Automatically Learning User Preferences for Personalized Service Composition

Yu Zhao, Shaohua Wang, Ying Zou
Queen's University, Kingston, Ontario, Canada
yu.zhao@queensu.ca, shaohua@cs.queensu.ca, ying.zou@queensu.ca

Joanna Ng, Tinny Ng
IBM CAS Research, Markham, Canada
{jwng, tng}@ca.ibm.com

***Abstract*—With the rapid growth of the web services technologies, users often leverage various web services to perform their daily activities, such as on-line shopping. Due to the massive amount of web services available, a user faces numerous choices to meet their personal preferences when selecting the desired services from the web services with the similar functionality. Therefore, it becomes tedious and cumbersome tasks for users to discover and compose services. To reduce user's cognitive burden, it is critical to support automated service composition and make efficient recommendation of personalized services to achieve user's overall goals. However, existing approaches only offer users with limited options designed for the interest of a group of users without considering individual users' interests.**

To allow users to compose personalized services without much manual specification, we propose a machine learning approach that applies a learning-to-rank algorithm, RankBoost, to automatically learn user preferences and the prioritization of the preferences from users' historical data. Moreover, our approach uses the multi-objective reinforcement learning (MORL) algorithm to make trade-offs among user preferences and recommends a collection of services to achieve the highest objective. We conduct an empirical study to evaluate our approach by collecting the historical data from 12 subjects. The results demonstrate that our approach outperforms the two well-established baseline approaches by *100%-200%* in terms of precision on recommending services.

I. Introduction

The massive amount of web services enables users to conduct various daily activities, such as on-line shopping. Generally speaking, a single service can not satisfy a user's goal. Users often need to visit multiple web services to fulfill their goals. For instance, to buy a pair of boots, a user may browse various E-commerce websites to compare the products and check user reviews of the products. Essentially, the user implicitly composes services for purchasing boots. However, a user faces numerous choices when selecting services. Various service information, such as price and user rating, can affect users' decisions of selecting services for achieving users' goals. Therefore, it becomes tedious and cumbersome tasks for users to discover and compose services. To reduce user's cognitive burden, it is critical to support automated service composition by efficiently recommending personalized services to achieve user's overall goals. However, in the current practices, users face the following challenges:

- *Limited and rigid options available for users to specify the personalized preferences.* Many service providers allow users to enter searching requirements for services to meet user preferences. For example, a user can specify the price range and brand names to search for boots on the Amazon[1] website. However, the number of predefined options is limited and such options are designed to address the interests of a group of users. The predefined options can not be customized for the individuals' needs.
- *Conflicting preferences.* A user might have several preferences in selecting services. However, such preferences can be contradictory. For instance, users who have a low budget for shopping for clothes may prefer boots with a lower price but a high user rating. However, a low price service may likely receive a low user rating. Therefore, a user needs to make trade-offs among the conflicting preferences before choosing a service.
- *No prioritization of various preferences.* A user can have a priority order among several preferences. For example, a user may prefer high user ratings than low prices. However, the priority order of preferences is not taken into account in the existing approaches [13][28] for the service recommendation.

To address the aforementioned challenges, the multi-objective reinforcement learning algorithm (MORL) [20] is used to solve the conflicting preferences in service composition [13][23][24]. Existing approaches, such as [23][24], use the MORL to model the user preferences for services as numeric weight values, and require users to manually assign the weight values. However, users can be reluctant to enter any additional information, since the meaning of the weight values is opaque for them.

In this paper, we propose an approach for personalized service composition. Our approach does not require users to manually specify preferences. Instead, user preferences and the priority orders of the preferences are automatically inferred from past service selection history. We automatically generate weight values for the MORL algorithm to compose services based on the identified user preferences. As a consequence, our approach can derive personalized composite services with minimal effort from users. In particular, our approach applies a machine learning algorithm, *i.e.*, a learning-to-rank algorithm, namely RankBoost [3], in order to identify the user preferences. We extract various learning features from service and

[1]https://www.amazon.com

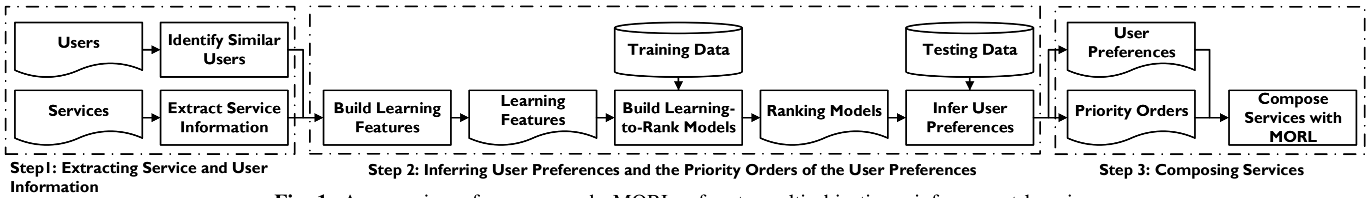

Users
Identify Similar Users
Services
Extract Service Information
Training Data
Testing Data
Build Learning Features
Learning Features
Build Learning-to-Rank Models
Ranking Models
Infer User Preferences
User Preferences
Priority Orders
Compose Services with MORL
Step1: Extracting Service and User Information
Step 2: Inferring User Preferences and the Priority Orders of the User Preferences
Step 3: Composing Services

Fig. 1: An overview of our approach. MORL refers to multi-objective reinforcement learning.

user information to build ranking models that can infer the prioritization of user preferences. The preference learning is conducted offline to ensure the minimal time delay for the real-time service recommendation. We conduct case studies on real user dataset. Comparing with the two well-established MORL baselines, our empirical results show that our approach can outperform the MORL baselines by 100% to 200% in terms of precision.

