COST-EFFECTIVE SERVICE NETWORK PLANNING FOR MASS CUSTOMIZATION OF SERVICES

Zhongjie Wang, Nan Jing, Fei Xu, Xiaofei Xu
School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
rainy@hit.edu.cn, jingnanjib@sina.com, xf_01001@126.com, xiaofei@hit.edu.cn

Abstract

In real-world scenarios, massive demands might be simultaneously raised by massive customers. Mass customization (MC) of services is a cost-effective approach to deal with this scenario by offering a customizable composite service that satisfies as many personalized requirements as possible. In this paper, Service Network (SN) is employed to represent a customizable composite service, so that MC of services is transformed into a Service Network Planning (SNP) problem. A heuristic algorithm called Iterative Enhancement based Algorithm (IEA) is proposed for SNP. Massive requirements are sorted in terms of their potential benefit in the descending order, and for each requirement to be dealt with, the algorithm iteratively enhances the preceding constructed SN by the least cost instead of constructing a new one, so that the final constructed SN keeps the minimal size and complexity and reaches cost-effectiveness. For comparison, two traditional service composition based algorithms called Solution Consolidation based Algorithm (SCA) and Requirement Grouping based Algorithm (RGA) are presented. Experiments are conducted to prove the superiority of the IEA, and factors that impact the performance of IEA are exploited elaborately.

Keywords: Service Composition, Mass Customization of Services, Service Network; Iterative Enhancement Algorithm, Cost-Effectiveness

1. INTRODUCTION

Mass customization (MC) refers to the design and production of personalized or custom-tailored goods or services to meet customers' diverse and changing needs, with the objective that every customer can have exactly what he wants by adding and/or changing certain functionalities of a core product (BusinessDictionary, 2014). It is of great significance to keep a wide variation in service offerings so that the personalized demands of different customers are met at high levels and at lower prices.

If we regard web services as the basic parts and a composite service constituted by multiple web services as a customizable product, it is apparent that MC of services has a similar philosophy as the MC of products, i.e., to assemble a set of fine-grained components into a coarse-grained composite solution. Nevertheless, the differences are obvious, too: (1) the reusable parts of customized products are manually designed by product designers, while available web services come from a mass of independent service providers who publicize their services on the Internet. This indicates that the MC of services is usually conducted in an open environment and there is not pre-defined reference architecture. (2) The quantity of components being a part of a customizable product is usually limited, while available web services on the Internet are comparatively numerous and their potential compositions are almost infinite. This implies that the customization degree of services is much higher than the one of the products and, as a side effect, the customization difficulty and cost is rather higher, too.

In a word, to realize the mass customization of services in the open Internet environment in a cost-effective and efficient way, is a challenging problem in services computing research.

Research of this paper aims at a typical broker-based service mass customization scenario. Numerous services are accumulated on an open platform of a broker, and they might either be publicized by their providers or be proactively collected by the broker. Customers raise their requirements to the platform, and the broker selects proper services and composes them into composite services to satisfy these requirements.

A widely adopted tactic in practice is "*service product*", i.e., the broker offers a composite service that owns particular customizable features, and each customer customizes these features according to his own preferences. For the broker, the cost of preparing such products is comparatively low; by contrast, on account that these customizable features are manually designed by the broker, the customization degree is limited.

Another tactic is "*service composition*" which goes to the other extreme, i.e., complete personalization. A composite service is not prepared in advance, but after a specific requirement is raised. Compared with service product, its personalization degree is higher, but its cost is also higher because each composite service fits perfectly for the functionality, QoS and context constraints of one specific requirement, therefore multiple composite services are needed for multiple requirements. Another deficiency is the *efficiency*: online service selection and composition from a large number of candidate web services is a time-consuming work.

To sum up, there is a tradeoff between the personalization/customization degree and the cost-effectiveness of a mass customizable service.

Service Network (SN) is a good choice of facilitating the MC of services by combining the two tactics together to exert their respective superiorities. Researchers have found there are explicit and implicit correlations among existing web services; as a consequence, these web services can be interconnected to form an SN which possesses many more customizable features (e.g., multi-source parameters and compound service nodes, which are introduced in Section 3.2) than "service product". Besides, due to the similarities between multiple personalized requirements, there are some frequently occurring composition patterns in the corresponding composite services, and an SN also includes such patterns that could be shared and reused by different requirements. For each incoming requirement, the SN is customized and a concrete composite service is identified.

This paper focuses on the Service Network Planning (SNP) problem, i.e., given a set of customer requirements and a set of available web services, an SN is built to meet these requirements in a mass customization way with the highest profit and the lowest cost. There are two objectives to be pursued in this problem:

(1) The degree to which massive personalized requirements could be satisfied by the customization of the SN. From the perspective of sociology, the crowd behaviors usually exhibit the "*continuity*" along with the time, i.e., there are common characteristics among the requirements appearing in consecutive time. This indicates that, if the SN could well satisfy the recent emerging requirements, then the following-up ones might be met well with a great probability. The higher the personalization degree to which the SN satisfies the requirements is and the more is the number of requirements that the SN could satisfy, accordingly the higher is the customer satisfaction degree, and as a consequence, the higher is the direct and potential benefit the SN brings to the broker.

(2) Cost-effectiveness of the SN. The cost of an SN includes the Planning Cost (PC), Customization Cost (CC) and Usage Cost (UC). PC is the cost at which the broker investigates the reputation of the services and negotiates with the service providers whose services are included in the SN, and the cost of connecting the selected service together as a network. CC refers to the cost of customizing the SN to satisfy each requirement. UC refers to the price of the final customized composite services offered to the customers. All the three cost are closely related to the size and complexity of the SN, the economic attributes of the services in the SN, and the number of customizable features in the SN.

There are three challenges in SNP problem:

(1) Under the situation that the number of available web services is very large, even the traditional one-requirement-oriented service composition problem is proving to be NP-hard, moreover the SNP problem in which *n* requirements are considered in the meanwhile. How to design a highly customizable and cost-effective SN in an acceptable time complexity?

(2) The high personalization degree requires that the SN is large enough to include more customizable features, while high cost-effectiveness is diametrically opposed. How to look for the compromise between the two objectives so that both are relatively acceptable?

(3) A personalized requirement includes two types of constraints, i.e., functionality and QoS. In most of the conventional service composition approaches, either the functional or QoS requirements are to be individually handled. How to synthesize both together?

