Applied Soft Computing Journal 85 (2019) 105830

Contents lists available at ScienceDirect

Applied Soft Computing Journal

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

Web service API recommendation for automated mashup creation using multi-objective evolutionary search

Nuri Almarimi [a], Ali Ouni [a,*], Salah Bouktif [b], Mohamed Wiem Mkaouer [c], Raula Gaikovina Kula [d], Mohamed Aymen Saied [e]

[a] Ecole de Technologie Superieure (ETS), University of Quebec, Montreal, QC, Canada
[b] College of Information Technology, UAE University, Al Ain, United Arab Emirates
[c] Rochester Institute of Technology, NY, USA
[d] Nara Institute of Science and Technology, Nara, Japan
[e] Concordia University, Montreal, QC, Canada

ARTICLE INFO

Article history:
Received 6 February 2019
Received in revised form 1 October 2019
Accepted 7 October 2019
Available online 17 October 2019

Keywords:
Service mashup
Web service
API recommendation
Search-based software engineering

推荐服务集合的方法：
定义为多目标组合问题；
用非支配排序基因算法去搜索最优子集

ABSTRACT

Modern software development builds on external Web services reuse as a promising way that allows developers delivering feature-rich software by composing existing Web service Application Programming Interfaces, known as APIs. With the overwhelming number of Web services that are available on the Internet, finding the appropriate Web services for automatic service composition, *i.e.*, mashup creation, has become a time-consuming, difficult, and error-prone task for software designers and developers when done manually. To help developers, a number of approaches and techniques have been proposed to automatically recommend Web services. However, they mostly focus on recommending individual services. Nevertheless, in practice, service APIs are intended to be used together forming a social network between different APIs, thus should be recommended collectively. In this paper, we introduce a novel automated approach, called *SerFinder*, to recommend service sets for automatic mashup creation. We formulate the service set recommendation as a multi-objective combinatorial problem and use the non-dominated sorting genetic algorithm (NSGA-II) as a search method to extract an optimal set of services to create a given mashup. We aim at guiding the search process towards generating the adequate compromise among three objectives to be optimized (*i*) maximize services historical co-usage, (*ii*) maximize services functional matching with the mashup requirements, and (*iii*) maximize services functional diversity. We perform a large-scale empirical experiment to evaluate *SerFinder* on a benchmark of real-world mashups and services. The obtained results demonstrate the effectiveness of *SerFinder* in comparison with recent existing approaches for mashup creation and services recommendation. The statistical analysis results provide an empirical evidence that *SerFinder*, significantly outperforms four state-of-the-art widely-used multi-objective search-based algorithms as well as random search.

1. Introduction

Reusing and composing third-party Web services has become a common practice in modern software systems engineering to provide high quality and feature rich systems. With the adoption of service-oriented architectures (SOA), several service ecosystems have emerged in recent years hosting a highly increasing number of published Web services by different providers on Internet [1] by different service providers. Hence, Programmable Web.com,[1] one of the largest online service Application Programming Interfaces (API) ecosystems, gather a wider variety of service APIs whose count goes beyond 19,000 service.

In the SOC (Service Oriented Computing) paradigm, existing services are reusable, programmable, and intended to be composed by developers to create Web service compositions, called *mashups*. Ultimately, mashups are created to meet comprehensive and complex functional requirements and provide an extra business values [2]. For example, a map management service requirement might be fulfilled by several services (*e.g.*, the *Google Maps* service, *Microsoft Bing Maps* service, *OpenTreeMap* service, etc.) to realize the required functionality at runtime. Looking at

* Corresponding author.
E-mail address: ali.ouni@etsmtl.ca (A. Ouni).

[1] www.programmableweb.com.

https://doi.org/10.1016/j.asoc.2019.105830
1568-4946/

ProgrammableWeb, for example, the map service *Google Maps* and the auction service *ebay* are composed to create a specific mashup called *BidNearBy*,[2] which provides a local auctions search based on a map view.

Indeed, mashup creation happens in a service API economy involving thousands of service API-providing companies offering a multitude of service choices across hundreds of categories. The problem of automated service mashup creation becomes more challenging especially with the increasing number of available competing services [3–9]. One key challenge is the ability to help users to find the appropriate service compositions that match their mashups to fulfill complex requirements. A typical instance of the service recommendation problem could be illustrated by an example of mashup service having 10 functionalities that are executed sequentially. Each of these functionalities can be performed by one of 10 available competing services. According to the smartest recommendation, such a mashup requires an exhaustive exploration of 10^{10} service compositions. Indeed, the problem is know to be NP-hard as pointed out by many researchers [10,11].

Several approaches and techniques have been introduced to address the service recommendation problem recently. Traditional approaches are based either on analyzing the requirement and service API descriptions, then recommend services using information retrieval techniques [5]; or on discovering frequent service composition patterns from historical mashups [8,9]. Other common approaches [12,13] recommend services through Quality of Service (QoS) optimization. The vast majority of these service recommenders take as input the developers' request, then a matching score is affected to every service, before outputting a ranked set of Web services that are ordered based on their score to the developer [6].

Despite the effort of current studies in recommending adequate mashup creations, we identify a number of relevant shortcomings. First, most of service recommendation approaches focus on recommending individual services in isolation. As a result, highly ranked services exhibit a redundancy due to sharing the same or similar functions provided by other co-recommended services. Evidently, in mashup design, the recommendation is more relevant if it contains a set of complementary, yet compatible services, *i.e.*, commonly used together, instead of similar services. In fact, the valuable knowledge which is extracted from existing mashups is ignored. Especially, knowing that some services are more likely to be composed together than others, as pointed out in previous research [14].

Second, existing service recommendation approaches based on service usage record tend to give little attention to the matching between the recommended services and the mashup requirements. That is, relying on frequent sequence is not appropriate in the situations where newly released services can be adopted, *i.e.*, when new services are published and have a better matching with mashup requirements but not commonly used with other services. We argue that basing the recommendation on the services usage record, may hurt other services having better functional matching. Ultimately, such challenges call for new recommendation approaches to better balance the '*wisdom of the crowd*' as reflected by service historical co-usage in service ecosystem, and functional matching between services and mashup requirements.

To tackle the service recommendation problem for automated mashup creation, we introduce in this paper an automated approach named as *SerFinder*. Given a developer's textual description of a given mashup requirement, *SerFinder* aims at recommending relevant service sets to the developer that are most appropriate to his mashup requirements. The recommendation process aims at finding a trade-off among three main objectives to be optimized (*i*) maximize service co-usage, *i.e.*, the recommended services have been commonly used together forming a social network between related services, (*ii*) maximize functional diversity, *i.e.*, the recommended service set should implement distinct functionality to prevent redundant/similar services, and (*iii*) maximize functional matching, *i.e.*, the recommended service set should match the developer' requirements.

SerFinder adopts the non-dominated sorting genetic algorithm (NSGA-II) [15], as an intelligent search algorithm to identify solutions having the optimal trade-off among the three abovementioned objectives. Our problem formulation is justified by the complexity of the automated mashup creation problem which is combinatorial as it consists in essence at finding service combinations through a search process guided by such conflicting considerations.

To evaluate *SerFinder*, we conduct an empirical study on a large and real-world benchmark that consists of 15,386 Web services and 3,564 mashups compositions mined from the ProgrammableWeb ecosystem. The empirical results provides evidence that *SerFinder* is efficient in recommending relevant service sets. The statistical analysis of the experimental results show that *SerFinder* outperforms four recent state-of-the-art approaches [3,6,8,16] with a precision score ranging from 30% to 41% and a recall ranging from 53% to 78%. Furthermore, the statistical analysis of the achieved results show that the used NSGA-II algorithms outperforms random search as well as four other widely-used multi-objective optimization algorithms for the problem of problem automated service mashup creation.

The rest of the paper is organized as follows. Section 2 introduces the paper's taxonomy. Section 3 reviews the existing studies related to our work. Section 4 details the problem definition underlying our approach. Section 5 describes our approach, *SerFinder*, for service set recommendation. The experiments are described in Section 6, along with the key findings, while Section 7 describes the potential threats that may affect the validity our study. The papers ends with Section 8 which draws our conclusions and presents our future perspectives.

2. Background

Before diving into the technicalities of our methodology, we first define the necessary keywords which will be used throughout the paper.

2.1. Definitions

service API. A collection of improved functions, given for public usage either for free or for a given fee. Service APIs are generally available through REpresentational State Transfer (REST)-based web services. The documentation associated with API usage is stored in ProgrammableWeb. The documentation mainly contains the function signature along with a brief and long description of its functionality.

A mashup represents the composition of one or many web service APIs into one single application, available as a service [17]. Mashups often mediate among a set of service APIs provided by a heterogeneous set of providers. Mashup are generally characterized by their novel integration of multiple APIs issued by various service providers. The quality of this structural composition has a direct impact on its integrity and on it quality of service [18].

[2] www.programmableweb.com/mashup/bidnearby.

3. Related work

3.1. Service recommendation

The problem of automated mashup creation has attracted many researchers and practitioners in recent years. Unlike recommending individual services, there are few recent approaches to recommending a set of services including SSR [6], DCCC [16], CDSR [3], SPR [8].

SSR [6] recommends service sets based on three main factors, namely the QoS preferences, composition preferences, along with functional preferences to drive service selection and ranking. CDSR [3] recommends services for composition by category, to avoid redundant recommendations. Services are initially categorized based on the similarity of their functionality. The SPR method [8] recommends services based on their matching with developers' requirement and based on their internal composability with each other. The internal composability is approximated based on tags and topics of services, and the frequency of composition between services. However, SPR recommends under the assumption that similar services are most likely to be together joint, but this does not hold in all scenarios [6]. In contrast, the DCCC method [16] uses divide-and-conquer technique on its proactive recommendation of any candidate collaborative services for a newly created service. Their method uses tags of categories and collaboration records along with text descriptions and finally the history of mashup-service usage. Furthermore, most service recommendation approaches employ either (*i*) keyword-based or LDA (Latent Dirichlet Allocation)-based for modeling the characteristics of latent topics between and queries of users and service descriptions [5], or (*ii*) neighborhood-based collaborative filtering [9], or combine both as hybrid methods. However, these approaches tend to recommend redundant services with the same functionality.