Paper Organization. Section 2 gives an overview of our approach. Section 3 presents the case studies. Section 4 describes the related literature. Section 5 concludes the paper.

II. Overview of Our Approach

To save users from manually specifying their preferences to compose services, our approach automatically infers user preferences utilizing a learning-to-rank (LtR) algorithm, Rank-Boost. Figure 1 gives an overview of our approach. Our approach consists of three main steps: 1) we extract service and user information from users' service composition history; 2) We identify learning features from the extracted service and user information to build ranking models. We use the built ranking models to infer user preferences; and 3) We apply the MORL algorithm to optimize the service composition by incorporating the identified user preferences into the service recommendation process. In the following subsections, we discuss each step in more details.

A. Extracting Service and User Information

We extract service and user information to build learning features to create a ranking model for various user preferences. A user can perform a small amount of services. The data related to services selected can be limited. The sparse data can make it hard to infer user preferences. Moreover, similar users sharing the same interests with a user can have the same preferences [9][19]. In our approach, we leverage the similar users' service selection history to infer the user's preferences. We examine the following information from various users.

1) Service Information: includes task description, service description, service context and user choices.

- **Task Description** states the intent of a user's activity. Usually, a web service is associated with the textual description of a task (*e.g.*, buy men's chukka boots). The task description can be provided by a user when a user specifies search criteria to describe the intents. Otherwise, it can be retrieved from the task description as specified in the business process definitions.
- **Service Description** specifies the functionality of a web service. It can be retrieved from the description of operations in Web Services Description Language (*i.e.*, WSDL) and web pages describing RESTful services or resources. Figure 2 shows an example of the associated task description and service description.

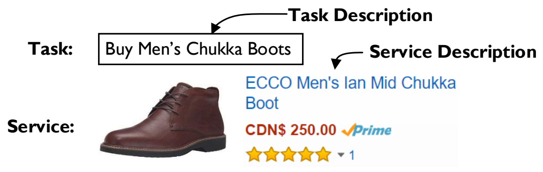


Fig. 2: An example of task description and service description.

- **Service Context** describes the circumstances that a service interacts with a user. The context includes *time* and *location*. Specifically, the time is when a user performs a task and can be obtained from the operating system. The location can be acquired from the IP address of the computer or the GPS in a mobile device.
- **User Choices** record the selection of services by a user. The value for a choice can be 0 or 1. 1 means that the user selects the web service and 0 denotes the service is not selected by the user. User choices can be mined from various sources, such as clicks on a URL link or invocation of a web service.

2) Identifying Similar Users: To utilize similar users' service selection history, we need to measure the similarity between users. Our approach identifies the user similarity by leveraging user's personal information, such as hobbies and education background. The user information can be extracted from the existing user profiles created in social networks (*e.g.*, Facebook[2]). We store user information in a key-value pair form. A key defines a label for user information (*e.g.*, hobby) and a value describes the contents (*e.g.*, swimming). We construct a bag of words (denoted as bag) for a user using the contents of the user information. We calculate the similarity of two users by comparing the differences of the corresponding two $bags$. A bag is normalized using the approach proposed by Zhao *et al.* [29]. We decompose words using special characters, such as "@", and capital cases if applicable. Suffixes containing numbers are removed. Non-English words are removed if the words are not contained in WordNet [11]. Moreover, we remove stop words (such as "a", "is" and "the") and perform word stemming ("reduced", "reducing" and "reduces" are normalized to reduce) [14]. Then the user similarity can be calculated using Jaccard index [10] between the two bags of words:

$$Sim(U_i, U_j) = \frac{|bag_i \cap bag_j|}{|bag_i \cup bag_j|} \tag{1}$$

where $|bag_i \cap bag_j|$ *denotes the number of common words in the two bags.* $|bag_i \cup bag_j|$ *states the number of words in the union sets of the two bags.*

[2]www.facebook.com

TABLE I: Twenty learning features. Learning features marked with * indicate unique features based on our dataset.