For (1), service composition research has made effective explorations. For example, the skyline-based approach filters out infeasible services before composition so that the search space is significantly reduced; the online-and-offline combined approach makes full use of the historical composition records to find out common composition patterns to narrow the search scope; and the approximate approaches, which have received the most attention, look for an acceptable solution in a limited time. We will use climbing-hilling combined with branch-and-bound to reduce the time complexity caused by the large number of candidate services.

For (2), we adopt an iterative enhancement strategy to look for an optimal tradeoff between the customization degree and cost-effectiveness. Our approach does not deal with the requirements one by one, but leverages the similarities and correlations among requirements so that the SN is iteratively constructed and reinforced using heuristic strategies to ensure that the least amount of services are composed to achieve the highest mass customization competency.

For (3), in our previous work we have proposed QoS-oriented and functionality-oriented SN planning, respectively. In this paper we will combine them together.

Briefly, the contributions of this paper are as follows: (1) we elaborate the service mass customization scenario which aims at the satisfaction of massive personalized requirements (functionality and QoS) and combine two traditional mass customization strategies together; (2) we propose a Service Network based approach to address the service mass customization issue, where an SN is a customizable composite service that could satisfy multiple personalized requirements ; (3) we put forward a Service Network Planning algorithm in which the iterative enhancement strategy is leveraged to construct an SN and achieve an optimal tradeoff between the customization degree and cost-effectiveness.

The rest of this paper is organized as follows. Section 2 surveys related work; section 3 defines the logical model of SN and clarifies how it facilitates mass customization; section 4 gives the mathematical model of the SNP problem; section 5 puts forward three potential approaches for SNP problem and the corresponding algorithms; section 6 is the experiments and comparative analysis; and finally is the conclusion and future work.

2. Related Work

2.1 One-Requirement-Oriented Service Composition

There are two types of service composition (SC) problems: the one with end-to-end QoS constraints (called QoS-aware service composition, QSC), and the one with semantics/functionality constraints (SSC).

QSC is a multidimensional, multi-objective, multi-choice knapsack problem (MMMKP) to look for the optimality of a single or multiple objectives under the multi-dimensional QoS constraints. Global optimality can be found by methods such as integer programming, but the most commonly used approach tries to look for acceptable solutions in the relatively lower time complexity using local search heuristics (Luo *et al*., 2011), or to make use of the historical composition records to find the common composition patterns (Wang, Wang, & Xu, 2012). Representative QSC methods include: greedy-based local search method (Zeng *et al*., 2004), skyline-based method (Alrifai, Skoutas, & Risse, 2010), genetic algorithm (Canfora *et al*., 2005), global and local search combined method (Alrifai, Risse, & Nejdl, 2012), multi-constraint optimal path based method (Yu, Zhang, & Lin, 2007), etc.

SSC is a typical AI planning problem. Given an initial and a target state, the planning methods identify a composite path for the transition from the initial to the target. Compared with QSC, in SSC there is not a pre-existing service process. But the difficulty is the same, i.e., when the number of candidate services is large, the search space is so huge that there are no effective algorithms with polynomial complexity (Lin *et al*., 2012). Various AI planners have been adopted for this challenge (Hatzi *et al*., 2013). Rao, & Su (2005), Peer (2005) and Oh, Lee, & Kumara (2006) have made comprehensive surveys on SSC research. Representative methods include: GraphPlan-based method (Zheng & Yan, 2008), heuristic rule based search (Hoffmann, 2000), and dynamic planning approaches (Kuzu & Cicekli, 2012).

In spite that QSC and SSC have achieved a great in theory, they are insufficient to be directly applied to the mass customization scenario, i.e., to deal with massive personalized requirements simultaneously. For example, the time complexity is more unacceptable because multiple requirements have to be dealt with one by one.

2.2 Multi-Requirement-Oriented Service Composition

A few research work have been conducted for the multi-requirement-oriented service composition problem (MSC), with the philosophy of transforming MSC problems into multiple traditional QSC/SSC ones by requirement grouping or QoS constraint decomposition, and meanwhile decreasing the execution times of QSC/SSC algorithms.

For example, Cardellini *et al*. (2007) proposed a method to group the multiple arrival requirements into a set of "*flows*" according to the similarities of QoS constraints, and requirements in the same flow are fulfilled by one composite service; Wang, Xu, & Xu (2012) presented an algorithm named Multi-Compositions-for-Multi-Requirements (MC4MR) to construct m ($\leq n$) composite services for n requirements with different QoS constraints to ensure the optimal cost-effectiveness. Jin *et al*. (2012) and Zou *et al*. (2013) found that those candidate services having higher cost performance will be more frequently selected to satisfy multiple requirements, and as a consequence, these services might be with higher load which would possibly violate their physical constraints. They proposed the methods to decompose the global QoS constraints into a set of local QoS constraints on each task and again decompose in terms of different users, then finally transform into the traditional QSC problem. But for n requirements, the same number of composite services must be constructed. Similar work could be found in (Liu *et al*., 2011), where a similar problem called global optimal service selection for multiple requesters (GOSSMR) is presented.

2.3 Methodology and Framework for Service Mass Customization

Research on the mass customization of services (Kannan & Healey, 2011; Hu *et al*., 2013) are generally conducted from the methodology and framework of service engineering, exploring the mechanism of how to develop a customizable service system for massive personalized requirements, instead of the narrow-sense composite services in service composition problem. Further, the customizable features are much broader, for example, in SaaS, a customizable feature might be located in the UI layer, business logic layer and data layer, and each customer makes configuration by his own preference to get a highly personalized SaaS instance. Typical techniques to implement such customization are meta-models and policy-based models (Shim, 2011).

This philosophy originates from the domain engineering and software product line (SPL) practice in software engineering (Alférez & Pelechano, 2011; Mohabbati *et al*., 2013). Two representative methods of this kind are configurable software engineering methodology (Becker *et al*., 2009) and Service Family (Moon *et al*., 2010; Seung *et al*., 2011). Semantic ontology (Liang *et al*., 2011), VxBPEL (Sun *et al*., 2010), and feature (tree) model (Nguyen & Colman, 2010), etc, are invented to describe the variable/customizable points and the dependencies between them (Hadaytullah, Koskimies, & Systa, 2009; Nguyen, Colman, & Han, 2011; Weidmann *et al*., 2011). These variations are then mapped to the implementation in a model-driven approach (Bucchiarone *et al*., 2010) and infect the service system with self-adaption capacities by techniques such as autonomic agent and autonomic event triggering.

