to their functionality. Since the PW dataset does not contain reliability and throughput, we use the qualities of Quality of Web Service (QWS)[2] Al-Masri and Mahmoud (2007a,b) to randomly assign each web service in PW dataset with the reliability and throughput.

In this experiment series, a scgraph is built for the PW dataset. There are a total number of 1104 nodes and 7024 edges in the scgraph. If two web services coexist in a same SBS, we assume they can be composed with each other and the corresponding nodes are linked bidirectionally in the scgraph.

[2] http://www.uoguelph.ca/~qmahmoud/qws/.

5.1.2. Query series

There are two concerns about choosing testing queries. Firstly, the queries should be meaningful for human but not randomly generated. To ensure that, we manually generate a set of candidate queries from many SBSs in the PW dataset, each of which is comprised of the representative keywords obtained from the description of each service in an SBS. Thus, the queries can match at least a real SBS. Secondly, the queries should have enough diverse results of search on the service connection graph, or it makes no sense to diversify their results. According to that, we choose the final 20 queries from the candidate queries by observation on their search results. Thus, these queries can support the following evaluation on diversity well. For example, an SBS in the PW dataset includes four services: *Amazon Product Advertising, YouTube, Yahoo Answers* and *Flickr*. The query generated based on this SBS contains four keywords: “advertising”, “video”, “yahoo answers”, “flickr” chosen from the service description. These keywords generalize the functionality or feature of the corresponding services, and many diverse compositions can be found by search them. Table 2 presents these 20 queries.

5.1.3. Compared algorithm

Table 3 refers to the algorithms we compare with each other. We compare ATD and PAD both based on JSC and CSTD. In addition, we compare our diversification method with the method that uses expansion ratio to diversify the compositions respectively based on JSC and CSTD. We add parameter α as a threshold to filter out the results with very low objective values for ATD and EXP, for reducing the execution time. Moreover, k and θ mean the size of returned results and dissimilarity respectively. We test the algorithms with varying values of these three parameters in our experiments. If there is no special statement in the following part, the default value of α, k and θ is 4, 3 and 0.3 respectively. α and θ are application-specific, so that they are evaluated empirically based on observation of the datasets. For k, if its value is too small we will have too few results to observe the diversity, and if its value is too large it may lead to a very long execution time, because there are not enough results dissimilar to each other in the dataset.

5.2. Effectiveness evaluation

In this paper, we just focus on the comparison of diversity not precision and recall. Due to the same effectiveness of *all-then-diversify* and *pop-and-diversify* that can be proved in Zhong et al. (2018), we just compare ATD and EXP based on JSC or CSTD.

We employ the metric proposed in Mei et al. (2010) to measure diversity. The reason is, the lower density of the induced subgraph, the less pairs of result trees that are too similar, and the more diverse the overall results. Ideally, the isolated results are the best results. The metric makes use of density of the induced sub-graph. The density of a graph is defined as the number of edges presenting in the graph divided by the maximal possible number of edges in the graph. Given a sub-graph S, the density is as followed:

$$Density(S) = \frac{|\{(u,v)|u \in S, v \in S, (u,v) \in E\}|}{|S| \cdot (|S|-1)} \tag{10}$$

In addition, to evaluate the redundancy of returned results, we use the following equation:

$$Redundancy(R) = \frac{\sum_{T \in R} |C(T)| - |\bigcup_{T \in R} C(T)|}{\sum_{T \in R} |C(T)|} \tag{11}$$

where R is the set of returned results and $C(T)$ is the set of categories of services in T.

We conduct experiments to study the performance of our approach with the diversity metric and make comparison with its competitors.

Table 2
20 queries composed by human users.

No.	Name of SBS	Keywords of query	Matched services
1	UK Beaches	region localization, map data, publish activities	5
2	Flash Earth	localization, JavaScript library, map control	6
3	Newsonmap.in	unstructured, localization	9
4	Moviegram	movies, trailers	10
5	Keep Up With Jones	map control, social, home	19
6	Map of the Dead	venue, interactive maps	29
7	PhoneDuty	SMS, alerting	33
8	qabic	SMS, interactive polling, payments	45
9	Junk Depot	localization, crawls, recommend payment	51
10	Happenic	event, social graph, localization	55
11	Usermeds	photo, advice	64
12	vizlingo	videos, social connections, Twitter	72
13	Find Best Three	photos, Amazon Product, videos, advice	80
14	Reptiles Now	ads, sources, videos, language	81
15	Gold Investment	precious metals,product	89
16	eBay Motors	product, localization	90
17	immipad	share, social network, events, ads	105
18	ShipStation	product, shipping, payment, online stores	129
19	Music Artist Cloud	music, videos, product	159
20	The Campus Atlas	localization, Product, ads, social	179

Table 3
The algorithms of experiments.