Category	Features	Description	Equation
Service Attributes	(F1) Price*	The price of using the service S_k	$Price(S_k) = price\ of\ using\ the\ service\ S_k$
	(F2) Discount*	The discount of using the service S_k	$(original\ price\ of\ using\ the\ service\ S_k)/Price(S_k)$
	(F3) User Rating*	The user rating of the service S_k	$user\ rating\ of\ the\ service\ S_k$
	(F4) Relevance*	To what extent that the service S_k is related to the task T_a	$returned\ page\ number\ of\ the\ service\ S_k\ for\ the\ task\ T_a$
Service Usage History	(F5) Service Usage Frequency	The frequency of the service S_k used by the user U_i in the past for T_a	$Freq(S_k, U_i) = \#\ of\ times\ S_k\ used\ for\ T_a\ by\ U_i$
	(F6) Service Usage Frequency by the Similar Users	The frequency of the service S_k used by n similar users of the user U_i in the past for T_a	$\sum_{j=1, j \neq i}^{n} Sim(U_i, U_j) \times Freq(S_k, U_j)$
	(F7) Service Composition Pattern*	The frequency of two services S_g and S_k occurring together in the past by the user U_i	$Pat(S_g, S_k, U_i) = \#\ of\ times\ S_g\ and\ S_k\ used\ together\ by\ U_i$
	(F8) Service Composition Pattern by the Similar Users*	The frequency of two services S_g and S_k occurring together in the past by n similar users of the user U_i	$\sum_{j=1, j \neq i}^{n} Sim(U_i, U_j) \times Pat(S_g, S_k, U_j)$
User Context	(F9) Usage of the Day	The frequency of the service S_k used by the user U_i in the past for T_a under the day D_t	$Day(D_t, S_k, U_i) = \#\ of\ times\ S_k\ used\ for\ T_a\ by\ U_i\ under\ D_t$
	(F10) Usage of the Hour	The frequency of the service S_k used by the user U_i in the past for T_a under the hour H_q	$Hour(H_q, S_k, U_i) = \#\ of\ times\ S_k\ used\ for\ T_a\ by\ U_i\ under\ H_q$
	(F11) Usage of the Place	The frequency of the service S_k used by the user U_i in the past for T_a under the place P_x	$Place(P, S_k, U_i) = \#\ of\ times\ S_k\ used\ for\ T_a\ by\ U_i\ under\ P_x$
	(F12) Usage of the Day by the Similar Users*	The frequency of the service S_k used by n similar users of the user U_i under the day D_t	$\sum_{j=1, j \neq i}^{n} Sim(U_i, U_j) \times Day(D_t, S_k, U_j)$
	(F13) Usage of the Hour by the Similar Users*	The frequency of the service S_k used by n similar users of the user U_i under the hour H_q	$\sum_{j=1, j \neq i}^{n} Sim(U_i, U_j) \times Hour(H_q, S_k, U_j)$
	(F14) Usage of the Place by the Similar Users	The frequency of the service S_k used by n similar users of the user U_i under the place P_x	$\sum_{j=1, j \neq i}^{n} Sim(U_i, U_j) \times Place(P_x, S_k, U_j)$
Textual Feature	(F15) Word Usage	The frequency of the words in the S_k used in the past by the user U_i	$Word\_Usage(S_k, U_i) = \sum_{p=1}^{m} \omega_{T_p}^{U_i}$; *where* $\omega_{T_p}^{U_i}$ *is the frequency of the word* T_p *used previously by the user* U_i.
	(F16) Average Word Usage	The average number of times that the words in the S_k used in the past by the user U_i	$Word\_Usage(S_k, U_i)/m_k$; *where* m_k *is the number of distinct words in the* Bag_S_k.
	(F17) Word Usage by the Similar Users	The frequency of the words in the S_k used in the past by n similar users of the user U_i	$\sum_{j=1, j \neq i}^{n} Sim(U_i, U_j) \times Word\_Usage(S_k, U_j)$
	(F18) Average Word Usage by the Similar Users	The average number of times that the words in the S_k used in the past by n similar users of the user U_i	$\frac{\sum_{j=1, j \neq i}^{n} Sim(U_i, U_j) \times Word\_Usage(S_k, U_j)}{m_k}$
	(F19) Task Similarity*	The textual similarity between the task description T_a and the service description S_k	$cos(Bag\_T_a, Bag\_S_k)$
	(F20) Brand*	Whether the service S_k is provided by brand-name corporations	$\begin{cases} 1 & \text{if the service description } S_k \text{ contains famous brand name} \\ 0 & \text{otherwise} \end{cases}$

B. Inferring User Preferences and the Priority Orders of the User Preferences

Users can have multiple preferences when they compose services (*e.g.*, price and user rating). The priority order of the preferences exists for users. To save users from manually specifying their preferences, our approach automatically infers user preferences and the priority orders.

1) Building Learning Features: We utilize a learning-to-rank (LtR) algorithm, named RankBoost [3], to build ranking models based on a set of learning features. A ranking model learns from different learning features to suggest an ideal ranking of services. Each learning feature describes a characteristic regarding a service and a user. We build the learning features from service and user information described in Section II-A. To build a ranking model, the LtR needs training and testing phases. The training dataset contains tasks and services. Given a task t^i, there are a number of services $S^i = \{s_1^i, s_2^i, ..., s_n^i\}$ associated with the task for performing the task, where s_j^i denotes the j^{th} service. A task-service pair (t^i, s_j^i) contains a relevance judgment representing if a user chooses the service s_j^i respect to the task t^i. We take the *user choices* for the services as the relevance judgment. A learning feature vector $\mathbf{X}_j^i$ is constructed for each task-service pair (t^i, s_j^i). $\mathbf{X}_j^i$ contains a set of learning features to describe the characteristics of the task-service pair. The training process aims to train the ranking model $f(\mathbf{X})$ that can assign a score to a given task and a respective service. The top ranked services have higher possibility that users choose.

We propose four categories of learning features as listed in Table I. We briefly discuss them as follows.

Service Attributes. Each web service has a set of attributes. Such attributes serve as indicators for users to select web services [28]. We can obtain service attributes from web pages of service providers and the runtime monitoring information [21]. Our approach identifies four types of service attributes: *price*, *user rating*, *discount* and *relevance* of a service to a task. The details of each learning feature are described in Table I.

Service Usage History. The history of users' service composition tracks user interests over time [19]. The past service usage is leveraged to recommend web services for users. The category of service usage history includes the following learning features:

- *Service Usage Frequency* and *Service Usage Frequency by the Similar Users.* Service frequency denotes how frequently a service is used in the past. The more frequently a service is used previously by a user U_i or by the similar users in the past, the higher chance that the service can be used again by the user U_i.
- *Service Composition Patterns* and *Service Composition Patterns by the Similar Users.* Service composition patterns capture re-occurrences of a set of services that are frequently composed together. The service composition patterns can be re-used by a user [16]. Moreover, similar users can have the similar or same composition behaviors.

User Contexts associated with the past selection of services can be used for ranking services [25]. For instance, the user could choose the hotel frequently booked for business trips for their own personal trips. Specifically, the user context includes the *usage of the day*, the *usage of the hour* and the *usage of the place*. The context of the usage by the similar users is also included, since the similar users can use services in the same context. We use the approach proposed by Lee *et al.* [6] to abstract numerical context values to categorical values. For example, the *hour* is categorized into four levels, *i.e.*, morning, noon, afternoon and evening.