Nevertheless, the service systems developed by the above-mentioned methods are essentially "*closed*", meaning that those customizable features are mostly identified and designed by the service designers based on their own experiences and domain knowledge. This limits the customization competency of service systems and violates the "*open*" characteristic of services.

2.4 Service Network Approaches

In order to improve the inadequate customization competency, a new approach called Service Network (SN) is emerging in recent years. Compared with the previous methods, it is not a single organization who dominates the design of the customizable service systems, but it is based on the open Internet environment and tries to build an open service network using an "*exhaustive spreading*" mechanism to aggregate various isolated web services and open APIs into an inter-connected network based on the similarities and correlations among the services. That is, an SN comes into being not so much by "*elaborate design*" as by "*naturally growing*".

Researchers used several different names for this phenomenon, such as Service Network (Chen, Han, & Feng, 2012), Service Eco-Systems (Huang, Fan, & Tan, 2012), Composition Service Network (Tao *et al*., 2012), Open Semantics Service Network (OSSN) (Cardoso, Pedrinaci, & De Leenheer, 2013), Global Social Service Network (Chen, Paik, & Huang, 2014), etc. A consensus is that, the more intensive business interoperability promotes the dynamic and complex interconnections between web-based software services, and a service-centric web is emerging (Danylevych, Karastoyanova, & Leymann, 2010). It has been found that an SN is a scale-free and small-world network composed of a few active services and a great amount of silent services, indicating that it follows the power law (Tao *et al*., 2012; Hwang, Altmann, & Kim, 2009; Kil *et al*., 2009). In essence, an SN represents the crowd intelligence when massive users spontaneously use these services.

Liang, Chen, & Feng (2013) adopted a snowball-based automatic service composition by exploring the semantics relations among massive web services, and based on the tracing of these potential relations, new services are constantly added into current SN so that it grows. Jung (2011) proposes a service chain identification method to discover explicit and implicit relations between services, and then uses Social Network Analysis (SNA) approach to merge multiple service chains for the flexible satisfaction of business demands. Maamar, Hacid, & Huhns (2011) and Chen, Paik, & Huang (2014) give web services the "sociability" competency, i.e., each web service knows which other web services collaborate, substitute, or compete with itself, so that the social networking approaches are utilized to build the SN. Common integration patterns are to be found in the service eco-systems to improve an SN's performance (Han, Chen, & Feng, 2014).

It is assumed that service network is a promising approach for mass customization of services. In the future, service network will become a social infrastructure, and when a service provider publicized a new service, it will be voluntarily connected into the SN; and along with more users' involvement into the SN, some relations between services might be strengthened and others be weakened. Let us envision the time when service network is like the Internet (connecting massive servers around the world) and social network (connecting massive people around the world); it connects all the publicized services around the world, no matter who offer the services and who use them. In this way, the high cost led by the one-requirement-oriented service composition is avoided. This is the reason why we use service network to facilitate the MC of services.

3. SERVICE NETWORK

3.1 Logical Architecture and Definition

Figure 1 shows an example of a simplified service network. From logical structure's point of view, a SN is composed of four parts: Input, Services, Output, and directed connections between them.

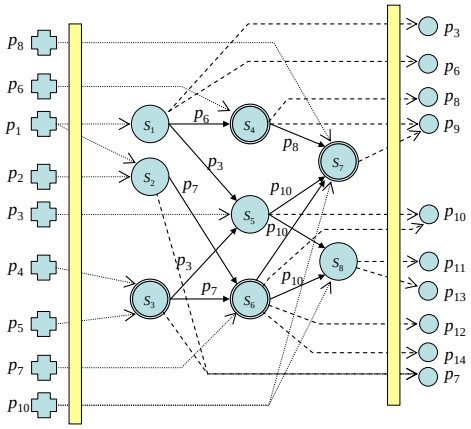


Figure 1. Logical Architecture of Service Network (SN)

Formally, a service network is defined by $SN= (I, O, S, F, G, H)$, where:

$I=\{p_i\}$ and $O=\{p_j\}$ are the input and output parameters of *SN*, respectively. Each parameter represents a distinct data item offered or expected by users, and is described in the form of standard public ontology.

$S=\{s_i\}$ is a set of abstract service nodes, and each $s_i=(I_i, O_i, A_i)$ is composed of the input parameters I_i, output parameters O_i and one or multiple concrete web services A_i. $|A_i|=1$ implies there is only one candidate service for s_i, and $|A_i|>1$ indicates s_i has multiple functionally-equivalent candidates which have the same input and output but different QoS. The j-th concrete service is represented by $a_{ij}=(I_i, O_i, T_{ij}, R_{ij}, P_{ij})$ where T_{ij}, R_{ij}, P_{ij} are the values of a_{ij}'s execution time, reliability and price, respectively.

$F=\{f_{ij}\}$ is a set of directed edges pointing from the input parameters to the service nodes, each edge $f_{ij}=p_i \rightarrow s_j$ ($p_i \in I$, $s_j \in S$) indicating an input parameter p_i is transferred to the service node s_j. $G=\{g_{ij}\}$ is the directed edges from service nodes to the output parameters, each $g_{ij}=s_i \rightarrow p_j$ ($s_i \in S$, $p_j \in O$) indicating the service node s_i will transfer its output parameter p_j directly to the output of *SN*. $H=\{h_{ikj}\}$, and $h_{ikj} = s_i \xrightarrow{p_k} s_j \left(s_i, s_j \in S\right)$ indicates that a parameter p_k is transferred between two service nodes s_i and s_j. The three types of edges jointly transform the input I to the output O layer by layer.

To note that $O(SN) = \bigcup_{s_i \in S(SN)} O(S_i)$. This implies that any output parameters of any service nodes in the *SN* could be as the output of *SN*.

An *SN* tends to come into being either spontaneously or planned by a predominant service provider on its own initiative. All the concrete services included in the service nodes are the open publicized web services or APIs. Two inter-connected service nodes have the "*related-to*" relation, and two concrete services belonging to the same service nodes have the "*similar-as*" relation.

3.2 How SN Enables Mass Customization

An *SN*'s customization refers that, given a requirement with functionality and QoS constraints, we look for an *SN*'s sub-network that satisfies both constraints. Here we briefly introduce why the *SN* is customizable.

