Optimizing Selection of Competing Services with Probabilistic Hierarchical Refinement

Tian Huat Tan[2] Manman Chen[1] Jun Sun[2] Yang Liu[3]
Étienne André[4] Yinxing Xue[3] Jin Song Dong[1]
[2]SUTD (Singapore), [1]NUS (Singapore), [3]NTU (Singapore), [4]Université Paris 13 (France)

ABSTRACT

Recently, many large enterprises (e.g., Netflix, Amazon) have decomposed their monolithic application into services, and composed them to fulfill their business functionalities. Many hosting services on the cloud, with different Quality of Service (QoS) (e.g., availability, cost), can be used to host the services. This is an example of competing services. QoS is crucial for the satisfaction of users. It is important to choose a set of services that maximize the overall QoS, and satisfy all QoS requirements for the service composition. This problem, known as *optimal service selection*, is NP-hard. Therefore, an effective method for reducing the search space and guiding the search process is highly desirable. To this end, we introduce a novel technique, called *Probabilistic Hierarchical Refinement (*PROHR*)*. PROHR effectively reduces the search space by removing competing services that cannot be part of the selection. PROHR provides two methods, probabilistic ranking and hierarchical refinement, that enable smart exploration of the reduced search space. Unlike existing approaches that perform poorly when QoS requirements become stricter, PROHR maintains high performance and accuracy, independent of the strictness of the QoS requirements. PROHR has been evaluated on a publicly available dataset, and has shown significant improvement over existing approaches.

1 Introduction

Web services provide an affordable and adaptable framework that can produce a significantly lower cost of ownership for the enterprises over time. Web services make use of open standards, such as WSDL [18] and SOAP [19], which enable the interaction among heterogeneous applications. This facilitates the dynamic integration among applications without costly and time-consuming programming.

Enterprises make use of service composition to integrate multiple services for complex interactions. The composed Web service is called a *composite service*, whereas the services that are used in the composed Web service are called *component services*. For each of the component services, there are usually many *competing services*, that provide the same functionality but different levels of Quality of Service (QoS) (e.g., response time, availability, cost) for selection. We provide two examples on such competing services.

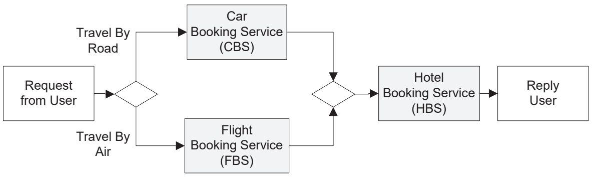

Travel By Road
Car Booking Service (CBS)
Request from User
Travel By Air
Flight Booking Service (FBS)
Hotel Booking Service (HBS)
Reply User

Figure 1: Travel Agency Service (TAS)

1. Many big players in the market (e.g., Netflix, Amazon, and Microsoft Azure) have adopted microservice architecture [3] by decomposing their existing monolithic applications into smaller, and highly decoupled services (also known as microservices). These services are then composed for fulfilling their business requirements. For example, Netflix decomposed their monolithic DVD rental application into services that work together, and that stream digital entertainment to millions of Netflix customers every day. Services of Netflix are hosted in a cloud provided by Amazon EC2 [2], which offers about 40 instance types (e.g., t2.micro, c4.4xlarge) for selection. All instance types provide the same functionality while having different QoS.

2. Many enterprises outsource some of their business tasks to external service providers (e.g., car booking service), for lowering the cost of operations. However, many external services may offer the same functionality, but with a different level of QoS. Figure 1 shows an example of a simple composite service, named Travel Agency Service (TAS). If the user requests to travel by road, the Car Booking Service (CBS) will be invoked to book a car for reaching the destination. If the user requests to travel by air, the Flight Booking Service (FBS) will be invoked to book the flight for reaching the nearest airport to the destination. In either case, the Hotel Booking Service (HBS) will be invoked to book the hotel. Finally, the system replies to the user with the status of booking. There may be many competing services for CBS, FBS and HBS.

QoS attributes, such as response time, availability, cost, etc., provide quantitative indicators on non-functional aspects of the composite service. The Service Level Agreement (SLA) is often used as an agreement between the composite service providers and the service users; it specifies the expected QoS level of the composite service. The SLA can be expressed as constraints over the QoS, e.g., the response time must be smaller than one second, or the availability must be higher than 99.99%. A composite service user typically has limited knowledge on the internal structure of the composite service, and component services that are involved. The service user is only concerned with the *end-to-end QoS* of the composite ser-

ICSE '16, May 14-22, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-3900-1/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2884781.2884861

vice, i.e., the QoS at the composite service level. The user requirements of QoS for the composite service can be specified as *global constraints*. Our objective is to select a set of concrete services that can satisfy the global constraints on the composite service, while maximizing the end-to-end QoS.

Following the SOA principles, a composite service contains a set of abstract services which have their functionalities defined. Given an abstract service (e.g., a hotel booking service), a concrete service (e.g., the Hilton Hotel booking service) is selected from a set of competing services to realize the abstract service at runtime. The problem of selecting concrete services while maximizing end-to-end QoS, known as the *optimal selection problem*, has received considerable attentions over the last decade [9, 7, 11, 10, 37]. The optimal selection problem becomes increasingly challenging as the number of concrete services increases. Given a composite service with 10 abstract services executed in a sequential manner, with each abstract service having 10 concrete service candidates, there are 10^{10} combinations to explore. In fact, it has been shown that the problem is NP-hard [10].

To address the scalability problem, approaches based on selecting representative component services have been proposed. We denote the optimality of QoS for a concrete component service with respect to other functional-equivalent services as *local optimality*. Existing state-of-the-art representative services approaches [9, 7] provide search space reduction based on local optimality of the services. Although it has been shown to provide good performance for a large number of candidates with less restrictive global constraints, it suffers from significant performance degradation when the global constraints become more restrictive. The primary reason is that they do not take the global constraints into account during service selection. In addition, the algorithm for selecting representative services (i.e., *k*-means clustering) has high worst case complexity (i.e., NP-hard).

In this work, we propose a novel approach, called *Probabilistic Hierarchical Refinement* (PROHR), to address this problem. The key idea behind PROHR is the usage of both global constraints and local optimality to achieve search space reduction. PROHR is divided into three stages, i.e., preprocessing stage, probabilistic ranking stage, and hierarchical refinement stage. In the *preprocessing stage*, the services that are verified impossible to be part of the optimal selection are pruned from the search space. Subsequently, in the *probabilistic ranking stage*, services are ranked according to both local optimality value and constraint satisfaction probability. Intuitively, local optimality value and constraint satisfaction probability of a service quantify the optimality of the service, and the likelihood that the service can contribute to the conformance of global constraints respectively. Following that, in the *hierarchical refinement stage*, initial representative services from the top hierarchy of competing services are chosen. The number of representative services is adjusted adaptively according to the constraint satisfaction probability of representative services. The representative services are then refined iteratively with more services from lower hierarchies until a solution for optimal service selection is found. In our approach, our representative selection has much lower worst case complexity than the existing state-of-the-art approaches [9, 7]. The complexity analysis of the algorithms is provided in Section 3.5.

Our main contributions are summarized below.

1. We introduce a pruning method called *constraint pruning*, that can effectively discard the service candidates that cannot satisfy the global constraints.
2. We propose probabilistic ranking and hierarchical refinement methods, which enable smart exploration of search space by effectively leveraging the information of global constraints and QoS for competing services.
3. We evaluate the benefits brought by the PROHR using a publicly available dataset. The results have shown significant improvement over existing approaches.

Outline. Section 2 introduces the QoS compositional model and necessary terminologies. Section 3 presents PROHR. Section 4 provides the evaluation of our approach. Section 5 discusses our approach. Section 6 reviews related works. Finally, Section 7 concludes, and outlines future work.

2 QoS-Aware Compositional Model

In this section, we define the QoS compositional model used in this work. Following the SOA principles, service providers need to characterize their services to define both the offered functionalities and the offered quality. An abstract service specifies the functionality of the service without referring to any concrete service instance. An *abstract service* can be defined as a service class $S = \{s_1, \ldots, s_n\}$ which contains n functionally equivalent concrete services s_i that can be used to realize the functionality specified by the abstract service. We use $|S|$ to denote the total number of concrete services in S. We use the notation $\widehat{S}$ to denote an abstract service S' that is a subset of S, i.e., $S' \subseteq S$.

An *abstract composite service* CS_a specifies the compositional workflows using a set of abstract services $CS_a = \langle S_1, \ldots, S_n \rangle$ for fulfilling the service requests. A *concrete composite service* $CS = \langle s_1, \ldots, s_n \rangle$ is defined as an instantiation of abstract composite service $CS_a = \langle S_1, \ldots, S_n \rangle$, where each abstract service S_i is replaced by a concrete service $s_i \in S_i$. We use $|CS_a|$ to denote the total number of abstract services in CS_a.