Textual Feature is related to the textual description of web services and tasks. We leverage the textual description to recommend services for users. To calculate the textual feature, we normalize the task description of T_a and the service description of S_k into two bags of words, denoted as Bag_T_a and Bag_S_k, respectively. We use term frequency and inverse term frequency ($tf-idf$) [10] to weight each word in the two bags. The category has the following six features.

- *Word Usage* and *Average Word Usage*. Sugiyama *et al.* [19] find that adapting service search results based on the word usage can achieve better accuracy for retrieving services. Service description containing more frequently used words can attract more attention from users.
- *Word Usage by the Similar Users* and *the Average Word Usage by the Similar Users*. The learning features rank the importance of a service for a user based on the words used in the previously selected services of the similar users.
- *Task Similarity*. A service sharing the highest textual similarity value with a task T_a could be selected by users for the task. We use the cosine similarity algorithm [10] (denoted as cos) to calculate the similarity value of the two weighted bags of words, *i.e.*, Bag_T_a and Bag_S_k.
- *Brand.* Users can choose services with a well-known brand. The learning feature has a binary value. Table I describes the calculation of the learning feature.

2) Inferring User Preferences and the Priority Orders: The user preferences are the learning features (listed in Table I) that are important to users. We determine the importances of learning features by evaluating the impact of each learning feature on the performance of our ranking models. The rational is that the higher impact of a learning feature on the performance of the ranking model, the higher is the priority order of the corresponding user preference.

We build an original ranking model R_{all} including all of the four categories of learning features. Since learning features can be correlated and redundant, we conduct the Spearman's rank correlation test and the redundancy analysis [22] to remove the correlated and redundant learning features, respectively. We exclude one correlated learning feature in the original ranking model R_{all}, if the correlation value of two learning features is higher than 0.8 [30]. Learning features that can be predicted and expressed by other features are also excluded in the R_{all}. Our approach consists of the following two steps to identify user preferences and their priority orders.

[**Step 1**]. *Identifying user preferences.* As learning features in one category can share the same properties, we study the effect of each category of learning features on the performance of a model [15]. We build alternative ranking models to analyze the impact of learning features in a category on the ranking performance. An alternative ranking model is built including all of the learning features used in the original model R_{all} **but excluding the learning features in a category**. The rational is that the impact of a category of learning features on the ranking performance can be observed when the learning features in the category are excluded. The first alternative ranking model R_{-sa} considers all the learning features used in the original ranking model R_{all} except for the learning features in the category *service attributes* (*i.e.*, price, user rating, discount and relevance). In total, we generate four alternative ranking models R_{-sa}, R_{-suh}, R_{-uc} and R_{-tf} where they include all the learning features except for the learning features in the category *service attributes*, *service usage history*, *user context* and *textual feature*, respectively.

We compare the performance of an alternative ranking model with the performance of the original model R_{all} to obtain the performance disparity. If the performance of the alternative ranking model is worse than that of the original model R_{all}, and the performance disparity is larger than a threshold T, we consider that the learning features in the category are all important to the user. Since the performance disparity can be contributed by an individual learning feature, we can not determine the priority order of the learning features within a category. Therefore, we temporarily assign the lowest priority order to all of the learning features within an identified category. In the next step, we identify the priority order for individual learning features.

[**Step 2**]. *Identifying the priority order of user preferences.* We determine the priority order of user preferences by analyzing the impact of each learning feature on the ranking performance. We build an individual ranking model for a learning feature F^{Test} by including all of the learning features used in the original model R_{all} **but excluding the learning feature** F^{Test}. Then, we compare the performance of the learned individual ranking model with the performance of the original model R_{all} to obtain the performance disparity for determining the significance of the learning feature F^{Test}. The larger is the performance disparity, more important is the learning feature F^{Test}. To compare the effect of two learning features F^a and F^b, we compare their performance disparity. If the difference of their performance disparity is larger than the threshold T, it indicates that a user prefers F^a over F^b. We compare every two individual ranking models and update the priority order of the learned important learning features in **step 1**.

C. *Composing Services*

To derive personalized composite services for users, our approach composes services using multi-objective reinforcement learning algorithm (MORL), and integrates the inferred user preferences and the priority orders identified in Section II-B.

1) Multi-Objective Reinforcement Learning (MORL): The service composition process is a sequential decision-making process, which can be modeled as a Markov Decision Process (MDP) [13][24]. A MDP involves choosing multiple tasks and services. The output of the service composition MDP is a composite service that maps from each task to a web service. The MDP problem can be optimized by the MORL algorithm [20]. We use a reinforcement learning approach, Q-learning [20], to learn optimal composite services. For a task and a web service, Q-learning initializes an arbitrary Q-value, such as 0. Q-learning conducts an exploration and exploitation strategy to update the Q-value recursively, until a convergence point is achieved. The Q-value update function considers the current reward as well as the long-term reward. More details of the algorithm can be referred to [20].

The reward is calculated by a reward function. We use the weight summation method as the reward function: $r = \sum_{i=1}^{N} \omega_i \times r_i$, where ω_i and r_i are the weight value and learning feature value for the i^{th} preference, respectively. A higher weight value of a preference means that a user puts more emphasis on the corresponding preference. We use the well-known ϵ-greedy policy [20] to explore and exploit web services. Under a task, Q-learning chooses to perform the optimal web service with a (1-ϵ) probability and perform a random service with a probability of ϵ. When the learning process ends, the optimal web services for each task form the optimal composite services.

2) Composing Services with the Inferred User Preferences and the Priority Orders: Rather than learning all the possible optimal composite services as the existing approach, we aim to identify the subset of the composite services that maximize the overall user preferences.