Abbreviation	Description
ATD-JSC	All-Then-Diversify algorithm based on JSC
ATD-CSTD	All-Then-Diversify algorithm based on CSTD
EXP-JSC	maximal expansion ratio algorithm based on JSC
EXP-CSTD	maximal expansion ratio algorithm based on CSTD
PAD-JSC	Pop-And-Diversify algorithm based on JSC
PAD-CSTD	Pop-And-Diversify algorithm based on CSTD

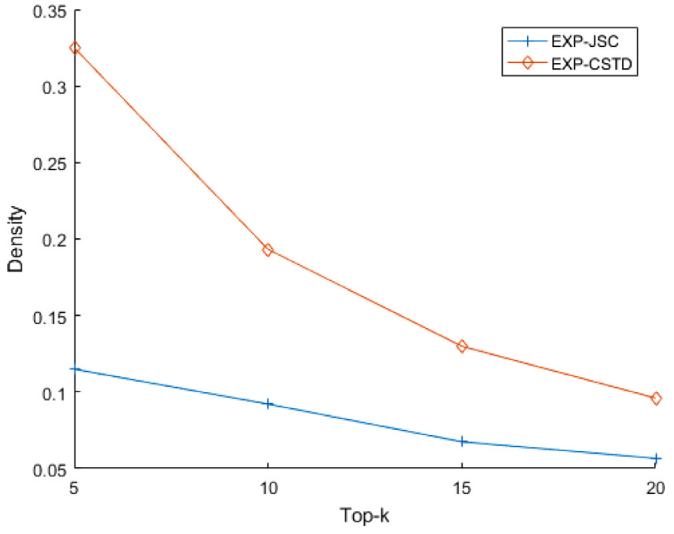


Fig. 6. Comparison based on density.

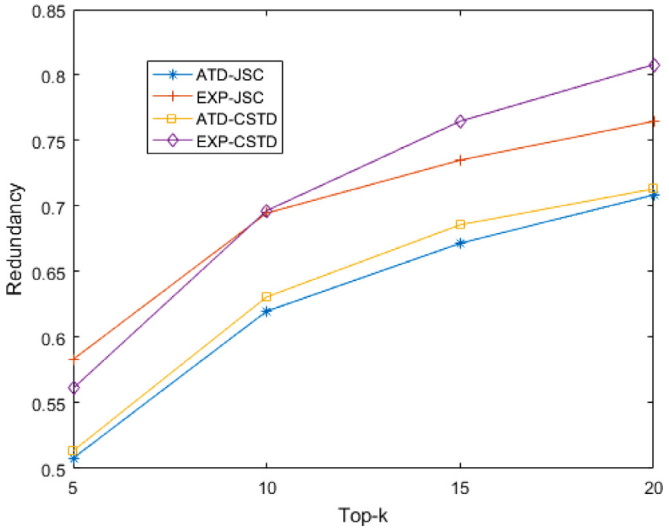


Fig. 7. Comparison based on redundancy.

As shown in Fig. 6 which omits ATD-JSC and ATD-CSTD, although the density of EXP decreases as k increases, our approach always achieves 0 for density since our approach is to find the maximal independent set which contains no edges between results. Thus, we can conclude that our approach must outperform EXP in terms of density.

As shown in Fig. 7, the redundancy of ATD-JSC and ATD-CSTD is lower than that of EXP-JSC and EXP-CSTD. Compared with the method using expansion ratio, our method ensures that the worst diversity between compositions is more than a threshold. In this way, our method can apparently reduce the redundancy. Therefore, we can conclude that EXP-JSC and EXP-CSTD algorithms exhibit poor performance in redundancy evaluation.

Besides, combined the two figures, we find that JSC outperforms CSTD in general no matter based on ATD or EXP. Because JSC is based on manually identified categories, which is more precise than CSTD based on latent topic extracted from descriptions.

For example, to show the effectiveness of our approach, we conducted top-3 experiments. We select Q ={photo, advice} extracted from SBS *Usermeds* which is consisted of "Flickr" and "Yahoo Answers" as an example based on JSC which is more effective. Table 4 shows the compositions of services related to Q and omits the structure of compositions.