Chune et al. [19] proposed methods that use developer's needs matching as an objective of service recommendation. These methods are qualified as functionality based methods. In Yu et al. [20], collaborative filtering (CF) models were used for service recommendation using the QoS prediction. This prediction model relies on mining any QoS information to approximate the user experience in terms of QoS from the previous usage of services. Tingting et al. [21] exploited data on social relationships among developers, mashups, and services in addition of location similarity to predict unobserved relations to support more accurate service recommendations.

Our work differs from these existing work by formulating service set recommendation as multi-objective search problem and uses meta-heuristic search instead of an exhaustive one. Besides, we define three competing objectives to drive the search process. The goal is to avoid recommending similar/redundant services while balancing the historical service collaborations and the functional matching with the mashup requirement.

Furthermore, a different approaches have recently been proposed to recommend software libraries in Object-Oriented (OO) software systems. Thung et al. [22] proposed an association rule mining and collaborative filtering based approach to recommend software libraries. Ouni et al. [11] used a search-based approach to find library reuse opportunities in the context of Java OO systems. Saied et al. [23] used mining and clustering technique to find library usage patterns for the Maven repository. Although different, all these previous works on libraries recommendation in OO systems constitute a strong background of expertise and ground knowledge when designing automated mashup creation algorithms in the context of SOC. Indeed, the proposed approach in this paper is largely inspired by our previous research works in OO library recommendation [11,23]. However, the dynamics of service APIs environments leads to many new challenges that are not featured in traditional OO software systems development due to the different technological, environmental and social contexts.

3.2. Service composition

Another category of works have explored evolutionary algorithms for Web services composition/binding. Ramírez et al. [24] studied the QoS Web service composition using evolutionary computation. The authors addressed the binding problem using multi- and many-objective algorithms. In their study, 9 QoS properties were involved. Results show that the evolutionary search can generate optimal solutions at the expense of an acceptable computational cost, which enables their application on decision-support and time-critical systems. In de Campos et al. [25], different variants of NSGA-II were proposed to deal with five QoS objectives. To reduce the high-dimensionality of the problem, preference relations were injected in the search space to enhance the performance of the proposed approach. Furthermore, Suciu et al. [26] deployed MOEA/D [27] to search for an optimal compromise between three objectives related to a variant of the QoS based Web service composition. A comparative study was conducted between MOEA/D, NSGA-II and GD3 [28], leading the authors to report the MOEA/D's ability of outperforming the other evolutionary algorithms.

Recently, Yu et al. [29] adopted evolutionary algorithms to handle many objectives when solving the high-dimensional the QoS-based Web service composition problem. They proposed F-MGOP, a variation of NSGA-III [30,31]. However, the study was limited to data-intensive services. Also, in the validation, the authors compared F-MGOP with traditional multi-objective algorithms such as SPEA2 [32] and NSGA-II. More recently, Pan et al. [33] used genetic-based clustering algorithm for mashup service clustering. The proposed approach quantifies the structural similarity, using SimRank, between potential combinations of mashup services. However, their goal was to cluster existing mashup services, instead of identifying suitable services for creating meaningful mashups automatically.

However, these works adopting evolutionary algorithms are merely based on QoS and ignore more important aspects including the history of services composition and the rich semantic content and functionalities provided by services as well as functional diversity which allow meaningful mashup creation.

4. Problem definitions

This section outlines the terminology and necessary concepts for the problem definition.

4.1. Mashups and service composition

Our approach aims at mining large service ecosystems to learn the '*wisdom from the crowd*'. In particular, we are interested in the composition (*i.e.*, co-usage), of services to create mashups.

Suppose $s \in \mathcal{R}$, where s is a service that belongs to a service repository, i.e., ecosystem, $\mathcal{R}$. As an example of popular super repositories that contain such services, we refer to `ProgrammableWeb.com`. One of the aspects that specifically interest us is the number of different mashups M adopting a service s. At this level, we are not concerned with the granularity of service versions.

In following, we provide the definitions of the different concepts underlying our approach:

Definition 1 (*Service Ecosystem*)**.** A service ecosystem $\mathcal{R}$ denotes a set of services and mashups history records. We define $\mathcal{R} = (S, M)$ where $S = \{s_1, s_2, \ldots, s_m\}$ and $M = \{m_1, m_2, \ldots, m_n\}$ denote the collection of services and mashups in the ecosystem $\mathcal{R}$, respectively. A mashup-service usage record exists if a mashup $m_i \in M$ invokes at least a service $s_j \in S$.

Definition 2 (*Usage*)**.** The service usage is defined as the number of existing mashups that previously used a service s. Let M_i a mashup, and s a service, the function $invoke(M_i, s)$ returns 1 if M_i invokes s, 0 otherwise. We formally define the usage as follows:

$$usage(s) = \sum_{i=1}^{p} invoke(M_i, s) \tag{1}$$

where p corresponds to the total number of mashups available in $\mathcal{R}$, and s is a given service in $\mathcal{R}$.

Definition 3 (*Co-usage*)**.** The service co-usage is defined as the number of existing mashups that used together a pairs of particular services. Let s_1 and s_2 two distinct services, then the co-usage is calculated as follows:

$$co\text{-}usage(s_1, s_2) = \sum_{i=1}^{p} (invoke(M_i, s_1) \times invoke(M_i, s_2)) \tag{2}$$

Based on the history of mashup creation, there are extractable patterns of a subset of services that are often called together. Such service *co-usage* is an indication of their affinity for creating mashups. Therefore, *co-usage* is a learning mechanism of such affinity among services from historical data. Also, it indicates a strong correlation between services in terms of their ease of collaborations.

Definition 4 (*Balanced-Usage*)**.** The services balanced-usage metric is defined as the co-usage average value proportionally to the total usage of different services s_1 and s_2. Formally, the Balanced-Usage is defined as follows:

$$balanced\text{-}usage(s_1, s_2) = \frac{1}{2} \times \left(\frac{co\text{-}usage(s_1, s_2)}{usage(s_1)} + \frac{co\text{-}usage(s_1, s_2)}{usage(s_2)} \right) \tag{3}$$

The balanced-usage corresponds to the normalized co-usage score between two services.

4.2. Service and mashup description similarity

Finding the similarity between service-to-service or between service-to-mashup requirements is of primary importance for efficient recommendation system. Using information retrieval (IR) and natural language processing (NLP) techniques, *SerFinder* adopts a semantic similarity measure to capturing similar concepts between services and mashups. We consider two levels of textual descriptions similarity that can exist (i) among two services and (ii) among a service and a developer's textual description of mashup requirement.

We define a service and mashup content as follows.

Definition 5 (*Service Content (S_c)*)**.** The content of a service s consists of a collection of words $S_c(s) = \{sw_1, sw_2, \ldots, sw_N\}$ to describe its functional abilities. S_c could be extracted from the service description.

Definition 6 (*Mashup Content (M_c)*)**.** Similarly to the service content, the content of a mashup m consists of a collection of words $M_c(m) = \{mw_1, mw_2, \ldots, mw_N\}$ to describe its provided functionalities. The mashup content could be extracted either from the textual description of a mashup or from the mashup requirements as specified by a developer' textual description.

4.3. Service and mashup description-based similarity

Once the service and mashup contents are formed, the goal is to identify the degree of similarity (i) among services, and (ii) among mashups and services, based on their textual descriptions.

Definition 7 (*Text Description-based Similarity (TD_{sim})*)**.** The text description based similarity quantifies the semantic similarity of two bags-of-words representing services/mashups.

To measure the TD_{sim} between two entities A and B (service-to-service or service-to-mashup), we employ *cosine similarity*, a lightweight and widely-used technique in information retrieval to compute the semantic similarity. This measure corresponds to the cosine between the vectors corresponding to A and B in the semantic space constructed by *tf–idf* (term frequency–inverse document frequency). Formally, TD_{sim} is computed by the following function:

$$TD_{sim}(A, B) = \frac{\sum_{i=1}^{N} w_{a_i} \times w_{b_i}}{\sqrt{\sum_{i=1}^{N} w_{a_i}^2} \times \sqrt{\sum_{i=1}^{N} w_{b_i}^2}} \tag{4}$$

where w_{a_i} and w_{b_i} denote the *tf–idf* weights corresponding to each of the terms $a_i \in A$ and $b_i \in B$, respectively.

5. Approach: SerFinder

We provide in this section the general structure of our approach *SerFinder*. Thereafter, we explain in details how we adopted a multi-objective search to recommend relevant service sets for automatic mashups creation.

5.1. Approach overview

Fig. 1 shows the general framework underlying the *SerFinder* approach. *SerFinder* takes as input a developer requirement for a new mashup creation, service usage record including the history of mashups compositions, service and mashup textual descriptions. As output, *SerFinder* returns a collection of optimal service sets that are relevant to the mashup requirement. As depicted in Fig. 1, our approach consists of two main phases : (i) data extraction and processing phase, and (ii) search and recommendation phase.

Phase 1: Data extraction and processing. In this phase, the necessary data is collected for our recommendation system including (i) mashups and services usage history, and (ii) mashups and services content description.

The mashups and services usage history is collected from existing service ecosystems, e.g., *programmableWeb.com*. The collected mashups and services usage history are organized following Definition 1 of the service ecosystem (Section 4.1).

The mashups and services content description is collected from their textual description. The textual description of a service or a mashup represents the main source of information that reflects its functionality. To extract relevant information embodied in such textual description, our approach pre-processes both service and mashup descriptions to remove potentially existing noise that may compromise the output of the natural language processing (NLP) techniques. The involved NLP steps are:

Tokenization. The text description of services and mashups is split into lists of unit words, called tokens. The tokenization process excludes numbers and meaningless tokens that generally provide useless information.