Given a new service composition job, we build learning features for services as described in Section II-B1. To ensure the learning feature values in the same range, we use the approach proposed in [28] to scale the values for learning features to a range of 0 to 1. Our approach optimizes composite services with user preferences. The reward function computes the weighted sum of the important learning features. We determine the weight values for user preferences based on the priority order identified in Section II-B2. Given the priority order $O = \{O_1, O_2, ..., O_N\}$ (N is the number of important learning features), we randomly assign weight values based on the following constraints:

$$\begin{cases} \omega_1 + \omega_2 + ... + \omega_N = 1 \\ \omega_i > \omega_j & \text{if } O_i > O_j \end{cases} \tag{2}$$

where ω_i is the weight for the i^{th} preference, and O_i is the priority order for the i^{th} preference.

The constraints guarantee that we always assign a larger weight value for an important user preference if the learning feature has a higher priority order.

As users tend to make choices from multiple recommended results [17], it is desirable to derive multiple possible composite services for users. The multiple composite services are derived by performing the MORL algorithm multiple runs with different weight settings [8]. In each run, the weight values are unchanged.

III. Case Study

We conduct experiments to evaluate our personalization approach. In this section, we introduce our case study setup and experimental results.

A. *Case Study Setup*

Collecting Web Services. To test the effectiveness of our approach, we collect service and user information from the E-commerce domain in which users can perform their tasks on regular basis. We collect RESTful services for the products sold on Amazon[3] as merchants offer web services for selling products on Amazon. A RESTful service for selling a product usually contains product search functionality. We take the category name of products (*e.g.*, headset and men jacket) as a task name and all of the products sold under the category as the associated web services. All of the extracted learning features are generic and applicable to other types of web services. We mine the web pages of Amazon website to collect services.

Our mined services fall into different domains: Clothing, Shoes & Handbags, Office Products, Sports & Outdoors, and Electronics. In total, we extract 64,888 web services associated with 27 tasks. Table II describes the distribution of the services and the tasks in each domain. *Service descriptions* and *service attributes* (*i.e.*, price, discount, user rating, and relevance) are extracted for each web service. For each category of products, Amazon provides "Top Brands" web pages to list famous brand names. We mine famous brand names from Amazon to determine the *brand* learning feature proposed in Section II-B.

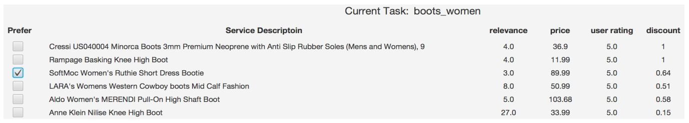
Collecting Historical Service Usage Data. To collect the user historical service usage dataset, we develop a tool for users to compose services. In our tool, a user enters various types of user information as described in Section II-A2. When a user submits a goal to describe the composed services in our tool, the user can choose a set of tasks from our extracted tasks in order to achieve the goal. For instance, to buy gifts, a user can select the tasks recommended by the tool: buying candy chocolates, buying activity trackers and buying fountain pens.

[3]https://www.amazon.com

TABLE II: The distribution of services and tasks in each domain. We also show the average number of tasks performed by the subjects in each domain.

Domain	# Tasks	# Services	Subjects Performed # Tasks
Clothing	10	34,725	14.5
Shoes & Handbags	3	9,066	4.2
Office Products	4	5,894	7.7
Sports & Outdoors	3	6,286	6.1
Electronics	7	8,917	19.0

In the initial stage, there are no sufficient historical data to use our approach. Therefore, the tool composes relevant services using the learning features known to users without historical information (*i.e.*, price, discount, user rating, and relevance). Our tool composes services using the MORL algorithm described in Section II-C. The returned composite services are ranked by the expected Q-values derived from the MORL algorithm. As users tend to go through a limited number of returned services [17], we list 20 web services relevant to carry out each task and show the related *service descriptions* and *service attributes* to users. A user can choose multiple preferred web services by clicking the respective services. Our tool records service information as described in Section II-A1. Figure 3 illustrates a screenshot of our tool for selecting services.



Current Task: boots_women

Prefer	Service Descriptoin	relevance	price	user rating	discount
☐	Cressi US040004 Minorca Boots 3mm Premium Neoprene with Anti Slip Rubber Soles (Mens and Womens), 9	4.0	36.9	5.0	1
☐	Rampage Basking Knee High Boot	4.0	11.99	5.0	1
✓	SoftMoc Women's Ruthie Short Dress Bootie	3.0	89.99	5.0	0.64
☐	LARA's Womens Western Cowboy boots Mid Calf Fashion	8.0	50.99	5.0	0.51
☐	Aldo Women's MERENDI Pull-On High Shaft Boot	5.0	103.68	5.0	0.58
☐	Anne Klein Nilise Knee High Boot	27.0	33.99	5.0	0.15

Fig. 3: A screenshot of our tool for selecting services.

User Study. To evaluate our approach, we recruit 12 subjects to participate in our user study (*i.e.*, 4 females and 8 males). All of the subjects are graduate students. They are not familiar with service composition techniques, but perform on-line tasks daily. The subjects spend approximately 8-10 hours on-line each day. We provide instructions for using the tool. The subjects use our tool for a period of two months. We do not set any restriction on the number of goals and tasks that a subject needs to perform. On average, the subjects have performed 51.5 tasks over a period of two months. The average number of tasks performed in each domain is listed in Table II.

B. Experimental Results

(RQ1) Is our approach effective to compose services?

Motivation. To save users from manually specifying their preferences, it is critical to automatically learn user preferences and the priority orders. We are interested to evaluate the effectiveness of our approach that composes services by considering the inferred user preferences.