For simplicity of illustration, we only consider the “travel by air” branch of the TAS example. This results in a composition where services FBS and HBS are running sequentially. We will use the modified TAS example as a running example in this work, and henceforth, simply refer to it as the TAS example. For the TAS example, there are two abstract services FBS and HBS. Suppose that the concrete services for FBS and HBS are $\{f_1, f_2, f_3, f_4\}$ and $\{h_1, h_2, h_3, h_4\}$ respectively. Then the abstract composite service TAS_a is $\langle FBS, HBS \rangle = \langle \{f_1, f_2, f_3, f_4\}, \{h_1, h_2, h_3, h_4\} \rangle$, and a possible concrete composite service of TAS_a can be $TAS = \langle f_1, h_2 \rangle$.

2.1 QoS Attributes

In this work, we deal with QoS attributes that can be quantitatively measured using metrics. In addition, we assume the values of QoS attributes for services to be known; we discuss how the value can be obtained in Section 5.

There are two classes of attributes: positive ones (e.g., availability) and negative ones (e.g., response time). Positive attributes have positive effects on the QoS, and therefore they need to be maximized. Conversely, negative attributes need to be minimized. For simplicity, we only consider negative attributes in this work, since positive attributes can be transformed into negative attributes by multiplying their values with –1. Given r QoS attributes of a concrete component service s, we use an attribute vector $Q_s = \langle q_1(s), \ldots, q_r(s) \rangle$ to represent the QoS attribute values of service s, where $q_i(s)$ is the ith QoS attribute value of s. Similarly, given a concrete composite service CS, we use the attribute vector $Q_{CS} = \langle q'_1(CS), \ldots, q'_r(CS) \rangle$ to represent it, where $q'_i(CS)$ is the ith end-to-end QoS attribute value of CS.

The SLA often specifies a set of constraints on the end-to-end QoS. Such constraints define the lower bound for positive constraints (e.g., the availability must be higher than 99.99%) and the upper bound for negative constraint (e.g., the response time must be less than 500 ms). Given a composite service CS with QoS attribute vector $Q_{CS} = \langle q'_1(CS), \ldots, q'_r(CS)\rangle$, the global constraints of CS can be represented as a vector $C_{CS} = \langle C_1, \ldots, C_r\rangle$ where $C_i \in \mathbb{R}$ and $q'_i(CS) \le C_i$. Without loss of generality, we use $C_i = \infty$ to denote the situation where $q'_i(CS)$ is unconstrained.

2.2 QoS for Composite Services

We introduce four elementary compositional structures for composing the component services: sequential, parallel, loop and conditional. Sequential composition of services $\{s_1, \ldots, s_n\}$ executes the services sequentially, one after another. Parallel composition of services $\{s_1, \ldots, s_n\}$ executes the services concurrently. A loop executes a service s_1 repeatedly up to k iterations. Conditional composition of services $\{s_1, \ldots, s_n\}$ executes exactly one of the services according to the evaluation of the guard conditions, where the guards are mutually exclusive.

The end-to-end QoS is aggregated from the QoS on the component services, based on the service compositional structures, and the types of QoS attributes. Table 1 shows the aggregation functions for response time, availability, and cost of component services with respect to the compositional structures. The response time $r \in \mathbb{R}_{\ge 0}$ is the delay between sending a request and receiving a response. For the sequential composition, the response time of the service composition is obtained by summing up the response time of the component services. For the parallel composition, it is equal to the maximum response time among the participating component services. For the loop composition, it is calculated by summing the involved component service k times, where k is the maximum number of loop iterations. For the conditional composition, since the evaluation of guards is not known at design time, the maximum response time of n services is chosen as the response time of the composite service. The availability $a \in \mathbb{R} \cap [0, 1]$ is the probability of the service being available. For the sequential composition, this implies that all the services are available during the sequential execution; therefore, the availability of the composite composition is the multiplication of the component services' availability. The cost $c \in \mathbb{R}_{\ge 0}$ is the price for utilizing a service. For the sequential and parallel composition, it is calculated by summing the cost of participating component services. For other compositions, their aggregation functions are similar.

Given an abstract composite service CS_a, a composite service CS'_a is a *subset* of CS_a, denoted by $CS'_a \subseteq CS_a$, if CS'_a and CS_a have the same compositional structure, and every service class of CS'_a is a subset of the corresponding service class of CS_a. Formally, $CS'_a \subseteq CS_a$ if CS'_a and CS_a share the same compositional structure C, with $|CS'_a| = |CS_a|$ and $\forall S_i \in CS'_a, \forall S_j \in CS_a : (i = j) \implies S_i \subseteq S_j$. Given a composite service CS_a, a *reduced abstract composite service* of CS_a, denoted by $\widehat{CS_a}$, is used to represent any composite service $CS'_a \subseteq CS_a$, e.g., an example of $\widehat{TAS_a}$ is $\langle\{f_1, f_2\}, \{h_1, h_2, h_3\}\rangle$.

2.3 Optimality Function

QoS Attribute	Sequential	Parallel	Loop	Conditional
Response Time	$\sum_{i=1}^{n} q(s_i)$	$\max_{i=1}^{n} q(s_i)$	$k * (q(s_1))$	$\max_{i=1}^{n} q(s_i)$
Availability	$\prod_{i=1}^{n} q(s_i)$	$\prod_{i=1}^{n} q(s_i)$	$(q(s_1))^k$	$\min_{i=1}^{n} q(s_i)$
Cost	$\sum_{i=1}^{n} q(s_i)$	$\sum_{i=1}^{n} q(s_i)$	$k * (q(s_1))$	$\max_{i=1}^{n} q(s_i)$

Table 1: Aggregation function

Concrete services have multi-dimensional attributes, and we need a methodology to facilitate their comparison in terms of their QoS. In this work, we use a Simple Additive Weighting (SAW) technique [34] to obtain a score for multi-dimensional attributes. SAW uses two phases: normalization and weighting, for producing the score. The normalization stage normalizes the values of QoS attributes so that they are independent of their units and ranges to allow comparison. The weighting stage allows users to specify their preferences on different QoS attributes. In the normalization stage, a service compares its QoS attribute values with the maximum and minimum QoS attributes of other service candidates within a service class. A composite service compares its aggregated QoS attributes with the maximum and minimum aggregated QoS attributes. The maximum (resp. minimum) aggregated QoS attributes can be obtained by aggregating maximum (resp. minimum) QoS attributes from each service class. Formally, we have

$$\begin{aligned} G_{min}(k) &= F_{(k)}{}_{i=1}^{n}(L_{min}(i,k)) \\ G_{max}(k) &= F_{(k)}{}_{i=1}^{n}(L_{max}(i,k)) \end{aligned} \tag{1}$$

with

$$\begin{aligned} L_{min}(i,k) &= \min_{\forall s \in S_i} q_k(s) \\ L_{max}(i,k) &= \max_{\forall s \in S_i} q_k(s) \end{aligned} \tag{2}$$

where $G_{min}(k)$ and $G_{max}(k)$ are the minimum and maximum aggregated values for the kth QoS attribute for the composite service, $F_{(k)}$ is the QoS aggregation function for attribute k which is given in Table 1, $L_{min}(i,k)$ and $L_{max}(i,k)$ are the minimum and maximum aggregated values for the kth QoS attribute for service class i.

Suppose each service has r QoS attributes. The local optimality function $L_i(s)$ computes the local optimality value of a concrete service s within service class S_i, where $s \in S_i$, as follows.

$$L_i(s) = \sum_{k=1}^{r} v(i,k,s) \cdot w_k \tag{3}$$

with

$$v(i,k,s) = \begin{cases} \dfrac{L_{max}(i,k) - q_k(s)}{L_{max}(i,k) - L_{min}(i,k)} & \text{if } L_{max}(i,k) \ne L_{min}(i,k) \\ 1 & \text{if } L_{max}(i,k) = L_{min}(i,k) \end{cases}$$

where $w_k \in \mathbb{R}_{\ge 0}$ is the weight of q_k and $\sum_{k=1}^{r} w_k = 1$.

The global optimality function $G(CS)$ computes the global optimality value of concrete composite service CS as follows.

$$G(CS) = \sum_{k=1}^{r} v'(k, CS) \cdot w_k \tag{4}$$

with

$$v'(k, CS) = \begin{cases} \dfrac{G_{max}(k) - q'_k(CS)}{G_{max}(k) - G_{min}(k)} & \text{if } G_{max}(k) \ne G_{min}(k) \\ 1 & \text{if } G_{max}(k) = G_{min}(k) \end{cases}$$

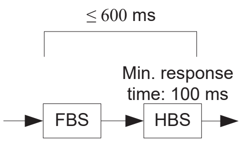
where $w_k \in \mathbb{R}_{\ge 0}$ is the weight of q'_k and $\sum_{k=1}^{r} w_k = 1$.