From Table 4, we can see all approaches can find the origin composition, i.e. "Flickr" and "Yahoo Answers". However, with no diversification, all results are of the same categories, i.e. "Photos" and "Q&A". When using EXP-JSC algorithm, the redundancy apparently decreases and there appears a composition of new category, i.e. "Social", but there still are the repeated services of the same category, i.e. "Google Picasa" and "Flickr". When using ATD-JSC, we find the web service composition of category are totally different

Table 4
Diversified top-3 results of Q ={photo, advice}.

	No diversification		EXP-JSC		ATD-JSC	
	services	categories	services	categories	services	categories
top-1	Flickr, Yahoo Answers	Photos, Q&A	Flickr, Yahoo Answers	Photos, Q&A	Flickr, Yahoo Answers	Photos, Q&A
top-2	Google Picasa, Yahoo Answers	Photos, Q&A	Google Picasa, Yahoo Answers	Photos, Q&A	Facebook, Yahoo Answers	Social, Q&A
top-3	Panoramio, Yahoo Answers	Photos, Q&A	Facebook, Yahoo Answers	Social, Q&A	FriendFeed, Yahoo Answers	Other, Q&A
Redun-dancy	66.7%		50%		33.3%	

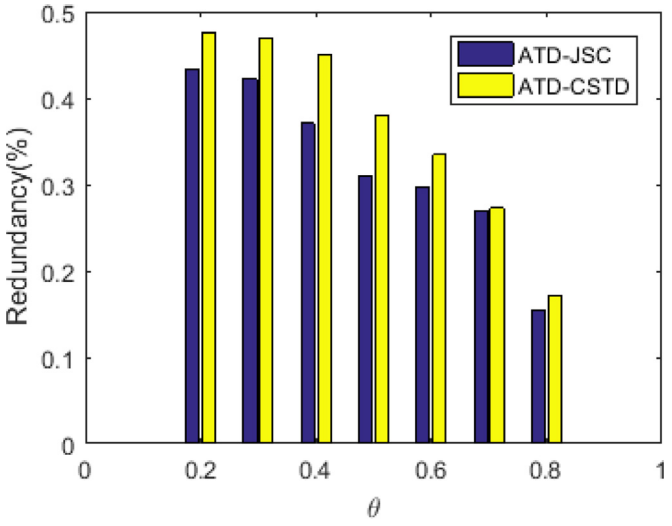


Fig. 8. Impact of θ the on redundancy.

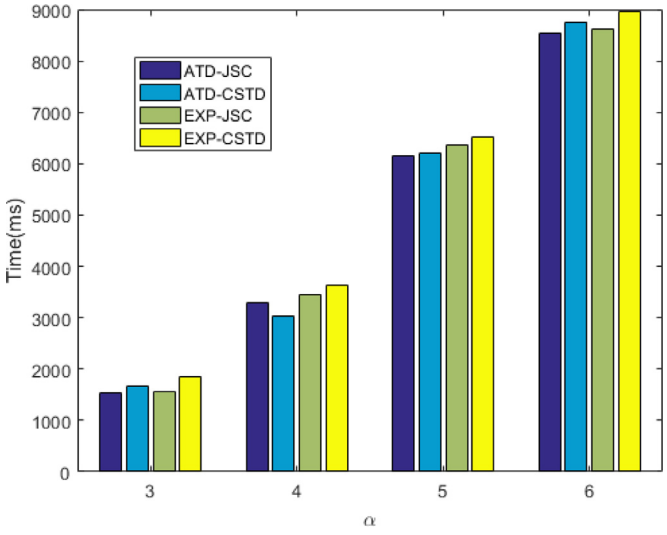


Fig. 9. Impact of α on the time costs.

from each other and there exists an extra composition of "Other". Therefore, our approach can achieve better diversity.

Next, we conduct experiments to study the effects of parameters in our approach. Specially, we study the impacts of θ for ATD and keep other parameters unchanged.

From Fig. 8, we can see that the redundancy decreases as θ increases, and ATD-JSC performs better than ATD-CSTD. Obviously, the higher the diversity threshold, the higher the dissimilarity between the results, the fewer the repeated categories, and the lesser the redundancy. From the observation, we can conclude that large θ and JSC causes low redundancy, thus, better diversity.

Overall, our approach outperforms the EXP approach in both redundancy and density compare. From the example, we can directly see that our approach achieves better diversity. Last but not least, our approach will get better diversity by increasing diversity threshold θ.

5.3. Efficiency evaluation

Fig. 9 shows the impact of bound α on the time costs. We can see that larger α causes longer time. As ATD and EXP all needs to first generate candidate result trees, the influence of α is apparent. To reduce the time costs of ATD and EXP, α should be set low. However, if we set α too low, the candidate result trees will be inadequate to diversify. Thus, PAD's role is even more important.