$\forall s_j \in S, \quad \exists p_k \in I(s_j), \quad \Phi(s_j, p_k) = \{f_{kj} \mid f_{kj} = p_k \rightarrow s_j\} \cup \{h_{ikj} \mid h_{ikj} = s_i \xrightarrow{p_k} s_j\}$ is the set of directed edges from the input of *SN* or other service nodes to s_j and that transfer the parameter p_k. This implies that all the edges in $\Phi(s_j, p_k)$ have the "*or*" relations, i.e., when *SN* is customized, as long as at least one edge in $\Phi(s_j, p_k)$ is kept in the final sub-network, will the input parameter p_k of s_j be provided. For example in Figure 1, the service node s_5 has three incoming edges all transferring the parameter p_3, one from the input of *SN*, the second from s_1, and the last from s_3. Similarly, $\forall p_k \in O$, all the directed edges in $\Theta(p_k) = \{g_{ik} \mid g_{ik} = s_i \rightarrow p_k\}$ have the "*or*" relations, such as the output parameters p_9 and p_{10} in Figure 1. The existence of the relation "*or*" implies that *SN* has multiple ways of producing expected data requested in a specific requirement.

Def. 1 (**Multi-Source Parameter**) $\forall s_j \in S$, $\exists p_k \in I(s_j)$, if $|\Phi(s_j, p_k)|>1$, then p_k is a multi-source parameter. $\forall p_k \in O$, if $|\Theta(p_k)|>1$, then p_k is a multi-source parameter, too.

Secondly, a compound node is also customizable, i.e., a concrete service is to be selected from A_j to satisfy the QoS constraint in a specific requirement. In Figure 1, the compound service node is shown as double-line cycles to be distinguished from those non-compound service nodes where $|A_j|=1$.

Def. 2 (**Compound Service Node**) $\forall s_j \in S$, if $|A_j|>1$, then s_j is a compound service node.

To sum up, there are two mechanisms that *SN* holds to facilitate the mass customization: (1) the variety how a specific parameter is produced, i.e., the multi-source parameter; (2) the variety that a specific service node exhibits different QoS levels, i.e., the compound service node. The former makes the customization results have personalized process structures, while the latter makes the results exhibit personalized QoS levels. From this point of view, *SN* can be considered as the fusion of two traditional service composition approaches (SSC and QSC): AI planning based SSC methods (e.g., GraphPlan) are good at producing the variety of service inter-connections (i.e., multi-source parameters), and QSC methods are good at producing the variety of global QoS of the final solutions (i.e., to select proper concrete services for the composition), respectively.

4. Problem Definition

4.1 Personalized Requirement

A personalized requirement is composed of three parts: functionality constraints, QoS constraints, and Willing to Pay (WTP). It is denoted by $r=<I^R, O^R, Q^R, W^R>$, where:

$I^R=\{p_i\}$ and $O^R=\{p_j\}$ are the data set offered and expected by a customer, respectively. They jointly represent the functionality constraints: the customer knows his initial state (I^R) and expected state (O^R), but does not know how to achieve the transformation from I^R to O^R.

$Q^R=\{<QParam_i, QCons_i>\}$ is a set of QoS attributes and the corresponding constraints acting on the QoS of the final customized solution. For example, <*Time*, ≤10hours> and <*Reliability*, ≥0.98> are two constraints on the execution time and reliability, respectively.

W^R is the amount of money the customer is willing to pay if all the expected data in O^R could be generated and all the constraints in Q^R could be satisfied by the final customized solution.

4.2 Candidate Services

A candidate service is abstractly defined as $a=<F, I^S, O^S, Q^S, NC, UC>$, where F is the functional description of a in the form of standard ontology, I^S and O^S are a's input and output parameters, respectively, and $Q^S=\{<QParam_i, QValue_i>\}$ is a set of QoS attributes and the corresponding values declared by a's provider.

NC and *UC* are two cost-related metrics. *NC* is the Negotiation Cost, referring to the cost that a broker pays for investigating the functionality, QoS, reputation, etc, of a, and for negotiating with a's owner (service provider) to get the permission that a is to be included in the *SN*. *UC* is the Usage Cost, the price that a customer pays to the provider if a participates in the customized solution that satisfies his requirement. Some web services and APIs are free, i.e.,

$UC(a)$=0. For those paid services, $UC(a)$ is the publicized price of a. No matter whether a is a free or paid service, there always has $NC(a)$>0. In the past service computing research, people usually focus on *UC* but basically ignore *NC*. To note that, $NC(a)$ is paid once for the first time when a is added into *SN* and never needs to be recalculated regardless of how many times it is used. But every time a participants into the satisfaction of a requirement, $UC(a)$ is paid once, i.e., *pay-per-use*.

For two candidate services a_i and a_j, $F(a_i)=F(a_j)$ indicates both offer the same functionality, denoted by $a_i \overset{F}{\Leftrightarrow} a_j$. On the premise of $a_i \overset{F}{\Leftrightarrow} a_j$, if $I^S(a_i)=I^S(a_j)$ and $O^S(a_i)=O^S(a_j)$, then a_i, a_j have the same input and output, which is denoted by $a_i \overset{IO}{\Leftrightarrow} a_j$. $\forall s_i \in S(SN)$, all the concrete services in its A_i satisfies the relation $\overset{IO}{\Leftrightarrow}$, but have difference QoS values. If $a_i \overset{F}{\Leftrightarrow} a_j$ holds but $a_i \overset{IO}{\Leftrightarrow} a_j$ does not, a_i and a_j cannot belong to the same service node in *SN*.

4.3 Problem Definition

The problem of Service Network Planning (SNP) is defined as follows. Given a set of personalized requirements $R=\{r_k\}$ (k=1, 2, ..., n) and a set of candidate services $CS=\{s_l\}$ (l=1, 2, ..., m), we build a customizable service network *SN*. The expected output is refined into two aspects:

(1) *SN*, having $I(SN) \subseteq \bigcup_{i=1}^{n} I^R(r_i)$, $O(SN) \supseteq \bigcup_{i=1}^{n} O^R(r_i)$;

(2) $CR \subseteq R$ being the requirements that could be satisfied by *SN*. $CR=R$ implies all the requirements are satisfied, and $\forall r_k \in CR$, $\exists CN_k$ is a sub-network of *SN* and satisfies the functionality and QoS constraints in r_k. From the service composition's standpoint, each CN_k is a composite service.