2.4 Problem Statement

We first define the notions of feasible and optimal selection.

Concrete Services	Response Time (ms)	Availability
f_1 / h_1	100	0.85
f_2 / h_2	300	0.92
f_3 / h_3	500	0.95
f_4	600	0.94
h_4	600	0.8

(a) Concrete services of TAS



(b) Service preprocessing

Figure 2: TAS Example

DEFINITION 1. *Given an abstract composite service* $CS_a = \langle S_1, \ldots, S_n \rangle$*, and global QoS constraints* $C = \langle C_1, \ldots, C_r \rangle$ *for* CS_a*, a* feasible selection *is a selection of concrete services* CS*, such that* CS *contains exactly one service for each service class* S_i *in* CS_a *and* CS *satisfies the global QoS constraints* C*, i.e.,* $q'_k(CS) \le C_k$.

DEFINITION 2. *An* optimal selection *is a feasible selection of concrete services* CS *that maximizes the global optimality value* $G(CS)$.

Given an abstract composite service CS_a, and a set of global constraints C, we are interested in finding the optimal selection CS. To address this problem, a naive approach is to exhaustively explore all combinations of concrete services for each service class. However, given n services in sequential composition, with each of them having l candidates, the total number of combinations is l^n. In fact, this problem is NP-hard, as it can be used to model a known NP-hard problem, i.e., the multi-dimensional multi-choice knapsack problem (MMKP) [26]. To allow runtime QoS-aware service composition, it is desirable to find a near-optimal selection at an acceptable cost, rather than finding an exact solution to the optimal selection problem at a very high cost [10].

One way to mitigate this problem is to select a subset of service candidates each time, instead of working on all services candidates. This is efficient especially when the number of service candidates is large and hard to be handled. The question that arises is how to identify a set of representative service candidates that not only satisfy the constraints, but also contribute to high global optimality value. In this work, our goal is to address the optimal selection problem efficiently and effectively by progressively exploring subsets of candidates that are likely to contribute to the optimal or near-optimal selection.

TAS Example. We provide the details of non-functional properties on the TAS example, which will be used in the following sections. Each of the abstract services of *TAS*$_a$ has four concrete services, where *FBS* $= \{f_1, f_2, f_3, f_4\}$ and *HBS* $= \{h_1, h_2, h_3, h_4\}$. The attribute vector of TAS example is in the form of $Q_{TAS} = \langle q'_1(TAS), q'_2(TAS) \rangle$, where attribute q'_1 provides the value of response time, and attribute q'_2 provides the value of availability. The values of response time and availability of all concrete services are given in Figure 2(a). In addition, the global constraints for response time and availability of TAS are 600 ms and 0.8 respectively.

3 Probabilistic Hierarchical Refinement (PROHR)

We introduce our method Probabilistic Hierarchical Refinement in this section. The workflow of PROHR is illustrated in Figure 3. PROHR is divided into three stages, i.e., preprocessing stage, probabilistic ranking stage, and hierarchical refinement stage. The details of each of the stages are introduced in the following subsections.

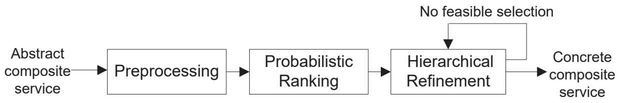


Figure 3: Probabilistic Hierarchical Refinement (PROHR)

3.1 Preprocessing Stage

The global constraints play an important role in selecting the services. For example, it has been shown that the heuristic method proposed in [9] has its performance degraded when the global constraints become more restrictive. In this section, we show that the global constraints can be used as an effective means for pruning service candidates. Consider the TAS example in Figure 2(b) where FBS and HBS are running sequentially, and the TAS global constraint for response time is 600 ms. We know that the concrete service that has the fastest response time for HBS is service s'_1, which has response time of 100 ms. Therefore, a service $s \in$ FBS must have a response time smaller than 500 ms, in order to fulfill the global constraint for response time. We refer to a service that may fulfill the global constraints as a *constraint-satisfiable service*.

Formally, given a composite service $CS = \langle S_1, \ldots, S_n \rangle$, and global constraints $C_{CS} = \langle C_1, \ldots, C_r \rangle$, a service $s_i \in S_i$ is a constraint-satisfiable service if the following condition holds:

$$\forall k \ : \ \underset{j \in \{1,\ldots,n\}}{F_{(k)}} \ (v_j) \le C_k \tag{5}$$

with

$$v_j = \begin{cases} q_k(s_i) & if\ j = i \\ L_{min}(j,k) & if\ j \ne i \end{cases}$$

where $F_{(k)}$ is the QoS aggregation function for attribute k. For example, given a service $s_i \in S_i$, and suppose all QoS aggregation functions $F_{(k)}$ are summations, then the condition becomes

$$\forall k \ : \ (q_k(s_i) + \sum_{j \in \{1,\ldots,n\} \setminus i} L_{min}(j,k)) \le C_k$$

We can safely prune all constraint-unsatisfiable services as they cannot satisfy the global constraints. For the TAS example, concrete services f_4 and h_4 are the only concrete-unsatisfiable services, because some of their QoS attributes do not satisfy Condition (5). In particular, it is the response times of f_4, h_4 and the availability of h_4 that do not satisfy the condition.

In addition to pruning using global constraints, we also include in the service preprocessing stage the pruning of non-*skyline services* [9]. Let us first recall the notion of *dominance*.

DEFINITION 3 (DOMINANCE). *Let* S *be a service class, and* s, s' *be two services, where* $s, s' \in S$ *and* $s \ne s'$*. Service* s dominates *service* s'*, denoted by* $s \prec s'$*, if the service* s *is at least as good as service* s' *in all QoS parameters and better than service* s' *in at least one QoS parameter, i.e.,* $\forall k \in \{1, \ldots, |Q_S|\}$*:* $q_k(s) \le q_k(s')$*, and* $\exists k \in \{1, \ldots, |Q_S|\}$*:* $q_k(s) < q_k(s')$.

A service $s \in S$ is denoted as a *skyline service* if there does not exist a service in service class S that dominates s (i.e., $\neg \exists s' \in S : s' \prec s$). In other words, a service $s \in S$ is a *non-skyline service*, if there exists a service $s' \in S$ that dominates s.

It can be shown that we can safely prune non-skyline services in service class CS, without affecting the result of optimal selection [7]. For our TAS example, concrete service h_4 is the only non-skyline service.

Algorithm 1: Service Preprocess (Preprocess)

```
input  : Abstract composite service $CS_a = \langle S_1, \ldots, S_n \rangle$
output: Preprocessed abstract composite service
        $CS'_a = \langle S'_1, \ldots, S'_n \rangle$
1 $CS'_a \leftarrow \langle \emptyset_1, \emptyset_2, \ldots, \emptyset_n \rangle$;
2 for $i = 1$ to $n$ do
3     $S''_i = \mathit{ConstraintPruning}(S_i)$;
4     $S'_i = \mathit{NonSkylinePruning}(S''_i)$;
5     $CS'_a[i] \leftarrow S'_i$;
6 return $CS'_a$;
```

Henceforth, we denote the pruning of constraint-unsatisfiable services and non-skyline services as *constraint pruning* and *non-skyline pruning* respectively. The algorithm of service preprocessing is shown in Algorithm 1. The abstract composite service CS'_a is initialized with empty service classes (line 1), where $\emptyset_i$ denotes ith empty service class. After that, each service class of CS_a will be undergone constraint pruning (line 3) and non-skyline pruning (line 4). The service class after pruning will be assigned to CS'_a (line 5). Finally, abstract composite service CS'_a which contains the preprocessed service classes will be returned (line 6). In the TAS example, the composite service TAS'_a after service preprocessing stage is $\langle \{f_1, f_2, f_3\}, \{h_1, h_2, h_3\} \rangle$.

After the initial preprocessing, we propose in the following a ranking based method that can allow us to limit the selection to a smaller set of services that have higher chance of contributing to the optimal selection.