Fig. 10 compares the ATD algorithm with the PAD algorithm for the time costs.

As shown in Fig. 10, PAD outperforms ATD, especially when k is small. Thus, we can conclude PAD can apparently reduce the time costs in the dataset. What is more, we can find that the time costs increases when k become larger.

From Fig. 10, we also know that similarity metric JSC outperforms CSTD. Because compared with JSC's area indexing, CSTD's area indexing is insufficient since the extracted description's high

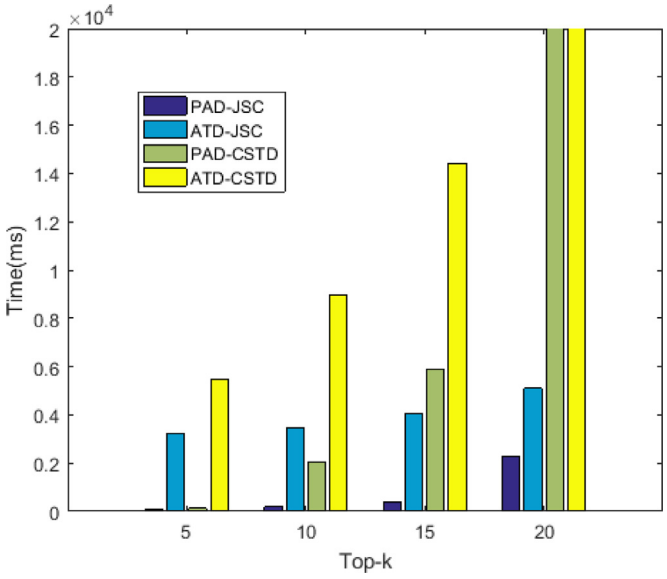


Fig. 10. Comparison of the ATD and PAD algorithms.

repetition. CSTD's lower area indexing causes stronger connectivity in similarity graph, which results in the MISs being insufficient to reach k. Thus, the algorithm needs to continually add candidate result trees to the similarity graph and find MISs until the MIS size is k. We know that larger similarity graph causes longer time to find MISs. Thus, CSTD always costs more time, especially when k is large.

From Fig. 11, we can see that the time costs of ATD almost keep unchanged when θ is low. Because lower dissimilarity threshold θ causes less edges,i.e. weaker connectivity, in similarity graph,and it

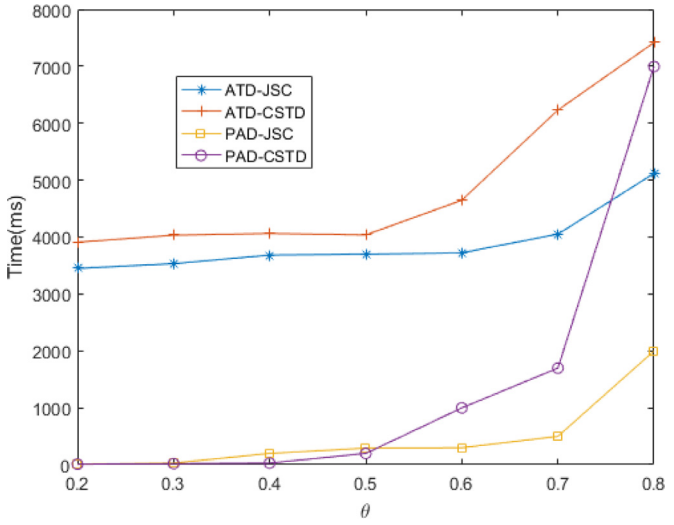


Fig. 11. Impact of θ on the time costs.

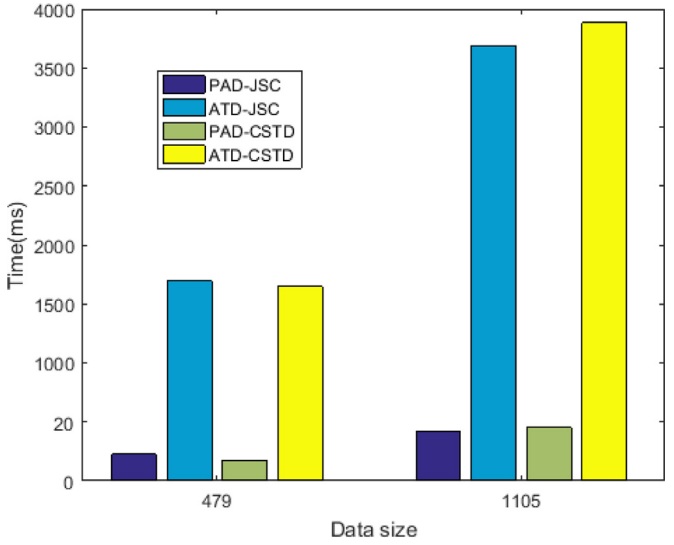


Fig. 12. Impact of data size on the time costs.