Filtering. The filtering step aims to reduce the non-informative and useless words. We use a stop-words list to exclude common

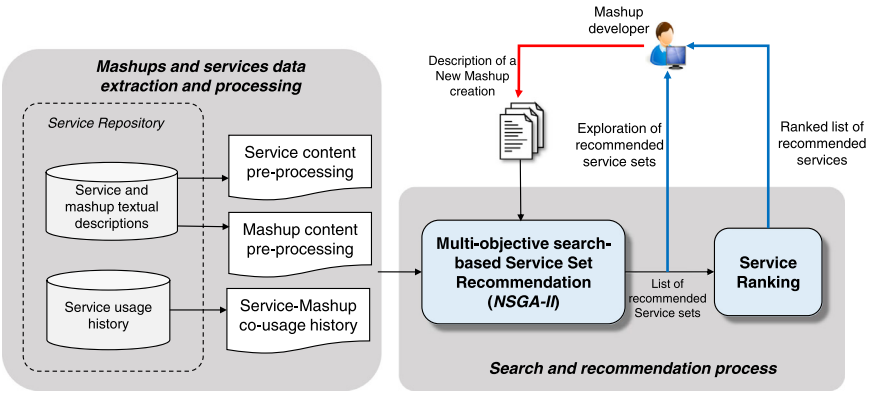

Fig. 1. Approach overview.

and frequent English words[3] that are frequently employed in English texts (*e.g.*, "the", "a", and "an"). Such stop-words generally add unnecessary noise and meaningless information to the NLP tasks and not directly related to the business abstraction domain. The remaining words are fed into the next step for lemmatization.

Lemmatization. This is the restitution of a given word to its basic form, called lemma. Lemmatization gathers all inflected forms and extensions of a given word, in order to consider them all as part of the word basis. The rationale behind this process is the consideration of all forms and instances, containing the same root, to be semantically similar and their meaning is close, thus, they would be treated as identical. For instance, for the word 'study', all keywords using that lemma are considered identical, such as 'studies', 'studying', 'student', and 'studied'. This process relies on the use of Stanford's CoreNLP[4] as the detector of word roots.

Vocabulary expansion. To enhance the semantic similarity and NLP tasks and make the vocabulary more exhaustive, we use `WordNet`[5] to generate synonyms for the extracted words. Such vocabulary expansion will provide more informative and meaningful sense to the collected and processed vocabulary for each service/mashup description. For instance, the keyword *student* may be related to various synonyms (*e.g.*, *scholar*, *pupil*, etc.), as long as they belong to a common domain concept. On the other hand, to find a trade-off between coherency and exhaustiveness, synonyms are considered, according to Wu et al. semantic relatedness (WUP) [34], with a threshold of 50% equality with the original list of extracted words.

As the end, the process returns a bag-of-words for each individual service and mashup in the ecosystem that reflects its content.

Phase 2: Search and recommendation phase. To explore the extracted data about services and mashups usage and content, efficient search techniques are needed. Our approach aims at supporting mashup developers by recommending relevant service sets that match their requirements. Specifically, the proposed approach aims at addressing the problem when mashup developers need a set of services to fulfill their required functionality, but they are not clear about the specific services required. Our approach recommends service sets based on two main aspects (1) functional matching with the developer requirements, and (2) the '*wisdom of the crowd*', *i.e.*, services commonly co-used together for mashups creation.

Indeed, the task of searching for commonly used service sets that match a given mashup requirements, is fastidious, complex and time-consuming. Hence, the number of potential solutions (*i.e.*, service combinations) could be very large or even infinite [35] as the number of available services grows. Thus, dedicated intelligent algorithms, namely the meta-heuristics, are known to solve such search problems with a high degree of objectives and constraints satisfaction.

NSGA-II finds multiple optimal, *i.e.*, non-dominated, solutions instead of single one, based on the *Pareto optimality principle* [15]. It shows a variety of optimal solutions in the search space, which allow the developer to explore a solution space that provides optimal trade-offs between the optimized objectives. Thus, the developer can choose one according to his preferences. *SerFinder* comprises an automatic ranking module for all recommended services in the different optimal solutions returned by NSGA-II, as detailed next.

Service Ranking. All the recommended services that appear in all the optimal solutions, *i.e.*, service sets, returned in the *Pareto front* by NSGA-II, are then sorted based on their redundancy count in all the returned service sets. The more the number of optimal service sets in which a service appears, the more the service is relevant for the mashup requirement. Our intuition is that if a service appears in several optimal solutions, then it is more relevant for the mashup. For instance, suppose the search algorithm returns 10 optimal solutions (service sets), where the service '*Google Maps*' appears in 7 solutions, and the service '*eBay*' appears in 5 solutions. Then, '*Google Maps*' will be ranked first with a score of 0.7, while the '*ebay*' rank will be 0.5.

In the next section, we detail how we adapted NSGA-II for the service set recommendation problem.

5.2. NSGA-II adaptation

We adopted the multi-objective search algorithm, NSGA-II [15] which is a widely used and powerful evolutionary algorithm. NSGA-II have show good performance in solving several software engineering problems [35–44]. Nonetheless, the generic nature of genetic algorithms, including NSGA-II, requires personalizing it to become a domain-specific algorithm. The design process consists of the following main steps: (*i*) solution representation, (*ii*) fitness function, and (*iii*) genetic change operators as part of the solution's evolution.

5.2.1. Solution representation

A candidate solution is typically represented in the form of a number of decision variables that need to be optimized through an evolutionary search process. We encoded a feasible solution for our problem as chromosomes of length n using a vector

[3] http://www.textfixer.com/resources/common-english-words.txt.
[4] nlp.stanford.edu/software/corenlp.shtml.
[5] wordnet.princeton.edu.

Fig. 2. Solution representation.

representation, where each gene, *i.e.*, vector dimension, represents a candidate service. Each candidate service has a unique numerical ID in the dataset which is used to identify it in the vector representation. The chromosome's length, n, is variable and corresponds to the number of individual services to be recommended for the mashup within an upper bound *maxSize* that is specified by the user. The order in which the services appear in the vector is not important. Indeed, in a real world scenario, a developer would use a number of relevant services to fulfill a given set of requirements for which the order is not important as long as the list of needed service is complete. Fig. 2 represents a simplified example of a chromosome that contains five recommended services. For the sake of clarity, our illustrative solution example shows the particular name of the service API, instead of it numerical ID.

5.2.2. Fitness evaluation

To evaluate how good is a candidate solution, we define a multi-objective function to optimize simultaneously the three following objectives.

Maximize service balanced-usage (SBU): Let *Sol* a candidate solution, that contains a set of services $Sol = \{s_1, \ldots, s_n\}$ for a given requirement of a mashup M. The balanced usage is computed based on the average of *balanced-usage* of all pairs of services belonging to the solution *Sol*, and calculated as follows:

$$SBU(Sol) = \frac{\sum_{\substack{\forall (s_i, s_j) \in Sol \\ s_i \neq s_j}} balanced\text{-}usage(s_i, s_j)}{\frac{|Sol| \times (|Sol|-1)}{2}} \tag{5}$$

where the function *balanced-usage* is given by Eq. (3), and $\frac{|Sol| \times (|Sol|-1)}{2}$ is the total number of possible pairs (s_i, s_j) to scale the objective function.

Maximize service functional diversity (SFD): This objective function aims at maximizing the diversity of the recommended services within a candidate solution in terms of their functionality as captured by the service textual description, *i.e.*, content. Let *Sol* a candidate solution that contains a set of services $Sol = \{s_1, \ldots, s_n\}$ for a given requirement of a mashup M. *SFD* is defined as the complement of the average of the *text description based similarity* (TD_{sim}) of all pairs of services belonging to the solution *Sol*, and calculated as follows:

$$SFD(Sol) = 1 - \frac{\sum_{\substack{\forall (s_i, s_j) \in Sol \\ s_i \neq s_j}} TD_{sim}(s_i, s_j)}{\frac{|Sol| \times (|Sol|-1)}{2}} \tag{6}$$

where the function TD_{sim} is given by Eq. (4), and $\frac{|Sol| \times (|Sol|-1)}{2}$ is the total number of possible pairs (s_i, s_j) to schale the objective function.

Maximize the matching between service content and mashup requirements (MSM): This objective function aims at maximizing the content relevance for the recommended services to the mashup requirements. Let *Sol* a candidate solution that contains a set of services $Sol = \{s_1, \ldots, s_n\}$ for a given requirement of a mashup M. The *MSM* is calculated as follows:

$$MSM(Sol) = \frac{\sum_{\forall s_i \in Sol} TD_{sim}(s_i, M)}{|\ Sol\ |} \tag{7}$$

where the function TD_{sim} is given by Eq. (4).

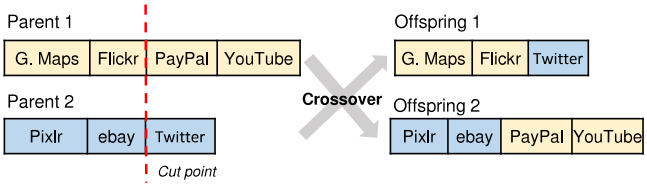

Fig. 3. Crossover operator.

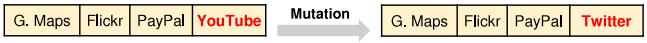

Fig. 4. Mutation operator.

5.2.3. Change operators

Population-based evolutionary algorithms including NSGA-II rely mainly on a set of genetic operators, *i.e.*, crossover and mutation, to change the solution's genotype. These changes are intended to evolve the solution into a different, and hopefully improved one. These operators are solicited in every iteration, starting from the initial iteration that generates totally random solutions. Change operators are the main drivers of the search to converge solutions to a near-optimal state.