3.2 Probabilistic Ranking Stage

The local optimality value of a concrete service can reflect the quality of a service. Nevertheless, it does not provide a direct indication on how likely the concrete service can contribute to the conformance of the global constraints. To address this problem, we propose an estimation on the probability of the constraint satisfaction of a concrete service. First, we introduce the notion of *local constraint*. The objective of local constraint is to provide an estimation on the average constraint value that has to be fulfilled by individual service classes. Given a global constraint C_i, we define the local constraint c_i as follows, where n is the number of service classes.

$$c_i = \begin{cases} C_i/n & \text{if } \mathit{agg\ is\ sum} \\ C_i^{1/n} & \text{if } \mathit{agg\ is\ mult} \wedge C_i \geq 0 \\ -|C_i|^{1/n} & \text{if } \mathit{agg\ is\ mult} \wedge C_i < 0 \\ C_i & \text{if } \mathit{agg\ is\ min, max} \end{cases} \tag{6}$$

Equation (6) is recursively applied to the compositional structure according to the aggregation functions (agg) – either summation (sum), multiplication ($mult$), minimum (min), or maximum (max). Assume that C_i is a global constraint for cost. Given the original TAS in Figure 1, which contains three services CBS, FBS and HBS, we first apply Equation (6) to the sequential structure, which contains a conditional structure and HBS. The conditional structure and HBS are both allocated with local constraint $C_i/2$ (since $\mathit{agg\ is\ sum}$ is true). Subsequently, we recursively apply Equation (6) to the conditional structure, where CBS and FBS are both allocated with $C_i/2$ (since $\mathit{agg\ is\ max}$ is true and the local constraint for the conditional structure is $C_i/2$).

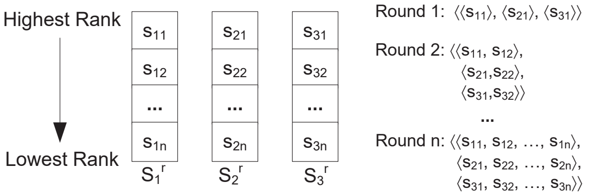

Highest Rank
Lowest Rank
s_{11}
s_{12}
...
s_{1n}
S_1^r
s_{21}
s_{22}
...
s_{2n}
S_2^r
s_{31}
s_{32}
...
s_{3n}
S_3^r
Round 1: $\langle\langle s_{11}\rangle, \langle s_{21}\rangle, \langle s_{31}\rangle\rangle$
Round 2: $\langle\langle s_{11}, s_{12}\rangle, \langle s_{21}, s_{22}\rangle, \langle s_{31}, s_{32}\rangle\rangle$
...
Round n: $\langle\langle s_{11}, s_{12}, \ldots, s_{1n}\rangle, \langle s_{21}, s_{22}, \ldots, s_{2n}\rangle, \langle s_{31}, s_{32}, \ldots, s_{3n}\rangle\rangle$

Figure 4: A scenario of hierarchical refinement

Given a composite service $CS_a = \langle S_1, \ldots, S_n \rangle$, and global constraints of CS_a as $C = \langle C_1, \ldots, C_r \rangle$, the *constraint satisfaction probability* of a service $s \in S_i$, denoted by $P_i(s)$, is calculated as follows:

$$P_i(s) = \gamma \Big(\sum_{k=1}^{r} p_k(s) \cdot w_k \Big) \tag{7}$$

where $w_k \in \mathbb{R}_0^+$ is the weight of $p_k(s)$, $\sum_{i=1}^{r} w_k = 1$ and $\gamma \in \mathbb{R} \cap [0, 1]$ is the *credibility factor* (explained later). The calculation of $p_k(s)$ is divided into two cases. When $q_k(s) > c_k$, then

$$p_k(s) = \frac{D_{max}(i,k) - (q_k(s) - c_k)}{D_{max}(i,k)} \tag{8}$$

with

$$D_{max}(i,k) = \max_{\forall s_i \in S_i} (q_k(s_i) - c_k)$$

and when $q_k(s) \leq c_k$, $p_k(s) = 1$. In the following, we explain the details of Equation (7) and Equation (8).

In Equation (7), $P_i(s)$ is calculated by using the SAW method that is introduced in Section 2.3, followed by multiplying with the credibility factor γ. The credibility factor γ is used to adjust the value of $P_i(s)$ based on how much we can trust $P_i(s)$. We set the value of credibility factor γ to $1/n$, where n is the number of service classes. The reason for this choice is that, with the increment of n, $P_i(s)$ would be more dependent on the choices made by other service classes. To estimate the probability more conservatively, we choose to lower the constraint satisfaction probability, given that the number of service classes n increases.

In Equation (8), $q_k(s) > c_k$ signifies that $q_k(s)$ has violated the local constraint (reminded that negative attributes are assumed here). The value of $p_k(s)$ is decided by the value of $q_k(s)$. If $q_k(s)$ has a smaller value, it has a higher chance for satisfying the global constraints, hence the higher value of $p_k(s)$. Given a service class S_i, $D_{max}(i,k)$ calculates the maximum difference between $q_k(s_i)$ and the local constraint c_k, where $s_i \in S_i$. $D_{max}(i,k)$ serves the purpose of normalizing the value of probability $p_k(s)$, such that $p_k(s) \in \mathbb{R} \cap [0, 1]$. For the case of $q_k(s) \leq c_k$, it indicates the conformance of $q_k(s)$ to the local constraint, and we simply set $p_k(s) = 1$. In TAS, the constraint satisfaction probabilities for concrete services are $P_1(f_1)$=$P_2(h_1)$=0.25, $P_1(f_2)$=$P_2(h_2)$=0.5, and $P_1(f_3)$=$P_2(h_3)$=0.25.

Given a composite service $CS_a = \langle S_1, \ldots, S_n \rangle$, the probabilistic ranking is performed by ordering the services in each service class S_i by the multiplication value of local optimality value and constraint satisfaction probability of each service $s \in S_i$, i.e., $L_i(s) \cdot P_i(s)$, in the descending order. We denote the ranked service class as $S_i^r = \langle s_1, \ldots, s_n \rangle$, which is an ordered sequence of all concrete services $s_i \in S_i$. The ranked composite service containing the ranked services as $CS_a^r = \langle S_1^r, \ldots, S_n^r \rangle$. For TAS example, given the composite service TAS'_a returned by service preprocessing, the ranked composite service $\mathit{TAS}_a^r = \langle \langle f_2, f_1, f_3 \rangle, \langle h_2, h_1, h_3 \rangle \rangle$, where the values of $L_i(s) \cdot P_i(s)$ for concrete services $f_1(h_1)$, $f_2(h_2)$, and $f_3(h_3)$ are 0.125, 0.3, and 0.125 respectively.

3.3 Hierarchical Refinement Stage

After obtaining the ranked composite service CS_a^r, the next stage is to perform service selection on CS_a^r. We illustrate the dynamic ser-

Algorithm 2: Optimization with ProHR (ProHR)

```
input : Abstract composite service to be solved $CS_a$
input : Termination threshold for constraint-satisfiability
        $\epsilon \in \mathbb{R} \cap [0,1]$
output: A feasible selection of concrete composite services
        $CS$
1  $CS'_a \leftarrow Preprocess(CS_a)$ ;
2  $CS^r_a \leftarrow PRank(CS'_a)$ ;
3  $\widehat{CS^r_a} \leftarrow \langle \emptyset_1, \emptyset_2, \ldots, \emptyset_n \rangle$ ;
4  $round \leftarrow 1$ ;
5  repeat
6      $\widehat{CS^r_a} \leftarrow HRefine(CS^r_a, \widehat{CS^r_a}, \epsilon, round)$ ;
7      $CS \leftarrow OptimalSelection(\widehat{CS^r_a})$;
8      if $CS \neq \emptyset$ then
9          return $CS$ ;
10     $round \leftarrow round + 1$;
11 until $\widehat{CS^r_a} = CS^r_a$ ;
12 return $\emptyset$;
```