is easier to find an MIS of which the size is greater than or equal to k. This results in early stop of the diversification. Thus, low θ causes little time for diversification. While θ is large, the time for diversification booms. Thus, the total time increases rapidly when θ is large. What is more, we find that no matter how θ changes, PAD always performs better than ATD, especially, when θ remains low. Because when θ remains low, PAD has no use for generating many candidate results to diversify. While θ is large, PAD has to generate adequate result trees that approach the quantity of ATD's result trees in order to find the large enough MIS, which means more time on generation. In addition, Fig. 11 shows that JSC performs better than CSTD due to the insufficiency of CSTD's area indexing once again.

In order to evaluate the efficiency of different data size, we extract 479 nodes that matched by queries from PW dataset and maintain 2297 edges between them. We conduct the experiments both on the extracted data and the origin data. As shown in Fig. 12, larger data size causes apparent time increase of ATD while PAD always costs little time.

To evaluate out approach on large graphs, we randomly generate different large graphs. We divide the queries into two categories, namely, one-hop queries and two hop queries. One-hop queries means the returned results can be achieved by combining the one-hop search paths while two-hop queries means two-hop search paths. From Fig. 13, we can find that when the ratio of nodes and edges maintains, time costs increase while nodes and edges increase and two-hop queries changes apparently. From Fig. 13, we can find that when the ratio of nodes and edges increases, time costs of two-hop queries increase apparently while one-hop queries have no change.

Overall, larger α causes longer time for ATD and EXP. More returned results and better diversity leads to longer time both for ATD and PAD. The impact of data size is apparent to ATD. Compared with CSTD, JSC have more advantages on time costs and area indexing. In general, PAD-JSC is a better algorithm to diversify the service compositions.

6. Related work

Recently, a number of research works leverage the graph for service composition. Different from the traditional service composition approaches, these works aim to find a bunch of services or micro-services (and their composition relationships) that can be used to compose a new service-based system, but not an instantly runnable system composed of services directly. For example, Rodriguez-Mier and et al. Rodriguez-Mier et al. (2016) define a composition framework by means of integration with fine-grained I/O service discovery that enables the generation of a graph-based composition which contains the set of services that are semantically relevant for an input-output request. He et al. (2017) propose Keyword Search for Service-based systems (abbr. KS3) that integrates and automates the three key tasks of service composition and offers a new paradigm for SBS engineering that can significantly save the time and effort during the system engineering process. To shorten the compositions, Diac (2017) propose a heuristic score that promotes the most relevant services when multiple are accessible. However, they can only find the optimal service composition with respect to QoS. Due to the inherently ambiguity of keywords, there should be a diverse set of service compositions to satisfy the different requirements of system engineers.

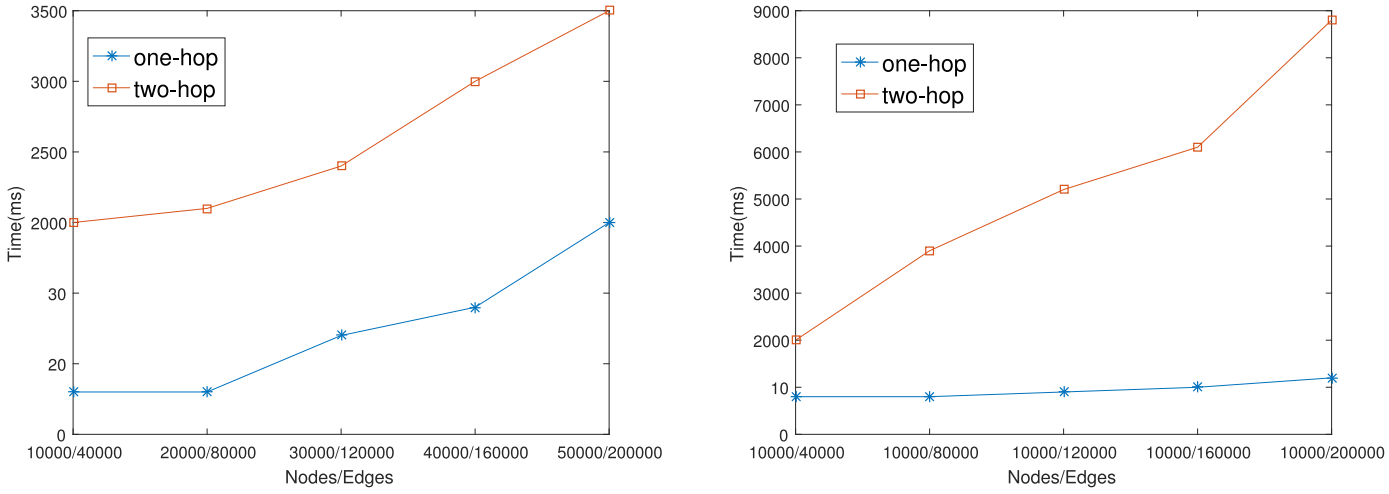
So far diversity has been considered in service computing. Nam et al. (2016b) propose an approach to measure semantic divergence of service communities. To improve the robustness of system, Wagner et al. (2016) consider functional and location diversity. To analyze the inherently ambiguity of keywords, Kang et al. (2016) explore service usage history to diversify web service recommendation results with the scores of relevance between historical search and potential interest in the web service graph. Then a service ranking algorithm (Aznag et al., 2013b; 2013a; 2014) for diversifying web services discovery results are proposed in order to minimize the redundancy in the search results. However, their attention is focused on service discovery diversity, not service composition diversity.