Approach. To compare the performance of our approach, we build two baseline approaches. In our approach, we use a user previously performed m goals as the training dataset, and compose services for the rest of the goals for the user using our proposed MORL algorithm (see Section II-C). We apply the approach to extract the learning features and identify the priority orders of user preferences. We follow the same data splitting strategy used in [25]. For the m goals, the learning-to-rank algorithm takes half of the dataset for training and validation, the other half for testing. We use Equation (3) to evaluate the performance of the learning-to-rank models. P@k is the ratio of the number of user selected services in the top k returned services to the number k.

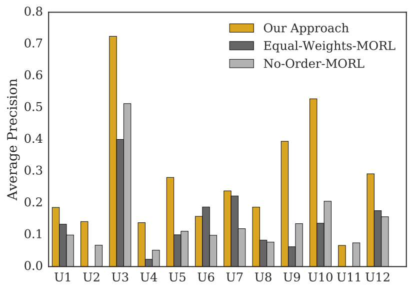


Fig. 4: Average precision of our approach and the two baseline approaches using 70% of the whole dataset for training. The x-axis denotes subject ids, Ui means the i^{th} user. y-axis is the average precision of recommending services.

$$P@k = \frac{\textit{\#Services Selected in Top k Results}}{k} \tag{3}$$

We compare the P@k to determine the preferences and the priority orders (see Section II-B2). If the difference between the P@ks of two individual feature models is higher than a threshold, these two learning features are considered to have different orders.

We design two baseline approaches, named Equal-Weights-MORL and No-Order-MORL. The two baseline approaches use the MORL algorithm, the weighted sum Q-learning algorithm [24], to compose services. Different from our approach, both baselines assume that user preferences and the priority orders are pre-defined. Both baseline approaches optimize the preferences under the category *service attributes* (*i.e.*, price, user rating, discount and relevance), since the learning features are visible and known to users.

Equal-Weights-MORL. The weights are the same for each preference and are set to $\boldsymbol{\omega}$ = (0.25, 0.25, 0.25, 0.25).

No-Order-MORL. This approach hypothesizes that users do not have specific priority orders over the preferences. The summation of the weight values is 1 and the weight values are randomly assigned in each run.

To ensure the efficiency of the MORL algorithm, we set the learning rate to 1, discount factor to 0.8, the ϵ to 0.7 and the number of iteration to converge as 1,000, same as [13]. To compose services for a goal, our approach and the two baseline approaches run the MORL algorithm for 1,000 times. We compute the precision using Equation (4). The precision is the ratio of the number of user selected services in the retrieved ones to the total number of retrieved services. For the goals that we compose services, we compute the precision for each goal and compare the average performances of the three approaches. The size of the training dataset (*i.e.*, m) can affect our results. Therefore, in our experiments, we take 50%, 60%, 70% and 80% of the number of user performed goals as our training dataset separately.

$$Precision = \frac{\textit{\#Correctly Identified Services}}{\textit{\#Retrieved Services}} \tag{4}$$

Results. **Our approach is more effective to help users select and compose services.** Figure 4 shows the average precision of recommending services for the three approaches with 70% of the whole dataset for training. We use P@1 to

TABLE III: Performance of the three approaches with different sizes of training dataset. **Abbreviation for method names**: OA, B1, and B2: our approach, Equal-Weights-MORL, and No-Order-MORL. We show the actual precision of our approach, and demonstrate the percentage of precision our approach improves or decreases the Equal-Weights-MORL and No-Order-MORL. "+" and "-" means improvement and decrease, respectively. Ui means the i^{th} user. The number colored with dark gray represents the highest number in the column, while the light gray denotes the lowest (same for other tables).

UID	Different Sizes of Training Dataset											
	50%			60%			70%			80%		
	OA	B1	B2	OA	B1	B2	OA	B1	B2	OA	B1	B2
U1	25	+69	+77	28	+101	+131	18	+39	+87	16	-15	+31
U2	19	+35	+48	12	$+\infty$	+34	14	$+\infty$	+110	11	$+\infty$	+65
U3	63	+16	+42	64	+28	+31	72	+81	+41	66	+50	+40
U4	14	+899	+251	13	+682	+214	13	+501	+167	15	+332	+177
U5	20	+93	+74	37	+195	+212	28	+180	+152	22	$+\infty$	+301
U6	29	0	+146	20	-19	+84	15	-15	+60	18	-10	+60
U7	20	0	+87	17	-31	+53	23	+7	+99	41	+150	+308
U8	17	+221	+97	4	-29	-48	18	+124	+142	11	$+\infty$	+75
U9	29	+673	+189	36	+626	+223	39	+530	+191	40	+302	+296
U10	50	+285	+158	42	+221	+109	52	+287	+156	47	+154	+112
U11	0	0	-100	24	$+\infty$	+165	6	$+\infty$	-10	23	$+\infty$	+281
U12	18	-26	-0.4	25	+9	+43	29	+65	+86	37	+40	+92

TABLE IV: Performance of our approach with different threshold values (%). "T" in the headers represents threshold. Ui means the i^{th} user.

UID	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12
T0.00	21.6	14.3	66.6	14.2	26.3	21.0	25.7	13.0	35.2	48.3	13.6	27.7
T0.01	21.8	17.9	63.9	11.5	26.0	21.2	25.0	12.2	35.6	45.2	12.1	27.3
T0.03	16.7	18.0	64.5	8.2	19.9	17.8	23.7	12.7	32.8	26.5	11.2	32.4
T0.05	14.7	17.8	60.6	6.4	14.7	16.3	20.9	12.2	29.1	27.6	8.6	28.2

evaluate our ranking models and set the threshold to determine the significance of learning feature as 0. If the performance disparity of two ranking models is higher than the threshold value 0, we consider that the learning feature is significant or has a higher priority order. Compared with *Equal-Weights-MORL* and *No-Order-MORL*, our approach achieves a better precision for 10 out of 12 subjects (*i.e.*, 83% of subjects, excepts for U6 and U11). The highest average precision our approach obtained is 72% on the subject U3. The subject U3 is a male graduate student majoring in the computer science.