This is a combinatorial optimization problem. Because there might be massive potential *SN* composed by the massive services in *CS*, the search space is large. The objective is to build the *SN* that satisfies as many requirements in *R* as possible with minimal cost. In other words, to maximize the benefit that the broker yields from the *SN*. This is called "*cost-effectiveness*" and denoted as:

max *Profit*(*SN*, *CR*)=*Benefit*(*CR*)–*Price*(*SN*, *CR*)–*CC*(*SN*)

Profit(*SN*, *CR*) is the net earnings from the construction of *SN* for satisfying the requirements in *CR*. Shown in Figure 2, it has three constituent parts:

- $Benefit(CR) = \sum_{r_i \in CR} W_i^R$ is the total revenue received from the customers whose requirements are included in *CR*;
- $Price(SN, CR) = \sum_{a_j \in \bigcup_{s_i \in S(SN)} A_i} \left(UC(a_j) \times |CR(a_j)| \right)$ is the Usage Cost (UC) that the broker pays to the service providers whose services participates into the satisfaction of the requirements in *CR*, and $CR(a_j) = \{ r_k \mid r_k \in R, a_j \in CN_k \}$ is the set of requirements where a_j participates.
- $CC(SN) = \sum_{a_j \in \bigcup_{s_i \in S(SN)} A_i} NC(a_j) + \mu \times |H(SN)|$ is the planning and maintenance cost borne by the broker. The former part is the Negotiation Cost (NC) for the services included in *SN*, and the latter is the maintenance cost for the inter-connections between service nodes in *SN*, where μ is the unit cost of service connections.

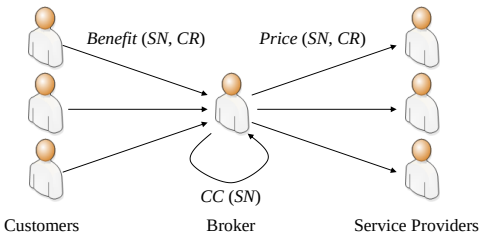


Figure 2. Cost and benefit issues in SNP problem

Constraints: $\forall r_i \in R$, $\forall QParam_{ij} \in Q_i^R$, the value of CN_i's global $QParam_{ij}$ satisfies $QCons_{ij}$.

From the optimization objective three conclusions can be drawn:

- The larger is the number of requirements that can be satisfied (i.e., $|CR|$) and the larger is the W^R that the satisfied requirements offer, the larger *Profit*(*SN*, *CR*) might be. Therefore, the SNP algorithm should give higher priority to those requirements that could bring more benefit;
- The smaller is the number of services included in *SN* and the smaller is the number of inter-connections between these services, the larger *Profit*(*SN*, *CR*) might be. Therefore, the SNP algorithm should cautiously import services and interconnections into *SN* unless they result in the profit growth of the *SN*;
- The lower UC and NC do the services in *SN* have and the higher is the number of requirements that each service participates in, the larger *Profit* (*SN*, *CR*) might be. Therefore, the SNP algorithm should try to use those services with higher cost performance and higher reusability.

But when we get down to the details, above three principles might contradict each other. For example, a requirement with higher WTP usually has more strict functionality and QoS constraints, and this implies the selected services should have better quality but consequently, higher price. Another example is: in order to use less services to satisfy more requirements, those services with higher quality will be usually selected to cover more requirements, but for those requirements with more relaxed QoS constraints, such service selection is a waste and possibly leads to the reduction of the profit; however, if we follow the "*just-enough*" principle to select the most suitable

services to exactly satisfy the QoS constraints of each requirement, the number of services and inter-connections will inevitably high. The challenge is to make an acceptable compromise on these conflicting objectives and plan an optimal *SN*.

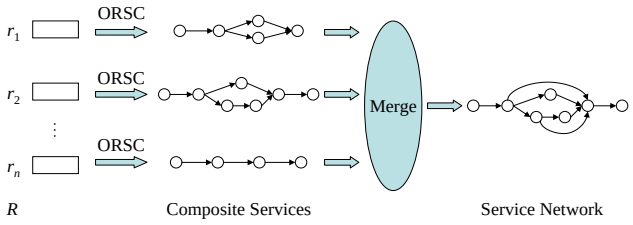
5. Optimization Algorithms

5.1 Three Potential Approaches

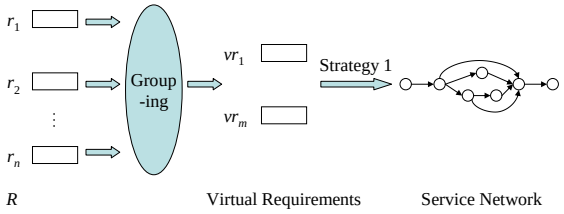
Compared with traditional service composition approaches where the number of requirements and the number of final composite services is 1:1, the SNP problem is that multiple requirements share one SN (, i.e., *n*:1). A critical step is how to transform *n*:1 SNP into one or several 1:1 service composition problem, then to apply traditional service composition algorithms to solve them, respectively. Here we use three strategies for such transformation:

- Strategy 1: To construct a composite service for each requirement, and then merge these composite services into an *SN*;
- Strategy 2: To group *n* requirements into *m* ($1 \le m \le n$) virtual ones, and then use Strategy 1 to plan the *SN*;
- Strategy 3: To sequentially deal with *n* requirements one by one and always keep one *SN* by iteratively enhancement.

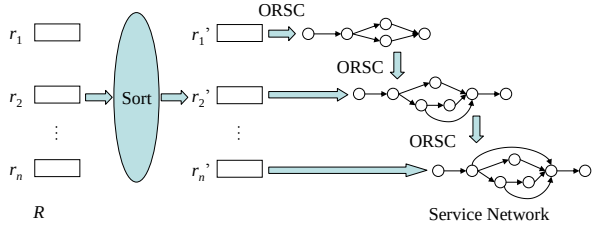
Processes of the three strategies are shown in Figure 3(a)(b)(c), in which ORSC refers to the traditional One-Requirement-oriented Service Composition algorithm.



(*a*) *Strategy 1: solution consolidation based SNP*



(*b*) *Strategy 2: requirement grouping based SNP*



(*c*) *Strategy 3: Iterative enhancement based SNP*

Figure 3. Three strategies for transforming SNP into traditional service composition problem

5.2 Solution Consolidation based Algorithm

This strategy is the direct application of traditional 1:1 service composition approaches into the MC scenario, i.e., using ORSC to plan a solution for each requirement, then merging all the solutions together and eliminating the repeated and unnecessary service nodes. The algorithm is called Solution-Consolidation-based Algorithm (SCA) and is listed below.