The *crossover* operator creates offspring solutions from existing ones that are generated during previous iterations, *i.e.*, a re-combination of solutions. In our NSGA-II adaptation, a single, random cut-point crossover operator is considered for the construction of a newborn solution, from two parent-solutions. The operation starts by randomly choosing then splitting two solutions from the current generation. Thereafter, the crossover operator outputs two offspring solutions by combining, for the first offspring, the left part of its first parent solution with the right part of the second parent solution, and vice versa for the second offspring. After the crossover operator takes place, a repair function checks potential redundant services in both offsprings and replaces them by random services from the pool of available services, if any. Fig. 3 depicts an example to illustrate the crossover mechanism.

The *mutation* operator aims at randomly introducing slight changes into selected solutions. This operator helps driving the search process into different solutions within the search space that cannot be reached via pure recombinations in order to avoid converging the population towards few or specific elite solutions. With service set recommendation, we employ a mutation operator that selects at random one or more services from their solution vectors and substitutes them by other services as shown is Fig. 4. To provide a better exploration of the search space, our mutation operator randomly selects a new service from the pool of available services regardless of its category. To be valid, the newly selected service should be different from the current services in the solution.

6. Empirical evaluation

This section describes the design of our experimental study, including the research questions to be answered, the datasets and the evaluation protocol for comparing our approach *SerFinder* against recent state-of-the-art service set recommendation techniques.

6.1. Research questions

To conduct the general evaluation of *SerFinder*, we experimentally address the three following research questions:

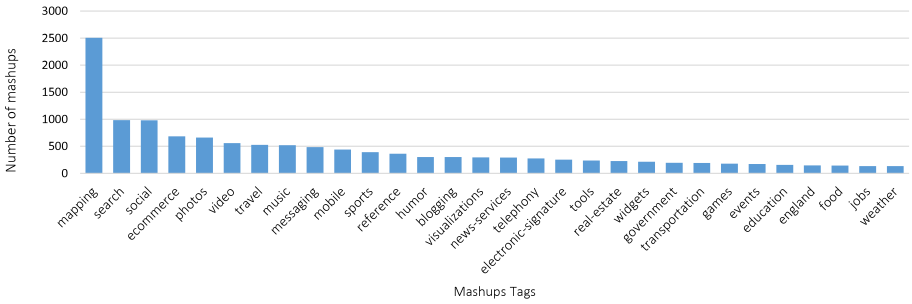

Fig. 5. Distribution of the mashups in the top-30 tags.

Table 1
Experimental dataset statistics.

Statistics	Value
# of services	15,386
# of mashups	7822
# of individual services that collaborate with other service(s)	1289
# of mashups composing two services or more	3564
# of terms in mashups text descriptions	185,208
# of different tags in the mashups corpus	408
Average number of tags per mashup	2.44

RQ_1. How accurate is *SerFinder* in service sets recommendation for automatic mashup creation?

RQ_2. Does *SerFinder* provide better ranking of recommended service sets?

RQ_3. How does *SerFinder* perform in comparison with random search and other popular multi-objective search methods?

6.2. Context selection

Dataset. We evaluate the proposed approach *SerFinder* on a real-world Web services dataset that is collected from the popular service ecosystem ProgrammableWeb, which contains a catalog of services and mashups since its foundation in 2005. In the data collection process, we crawled the metadata the current Web services and mashups in the period from September 2005 to August 2016. For each service, the following informations were collected: name, description text, publishing date, and category while the metadata for each mashup includes its requirement text, summary, mashup tags, as well as the list of invoked services. The working data set used in our study is publicly available [45]. Table 1 summarizes the statistical properties of the dataset.

As reported in the table, our mashups dataset is highly diversified in terms of mashups requirements belonging to 408 tags, *i.e.*, application domains. On average, the collected mashups belong to 2.44 different tags. Moreover, Fig. 5 shows the distribution of the top-30 tags among our mashups dataset reflecting a high diversity in the application domains, i.e., *mapping*, *search*, *ecommerce*, *photos*, etc.

Comparative study. To evaluate the accuracy of *SerFinder*, we compare it against four recent state-of-the-art methods, SSR [6], DCCC [16], CDSR [3], and SPR [8] (c.f., Section 3 for details), to investigate what improvements our approach could achieve.

6.3. Experimental setting

6.3.1. Analysis method

To answer **RQ_1**, we assess the recommendation accuracy using traditional precision and recall ratios, widely used for comparing recommendation systems. *Precision* represents the percentage of the number of correctly recommended services over the total number of recommended services, whereas the *Recall* refers to the percentage of number of correctly recommended services over the total number of appropriate services, *i.e.*, the ground truth (services actually invoked in the mashup). Specifically, we denote *precision@k* and *recall@k* to assess the number of recommended services' impact on the accuracy. The overall *Precision@k* and *Recall@k* of the recommendation is the average of all precisions and recalls achieved for each $m_i \in M$.

To answer **RQ_2**, we aim at evaluating our recommendation approach as a ranking task, meaning that we are mainly interested in a relatively few items, *i.e.*, service APIs, that we consider most relevant which is known as top-K recommendation. We, thus, assess the quality of our recommendations ranking quality, as described in Section 5.1. The goal is to assess the relevance of a recommended service based on its position in the resulting ranked list. To evaluate the ranking quality, we use two common metrics: the Normalized Discounted Cumulative Gain @top k (*NDCG@k*), and the Mean Average Precision @top k (*MAP@k*) [3,16,46]. The metrics NDCG@k and MAP@k are given in Eqs. (8) and (9), respectively. The higher *MAP@k* and *NGCD@k*, the better is the recommendation accuracy. *NDCG@k* emphasizes on the precision of the first few ($1st$, $2nd$, $3rd$, ...) recommendations.

$$NDCG@k = \frac{1}{S_k} \times \sum_{j=1}^{k} \frac{2^{r(j)} - 1}{log_2(1+j)} \in [0, 1] \tag{8}$$

where $r(j)$ is associated with jth recommended service on the ranking list, as being its relevant score, and whose value is binary, *i.e.*, 1 if the service appears in the top k services, or 0 otherwise. S_k is the best (maximum) score that any element can cumulatively reach.

$$MAP@k = \frac{\sum_{j=1}^{k} (\frac{k_r}{r} \bullet I(r))}{k_{used}} \times \in [0, 1] \tag{9}$$

where k_r represents the number of currently utilized service instances in the top r services of the ranking list. $I(r)$ indicates whether the service at a given ranking position r is actually used, and k_{used} is the total number of actual used services in the mashup.

To answer **RQ_3**, we first conduct a comparative study of the NSGA-II formulation against random search (RS) [47] in terms of performance. In general, the goal of such an initial comparison with RS is to check whether there an intelligent search method is required to explore such a large search space of possible service combination solutions to create relevant mashups. In addition, to justify the use of NSGA-II, we design a comparative between our NSGA-II based approach against four popular state-of-the-art multi-objective search algorithms namely Indicator-Based Evolutionary Algorithm (IBEA) [48], the Strength Pareto Evolutionary Algorithm (SPEA2) [32], and Multi-objective evolutionary algorithm based on decomposition (MOEA/D) [27], and

Multi-Objective Particle Swarm Optimization (MOPSO) [49]. **RQ_3** is considered as a standard 'baseline' question that fulfill a *sanity check* used at any attempt at using search algorithms to solve software engineering problems [35,50].

Since multi-objective search algorithms converge towards a set of optimal solutions (also called *non-dominated*, or *Pareto optimal* solutions), the performance of such algorithms needs to be assessed to make sure that the best algorithm is used. A set of performance and convergence metrics have been deployed in multi-objective optimization for the purpose of comparison between algorithms and in order to better understand their evolution during the search [51–53]. We evaluate the performance of each algorithm based on Zitzler et al. measures [53], based on three measures (1) the quality of the generated Pareto fronts, (2) the convergence to the exact Pareto front, and (3) the diversity of the produced solutions. In particular, we measure the six following performance metrics:

- *Hypervolume (HV)* [54]: it approximates the volume of the subspace covered by a given front. This metric is used as an evaluation of how well the approximate front achieves the optimization objectives. Thus, the higher the value of the Hypervolume, the better is the performance of the front.
- *Generational Distance (GD)* [52]: this metric takes as input a set of solutions *S*, then calculates how far they are from the reference set *RS*. The computation of the distance between *S* and *RS*, for a given *n* objective space, is the average n-dimensional Euclidean distance between each point in *S* and its nearest neighboring point in *RS*. *GD* is a convergence indicator representing how "far" the mashup creations *S* is from *RS*, *i.e.*, an error measure.
- *Inverted Generational Distance (IGD)* [52,53]: this metrics is considered as an indicator of both convergence and diversity. In contrast with *GD*, *IGD* represents the average Euclidean distance separating each reference solution set (*RS*) from its closest non-dominated one *S*. The smaller is the value of *IGD*, the better is the convergence of solutions, when compared to the ones in the Pareto front. Also, a smaller values of *IDG* indicates a good distribution of mashup creations in comparison with the ones in the Pareto front.
- *Generalized Spread (GS)* [15,55]: it computes the spread of the evolved solutions, belonging to an approximate front. This metric serves as an evaluator of how well distributed the solutions are, with respect to all objectives. Thus, the higher the value of *GS* is, the better is the convergence of evolving mashup creations towards the Pareto optimal ones.
- *Pareto Front Size (PFS)* [56]: it simply calculates how many solutions are inside the Pareto front. Depending on the nature of the problem, a relatively higher *PFS* is considered better since it provides decision makers with a larger variety of equivalent mashup creations to choose from.
  A higher size of the generated Pareto front is preferred, as multiple and a variety of options for mashup creations will be given to the user.
- *Contribution (IC)* [57]: this metric measures percentage of non-dominated solutions a given algorithm *A* provides to the cardinality of the Reference Front (*RF*). While *IC* is an easy to calculate metric, it depends on the number of generated solutions, unfavorably penalizing an algorithm if it generates 'few but excellent' solutions. Higher values for the IC metric mean better performance.