Algorithm 3: Hierarchical Refinement (HRefine)

```
input : Abstract composite service $CS^r_a = \langle S^r_1, \ldots, S^r_n \rangle$
input : Reduced abstract composite service
        $\widehat{CS^r_a} = \langle \widehat{S^r_1}, \ldots, \widehat{S^r_n} \rangle$
input : Termination threshold for constraint-satisfiability
        $\epsilon \in \mathbb{R} \cap [0,1]$
input : Current round, $round$
output: Abstract composite service $\widehat{CS^r_a}'$, where
        $\widehat{CS^r_a} \subseteq \widehat{CS^r_a}' \subseteq CS^r_a$
1  $\widehat{CS^r_a}' \leftarrow \langle \emptyset_1, \emptyset_2, \ldots, \emptyset_n \rangle$;
2  for $i = 1$ to $n$ do
3      $serviceCount \leftarrow |\widehat{S^r_i}|$ ;
4      $prob \leftarrow 1$ ;
5      for $j = |\widehat{S^r_i}|$ to $|S^r_i|$ do
6          $prob \leftarrow prob \cdot (1 - P_i(S^r_i[j]))$;
7          $serviceCount \leftarrow serviceCount + 1$;
8          $p \leftarrow 2^{round-1}$;
9          if $(1 - prob) \geq \epsilon^{1/p}$ then
10             break ;
11     $\widehat{CS^r_a}'[i] \leftarrow \{s_j \in S^r_i \mid j \in [1, serviceCount]\}$ ;
12 return $\widehat{CS^r_a}'$ ;
```

vice selection with an example as shown in Figure 4. Consider that $CS^r_a = \langle S^r_1, S^r_2, S^r_3 \rangle$, where the services of each service class S^r_i can be found in Figure 4. The hierarchical refinement uses multiple rounds of selection for selecting the service representatives. At the first round, one service is chosen from each service class; they form a *reduced* composite service $\widehat{CS^r_a} = \langle \widehat{S^r_1}, \widehat{S^r_2}, \widehat{S^r_3} \rangle$, where $\widehat{S^r_1} = \langle s_{11} \rangle$, $\widehat{S^r_2} = \langle s_{21} \rangle$, and $\widehat{S^r_3} = \langle s_{31} \rangle$. The reduced composite service $\widehat{CS^r_a}$ is then solved, e.g., by MIP solver, for optimal selection (details will be given in Section 3.4). Note that the reduced composite service $\widehat{CS^r_a}$ contains less services than CS^r_a. Therefore, the optimal selection can be performed more efficiently. If a solution is found, then the result is returned to the user. Otherwise, we proceed to next round by selecting more services from each service class. The process is repeated until a solution is found or until all services in the service classes have been explored. In the latter case, the selection approach guarantees that it does not miss out a solution if one exists.

The next problem is how to determine the number of representatives for each service class S^r_i at each round of selection: the number of services should be large enough for finding a solution to the service composition, while small enough to allow for efficient computation. We propose to use constraint satisfaction probability to address the problem. We first extend the definition of constraint satisfaction probability to a set of services. Given $\mathbb{S}$ as a subset of services from service class S_i, i.e., $\mathbb{S} \subseteq S_i$, we define constraint satisfaction probability of $\mathbb{S}$, denoted as $P_i(\mathbb{S})$, as the probability that at least one of the services in $\mathbb{S}$ succeed in satisfying the global constraints after composition. $P_i(\mathbb{S})$ is calculated as follows:

$$P_i(\mathbb{S}) = 1 - \prod_{s \in \mathbb{S}} (1 - P_i(s)) \qquad (9)$$

We observe that the reason that the optimal selection cannot produce a solution is due to global constraint violation. Therefore, the capacity of the set of representatives $\mathbb{S}$ should be large enough, so that $P_i(\mathbb{S})$ can exceed ϵ, which is a parameter provided by the user on the intended termination threshold.

Algorithm 2 presents the main ProHR algorithm. Initially, the input abstract composite service CS_a passes through two initial stages of ProHR: service preprocess and service ranking, using functions *Preprocess* and *PRank* respectively (lines 1, 2). Function *Preprocess* has been introduced in Algorithm 1, and function *PRank* sorts the services for each service class $S_i \in CS'_a$ by the value of $L_i(s) \cdot P_i(s)$, where $s \in S_i$, in the descending order. The details of *PRank* can be found in Section 3.2. The reduced abstract composite service $\widehat{CS^r_a}$ is initialized with empty service classes (line 3), and the current round number $round$ is initialized to value 1 (line 4). After that, *HRefine* (see Algorithm 3) is called to populate each service class of the reduced composite function $\widehat{CS^r_a}$ with the representative services (line 6). Following that, it is passed to the *OptimalSelection* function (line 7) for optimal selection, where the details will be introduced in Section 3.4. The result of the optimal selection is stored in CS. If the optimal selection is successful, i.e., when CS is not empty, the result is returned (line 9); otherwise, we proceed to the next round of ProHR. This continues until all concrete services are explored, i.e., when $\widehat{CS^r_a} = CS^r_a$ (line 11). If the optimal selection cannot find a result, an empty result is returned (line 12).

Algorithm 3 presents the dynamic selection algorithm. Initially, the reduced abstract composite service $\widehat{CS^r_a}'$ is initialized with empty service classes. For each service class (line 2), the $serviceCount$ is initialized to the number of services in service class $\widehat{S^r_i}$ (line 3), and the accumulated probability $prob$ is initialized with 1 (line 4). Starting from index $|\widehat{S^r_i}|$, we accumulate the constraint satisfaction probability of service $S^r_i[j]$, until it exceeds the threshold $\epsilon^{1/p}$, where $p = 2^{round-1}$ (lines 5–10). Note that the threshold $\epsilon^{1/p}$ increases with the number of rounds. The reason is based on the assumption that if there exists an optimal selection, concrete services in the optimal selection are likely to be ranked higher by the ranking algorithm in respective service classes. Therefore, at the starting round, choosing a smaller amount of services that are ranked higher from each service class will decrease the solving time for the optimal selection. Conversely, the chance of getting a feasible selection decreases when the number of rounds increases. Hence, choosing a larger amount of services at later rounds will lead to a faster exploration of all concrete services, especially in the case where there does not exist a feasible selection.

Afterwards, the respective service classes in $\widehat{CS^r_a}'$ are then populated with the representative services with the amount equals to $serviceCount$ (line 11). Finally, we return the reduced composite service $\widehat{CS^r_a}'$, which contains the newly inserted representatives (line 12).

For TAS, the number of services that are chosen at each round depends on the termination threshold for constraint-satisfiability ϵ. If $\epsilon = 0.4$, then for the first round, we get the reduced composite service $\widehat{TAS^r_a} = \langle\langle f_2\rangle, \langle h_2\rangle\rangle$. $OptimalSelection(\widehat{TAS^r_a})$ will return a feasible selection $TAS = \langle f_2, h_2\rangle$, which is also the optimal selection for TAS example. Since a feasible selection is found, TAS terminates at the first round of $HRefine$. If $\epsilon = 0.9$, for the first round, we get $\widehat{TAS^r_a} = \langle\langle f_2, f_1, f_3\rangle, \langle h_2, h_1, h_3\rangle\rangle = TAS^r_a$. $OptimalSelection(\widehat{TAS^r_a})$ will return the same optimal solution as in the former case where $\epsilon = 0.4$. However, since there are more concrete services, it might take longer time to solve for the latter case where $\epsilon = 0.9$. We will investigate how different values of ϵ affect the service selection process in Section 4.

3.4 Solving for Optimal Selection

In this section, we introduce the use of Mixed Integer Programming (MIP) to realize the $OptimalSelection$ at line 7 of Algorithm 2. Mixed Integer Programming (MIP) is a technique for minimization or maximization of an objective function subjected to a set of constraints. A binary decision variable x_{ij} is used to represent the selection of service candidates s_{ij}. If a service candidate is selected, then x_{ij} is set to 1, and to 0 otherwise. By Equation (4), an MIP model can be specified as a maximization of the objective function below:

$$\sum_{k=1}^{r} \frac{G_{max}(k) - F_{(k)}{}_{j=1}^{n} \sum_{i=1}^{|S_j|} q_k(s_{ij}) \cdot x_{ij}}{G_{max}(k) - G_{min}(k)} \cdot w_k \tag{10}$$

subjected to the QoS constraints

$$(F_{(k)}{}_{j=1}^{n} \sum_{i=1}^{|S_j|} q_k(s_{ij}) \cdot x_{ij}) \leq C_k, 1 \leq k \leq r \tag{11}$$

where r and n are the total number of attributes and service classes respectively. In addition, since we only choose one service per service class, the following constraint must be hold.

$$\sum_{i=1}^{|S_j|} x_{ij} = 1, 1 \leq j \leq n \tag{12}$$

There are several tools or libraries that can be used to solve the MIP model, examples are Gurobi solver [20] and lpsolve solver [13]. We omit the technical details of MIP, and refer the readers to [32]. Note that to solve the above MIP model requires the linearization of the objective function and the QoS constraints, the reader can refer [8] for more details.

3.5 Complexity Analysis

In this section, we analyze the complexity of ProHR (Algorithm 2). Let n be the number of service classes, m be the number of competing services for the service class with the largest number of competing services, and k be the total number of QoS attributes for each service.