Algorithm 1: Solution-Consolidation-based Algorithm (SCA)

Input: *CS*, *R*
Output: *SN*, *CR*, CN_i ($1 \le i \le n$)
(1) **for** $\forall r_i \in R$
(2) $CN_i \leftarrow ORSC(r_i, CS)$
(3) **if** $CN_i \neq \varnothing$
(4) $CR \leftarrow CR \cup \{r_i\}$
(5) **end for**
(6) $SN \leftarrow Merge(CN_1, CN_2, ..., CN_n)$
(7) **return** *SN*, *CR*, $CN_{i...n}$

The advantage of SCA is straightforward: it is simple and a traditional ORSC algorithm is applied *n* times with no need for designing a new algorithm. The disadvantages are obvious, too, i.e., it does not consider the correlations between requirements (indicating that the selected services in step (2) are only used to satisfy one requirement) so that the final consolidated solution in step (6) will include a large number of services with high complexity of interconnections and at high cost.

The time complexity of SCA is subject to the number of requirements (*n*), being equivalent to *n* times of the one of ORSC algorithm.

5.3 Requirement Grouping based Algorithm

This algorithm is based on SCA, but further considers the correlations and similarities between the *n* requirements and divides them into groups according to the *coverage* relations. A virtual requirements is constructed as the representative of all the requirements belonging to the same group, and ORSC is applied for each virtual requirement. In this way, the number of requirements is compressed and the time complexity is reduced. The philosophy is straightforward: if a composite service can meet a requirement with strict constraints, then it can surely satisfy other requirements with more relaxed constraints. Here we firstly define two coverage relations between requirements and use them for requirement grouping.

Def. 3 (**Functionality Coverage between two requirements**). For two requirements r_i and r_j, the conditions $I^R(r_i) \subseteq I^R(r_j)$ and $O^R(r_i) \supseteq O^R(r_j)$ implies that a composite service that could satisfy the function constraint of r_i must satisfy the one of r_j. We call that r_i "*functionally covers*" r_j, denoted as $FCover(r_i, r_j)$.

Def. 4 (**QoS Coverage between two requirements**). For r_i and r_j where there is $FCover(r_i, r_j)$, if each QoS constraint in r_i is more strict than the constraint on the same QoS attribute in r_j, we called that r_i "*QoS covers*" r_j, denoted as $QCover(r_i, r_j)$.

In terms of the requirement grouping, there are two steps: (1) Partition R into multiple mutually disjoint subsets $\{R_1, R_2, ..., R_p\}$ by the *FCover* relation, i.e., for each $\forall r_i \in R_k$ $(1 \le k \le p)$, there is at least one $r_i' \in R_k$ that makes $FCover(r_i', r_i)$, and for $\forall r_j \in R \backslash R_k$, both $FCover(r_i, r_j)$ and $FCover(r_j, r_i)$ are false; (2) Similarly, for each $k \in [1, p]$, partition R_k into multiple mutually disjoint subsets by the *QCover* relation. Finally R is grouped into $\{R_1, R_2, ..., R_m\}$ and $1 \le p \le m \le n$.

Algorithm 2: Requirement-Grouping-based Algorithm (*RGA*)

Input: CS, R
Output: $SN, CR, CN_i (1 \le i \le n)$
(1) Partition R into $\{R_1, R_2, ..., R_m\}$ *w.r.t. FCover* and *QCover*, $\bigcup_{j=1}^{m} R_j = R$
(2) **for** $\forall j \in [1, m]$
(3) $vr_j \leftarrow \left(I_j^R, O_j^R, Q_j^R, W_j^R\right)$
where $I_j^R \leftarrow \bigcap_{r_k \in R_j} I_k^R$, $O_j^R \leftarrow \bigcup_{r_k \in R_j} O_k^R$,
$W_j^R \leftarrow \sum_{r_k \in R_j} W_k^R$, $Q_j^R \leftarrow Strictest_{r_k \in R_j} Q_k^R$
(4) **end for**
(5) $VR \leftarrow \{vr_1, vr_2, ..., vr_m\}$
(6) **return** $SCA(CS, VR)$

Step (1) partitions the initial requirements R into m subsets and each subset is a partially ordered set in terms of *FCover* and *QCover*. For each R_j, step (2) and (3) construct a virtual requirement vr_j with the least input parameters, the most output parameters, the total WTP, and the strictest QoS constraints of all the requirements in R_j. Step (6) invoke SCA algorithm to plan the final SN.

Compared with SCA, RGA compresses the requirements and transforms the initial n:1 composition problem into an m:1 ($m \le n$) one. Therefore, the time complexity decreases n/m times. In the worst case where $m=n$, i.e., there are not any *FCover* and *QCover* relations among all the requirements, so SCA and RGA have the same efficiency. In the best case where $m=1$, i.e., all the requirements are mutually covered each other, the time complexity of RGA is equivalent to the one of ORSC. Besides, the number of service nodes and connections in the final SN has lowered a great deal because multiple requirements in the same group share the common composite solution.

But as mentioned before, for two requirements r_i, r_j in the same equivalence class, if r_i's QoS constraints is extraordinarily stricter than r_j's, the virtual requirement must follow the constraints of r_i. This violates the "*just-enough*" principle and results in cost-ineffectiveness. From this perspective, the output SN is not optimal. On the other hand, RGA only considers *FCover* and *QCover* relations which look somewhat strict. For example, suppose r_1 expects to get the output $\{a, b, c\}$ by offering the input $\{y, z\}$, and r_2 expects to get $\{b, d\}$ by offering $\{x, z\}$; in RGA, r_1 and r_2 do not satisfy the *FCover* relation, hence the ORSC algorithm is to be invoked twice to construct the solutions for each, respectively. However, there are indeed some similarities, e.g., both provide b and expect z. To further improve the optimality, we go to the third strategy, Iterative Enhancement based Algorithm (IEA).

5.4 Iterative Enhancement based Algorithm

From above analysis, we get a heuristic rule: it had better utilize more shared services and connections that have the capacity of satisfying multiple requirements, so that the negotiation and maintenance cost are to be averaged, and consequently, the total cost is to be reduced.