It is important to mention that, during the calculation of the aforementioned metrics, for each system, the Pareto front is approximated using the set of non-dominated solutions that are generated by all algorithms over all runs as reference solutions.

6.3.2. Statistical analysis and tests

The stochastic nature of search-based algorithms allows multiple solutions to be considered appropriate for a given problem regardless of their different characteristics. Furthermore, for the same problem, each algorithm may produce different results from one execution to another. Therefore, during the experiments, each search algorithm is evaluated using 31 independent simulation runs for each problem instance and the obtained results are statistically verified using the Mann–Whitney test [58,59] with a 95% confidence level ($\alpha = 0.05$). We adjusted the p-values using Bonferroni correction [60,61]. The Mann–Whitney test is a non-parametric statistical test that allows the comparison of obtained results through the verification of the following null hypothesis (H_0):

H_0: the generated outcomes of the different algorithms represent samples from continuous distributions with equal medians.

As for H_1, the Mann–Whitney *p*-value represents the *reject* probability of H_0 while it is true (type I error). Therefore, when the *p*-value is less than or equal to α ($\leqslant 0.05$), it indicates the acceptance of H_1, and the rejection H_0. Otherwise, when the *p*-value is strictly higher than α (>0.05), H_0 is accepted while H_1 is rejected.

While the Mann–Whitney test verifies the statistical significance of outcomes, it does not control the difference in magnitude. Hence, we use the non-parametric effect Cliff's delta (δ) [62] to compute the effect size. The value of effect size is statistically interpreted as :

- *negligible* if $| d | < 0.147$,
- *small* if $0.147 \leq | d | < 0.33$,
- *medium* if $0.33 \leq | d | < 0.474$, or
- *high* if $| d | \geq 0.474$.

6.3.3. Parameter setting and tuning

The performance of evolutionary algorithms are significantly influenced by their parameters setting, for a given problem instance. Consequently, we perform, for each algorithm in our comparative study, series of exploratory experiments for the purpose of finding the most adequate population size by varying it, in every simulation run, using the following values: 10, 20, 50, 100, 150, and 200. These values were initially selected as we have manually investigated the minimum, the maximum and the average length of mashups. Then, we noticed that the results of executions, in terms of performance indicators, is no longer statistically significant when the population size goes beyond 50. Therefore, we set the maximum size of the population to 50, as we want to minimize the execution time. We also set the stopping criterion to 100,000 fitness evaluations in order to ensure the comparison fairness between algorithms.

The remaining parameters were tuned by trial and error [63]. We set the probability of the crossover operator to 0.8. As for the mutation probability, we set it to 0.1, while the gene modification probability is set to 0.2. Moreover, For the tuning of MOPSO, the cognitive scaling parameter c_1 is set to 1.5, similarly to the social scaling parameter c_2. For the inertia weighting coefficient w, we gradually decrease it from 1.0 to 0.3. The maximum length of candidate solutions is set to 50 (*i.e.*, the number of recommended services per mashup). Finally, we set the IBEA's archive size to 100.

6.4. Results and discussions

Results for RQ_1. Figs. 6 and 7 report the *precision@k* and *recall@k* results of the *SerFinder* method compared to the four state-of-the-art methods for $k = 3, 5, 10, 15$ and 20. Our results show that *SerFinder* achieves a better performance in comparison

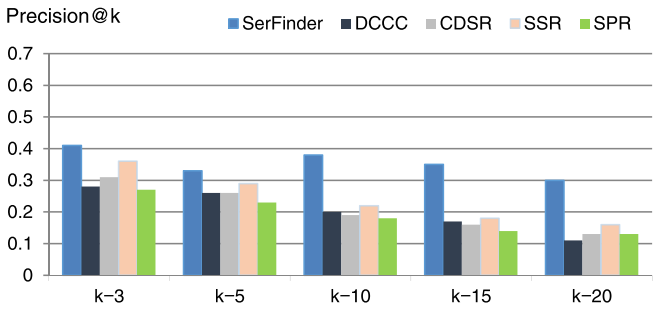

Fig. 6. Precision@k performance results for all approaches.

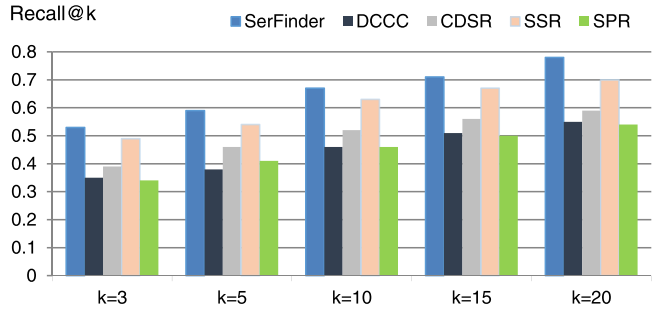

Fig. 7. Recall@k performance results for all approaches.

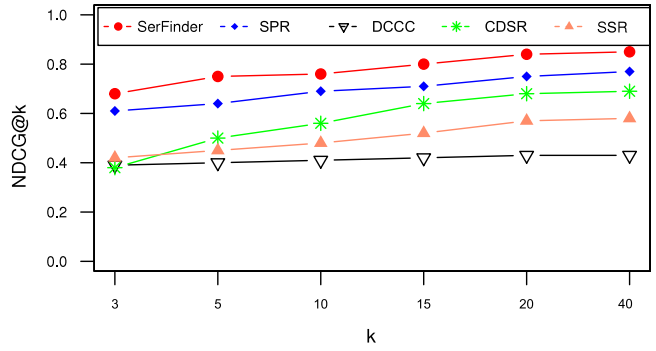

Fig. 8. NDCG@k performance results for all approaches.

with four competing methods DCCC, CDSR, SSR and SPR with an average of 35% and 66% of precision and recall, respectively, $\forall k$. Clearly, SPR turns out to achieve the lowest performance with an average precision of 19% and recall of 45% $\forall k$. We speculate that such drawback of SPR is partly because it considers that frequently composed services having similar functionality are likely to be recommended together. As a result, it recommends similar services. However, similar services, in terms of functionality, are not necessarily belonging to the same composition instead, we observe, on the generated mashups, that services with various functionalities, are also composed together. We observe that DCCC provides better results than SPR but still less than the other methods as it gives higher weight to recommend newborn services rather than popular services.

We also note that both CDSR and SSR outperform SPR and DCCC as they recommend functionally heterogeneous services while considering preferences of composition among service categories. However, since preferences of composition are represented in terms of models at the category level, the modeling tend to be too coarse-grained. This might be one of the aspects that justifies the superiority of our approach which, rather, searches for the optimal trade-off between the three explicit objectives,

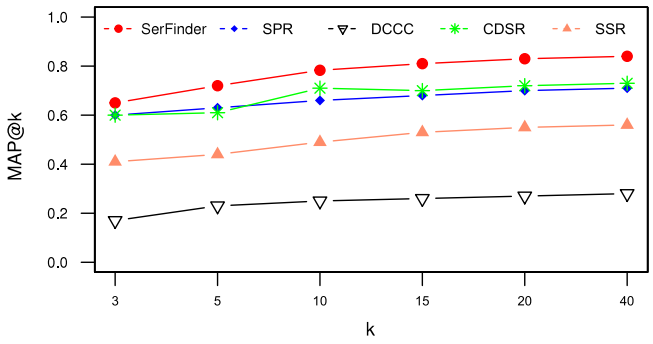

Fig. 9. MAP@k performance results for all approaches.

namely, historical service compositions, functional diversity, and functional requirements matching. Furthermore, our employed fine-grained semantic similarity model at the vocabulary level is more adequate to better capture both the functional matching and diversity.

We also observe from the results of Figs. 6 and 7 that the decrease or increase in both precision and recall with an increase of the value of k is gradual, for which no drastic changes were noted. Moreover, the value of k tends to have more influence on precision for the compared approaches, whereas a slightly higher variation is observed for *SerFinder* in terms of recall. This variation might be justified also by the fact larger values of k could provide higher choices in the list which result into an increased recall score with the cost of a decreased precision. In addition, the diversity objective function seems to be efficient in discovering different areas of the search space for which a higher matching with mashup requirement.

We also speculate that the preprocessing step might have an influence on the accuracy results. While the quality of mashup textual descriptions is an important factor in finding appropriate service recommendation, the preprocessing of such descriptions can have its impact. Indeed, unlike existing approaches, *SerFinder* uses a semantic analysis based on the WordNet dictionary to reduce the heterogeneity in both mashups and services textual descriptions. As part of our future work, we will explore more natural language processing (NLP) techniques to improve the recommendation accuracy of service recommendations and conduct an empirical comparison between different existing NLP techniques to better assess the impact of the mashups pre-processing.

It is also noteworthy to mention that while we did not find a clear pattern of convergence towards a particular mashup length, we noticed that the mashup's converging length has a positive correlation of 0.32 with the actual mashup's requirement description length (number of words), and a correlation of 0.37 with the number of tags the mashup has. Moreover, we observed that the length of the mashups tends to converge to the fixed maximum mashup length when the number of mashup tags is greater than 3. As part of our future work, we plan to investigate more on the impact of different factors on the mashup length. We also plan to incorporate the developers preferences on the number of recommended mashups. Indeed, while it is desirable to provide multiple recommendations to developers, typically developers are unlikely to go through a large list of recommended services

Results for RQ_2. Figs. 8 and 9 report the *NDCG@k* and *MAP@k* results achieved by *SerFinder* and the four competing methods. In terms of *NDCG@k*, results show that *SerFinder* achieves superior performance in recommending accurate services in the first few recommendations of 0.76 for the top 10 recommended services, i.e., $k = 10$. Whereas, DCCC achieves the lowest performance

(0.41 for $k = 10$) which could be due to the combination of coarse-grained category tags with recent collaboration records, thus failing in capturing more popular and/or functionally relevant services. SSR achieves better *NDCG@k* results than DCCC which indicate that the employed hypergraph-based model for measuring composition patterns among services is interesting. We also observe that CDSR and SPR achieved better performance than DCCC and SSR by providing better mechanisms to avoid the redundancy issue such as recommending services from distinct service category groups.