Representative selection. The *Preprocess* function (Algorithm 1) makes use of two functions *ConstraintPruning* and *NonSkylinePruning*. The complexity of *ConstraintPruning* for n service classes is $\mathcal{O}(nm)$. The complexity of *NonSkylinePruning* for n service classes is $\mathcal{O}(nm \cdot \log m)$ for $k = 2, 3$ and $\mathcal{O}(nm \cdot (\log m)^{k-2})$ for $k \geq 4$ [14, 23]. As a side note, we intentionally let *ConstraintPruning* run before *NonSkylinePruning*, because *ConstraintPruning* has lower complexity – after *ConstraintPruning*, some competing services might be pruned, and it will result in faster *NonSkylinePruning*. The complexity of the *PRank* function is $\mathcal{O}(nm \cdot (\log m))$, since it only incurs sorting in each service class. The complexity of iteratively invoking *HRefine* (Algorithm 3) is $\mathcal{O}(nm)$, since it runs iteratively until all competing services are included. To conclude, the complexity for representative selection (i.e., Algorithm 2 without the *OptimalSelection* function) is bounded by the complexity of *NonSkylinePruning*.

0 - 1 Integer Linear Programming Problem. *OptimalSelection* function is essentially solving a 0 - 1 integer linear programming problem, which is a well-known NP-complete problem. Therefore, Algorithm 2 is NP-complete. The ultimate goal of the representative selection is to reduce the number of competing services required to be solved by the *OptimalSelection* function, in order to improve the overall performance for Algorithm 2.

4 Evaluation

We conducted experiments to evaluate our approach to service selection using ProHR.

4.1 Research Quesions

We attempted to answer the following research questions (RQ1 – RQ4).

RQ1. How is the *performance* of ProHR in comparison to the state-of-the-art?

We analyze how different stages contribute to the performance of ProHR. Firstly, we investigate how well service preprocessing performs. In particular, what is the number of services that are pruned in the service preprocessing stage. Secondly, we also look into the performance of ProHR algorithm without the constraint pruning preprocessing. The reason is to facilitate the comparison with the state-of-the-art, as well as to enable us to evaluate how well the probabilistic ranking and hierarchical refinement work without utilizing the constraint pruning preprocessing. Lastly, we evaluate the performance of ProHR as a whole.

RQ2. How is the *accuracy* of the concrete services that are selected by ProHR?

We measure the accuracy using the following formula:

$$accuracy = \frac{G(CS_{heu})}{G(CS_{exact})} \tag{13}$$

where $G(CS_{heu})$ and $G(CS_{exact})$ are the global optimality values of concrete services returned by heuristic method and exact method respectively.

RQ3. How *scalable* is ProHR?

RQ4. How do different termination threshold values ϵ influence the selection process of ProHR?

4.2 Experiment Setup

Implementation. We implemented all algorithms in C#. For solving the mixed integer programming models, we used the Gurobi solver [20]. The experiments were conducted on an Intel Core i7-4600U CPU with 8 GB RAM, running on Windows 7.

Experimental Setup. To answer the previous research questions, we evaluate ProHR using a real-world dataset [4]. We compare our methods with [9] which is the existing state-of-the-art for the optimal selection of services, to the best of our knowledge. For effective comparison, we choose to use dataset of anti-correlated distribution, as it presents the most challenging dataset for methods that make use of non-skyline pruning [9]. Note that both [9] and our methods utilize non-skyline pruning in the preprocessing stage.

Methods to compare. We compare the efficiency of the following methods.

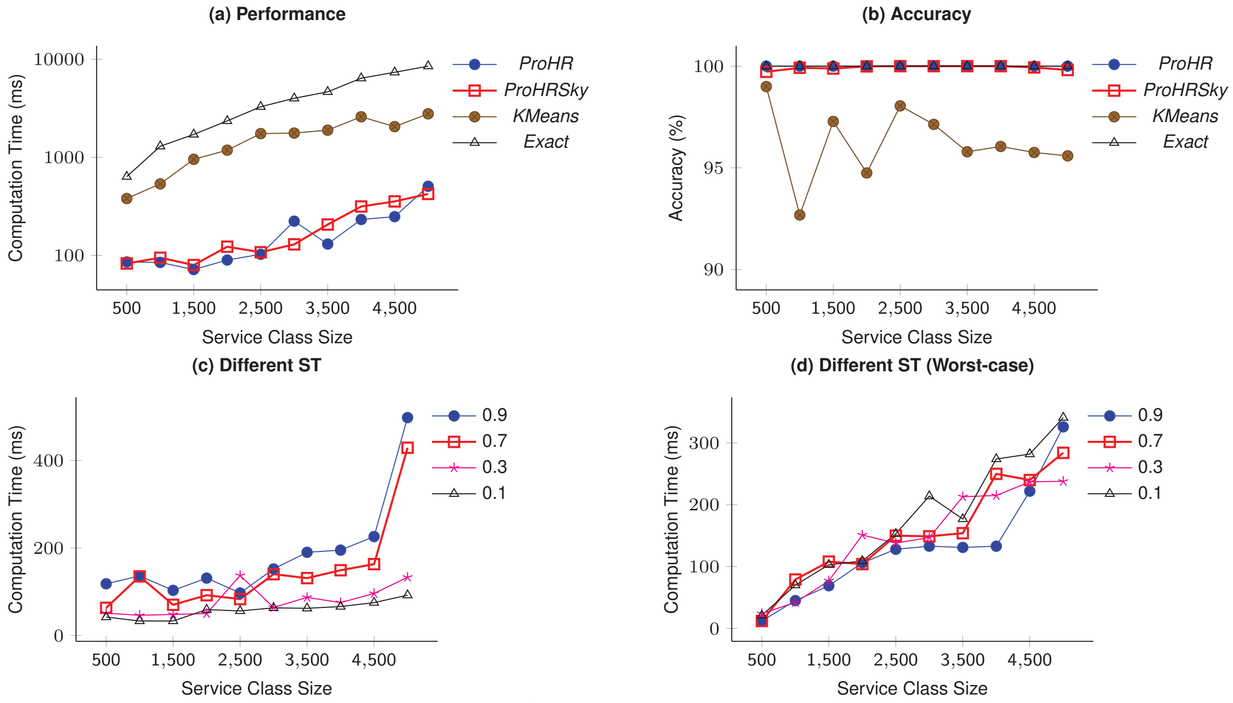


Figure 5: Experiment results

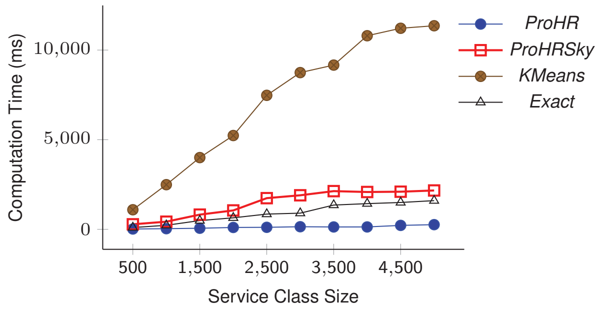


Figure 6: Worst-case performance

Name	Agg.	Type	Name	Agg.	Type
Response Time	Sum	-	Compliance	Mult	+
Availability	Mult	+	Best Practices	Mult	+
Throughput	Min	+	Latency	Sum	-
Successability	Mult	+	Documentation	Mult	+
Reliability	Mult	+			

Figure 7: QoS attributes

Size	1,000	2,000	3,000	4,000	5,000
NonSkyline (%)	23.1	29.7	34.4	38.0	40.2
Constraint (%)	91.8	91.1	91.0	90.9	90.6
Preprocess (%)	92.1	91.5	91.7	91.8	91.6

Figure 8: Preprocess results

1. ***ProHR*****:** Our method described in Section 3.
2. ***ProHRSky*****:** Our method described in Section 3, with the modification that only non-skyline pruning is used in the preprocessing stage.
3. ***KMeans*****:** The hierarchical k-means clustering method proposed in [9].
4. ***Exact*****:** The standard global optimization problem for all service candidates represented in the MIP model.

The reason for adding *ProHRSky* is because *KMeans* is also using non-skyline pruning as their preprocessing method. Using the same preprocessing method will allow both methods to reduce the same amount of services at the preprocessing stage, therefore enable us to compare the strength of the service selection algorithms for these two methods.

Dataset. We make use of the QWS dataset [4], which is a public dataset collected from public registries, search engines and service portals, using a specialized Web crawler. The dataset contains 2,507 Web services and there are a total of 9 QoS attributes measured using commercial benchmark tools. More details on the dataset can be found at [5, 6]. In our experiments, there are five service classes. For each round of experiments, the concrete services for the service classes are randomly chosen from the QWS dataset. We generate 10 sets of QoS constraints $Q = \{Q_1, \ldots, Q_{10}\}$, where $i{*}5\%$ of the QWS services satisfy constraints Q_i. For example, 5% of the QWS services satisfies Q_1. For each round of experiments, we randomly select a QoS constraint from Q.