Although Nam et al. (2016a) propose a probabilistic approach for diversifying web services discovery and composition, their diversity of composition is the scalability of compositions. In contrast to the above research works, we consider the service composition diversification in the scenario of using keywords to find compositions in a service connection graph. Our approach can return the top-k compositions in terms of QoS that are dissimilar to each other.

7. Limitations and threats to validity

7.1. Limitations

There are some limitations in our approach. Firstly, as a keyword search based approach, our approach is not an end-to-end



(a) Fixed node#/edge#. (b) Varying node#/edge#.

Fig. 13. Time costs on large graph.

approach and only returns compositions that loosely connect services matched by keywords. Such compositions are not the final runnable systems and even may not have the exact structures required by users. Thus, we actually need to perform a few of post-processing tasks to deal with the returned compositions. The services in a composition would be manually picked and assembled to a real system by system engineers. For example, a system engineer may replace a service like "Google Map" in a composition by further initiating a post-processing task like finding alternative services of "Google Map". We do not discuss the post-processing detailedly in this paper since it is quite trivial and not the focus.

Another limitation is that, our approach may exclude some potential compositions with high QoS from the top-k list. Because the diversification requires each pair of returned compositions are dissimilar to each other, so that some compositions with high QoS may be abandoned. In some cases, e.g., the users see the QoS as the highest priority, it could violate the original requirements of service composition. A possible solution is to adjust the threshold of dissimilarity between compositions. However, it is usually very difficult to evaluate the threshold.

7.2. Threats to validity

There are a number of internal and external validity threats in our evaluation that should be discussed.

A major threat to the internal validity is the lack of ground truth of how users are satisfied by diversification. Due to that, the precision and recall, which are widely used in the evaluation of information retrieval research, are however not adopted in this paper. It is a common problem for diversification research. To mitigate this threat, we evaluated the effectiveness by using the indirect metrics like density and redundancy. Moreover, the small number of test queries is another threat to internal validity. For effectiveness evaluation, we simulated 20 queries from the existing SBSs in the PW dataset, as shown in Table 2. We cannot batch the query generation by means like choosing random keywords, since the queries should reflect the real user requirements. As a result, the number of queries is relatively limited.

One external validity threat is the representativeness of our service connection graph. We generated the graph from ProgrammableWeb.com, a well known dataset widely used in service selection and composition. Although the services are all real, the composibility between them are actually unclear, since we do not know the internal structures of SBSs. Therefore, we built a connection between two services if they appear in at least one same SBS, though not all services in an SBS are certainly composable. It could lead to that many nonexistent connections are introduced to the graph. As a result, the graph in our experiments might be too denser and thereby not the exact representative of the real-world service connection graphs.

8. Conclusion

In this paper, we study how to extend the emerging keyword search based service composition approach to satisfy different user requirements and overcome the problem of explosive combinations of composition candidates, which may incur unnecessary overheads of diversification. To address the problem, we propose a novel approach that aims to derive the top-k diverse compositions with respect to their QoS. Based on the All-Then-Diversify algorithm that enumerates all possibly massive results and derives the diverse subsets, our Pop-And-Diversify algorithm maintains a similarity graph of only the top results that have been found so far during the search, and computes the maximal independent sets of the similarity graph incrementally with its gradual growth until a set of k results have been found. From the experimental results on a real world web service portal, we observe that our Pop-And-Diversify algorithm is both effective and efficient.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Huanyu Cheng: Methodology, Software, Formal analysis, Investigation, Writing - original draft. **Ming Zhong:** Conceptualization, Methodology, Formal analysis, Writing - original draft, Writing - review & editing, Supervision. **Jian Wang:** Resources, Writing - review & editing.