In Table III, we show a breakdown of Figure 4, and the precision of the three approaches with different sizes of training data. We calculate the improvement of our approach compared with the baseline approaches using Equation (5). On average, our approach improves *Equal-Weights-MORL* by 205%, 176%, 180% and 125%, and *No-Order-MORL* by 89%, 103%, 107% and 153%, when we use 50%, 60%, 70% and 80% of the whole dataset for training, respectively. We exclude the $+\infty$ to calculate the average, as the average would be infinity. With only 50% of the whole dataset for training, we improve *Equal-Weights-MORL* and *No-Order-MORL* by 205% and 89%, respectively. As the increase of the size of training dataset, the improvement of our approach compared with the *No-Order-MORL* is more significant. Among the 12 subjects, the subject U4 achieves the highest improvement using our approach, compared with *Equal-Weights-MORL* (604% improvement) and *No-Order-MORL* (202% improvement).

Why our approach works? Our approach achieves a better performance for recommending services. It demonstrates that the inferred user preferences and the priority orders are valid for users. The *Equal-Weights-MORL* assumes that users view all of the user preferences equally. Thus user preferred services may not be contained in the retrieved services. The *No-Order-MORL* locates all the optimal (or sub-optimal) composite services. However, users can only prefer a small subset of the possible service compositions. The inferred user preferences and the priority orders help our approach to accurately locate the user preferred optimal composite services.

$$Improvement = \frac{Precision(OA) - Precision(B)}{Precision(B)} \tag{5}$$

where B means a baseline and OA means our approach.

(RQ2) What is the best configuration for using our approach?

Motivation. The user preferences and the priority orders of the preferences can be affected by two parameters, *i.e.*, the threshold of ranking models, and the precision metrics used to evaluate the models. In this question, we study the effect of the two parameters on the performance of our approach.

Approach. We conduct two experiments to study the two parameters separately. The threshold is used to compare ranking models, so as to determine the significant learning features. The precision metrics is used to evaluate the performance of ranking models to rank services. We test various threshold values from 0 to 0.05 and precision metrics P@1, P@3 and P@5. To test the effect of the threshold, we use P@1 as the precision metric to evaluate ranking models. For each threshold value, we use our approach to determine the priority orders of learning features and compose services.

Results. **The performance of our approach is affected by the threshold of ranking models.** Table IV shows the results of our approach with different threshold values. Each value in the table is the average value of the results of our approach using four sizes of training datasets (*i.e.*, 50%, 60%, 70% and 80%). For threshold values 0, 0.01, 0.03 and 0.05, the average precisions over all the subjects are 27.2%, 26.6%, 23.7% and 21.4%, respectively. The lower is the threshold value, the better is the performance. A high threshold value can ignore important learning features. Hence, we set the threshold value as 0 for the rest of the experiments. We determine the important learning features and the priority orders by comparing the performances of the ranking models. If the performance disparity is higher than the threshold value 0, we conceive the learning feature is significant or has a higher priority order.

Table V shows the results of the effect of precision metrics on our approach. Each value in the table is the average results from the four training datasets. The average precision of all the subjects is 27.2%, 28.2% and 27.0% for P@1, P@3 and P@5, respectively. We do not observe significant differences with various precision metrics. It shows that the our approach is not affected by the precision metrics.

TABLE V: Performance of our approach with different precision metrics (%). Ui means the i^{th} user.

UID	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12
P@1	21.6	14.3	66.6	14.2	26.3	21.0	25.7	13.0	35.2	48.3	13.6	27.7
P@3	22.1	13.4	66.8	15.5	24.5	19.5	30.7	14.8	31.2	53.4	18.1	28.9
P@5	19.9	11.1	65.9	13.4	26.4	12.4	23.6	22.0	27.7	53.8	19.3	29.0

IV. Related Work

We summarize the related work on the personalized service composition and multi-objective optimization.

Personalized Service Composition. A large amount of work aims to compose services by combining user preferences. Lin *et al.* [7] and Sohrabi and McIlraith [18] allow users to specify preferences in the form of PDDL3, the Planning Domain Definition Language. Kang *et al.* [4] and Zeng *et al.* [28] refer the preferences as the weights of learning features. The above approaches optimize composed services by maximizing the objective function over preferences. Balke and Wagner [2] specify preferences with the SQL-like syntax. The selected services guarantee the availability of user required parameters. Aggarwal *et al.* [1] and Lamparter *et al.* [5] develop an ontology model to describe functional and non-functional requirements. All of the aforementioned approaches demand the basic programming knowledge from users. Our approach infers the preferences automatically.

Multi-Objective Optimization. Moustafa and Zhang [13] utilize a multiple-policy MORL approach to derive a complete set of optimal composite services. However, it is inefficient to obtain all the composite services for both machines and users. Yu and Bouguettaya [27] leverage indices on service operations and an aggregate function to compute optimal composite services. Wang *et al.* [23][24] use a linear weighted sum function as the reward function for the MORL algorithm. Mouaddib [12] and Wray *et al.* [26] also represent preferences over learning features as an ordering relation. Users are required to specify preferences manually in these approaches. Unlike the above approaches, we automatically identify user preferences and the priority orders of the preferences to compose services for users.