QoS Attributes. The details of these QoS attributes can be found in Figure 7. The *Name* and *Agg.* columns in Figure 7 provide the names of the QoS attributes and the aggregation functions (Summation (*Sum*), Multiplication (*Mult*), Minimum (*Min*)) for those attributes. The *Type* column shows whether these attributes are negative (–) or positive (+). We randomly partition the dataset into 10 service classes and the QoS constraints are set randomly. We run each experiment for 500 times and report the average values of the metrics.

4.3 Experiments

We conduct several experiments (E1 – E6) to answer the research questions (R1 – R4). For experiments E1 to E4, we set the termination threshold at 0.9, and different termination thresholds are evaluated in experiments E5 and E6. For experiments E1 to E3, there exists a feasible selection that can satisfy the global constraints. We report our findings in the following.

E1: We record the number of services that have been pruned using non-skyline pruning (*NonSkyline*), constraint pruning (*Constraint*), and the combination of both pruning methods (*Preprocess*). The numbers reported are the percentage of services that are pruned with respect to the total number of services. The experiment results can be found in Figure 8.
Results. The non-skyline pruning has achieved 23.1% – 40.2% reduction rate. As the dataset for experiment has anti-correlated distribution, and it is known that the non-skyline pruning does not perform well in such dataset [9], therefore we can expect the method to perform better for datasets of different distributions (e.g., correlated distribution). For constraint pruning, it has achieved 90.6% – 91.8% reduction rate. The high reduction rate is attributed to the reason that a seemingly lenient global constraint of composite service can result in a high expectation of the QoS attribute of component services. For example, the global constraint of availability is set to 0.1, and since there are 10 service classes, the average availability of the involved component services that need to fulfill the global constraint is $(0.1)^{1/10} \approx 0.8$. Therefore, it results in high reduction rate when combined with the constraint reduction using response time attribute. Another observation is that, when these two pruning methods are combined (*Preprocess*), it can achieve a reduction rate that is greater than any individual pruning method applied alone.

E2: We compare the computation time with respect to the number of services for each service class, which varies from 500 to 5000. The experiment results can be found in Figure 5(a).
Results. We notice that both *ProHR* and *ProHRSky* have significant improvement over *Exact* and *KMeans* methods. Since *ProHRSky* and *KMeans* are both using the same preprocessing method (non-skyline pruning), this result suggests that probabilistic ranking and hierarchical refinement of *ProHR* have provided faster performance over *KMeans*. Interestingly, although substantial number of services have been pruned by constraint pruning, *ProHR* has only achieved slight improvement of performance over *ProHRSky*. This result suggests that the services that are pruned by constraint pruning are ranked lower by probabilistic ranking algorithm, and services that are part of the feasible selection are ranked higher. Therefore, the hierarchical refinement effectively locates a feasible selection by using a significant smaller subset of the services in a service class.

E3: We compare the accuracy with respect to the number of services for each service class, which varies from 500 to 5000. The experiment results can be found in Figure 5(b).
Results. We notice that all methods achieve accuracy higher than 90%. For *ProHR* and *ProHRSky*, both have outperformed *KMeans* by achieving almost 100% accuracy. This result shows that the service ranking algorithm of *ProHR* has accurately ranked the service, such that the feasible selection that has been chosen by the hierarchical refinement algorithm is near-optimal or optimal selection.

E4: We compare the computation time with respect to the number of services for each service class, which varies from 500 to 5000, in the case where it does not exist a selection that can satisfy the global constraints. The experiment results can be found in Figure 6.
Results. We notice that *KMeans* has fast-growing computation time. The reason is that *KMeans* applies k-means clustering at each round of selection, and it is known that k-means clustering is NP-hard in the worst case. Therefore, this makes the method sensitive to the number of services. We also notice that *ProHRSky* outperforms *KMeans* significantly. There are two reasons for this. Firstly, *ProHRSky* only incurs sorting at the start of the selection, and it does not incur extra operations between rounds of selection. Secondly, the number of services that are used for selection increases significantly at each round; therefore, it effectively explores all services in the service classes in a few number of rounds. *ProHRSky* is slower than *Exact* due to the extra time that it spent on multiple rounds of hierarchical refinements before all concrete services in the service classes are explored at the final round – while for *Exact*, it explores all services from the beginning. We also observe that *ProHR* outperforms *Exact*. This is attributed to the constraint pruning preprocessing in *ProHR*, since most of the services are pruned by constraint pruning due to the strict global constraints that make no feasible selection, and this let *ProHR* achieve higher performance than *Exact*.

E5, E6: We compare *ProHR* using different termination thresholds with respect to the number of services for each service class, which varies from 500 to 5000, in the case where it exists (E5) and does not exist (E6) a selection that can satisfy the global constraints respectively. The results of experiments E5 and E6 are shown in Figure 5(c) and Figure 5(d) respectively.
Results. It suggests that the smaller the value of termination threshold, the faster it tends to complete. For example, the dynamic selection algorithm with termination threshold of 0.1 tends to complete faster than the other termination thresholds. This result is due to the fact that the smaller the termination threshold, the fewer elements will be chosen in each round by the dynamic selection algorithm. We illustrate why this can end up in a faster searching time using an example. Suppose the services that are part of a feasible selection are all ranked at fifth position in their service classes. Dynamic selection algorithm with termination threshold of 0.1 can choose fewer services (say five services) at a single round, while dynamic selection algorithm with termination threshold of 0.9 can choose more services (say ten services) at single round. Although the services selected by dynamic selection algorithm for termination thresholds 0.1 and 0.9 both contain the feasible services, but dynamic selection algorithm for termination thresholds 0.1 will be solved faster by the MIP solver since it contains fewer services. Nevertheless, there is a disadvantage for choosing a smaller termination threshold. Experiment E6 has shown that the dynamic selection algorithm with smaller termination threshold tends to finish slower, since fewer items that are chosen at each round will lead to more rounds to iterate before all concrete services in each service class are explored. It is therefore a tradeoff to choose a smaller termination threshold over a larger one.

Answer to Research Questions.. To answer the research questions RQ1–RQ3, we can see that both *ProHR* and *ProHRSky* outperform *KMeans* in terms of performance, accuracy and scalability from experiments E1–E4. Research question RQ4 is answered by the analyses of experiments E5 and E6.
Threats to Validity. There are several threats to validity. The first threat to validity is due to the fact that competing services are selected from 2,507 services offered by the QWS public dataset. Although the number of services is reasonably large, there could still be observations that do not make obvious by the set of QoS attributes we obtained from these services. The second threat to validity is stemmed from our choice to use a few example values as experimental parameters, that include global constraints and termination thresholds, in order to cope with the combinatorial explosion

of options. To address these threats, it is clear that more experimentations with different datasets and experimental parameters are required, so that we can further investigate the effects that has not been made obvious by our dataset and experimental parameters.

5 Discussion

Worst case scenario. For our method, the worst case scenario happens when no optimal service selection exists. In such a case, our method would need to go through multiple rounds of hierarchical refinement (Algorithm 3) and MIP solving until all services are included in representative services. We can then conclude the non-existence of optimal service selection. However, the non-existence of optimal service selection is normally due to overly restrictive global constraints, which no service selection is able to satisfy. This signifies that the preprocessing algorithm (Algorithm 1) may have removed most of the competing services from the beginning. This makes our algorithm still perform well in the worst case scenario. We have evaluated PROHR in the worst case scenario in Section 4.

Estimation of QoS values. In practice, the values of QoS attributes may be obtained from the SLA of the service; for example, Amazon EC2 has guaranteed that the monthly uptime percentage of their services would be at least 99.95% [1]. The QoS values can also be estimated based on past invocations of the service, by using methods such as [38, 27]. Based on the observation from past invocations, the user can use an appropriate estimation method, such as expected value or i-th percentile, according to the requirements of the user. If a user requires an estimation of QoS value that holds for most of the time, the user can make use of the expected value of the observed values. On the other hand, if the user requires a more conservative estimation, the user can, for example, make use of the QoS value at the 90-th percentile (i.e., the QoS value is worse than 90% of all observed values).

Pareto optimal solution. Given two service selection CS_1 and CS_2, where both satisfy all global constraints, the selection CS_1 is said to *dominate* CS_2, if $\forall i : q'_i(CS_1) \leq q'_i(CS_2) \land \exists j : q'_i(CS_1) < q'_i(CS_2)$, where $i, j \in \{1, \ldots, k\}$, and k is the number of QoS attributes. CS is called a *Pareto-optimal solution* if CS is not dominated by any other CS' that satisfies the global constraints. Our method with *weighed QoS attributes* (Equation (10)) is provable to produce a Pareto-optimal solution. For the proof, interested readers can refer to [25], Theorem 3.1.2. To obtain multiple Pareto optimal solutions, the user can run our method for several times, with different weights for each QoS attribute each time.