Acknowledgements

This work was supported by the National Key Research and Development Program of China (no. 2017YFB1400602), National Nat-

ural Science Foundation of China (no. 61202036) and Natural Science Foundation of Hubei Province (no. 2018CFB616).

References

Agrawal, R., Gollapudi, S., Halverson, A., Ieong, S., 2009. Diversifying search results. In: International Conference on Web Search and Data Mining (WSDM), pp. 5–14.

Al-Masri, E., Mahmoud, Q.H., 2007. Discovering the best web service. In: International Conference on World Wide Web, pp. 1257–1258.

Al-Masri, E., Mahmoud, Q.H., 2007. Qos-based discovery and ranking of web services. In: International Conference on Computer Communications and Networks, pp. 529–534.

Alho, P., Mattila, J., 2015. Service-oriented approach to fault tolerance in cpss. J. Syst. Softw. 105, 1–17.

Aznag, M., Quafafou, M., Jarir, Z., 2013. Correlated topic model for web services ranking. Int. J. Adv. Comput. Sci. Appl. 4 (6), 283–291.

Aznag, M., Quafafou, M., Jarir, Z., 2014. Leveraging formal concept analysis with topic correlation for service clustering and discovery. In: IEEE International Conference on Web Services, pp. 153–160.

Aznag, M., Quafafou, M., Rochd, E.M., Jarir, Z., 2013. Probabilistic topic models for web services clustering and discovery. In: Service-oriented and Cloud Computing, pp. 19–33.

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S., 2002. Keyword searching and browsing in databases using banks. In: International Conference On Data Engineering, pp. 431–440.

Calinescu, R.C., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G., 2011. Dynamic qos management and optimization in service-based systems. IEEE Trans. Softw. Eng. 37 (3), 387–409.

Capannini, G., Nardini, F.M., Perego, R., Silvestri, F., 2011. Efficient diversification of web search results. Proc. Vldb Endowment 4 (7), 451–459.

Demidova, E., Fankhauser, P., Zhou, X., Nejdl, W., 2010. Divq : diversification for keyword search over structured databases. In: International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 331–338.

Diac, P., 2017. Warp: Efficient automatic web service composition. In: 2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 284–285.

Dojchinovski, M., Kuchar, J., Vitvar, T., Zaremba, M., 2012. Personalised graph-based selection of web apis. In: International Conference on the Semantic Web, pp. 34–48.

Feng, Z., Lan, B., Zhang, Z., Chen, S., 2014. A study of semantic web services network. Comput. J. 58 (6), 1293–1305.

Fraternali, P., Martinenghi, D., Tagliasacchi, M., 2012. Top-k bounded diversification. In: ACM SIGMOD International Conference On Management of Data, pp. 421–432.

Golenberg, K., Kimelfeld, B., Sagiv, Y., 2008. Keyword proximity search in complex data graphs. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 927–940.

He, H., Wang, H., Yang, J., Yu, P.S., 2007. Blinks: ranked keyword searches on graphs. In: ACM SIGMOD International Conference On Management of Data, pp. 316–356.

He, Q., Xie, X., Wang, Y., Ye, D., Chen, F., Jin, H., Yang, Y., 2017. Localizing runtime anomalies in service-oriented systems. IEEE Trans. Serv. Comput. 73–80.

He, Q., Yan, J., Jin, H., Yang, Y., 2014. Quality-aware service selection for service-based systems based on iterative multi-attribute combinatorial auction. IEEE Trans. Softw. Eng. 40 (2), 192–215.

He, Q., Zhou, R., Zhang, X., Wang, Y., Yang, Y., 2017. Keyword search for building service-based systems. IEEE Trans. Softw. Eng. 43 (7), 658–674.

Hoffmann, J., Bertoli, P., Pistore, M., 2007. Web service composition as planning, revisited: in between background theories and initial state uncertainty. In: Twenty-Second AAAI Conference on Artificial Intelligence, pp. 1013–1018.

Huang, K., Fan, Y., Wei, T., 2012. An empirical study of programmable web: a network analysis on a service-mashup system. In: IEEE International Conference on Web Services, pp. 552–559.

Hugo Haas, A. B., 2004. Web services glossary. https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice.

Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H., 2005. Bidirectional expansion for keyword search on graph databases. Very Large Data Bases 505–516.