Similar comparison results were obtained in terms *MAP@k*. As can be seen from Fig. 9, *SerFinder* achieves a much higher *MAP@k* score than the four other approaches. We observe also that CDSR and SPR are still achieving better performance than DCCC and SSR. The possible reason of such clear superiority of *SerFinder* is that the problem is by nature a search-based optimization problem that is well-suited to meta-heuristic algorithms such as NSGA-II to provide high degree of objectives satisfaction. Indeed, *SerFinder* takes advantage of both multi-objective search formulation as well as the fine-granularity vocabulary similarity measure.

Results for RQ_3. To answer this third research question (RQ3), the obtained Pareto fronts were compared for statistically significant differences of our algorithm NSGA-II with random search (RS) as well as four popular multi-objective search algorithms, MOPSO, IBEA, PEA2, and MOEA/D. To this end, our comparison is based on six different performance metrics as defined in Section 6.3.1.

Tables 2 and 3 show the average results achieved by the six algorithms for the considered mashups benchmark collected from ProgrammableWeb as described in Table 1, and the statistical significance and effect size comparative results, respectively. Furthermore, as illustrative examples, we report the results of 10 randomly selected mashups with different sizes from our benchmark, namely Magic Mosaic,[6] UsersThink,[7] Feelter,[8] Rainbow Data Sync,[9] Slick Deals Price Tracker,[10] Soccer Shots,[11] AyeDeals,[12] Jobfinder,[13] UK Beaches,[14] Early Movie Trailers.[15]

As shown in Tables 2 and 3, NSGA-II clearly outperforms RS in all the benchmark mashups with a 'high' Cliff's delta effect size in terms of the six performance indicators *HV*, *GD*, *IGD*, *GS*, *PFS*, and *IC*. This is mainly due to such a large search space to be explored to find suitable combinations of service APIs. This requires an intelligent metaheuristic search rather than a random search.

Furthermore, we observe that NSGA-II stands out in terms of all the quality indicators except for the Generational Spread (SD) where IBEA tends to provide slightly better results on average even though the results are not statistically different (cf., Table 3). The remarkable gains achieved by NSGA-II become obvious with the bigger and more complex mashups where more than four service APIs are used, e.g., the mashups *Rainbow Data Sync*, *Soccer Shots*, *AyeDeals*, etc. In particular, we notice that, for smaller mashups, better average spread results were achieved by IBEA, even though it is not based on crowd pruning measures in its dominance criteria. As shown in Table 3, while IBEA provides better average spread results (GS), there is no statistically significant results in terms of any performance indicator, for which IBEA is better than NSGA-II including the GS performance indicator.

The remaining algorithms (MOPSO, SPEA2, MOEA/D, and IBEA) perform moderately with smaller and medium size mashups. In particular, with medium size mashups, *e.g.*, *UsersThink*, MOEA/D achieves slightly better performance in therms of the inverted generational distance (IGD) metric compared to NSGA-II and the other algorithms.

To conclude, the obtained results provide evidence that NSGA-II is the best search technique for the automated mashup composition problem, particularly for the larger and more complex mashups. Consequently, we can concur that there is an empirical evidence that our formulation successfully pass the baseline sanity check (RQ3).

7. Threats to validity

This section reports potential factors that may impact the methodology of our work along with observed outcomes. These factors are identified in three main types : construct, conclusion, and external validity.

Construct threats to validity could arise due to the employed measures used in our evaluation, *e.g.*, NDCG@k, and MAP@k. While the adoption of these metrics is popular to assess the efficiency of algorithms and to conduct comparative studies [11,22,64], and in particular in mashup recommendation state-of-the-art approaches [3,6,8,16], we cannot neglect the existence of a slight bias towards the usage of the aforementioned metrics. Yet, these metrics are independent from the algorithms under comparison and they do not favor an approach over another. Furthermore, the evaluation of recommended results can be seen as a ranking problem, where each algorithm is rewarded based on how many correct items in the upper rank of its recommendation [64]. For example, in the problem of service APIs recommendation, a mashup might not use a particular service API due to several factors that can be as simple as the mashup's developer does not know about the existence of such a service API, especially with the explosion of available service APIs. Other reasons can be related to the time and budget limitations that typical developers experience on a regular basis. To better analyze the impact of these constraints, we need to conduct an empirical study on *SerFinder* with developers in an industrial context.

Moreover, while our goal in the experimental study was to use a variety of performance indicators to cover more performance aspects of the compared algorithms (*i.e.*, convergence, distribution, and spread), each indicator typically focuses on some particular characteristics while ignoring information in others. Indeed, each indicator has its particular characteristics and no indicators can substitute others completely. Therefore, a fixed number of indicators seems not sufficient to make a comprehensive measure for MOEAs [65,66]. To address this issue, we plan in our future work to use ensemble performance indicators [67–70].

Conclusion validity: Due to the stochastic nature of the implemented algorithms, we used the Mann–Whitney statistical test and Cliff's delta effect size measures more than 31 repeated executions for each algorithm with a 95% confidence level. The aim is to assess there exists statical differences between the measurements for different evaluation datasets. The suitability of the used statistical non-parametric methods with data ordinality, along with no assumption on their distribution raises our confidence about the significance of the analyzed statistical relationships.

External validity: It concerns the possible biases related to the choice of experimental subjects/objects. Although we have analyzed a large scale dataset of 3,465 subject mashups and 15,386 services collected from the largest service API repository, namely *ProgrammableWeb*, that have a high degree of diversity in terms

6 https://www.programmableweb.com/mashup/magic-mosaic.
7 https://www.programmableweb.com/mashup/usersthink.
8 https://www.programmableweb.com/mashup/feelter.
9 https://www.programmableweb.com/mashup/rainbow-data-sync-ismart-solutions.
10 https://www.programmableweb.com/mashup/slick-deals-price-tracker.
11 https://www.programmableweb.com/mashup/soccer-shots.
12 https://www.programmableweb.com/mashup/ayedeals.
13 https://www.programmableweb.com/mashup/jobfinder.
14 https://www.programmableweb.com/mashup/uk-beaches.
15 https://www.programmableweb.com/mashup/early-movie-trailers.

Table 2
Experimental results achieved the six algorithms, NSGA-II, IBEA, MOPSO, SPEA2, MOEA/D and RS in terms of hypervolume (HV), generational distance (GD), inverted generational distance (IGD), generalized spread (GS), Pareto front size (PFS) and contribution (IC) applied to all the ProgrammableWeb benchmark and 10 random mashup samples.