6 Related Work

The problem of QoS-aware Web service selection and composition has received considerable attention during recent years. In [36, 37], the authors present an approach that makes use of global planning to search dynamically for the best concrete services for service composition. Their approach involves the use of mixed integer programming (MIP) techniques to find the optimal selection of component services. Ardagna *et al.* [11] extend the MIP methods to include local constraints. Cardellini *et al.* [15] propose a methodology to integrate different adaptation mechanisms for combining concrete services to an abstract service, in order to achieve a greater flexibility in facing different operating environments. Our work is orthogonal to aforementioned works, as it does not assume particular formulation of the MIP problems. Although the method in aforementioned works efficiently for small case studies, it suffers from scalability problems when the size of the case studies becomes larger, since the time required grows exponentially with the size of concrete services.

Yu *et al.* [35] propose a heuristic algorithm that can be used to find a near-optimal solution. The authors propose two QoS compositional models, a combinatorial model and a graph model. The time complexity for the combinatorial model is polynomial, while the time complexity for the graph model is exponential. However the algorithm does not scale with the increasing number of web services. To address this problem, Alrifai *et al.* [9] present an approach that prunes the search space using skyline methods, and they make use of a hierarchical k-means clustering method for representative selection. The work of Alrifai *et al.* is the closest to ours. Our approach has several advantages over their approach. Firstly, their work does not take into account of the provided global constraints for representative selection. Therefore, it does not scale well with respect to the number of attributes, and the performance can be significantly degraded by providing restrictive constraints. Secondly, making use of k-means clustering for the purpose of representative selection can be expensive since the operation is NP-hard in general, while in our work, the worst-case performance of representative selection is much lower (see details in Section 3.5).

Dionysis *et al.* [12] propose a method that allows users to specify their perception of quality in terms of user-defined quality model. Their method is based on a k-means approach to match the user defined quality model to the search engine's quality model automatically. Their work focuses on providing intuitive quality abstraction, and is not related to the optimal selection of services. Stephan *et al.* [33] propose a QoS-based service ranking and selection approach. Their approach ranks the services according to their satisfactory scores and selects the optimal service that satisfies users' QoS requirements. Raed *et al.* [22] propose a method that makes use of analytical network process (ANP) to calculate the weight associated with each QoS attribute and rank the service based on users' satisfaction degrees. [33] and [22] can only be used to choose for a single optimal service that can satisfy the users' QoS requirements. In contrast, our work aims to select a set of services that are optimal for a service composition.

In [29, 24], we propose approaches to synthesize the local time requirement for component services given the global time requirement of composite service. In this work, we focus on the set of component services that not only can satisfy the global QoS constraint, but also provide the overall optimal QoS for the composite service. In [31], we propose the usage of evolutionary algorithm to optimize the selection of competing features in software product line. In this work, our focus is to optimize the selection of competing services in service composition.

This work is related to analysis of service composition. In [16, 17, 28], we propose a method that enables integrated verification of functional and non-functional properties of service composition. In [30], we leverage genetic algorithm in calculating the optimal recovery plan during service failure. In this work, our focus is on optimizing the service selection in service composition.

7 Conclusion and Future Work

In this paper, we have addressed the optimal selection problem by proposing a new technique, namely the Probabilistic Hierarchical Refinement (PROHR). The technique considerably improves the current service selection approaches, by considering only a subset of representatives that are likely to succeed, before exploring a larger search space. The full search space will be explored only if all the smaller search spaces have failed to produce a result. The evaluation has shown great improvement over the existing methods. For future work, we plan to investigate how PROHR can be extended to other search-based software engineering [21] problems.

8 References

[1] Amazon EC2 service level agreement. http://aws.amazon.com/ec2/sla/.

[2] Amazon Elastic Compute Cloud (Amazon EC2). https://aws.amazon.com/ec2/.

[3] Microservices. http://microservices.io/patterns/microservices.html.

[4] The QWS dataset. http://www.uoguelph.ca/~qmahmoud/qws/.

[5] E. Al-Masri and Q. H. Mahmoud. Discovering the best Web service. In *WWW*, pages 1257–1258, 2007.

[6] E. Al-Masri and Q. H. Mahmoud. QoS-based discovery and ranking of Web services. In *ICCCN*, pages 529–534, 2007.

[7] M. Alrifai and T. Risse. Combining global optimization with local selection for efficient qos-aware service composition. In *WWW*, pages 881–890, 2009.

[8] M. Alrifai, T. Risse, and W. Nejdl. A hybrid approach for efficient web service composition with end-to-end qos constraints. *TWEB*, 6(2):7, 2012.

[9] M. Alrifai, D. Skoutas, and T. Risse. Selecting skyline services for QoS-based web service composition. In *WWW*, pages 11–20, 2010.

[10] D. Ardagna and B. Pernici. Global and local qos guarantee in Web service selection. In *Business Process Management Workshops*, pages 32–46. Springer, 2006.

[11] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Software Eng.*, 33(6):369–384, 2007.

[12] D. Athanasopoulos, A. Zarras, and P. Vassiliadis. Service selection for happy users: making user-intuitive quality abstractions. In *SIGSOFT FSE*, pages 32–35, 2012.

[13] M. Berkelaar, K. Eikland, and P. Notebaert. Open source (mixed-integer) linear programming system. http://lpsolve.sourceforge.net/.

[14] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 421–430, 2001.

[15] V. Cardellini, E. Casalicchio, V. Grassi, F. L. Presti, and R. Mirandola. Qos-driven runtime adaptation of service oriented architectures. In *SIGSOFT FSE*, pages 131–140, 2009.

[16] M. Chen, T. H. Tan, J. Sun, Y. Liu, and J. S. Dong. VeriWS: a tool for verification of combined functional and non-functional requirements of web service composition. In *ICSE*, pages 564–567, 2014.

[17] M. Chen, T. H. Tan, J. Sun, Y. Liu, J. Pang, and X. Li. Verification of functional and non-functional requirements of web service composition. In *ICFEM*, pages 313–328, 2013.

[18] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (wsdl) version 2.0. http://www.w3.org/TR/wsdl20/.

[19] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. Simple Object Access Protocol (SOAP) Version 1.2. http://www.w3.org/TR/soap12/.

[20] I. Gurobi Optimization. Gurobi optimizer reference manual. http://www.gurobi.com.

[21] M. Harman and B. F. Jones. Search-based software engineering. *Information & Software Technology*, 43(14):833–839, 2001.

[22] R. Karim, C. Ding, and C.-H. Chi. An enhanced PROMETHEE model for QoS-based Web service selection. In *SCC*, pages 536–543, 2011.

[23] H.-T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM (JACM)*, 22(4):469–476, 1975.

[24] Y. Li, T. H. Tan, and M. Chechik. Management of time requirements in component-based systems. In *FM*, pages 399–415, 2014.

[25] K. Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 1998.

[26] D. Pisinger. *Algorithms for knapsack problems*. PhD thesis, University of Copenhagen, 1995.

[27] M. Silic, G. Delac, and S. Srbljic. Prediction of atomic web services reliability based on k-means clustering. In *ESEC/SIGSOFT FSE*, pages 70–80, 2013.

[28] T. H. Tan. Towards verification of a service orchestration language. In *SSIRI*, pages 36–37, 2010.

[29] T. H. Tan, É. André, J. Sun, Y. Liu, J. S. Dong, and M. Chen. Dynamic synthesis of local time requirement for service composition. In *ICSE*, pages 542–551, 2013.

[30] T. H. Tan, M. Chen, É. André, J. Sun, Y. Liu, and J. S. Dong. Automated runtime recovery for QoS-based service composition. In *WWW*, pages 563–574, 2014.

[31] T. H. Tan, Y. Xue, M. Chen, J. Sun, Y. Liu, and J. S. Dong. Optimizing selection of competing features via feedback-directed evolutionary algorithms. In *ISSTA*, pages 246–256, 2015.

[32] L. A. Wolsey. Mixed integer programming. *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

[33] S. S. Yau and Y. Yin. QoS-based service ranking and selection for service-based systems. In *IEEE SCC*, pages 56–63, 2011.

[34] K. Yoon and C. Hwang. *Multiple attribute decision making: an introduction*. Sage Publications, Incorporated, 1995.

[35] T. Yu, Y. Zhang, and K. Lin. Efficient algorithms for Web services selection with end-to-end QoS constraints. *TWEB*, 1(1):6, 2007.

[36] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven Web services composition. In *WWW*, pages 411–421, 2003.

[37] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for Web services composition. *IEEE Trans. Software Eng.*, 30(5):311–327, 2004.

[38] Y. Zhang, Z. Zheng, and M. R. Lyu. Wspred: A time-aware personalized qos prediction framework for web services. In *ISSRE*, pages 210–219, 2011.