V. Conclusion

To overcome the complexity of manually specifying user preferences by users in the service composition process, we propose a learning-to-rank approach to automatically learn user preferences and the priority orders of the preferences using the historical service composition dataset. Our learned priority orders of user preferences are integrated into the multi-objective reinforcement learning (MORL) algorithm, to efficiently derive optimal composite services. We conduct experiments to evaluate the effectiveness of our approach. Our experimental results show that the inferred user preferences are helpful for selecting services. Compared with two well-established baseline approaches, our approach improves them by 100%-200% on precision for service selection. In the future, we plan to consider more categories of learning features to capture more aspects of user preferences. We would also like to evaluate our approach using services from more domains.

References

[1] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, “Constraint driven web service composition in meteor-s,” in *IEEE. SCC*, 2004.

[2] W.-T. Balke and M. Wagner, “Towards personalized selection of web services.” in *WWW. ACM*, 2003, pp. 20–24.

[3] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, “An efficient boosting algorithm for combining preferences,” *Journal of machine learning research*, vol. 4, no. Nov, pp. 933–969, 2003.

[4] G. Kang, J. Liu, M. Tang, X. Liu, B. Cao, and Y. Xu, “Awsr: Active web service recommendation based on usage history,” in *ICWS*. IEEE, 2012, pp. 186–193.

[5] S. Lamparter, A. Ankolekar, R. Studer, and S. Grimm, “Preference-based selection of highly configurable web services,” in *WWW. ACM*, 2007, pp. 1013–1022.

[6] S. Lee, J. Chang, and S.-g. Lee, “Survey and trend analysis of context-aware systems,” *Information-An International Interdisciplinary Journal*, vol. 14, no. 2, pp. 527–548, 2011.

[7] N. Lin, U. Kuter, and E. Sirin, “Web service composition with user preferences,” in *ESWC*. Springer, 2008, pp. 629–643.

[8] C. Liu, X. Xu, and D. Hu, “Multiobjective reinforcement learning: A comprehensive overview,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 3, pp. 385–398, 2015.

[9] A. Maaradji, H. Hacid, J. Daigremont, and N. Crespi, “Towards a social network based approach for services composition,” in *ICC IEEE*, 2010, pp. 1–5.

[10] C. D. Manning, P. Raghavan, H. Schütze *et al.*, *Introduction to information retrieval*. Cambridge university press, 2008, vol. 1, no. 1.

[11] G. A. Miller, “Wordnet: a lexical database for english,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[12] A.-I. Mouaddib, “Multi-objective decision-theoretic path planning,” in *ICRA*, vol. 3. IEEE, 2004, pp. 2814–2819.

[13] A. Moustafa and M. Zhang, “Multi-objective service composition using reinforcement learning,” in *ICSOC*. Springer, 2013, pp. 298–312.

[14] M. F. Porter, “An algorithm for suffix stripping,” *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[15] J. Qiu, Y. Li, J. Tang, Z. Lu, H. Ye, B. Chen, Q. Yang, and J. E. Hopcroft, “The lifecycle and cascade of wechat social messaging groups,” in *WWW. ACM*, 2016, pp. 311–320.

[16] S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati, “Baya: assisted mashup development as a service,” in *WWW. ACM*, 2012, pp. 409–412.

[17] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, “Analysis of a very large web search engine query log,” in *SIGIR Forum*, vol. 33, no. 1. ACM, 1999, pp. 6–12.

[18] S. Sohrabi and S. A. McIlraith, “Preference-based web service composition: A middle ground between execution and search,” in *International Semantic Web Conference*. Springer, 2010, pp. 713–729.

[19] K. Sugiyama, K. Hatano, and M. Yoshikawa, “Adaptive web search based on user profile constructed without any effort from users,” in *WWW. ACM*, 2004, pp. 675–684.

[20] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[21] B. Upadhyaya, Y. Zou, I. Keivanloo, and J. Ng, “Quality of experience: User’s perception about web services,” *IEEE. TSC*, vol. 8.

[22] A. L. Van Den Wollenberg, “Redundancy analysis an alternative for canonical correlation analysis,” *Psychometrika*, vol. 42, no. 2, pp. 207–219, 1977.

[23] H. Wang, X. Chen, Q. Wu, Q. Yu, Z. Zheng, and A. Bouguettaya, “Integrating on-policy reinforcement learning with multi-agent techniques for adaptive service composition,” in *ICSOC*. Springer, 2014, pp. 154–168.

[24] H. Wang, X. Zhou, X. Zhou, W. Liu, and W. Li, “Adaptive and dynamic service composition using q-learning,” in *22nd International Conference on Tools with Artificial Intelligence*, vol. 1. IEEE, 2010, pp. 145–152.

[25] S. Wang, Y. Zou, J. Ng, and T. Ng, “Learning to reuse user inputs in service composition,” in *ICWS*. IEEE, 2015, pp. 695–702.

[26] K. H. Wray, S. Zilberstein, and A.-I. Mouaddib, “Multi-objective mdps with conditional lexicographic reward preferences.” in *AAAI*, 2015, pp. 3418–3424.

[27] Q. Yu and A. Bouguettaya, “Multi-attribute optimization in service selection,” *World Wide Web*, vol. 15, no. 1, pp. 1–31, 2012.

[28] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, “Qos-aware middleware for web services composition,” *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, 2004.

[29] Y. Zhao, S. Wang, Y. Zou, J. Ng, and T. Ng, “Mining user intents to compose services for end-users,” in *ICWS*. IEEE, 2016, pp. 348–355.

[30] Y. Zhao, F. Zhang, E. Shihab, Y. Zou, and A. E. Hassan, “How are discussions associated with bug reworking?: An empirical study on open source projects,” in *ESEM, ACM*, 2016, p. 21.