Mashup	#services	Performance metric	NSGA-II	IBEA	MOPSO	SPEA2	MOEA/D	RS
ProgrammableWeb Benchmark (average)	-	HV	2.477E−01	2.456E−01	2.370E−01	2.399E−01	2.388E−01	6.871E−02
		GD	4.332E−03	5.329E−03	4.810E−03	6.983E−03	5.987E−03	1.999E−01
		IGD	3.654E−03	4.255E−03	3.789E−03	4.875E−03	3.896E−03	2.336E−02
		GS	6.827E−01	6.974E−01	6.046E−01	5.441E−01	6.023E−01	1.056E−01
		PFS	86.40	74.50	81.20	69.30	72.20	23.10
		IC	4.559E−01	4.029E−01	4.133E−01	3.949E−01	3.885E−01	6.066E−02
Magic Mosaic	4	HV	3.013E−01	2.836E−01	2.856E−01	2.569E−01	2.190E−01	1.024E−01
		GD	6.509E−03	7.900E−03	6.900E−03	9.800E−03	1.080E−02	3.450E−01
		IGD	4.259E−03	6.890E−03	4.899E−03	5.015E−03	4.987E−03	8.541E−02
		GS	8.974E−01	6.987E−01	7.045E−01	8.148E−01	7.845E−01	2.369E−02
		PFS	87.30	76.90	84.08	56.13	75.30	21.40
		IC	4.568E−01	4.065E−01	5.442E−01	3.654E−01	3.432E−01	9.834E−02
UsersThink	5	HV	1.981E−01	1.357E−01	2.006E−01	1.699E−01	1.870E−01	3.590E−02
		GD	3.901E−03	3.901E−03	3.901E−03	3.901E−03	3.901E−03	3.901E−03
		IGD	6.987E−04	2.547E−03	9.014E−04	8.954E−04	6.930E−04	2.302E−02
		GS	4.896E−01	4.266E−01	4.459E−01	3.987E−01	4.025E−01	1.459E+00
		PFS	127.30	112.90	119.30	93.00	69.30	77.80
		IC	3.445E−01	3.445E−01	3.445E−01	3.445E−01	3.445E−01	3.445E−01
Feelter	4	HV	2.375E−01	2.325E−01	2.156E−01	1.989E−01	2.070E−01	1.227E−02
		GD	1.876E−03	2.011E−03	2.763E−03	9.981E−03	3.180E−03	8.201E−01
		IGD	1.987E−03	2.697E−03	2.015E−03	3.698E−03	6.970E−03	1.004E−02
		GS	5.789E−01	5.897E−01	5.148E−01	4.987E−01	4.587E−01	8.974E−02
		PFS	37.90	22.60	28.40	31.70	32.90	15.90
		IC	4.553E−01	4.322E−01	4.098E−01	4.211E−01	3.909E−01	9.923E−02
Rainbow Data Sync	10	HV	7.560E−02	6.240E−02	7.159E−02	3.699E−02	6.004E−02	1.027E−02
		GD	2.999E−03	3.098E−03	8.763E−03	3.387E−03	4.409E−03	1.124E+00
		IGD	5.489E−04	6.099E−04	7.890E−04	8.974E−04	5.901E−04	3.690E−03
		GS	6.987E−01	6.021E−01	6.270E−01	5.099E−01	6.026E−01	1.037E−01
		PFS	169.30	142.60	146.70	105.30	96.80	42.50
		IC	6.332E−01	5.332E−01	6.102E−01	6.100E−01	5.443E−01	1.988E−01
Slick Deals Price Tracker	5	HV	1.656E−01	1.604E−01	1.549E−01	1.370E−01	1.369E−01	9.987E−02
		GD	3.467E−03	6.550E−03	3.980E−03	4.332E−03	4.098E−03	7.023E−01
		IGD	8.014E−04	9.470E−04	8.369E−04	8.907E−04	9.914E−04	5.470E−03
		GS	1.070E+00	1.034E+00	1.061E+00	1.015E+00	1.027E+00	5.070E−01
		PFS	47.20	41.40	52.40	29.40	36.00	14.20
		IC	5.983E−01	5.122E−01	5.845E−01	4.663E−01	5.543E−01	1.221E−01
Soccer Shots	5	HV	2.989E−01	2.002E−01	2.599E−01	2.799E−01	2.270E−01	4.896E−02
		GD	2.893E−04	4.332E−04	2.989E−04	3.001E−04	4.332E−04	2.315E−01
		IGD	2.780E−04	3.024E−04	2.914E−04	3.214E−04	2.987E−04	1.087E−03
		GS	8.070E−01	6.605E−01	7.449E−01	7.471E−01	6.887E−01	1.070E−02
		PFS	18.40	17.90	18.10	16.40	17.10	11.20
		IC	3.450E−01	3.098E−01	3.233E−01	3.122E−01	2.984E−01	9.086E−02
AyeDeals	5	HV	1.867E−01	1.488E−01	1.699E−01	1.815E−01	1.802E−01	2.237E−02
		GD	5.180E−03	6.554E−03	5.309E−03	6.554E−03	5.889E−03	1.098E−01
		IGD	4.871E−03	5.011E−03	4.907E−03	6.587E−03	5.471E−03	8.971E−03
		GS	1.826E−01	1.193E−01	1.457E−01	1.599E−01	1.415E−01	8.257E−02
		PFS	61.10	52.50	66.40	54.00	60.30	23.50
		IC	4.399E−01	4.122E−01	4.233E−01	4.094E−01	3.995E−01	9.949E−02
Jobfinder	4	HV	2.237E−01	2.156E−01	2.048E−01	2.148E−01	1.987E−01	1.148E−01
		GD	2.341E−03	2.877E−03	2.399E−03	3.200E−03	3.520E−03	1.776E+00
		IGD	1.657E−04	3.699E−04	1.874E−04	1.987E−04	1.789E−04	9.874E−04
		GS	4.875E−01	3.699E−01	4.145E−01	3.657E−01	2.987E−01	1.257E−01
		PFS	35.40	33.10	34.20	31.40	33.90	12.30
		IC	3.452E−01	3.103E−01	2.432E−01	3.200E−01	2.952E−01	1.003E−01
UK Beaches	3	HV	1.509E−01	1.497E−01	1.370E−01	1.237E−01	1.365E−01	5.801E−02
		GD	8.776E−02	4.556E−02	4.121E−02	5.231E−02	5.902E−02	1.801E+00
		IGD	8.473E−03	9.874E−03	8.987E−03	9.874E−03	9.024E−03	5.699E−02
		GS	8.045E−01	8.270E−01	7.895E−01	7.367E−01	7.046E−01	7.045E−02
		PFS	78.80	78.30	71.50	66.00	72.10	24.90
		IC	4.094E−01	3.978E−01	3.763E−01	3.898E−01	3.984E−01	9.485E−02
Early Movie Trailers	3	HV	1.718E−01	1.656E−01	1.500E−01	1.570E−01	1.648E−01	1.066E−01
		GD	3.412E−03	4.087E−03	3.899E−03	4.987E−03	5.335E−03	5.331E−01
		IGD	3.697E−04	5.746E−04	4.016E−04	3.987E−04	5.501E−04	2.897E−03
		GS	5.887E−01	5.949E−01	5.211E−01	5.015E−01	4.887E−01	1.479E−01
		PFS	49.20	46.70	50.10	44.80	46.30	30.10
		IC	5.450E−01	5.023E−01	5.122E−01	4.856E−01	4.873E−01	9.940E−02

of size, application domains, and IT development companies, we cannot claim that our results can be generalized beyond these subject mashups to other service repositories or other industrial contexts. To address this issue we plan to perform new

Table 3
Statistical analysis results p-value ($\alpha = 0.05$) and Cliff's Delta effect size achieved by the algorithm NSGA-II in comparison with each of IBEA, MOPSO, SPEA2, MOEA/D and RS in terms of the six performance metrics hypervolume (HV), generational distance (GD), inverted generational distance (IGD), generalized spread (GS), Pareto front size (PFS) and contribution (IC) applied to all the ProgrammableWeb benchmark.

Metric	Stat.	NSGA-II vs. IBEA	NSGA-II vs. MOPSO	NSGA-II vs.SPEA2	NSGA-II vs. MOEA/D	NSGA-II vs. RS
HV	p-value	2.922E−03	5.453E−03	2.334E−03	4.431E−03	5.031E−09
	Effect size	High	High	High	High	High
GD	p-value	9.384E−03	2.938E−04	4.450E−04	4.431E−03	4.230E−08
	Effect size	High	High	High	High	High
IGD	p-value	4.822E−03	3.842E−04	5.554E−03	9.884E−03	8.430E−08
	Effect size	High	High	High	High	High
GS	p-value	6.456E−02	8.443E−03	1.223E−03	9.430E−04	1.374E−08
	Effect size	Small	High	High	High	High
PFS	p-value	8.344E−03	6.320E−03	8.454E−04	4.940E−05	1.934E−09
	Effect size	High	High	High	High	High
IC	p-value	9.342E−03	3.930E−05	9.942E−04	3.342E−04	4.332E−07
	Effect size	High	High	High	High	High

experiments in an industrial context in order to evaluate our automated mashups recommendation approach from a developer's perspective.

8. Conclusion and future work

In this paper, we have proposed *SerFinder*, a new multi-objective search-based approach for service set recommendation. Specifically, *SerFinder* adopts the non-dominated sorting genetic algorithm (NSGA-II) to find the best trade-off between service historical co-usage, functional matching with the requirement of a mashup, and functional diversity of the recommended services. We also present a lower-granularity semantic similarity model to enhance the functional matching. Our experimental study demonstrates that *SerFinder* achieves significant improvement in recommendation accuracy as compared to four recent state-of-the-art methods for service mashup recommendation. We also conducted an empirical study to assess the efficiency of the used algorithm NSGA-II compared to other existing multi-objective algorithms. The statistical analysis of the obtained results provide evidence that NSGA-II has the best performance among all algorithms.

As part of our future work, we plan to extend *SerFinder* by incorporating a temporal dimension to better capture the intricate service composition patterns. Indeed, service ecosystems change frequently over time as new services may appear and others will be deprecated. We also plan to put the developer in the loop to guide the search process through an interactive services recommendation method.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.asoc.2019.105830.

Acknowledgments

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Research Startup (2) 2016 Grant G00002211 funded by UAE University, United Arab Emirates, and the JSPS, Japan KAKENHI Grant Number 18H04094.

References

[1] V. Andrikopoulos, S. Benbernou, M.P. Papazoglou, On the evolution of services, IEEE Trans. Softw. Eng. 38 (3) (2012) 609–628.
[2] X. Liu, Y. Hui, W. Sun, H. Liang, Towards service composition based on mashup, in: Services, 2007 IEEE Congress on, IEEE, 2007, pp. 332–339.
[3] B. Xia, Y. Fan, W. Tan, K. Huang, J. Zhang, C. Wu, Category-aware API clustering and distributed recommendation for automatic mashup creation, IEEE Trans. Serv. Comput. 8 (5) (2015) 674–687.
[4] X. Dong, A. Halevy, J. Madhavan, E. Nemes, J. Zhang, Similarity search for web services, in: International Conference on Very Large Data Bases-Volume 30, VLDB Endowment, 2004, pp. 372–383.
[5] C. Li, R. Zhang, J. Huai, X. Guo, H. Sun, A probabilistic approach for web service discovery, in: IEEE International Conference on Services Computing, IEEE, 2013, pp. 49–56.
[6] W. Gao, J. Wu, A novel framework for service set recommendation in mashup creation, in: IEEE International Conference on Web Services (ICWS), IEEE, 2017, pp. 65–72.
[7] W. Gao, L. Chen, J. Wu, H. Gao, Manifold-learning based API recommendation for mashup creation, in: International Conference on Web Services, 2015, pp. 432–439.
[8] J. Cao, Y. Lu, N. Zhu, Service package recommendation for mashup development based on a multi-level relational network, in: Lecture Notes in Computer Science, in: LNCS, vol. 9936, Springer, Cham, 2016, pp. 666–674.
[9] Z. Zheng, H. Ma, M.R. Lyu, I. King, Wsrec: A collaborative filtering based web service recommender system, in: Int. Conference on Web Services, 2009, pp. 437–444.
[10] D. Ardagna, B. Pernici, Global and local qos guarantee in web service selection, in: Business Process Management Workshops, vol. 3812, Springer, 2005, pp. 32–46.
[11] A. Ouni, R.G. Kula, M. Kessentini, T. Ishio, D.M. German, K. Inoue, Search-based software library recommendation using multi-objective optimization, Inf. Softw. Technol. 83 (2017) 55–75.
[12] W. Ahmed, Y. Wu, W. Zheng, Response time based optimal web service selection, IEEE Trans. Parallel Distrib. Syst. 26 (2) (2015) 551–561.
[13] Z. Zheng, H. Ma, M.R. Lyu, I. King, Collaborative web service qos prediction via neighborhood integrated matrix factorization, IEEE Trans. Serv. Comput. 6 (3) (2013) 289–299.
[14] Z. Gao, Y. Fan, C. Wu, W. Tan, J. Zhang, Y. Ni, B. Bai, S. Chen, SeCo-LDA: Mining service co-occurrence topics for recommendation, in: IEEE International Conference on Web Services (ICWS), IEEE, 2016, pp. 25–32.
[15] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[16] J. Zhang, Y. Fan, W. Tan, J. Zhang, Recommendation for newborn services by divide-and-conquer, in: IEEE 24th International Conference on Web Services, 2017, pp. 57–64.
[17] S. Murugesan, Understanding web 2.0, IT Prof. Mag. 9 (4) (2007) 34.
[18] C. Cappiello, F. Daniel, M. Matera, A quality model for mashup components, in: International Conference on Web Engineering, Springer, 2009, pp. 236–250.
[19] C. Li, R. Zhang, J. Huai, H. Sun, A novel approach for API recommendation in mashup development, in: IEEE International Conference on Web Services (ICWS), 2014, pp. 289–296.
[20] Q. Yu, Z. Zheng, H. Wang, Trace norm regularized matrix factorization for service recommendation, in: IEEE International Conference on Web Services (ICWS), 2013, pp. 34–41.