Kang, G., Tang, M., Liu, J., Liu, X.F., Cao, B., 2016. Diversifying web service recommendation results via exploring service usage history. IEEE Trans. Serv. Comput. 9 (4), 566–579.

Kil, H., Oh, S.C., Elmacioglu, E., Nam, W., Lee, D., 2009. Graph theoretic topological analysis of web service networks. World Wide Web-internet Web Inf.Syst. 12 (3), 321–343.

Klusch, M., Fries, B., Sycara, K., 2009. Owls-mx: a hybrid semantic web service matchmaker for owl-s services. Web Semantics Sci. Serv. AgentsWorld Wide Web 7 (2), 121–133.

Kster, U., Knig-Ries, B., Stern, M., Klein, M., 2007. Diane: an integrated approach to automated service discovery, matchmaking and composition. In: Proceedings of the 16th international conference on World Wide Web, pp. 1033–1042.

Lyu, S., Liu, J., Tang, M., Kang, G., Duan, Y., 2014. Three-level views of the web service network: an empirical study based on programmableweb. In: IEEE International Congress on Big Data, pp. 374–381.

Mei, Q., Guo, J., Radev, D.R., 2010. Divrank: the interplay of prestige and diversity in information networks. In: International Conference on Knowledge Discovery and Data Mining, pp. 1009–1018.

Nam, H., Aznag, M., Quafafou, M., Durand, N., 2016. Probabilistic approach for diversifying web services discovery and composition. In: IEEE International Conference on Web Services, pp. 73–80.

Nam, H., Aznag, M., Quafafou, M., Durand, N., 2016. Semantic divergence based evaluation of web service communities. In: the 13th IEEE International Conference on Services Computing, pp. 21–28.

Ni, Y., Fan, Y., Wei, T., Huang, K., Jing, B., 2016. Ncsr: negative-connection-aware service recommendation for large sparse service network. IEEE Trans. Autom. Sci.Eng. 13 (2), 579–590.

Pistore, M., Marconi, A., Bertoli, P., Traverso, P., 2005. Automated composition of web services by planning at the knowledge level. In: IJCAI'05 Proceedings of the 19th international joint conference on Artificial intelligence, pp. 1252–1259.

Psiuk, M., Zielinski, K., 2015. Goal-driven adaptive monitoring of soa systems. J. Syst. Softw. 110, 101–121.

Rodriguez-Mier, P., Pedrinaci, C., Lama, M., Mucientes, M., 2016. An integrated semantic web service discovery and composition framework. IEEE Trans. Serv. Comput. 9 (4), 537–550.

Vee, E., Srivastava, U., Shanmugasundaram, J., Bhat, P., Yahia, S.A., 2008. Efficient computation of diverse query results. In: International Conference on Data Engineering, pp. 228–236.

Wagner, F., Ishikawa, F., Honiden, S., 2016. Robust service compositions with functional and location diversity. IEEE Trans. Serv. Comput. 9 (2), 277–290.

Wang, Y., Qiang, H., Zhang, X., Ye, D., Yun, Y., 2017. Efficient qos-aware service recommendation for multi-tenant service-based systems in cloud. IEEE Trans. Serv. Comput. 40 (5), 461–482.

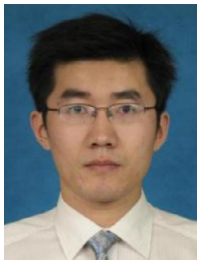
Yu, C., Lakshmanan, L.V.S., Amer-Yahia, S., 2009. It takes variety to make a world: diversification in recommender systems. In: International Conference on Extending Database Technology, pp. 368–378.

Yu, C., Lakshmanan, L.V.S., Amer-Yahia, S., 2009. It takes variety to make a world: diversification in recommender systems. In: Proceedings of the 12th International Conference on Extending Database Technology, pp. 368–378.

Zhong, M., Wang, Y., Zhu, Y., 2018. Coverage-oriented diversification of keyword search results on graphs. In: Database Systems for Advanced Applications, pp. 166–183.

Huanyu Cheng received the B.S. degree in 2017 from Wuhan University, China. He is now a M.S. candidate in School of Computer Science, Wuhan University, China. His current research interests include diversity and services computing.

Ming Zhong received the Ph.D. degree in computer software and theory from Wuhan University, China. He is an associate professor in School of Computer Science, Wuhan University. His current research interests include Big data management and analysis, Graph data management and analysis, Social network analysis, etc.



Jian Wang received the Ph.D. degree in 2008 from Wuhan University, China. He is now a lecturer in School of Computer Science, Wuhan University, China. His current research interests include software engineering and services computing.