Here we introduce the iterative enhancement strategy based on above rule. Suppose that a service network $SN^{(i)}$ has been constructed for the first i requirements $\{r_1, r_2, ..., r_i\}$, we first check whether r_{i+1} could be fully satisfied by $SN^{(i)}$. If yes, it is not necessary to invoke the ORSC algorithm for r_{i+1}; otherwise, ORSC is called to import new services and connections and merge them into $SN^{(i)}$ to get a larger $SN^{(i+1)}$. During the ORSC, it is better to reuse the existing services and connections in $SN^{(i)}$ because their *CC* have already been "*sunk cost*", and secondly, the new imported services should follow the "*just-enough*" principle for the satisfaction of the constraints of r_{i+1}. Such iterations continue until all the requirements have been coped with and a final $SN^{(n)}$ is obtained. In each iteration, the criterion whether a requirement is to be satisfied or not is based on the cost-effectiveness, i.e., the satisfaction of a requirement will increase the profit of $SN^{(i)}$.

A critical point in this process is in what order the requirements are dealt with. We use a metric "*Potential Benefit*" to sort the requirements with the objective that the requirements which would bring more benefit to the broker stand in the front of the queue.

Algorithm 3: Iterative Enhancement based Algorithm (IEA)

Input: CS, R
Output: $SN, CR, CN_i (1 \le i \le n)$
(1) $R \leftarrow SortByPotentialBenefit(R)$
(2) **if** $|R|=1$
(3) $SN^{(n)} \leftarrow ORSC(r_n, CS)$, $CN_n \leftarrow SN^{(n)}$, $CR \leftarrow CR \cup \{r_n\}$
(4) **return**
(5) **else**
(6) $SN^{(n-1)} \leftarrow IEA\left(R \setminus \{r_n\}, CS\right)$
(7) $\overline{SN^{(n-1)}} \leftarrow Prune\left(SN^{(n-1)}, I\left(SN^{(n-1)}\right) \setminus I_n^R\right)$
(8) **if** $O_n^R \subseteq O\left(\overline{SN^{(n-1)}}\right) \wedge \overline{SN^{(n-1)}}$ satisfies Q_n^R

(9) $SN^{(n)} \leftarrow SN^{(n-1)}, CN_n \leftarrow SN^{(n)}, CR \leftarrow CR \cup \{r_n\}$
(10) **return**
(11) **for** $\forall a_i \in \bigcup_{s_j \in S\left(\overline{SN^{(n-1)}}\right)} A_j$
(12) $uFlag(a_i) \leftarrow 1, NC(a_i) \leftarrow 0$
(13) **for** $\forall h \in H\left(\overline{SN^{(n-1)}}\right)$
(14) $uFlag(h) \leftarrow 1$
(15) $SN^{(n)} \leftarrow ORSC(r_n, CS)$
(16) $SN^{(n)} \leftarrow Merge(SN^{(n-1)}, SN^{(n)})$
(17) **return**

Step (1) calls *SortByPotentialBenefit*() to sort the requirements in terms of the potential benefit $PB(r_i)$ in the descending order. Step (2)-(4) deals with the case when n=1, and the ORSC algorithm is invoked to get the solution. If n>1, step (6) recursively calls IEA algorithm and get a service network $SN^{(n-1)}$ for the first n–1 requirements. Step (7) calls *Prune*() to prune $SN^{(n-1)}$ to eliminate the input parameters $I\left(SN^{(n-1)}\right) \setminus I_n^R$ that are not offered by r_n, and the resulted unavailable service nodes and output parameters, then gets a result $\overline{SN^{(n-1)}}$ all the constituents of which possibly participate into the satisfaction of r_n. Step (8) checks whether $\overline{SN^{(n-1)}}$ could generate all the expected output paramters of r_n and satisfy the QoS constraints of r_n. If so, it is not necessary to re-plan a composite solution for r_n and the algorithm ends in step (9) and (10). Otherwise, in step (11)-(14) all the already-existing concrete services and connections in $\overline{SN^{(n-1)}}$ are marked by a flag "1" and set their negotiation cost as 0, indicating that these services are no longer required to pay the negotiation cost. In step (15) the ORSC algorithm are again called to get the solution for r_n and merge with $SN^{(n-1)}$ to get $SN^{(n)}$.

There are two extreme cases:

(a) $O_n^R \cap O\left(\overline{SN^{(n-1)}}\right) = \varnothing$: indicating that $\overline{SN^{(n-1)}}$ cannot generate any expected output parameters of r_n, i.e., the services and connections in $SN^{(n-1)}$ have no any help to r_n;

(b) $O_n^R \subseteq O\left(\overline{SN^{(n-1)}}\right)$ and $\overline{SN^{(n-1)}}$ satisfies Q_n^R: indicating that $\overline{SN^{(n-1)}}$ could generate all the expected output parameters of r_n and satisfy all the QoS constraints, i.e., no enhancement is required.

The time complexity of IEA depends on the times that the ORSC algorithm is invoked. If for each iteration the case (b) is always true, ORSC is invoked only once (the best case). If each iteration falls into the case (a), ORSC is called n times (the worst case). The key is to let each iteration fall into the case (b) and avoid (a) as far as possible. This relies heavily on the requirement sorting in step (1). We go to the introduction of how to estimate the potential benefit of a requirement in the next section.

5.5 Estimation of Potential Benefit of a Requirement (PB)

$PB(r_i)$ measures the potential benefit that is produced by the composite solution of r_i. It is composed of two parts: (1) the benefit received from the payment of the customer who raises r_i; (2) if r_i's composite solution can satisfy other requirements $\{r_{j1}, r_{j2}, ...\}$, then it is not necessary to construct new solutions for these requirements, and the payment of the corresponding customers is considered as r_i's benefit, too. The larger is $PB(r_i)$, the higher priority r_i is to be dealt with during IEA, and the higher probability the follow-up iterations fall into the case (b) with.

$PB(r_i)$ cannot be precisely measured before r_i's composite solution is actually constructed. Here we use an estimation based on the "*similarity*" between two requirements, i.e., $D^F(r_i, r_j)$ and $D^Q(r_i, r_j)$. The former measures the similarities between the input and output parameters of r_i and r_j, and the latter measures the similarities between the QoS constraints of r_i and r_j. Larger similarity implies that the corresponding solutions might be more similar, and the satisfaction of r_i will have a high probability of the satisfaction of r_j.

For the functional similarity between r_i and r_j, $D^F\left(r_i, r_j\right) = \xi^O\left(r_i, r_j\right) \times \xi^I\left(r_i, r_j\right)$, where $\xi^O\left(r_i, r_j\right) = \frac{\left|O_i^R \cap O_j^R\right|}{\left|O_j^R\right|}$ and $\xi^I\left(r_i, r_j\right) = \frac{\left|I_i^R \cap I_j^R\right|}{\left|I_i^R\right|}$, and $D^F(r_i, r_j) \in (0,1]$.