[21] T. Liang, L. Chen, J. Wu, G. Xu, Z. Wu, SMS: A framework for service discovery by incorporating social media information, IEEE Trans. Serv. Comput. (2016).

[22] F. Thung, D. Lo, J. Lawall, Automated library recommendation, in: 20th Working Conference on Reverse Engineering (WCRE), 2013, pp. 182–191.

[23] M.A. Saied, A. Ouni, H. Sahraoui, R.G. Kula, K. Inoue, D. Lo, Improving reusability of software libraries through usage pattern mining, J. Syst. Softw. 145 (2018) 164–179.

[24] A. Ramírez, J.A. Parejo, J.R. Romero, S. Segura, A. Ruiz-Cortés, Evolutionary composition of QoS-aware web services: a many-objective perspective, Expert Syst. Appl. 72 (2017) 357–370.

[25] A. de Campos Jr, A.T. Pozo, S.R. Vergilio, T. Savegnago, Many-objective evolutionary algorithms in the composition of web services, in: Eleventh Brazilian Symposium on Neural Networks (SBRN), IEEE, 2010, pp. 152–157.

[26] M. Suciu, D. Pallez, M. Cremene, D. Dumitrescu, Adaptive MOEA/D for QoS-based web service composition, in: European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, 2013, pp. 73–84.

[27] Q. Zhang, H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (6) (2007) 712–731.

[28] S. Kukkonen, J. Lampinen, GDE3: The third evolution step of generalized differential evolution, in: Evolutionary Computation, 2005. the 2005 IEEE Congress on, vol. 1, IEEE, 2005, pp. 443–450.

[29] Y. Yu, H. Ma, M. Zhang, F-MOGP: A novel many-objective evolutionary approach to QoS-aware data intensive web service composition, in: Evolutionary Computation (CEC), 2015 IEEE Congress on, 2015, pp. 2843–2850.

[30] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints, IEEE Trans. Evol. Comput. 18 (4) (2014) 577–601.

[31] W. Mkaouer, M. Kessentini, A. Shaout, P. Koligheu, S. Bechikh, K. Deb, A. Ouni, Many-objective software remodularization using NSGA-III, ACM Trans. Softw. Eng. Methodol. 24 (3) (2015) 17.

[32] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm, TIK-report 103, Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), 2001.

[33] W. Pan, C. Chai, Structure-aware mashup service clustering for cloud-based internet of things using genetic algorithm based clustering algorithm, Future Gener. Comput. Syst. (2018).

[34] X. Wu, L. Zhang, Y. Yu, Exploring social annotations for the semantic web, in: Proceedings of the 15th International Conference on World Wide Web, ACM, 2006, pp. 417–426.

[35] M. Harman, S. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Comput. Surv. 45 (1) (2012) 11.

[36] A. Ouni, M. Kessentini, K. Inoue, M. O.C.inneide, Search-based web service antipatterns detection, IEEE Trans. Serv. Comput. PP (99) (2016) http://dx.doi.org/10.1109/TSC.2015.2502595.

[37] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, M.S. Hamdi, Improving multi-objective code-smells correction using development history, J. Syst. Softw. (2015).

[38] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, K. Deb, Multi-criteria code refactoring using search-based software engineering: An industrial case study, ACM Trans. Softw. Eng. Methodol. 25 (3) (2016) 23.

[39] A. Ouni, M. Kessentini, H. Sahraoui, M. Boukadoum, Maintainability defects detection and correction: a multi-objective approach, Autom. Softw. Eng. 20 (1) (2013) 47–79.

[40] S. Boukharata, A. Ouni, M. Kessentini, S. Bouktif, H. Wang, Improving web service interfaces modularity using multi-objective optimization, Autom. Softw. Eng. 26 (2) (2019) 275–312.

[41] A. Ouni, H. Wang, M. Kessentini, S. Bouktif, K. Inoue, A hybrid approach for improving the design quality of web service interfaces, ACM Trans. Internet Technol. 19 (1) (2018) 4.

[42] A. Ouni, M. Kessentini, M. Ó Cinnéide, H. Sahraoui, K. Deb, K. Inoue, MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells, J. Softw. Evol. Process 29 (5) (2017) e1843.

[43] A. Ouni, M. Kessentini, H. Sahraoui, M.S. Hamdi, Search-based refactoring: Towards semantics preservation, in: IEEE International Conference on Software Maintenance (ICSM), 2012, pp. 347–356.

[44] I. Saidani, A. Ouni, M.W. Mkaouer, M.A. Saied, Towards automated microservices extraction using muti-objective evolutionary search, in: 17th International Conference on Service-Oriented Computing (ICSOC), 2019.

[45] SerFinder: dataset used in the study, 2019, Available at : https://github.com/ouniali/SerFinder.

[46] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, T.-Y. Liu, A theoretical analysis of NDCG ranking measures, in: 26th Annual Conference on Learning Theory (COLT 2013), vol. 8, 2013, p. 6.

[47] D.C. Karnopp, Random search techniques for optimization problems, Automatica 1 (2) (1963) 111–121.

[48] E. Zitzler, S. Künzli, Indicator-based selection in multiobjective search, in: International Conference on Parallel Problem Solving from Nature, Springer, 2004, pp. 832–842.

[49] C.A.C. Coello, G.T. Pulido, M.S. Lechuga, Handling multiple objectives with particle swarm optimization, IEEE Trans. Evol. Comput. 8 (3) (2004) 256–279.

[50] M. Harman, B.F. Jones, Search-based software engineering, Inf. Softw. Technol. 43 (14) (2001) 833–839.

[51] D.A. Van Veldhuizen, G.B. Lamont, Multiobjective Evolutionary Algorithm Research: A History and Analysis, Citeseer, 1998.

[52] C.A. Coello, N.C. Cortés, Solving multiobjective optimization problems using an artificial immune system, Genet. Program. Evol. Mach. 6 (2) (2005) 163–190.

[53] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, V.G. Da Fonseca, Performance assessment of multiobjective optimizers: An analysis and review, IEEE Trans. Evol. Comput. 7 (2) (2003) 117–132.

[54] D. Brockhoff, T. Friedrich, F. Neumann, Analyzing hypervolume indicator based algorithms, in: International Conference on Parallel Problem Solving from Nature, Springer, 2008, pp. 651–660.

[55] K. Deb, M. Mohan, S. Mishra, Towards a quick computation of well-spread pareto-optimal solutions, in: International Conference on Evolutionary Multi-Criterion Optimization, Springer, 2003, pp. 222–236.

[56] C. Henard, M. Papadakis, M. Harman, Y. Le Traon, Combining multi-objective search and constraint solving for configuring large software product lines, in: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, 2015, pp. 517–528.

[57] H. Meunier, E.-G. Talbi, P. Reininger, A multiobjective genetic algorithm for radio network optimization, in: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 1, 2000, pp. 317–324.

[58] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Software Engineering (ICSE), 2011 33rd International Conference on, IEEE, 2011, pp. 1–10.

[59] A. Arcuri, G. Fraser, On parameter tuning in search based software engineering, in: Search Based Software Engineering, Springer, 2011, pp. 33–47.

[60] Y. Hochberg, A sharper Bonferroni procedure for multiple tests of significance, Biometrika 75 (4) (1988) 800–802.

[61] A. Arcuri, L. Briand, A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering, Softw. Test. Verif. Reliab. 24 (3) (2014) 219–250.

[62] N. Cliff, Dominance statistics: Ordinal analyses to answer ordinal questions, Psychol. Bull. 114 (3) (1993) 494.

[63] A.E. Eiben, S.K. Smit, Parameter tuning for configuring and analyzing evolutionary algorithms, Swarm Evol. Comput. 1 (1) (2011) 19–31.

[64] I. Avazpour, T. Pitakrat, L. Grunske, J. Grundy, Dimensions and metrics for evaluating recommendation systems, in: Recommendation Systems in Software Engineering, Springer, 2014, pp. 245–273.

[65] E. Zitzler, Evolutionary algorithms for multiobjective optimization: Methods and applications, vol. 63, 1999.

[66] M. Ravber, M. Mernik, M. Črepinšek, The impact of quality indicators on the rating of multi-objective evolutionary algorithms, Appl. Soft Comput. 55 (2017) 265–275.

[67] G.G. Yen, Z. He, Performance metric ensemble for multiobjective evolutionary algorithms, IEEE Trans. Evol. Comput. 18 (1) (2013) 131–144.

[68] Z. He, G.G. Yen, Many-objective evolutionary algorithm: Objective space reduction and diversity improvement, IEEE Trans. Evol. Comput. 20 (1) (2015) 145–160.

[69] M. Ravber, M. Mernik, M. Črepinšek, Ranking multi-objective evolutionary algorithms using a chess rating system with quality indicator ensemble, in: IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 1503–1510.

[70] M. Črepinšek, S.-H. Liu, M. Mernik, Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them, Appl. Soft Comput. 19 (2014) 161–170.