For the QoS similarity between r_i and r_j, $D^Q\left(r_i, r_j\right) = \sum_{QParam_k \in Q_i^R, Q_j^R} Strict_k\left(r_i, r_j\right)$, where $\forall QParam_k \in Q_i^R, Q_j^R$ being a common QoS attribute included in r_i and r_j, if $QCons_k(r_i) \geq QCons_k(r_j)$ (indicating r_i requests a stricter $QParam_k$ than r_j), then $Strict_k(r_i, r_j)$=1. $D^Q(r_i, r_j)$ is a non-negative integer.

To sum up, the potential benefit is estimated by:

$$PB\left(r_i\right) = W_i^R + \sum_{r_j \in R, r_j \neq r_i} \left(D^F\left(r_i, r_j\right)^{\frac{1}{D^Q(r_i, r_j)+1}} \times W_j^R \right)$$

where W_i^R and W_j^R are the benefit gained from r_i and r_j, respectively, and $D^F\left(r_i, r_j\right)^{\frac{1}{D^Q(r_i, r_j)+1}}$ represents how much probability there is that the satisfaction of r_i will lead to the satisfaction of r_j. The reason why $D^Q(r_i, r_j)$ is placed as an exponent of $D^F(r_i, r_j)$ is that functional similarity has greater impact on the final solution than the QoS similarity. And the greater are the $D^F(r_i, r_j)$ and $D^Q(r_i, r_j)$, the larger $PB(r_i)$ is.

5.6 One-Requirement Oriented Service Composition Algorithm (ORSC)

Here we introduce the ORSC algorithm used in SCA, RGA and IEA. ORSC gets the input of a requirement and a set of candidate services, and generates a composite service that satisfies the functionality and QoS constraints in the requirement. Obviously there is not a pre-existing abstract

service process, so traditional QSC algorithms are unfit for ORSC. Here we propose the ORSC algorithm based on the traditional SSC approaches.

In AI planning based SSC algorithms, the input parameters in the requirement are considered as the initial state, the expected output parameters are considered as the target state, and each candidate service is considered as the "*action*", then specific AI planners are employed to look for a composite service that transform from the initial to the objective state. In the planning process, the composite solution gradually comes into being, i.e., before the functionality constraints are fully satisfied, the solution is always partial. It is easy to check the satisfiability of functionality constraints (e.g., by judging whether all the expected output parameters have been produced) but it is difficult to check the satisfiability of QoS constraints because this checking should be based on a complete solution instead of a partial one. For example, different process structures lead to different calculations of getting the global *execution time* from the execution time of each constituent web service included in the process.

To address this issue, we present a Planning Graph based algorithm combined with *branch-and-bound* and *hill-climbing* strategies. Aiming at a requirement r and the candidate services CS, the algorithm looks for a cost-effective composite solution CN.

Algorithm 4: ORSC

Input: CS, r
Output: CN
(1) $PG \leftarrow PlanningGraph(CS, r)$
(2) **if** $PG \neq \varnothing$
(3) **return** $SolutionTreeSearch(PG, r)$
(4) **return** $\varnothing$

In step (1) a planning graph PG is generated by the classical forward chaining approach. A planning graph is a layered directed graph, denoted by $PG=<P_0, S_1, P_1, .., S_T, P_T, M>$ where $\{P_0, P_1, ..., P_T\}$ are parameter layer, $\{S_1, S_2, ..., S_T\}$ are service layers, and M is a set of parameter transferring relations (directed edges) between parameter and service layer. P_0 is the "initial state" and P_T is the "target state" of the planning, and PG contains all the possible composite solutions of r. For the algorithm $PlanningGraph(CS, r)$ that generates PG, please refer to the algorithm *Compose*() and *Expand*() in Zheng & Yan (2008) and we will not repeat here. If some expected output parameters in r cannot be generated by services in CS, then the algorithm returns $PG=\varnothing$. Figure 4 shows an example of planning graph with two service layers and three parameter layers.

Next, we use the hill-climbing and branch-and-bound strategies to conduct the backward search in the planning graph and look for an optimal solution. This is called $SolutionTreeSearch(PG, r)$ in step (3). $PG \neq \varnothing$ implies there must be at least one feasible solution that satisfies the functional constraints of r, but whether the QoS constraints are satisfied should be checked during the search.

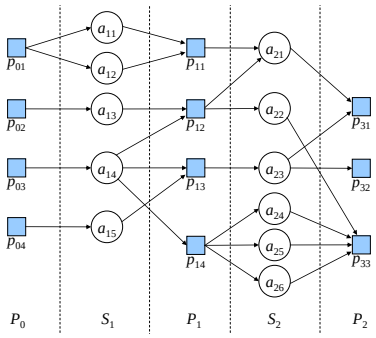


Figure 4. A planning graph example

Def. 5 (Solution Tree). A solution tree $ST=<root, N, E>$ is a layered tree representing the dynamic search process on the planning graph PG, where $root \in N$ is the root node of ST; $N=\{node_i\}$ is a set of nodes each of which represents a partial or complete solution obtained during the search; $<node_i, node_j> \in E$ is a directed edge between nodes, $node_j$ is generated by adding a new concrete service and the corresponding edges into $node_i$, and there have $node_i=Father(node_j)$ and $node_j \in Children(node_i)$. If $<node_i, node_j> \in E$ and $<node_i, node_k> \in E$, then $node_j$ and $node_k$ are siblings, indicating that they are two different extensions of $node_i$.

A node in N is further refined as $node=<SList, LPList, DPList>$, where

- $SList=\{a_k\}$ is the set of concrete services included in this solution;
- $LPList$ is a set of input parameters that are required by this solution, but cannot be offered by r;
- $DPList \subseteq O^R(r)$ is a subset of r's output parameters that could be produced by this solution.

For the *root* being a special solution, there are $SList=\varnothing$, $LPList=O^R(r)$ and $DPList=\varnothing$, i.e., the initial solution where no any services are selected from PG. Each leaf-node of ST represents a feasible solution of r, i.e., $DPList=O^R(r)$ and $LPList=\varnothing$. Each non-leaf node is a partial solution, i.e., $DPList \subset O^R(r)$. The algorithm $SolutionTreeSearch(PG, r)$ is to find an optimal leaf-node with the best profit by iteratively and heuristically extending and pruning the ST according to the structure of PG, until there are not any nodes that can be further extended and ST becomes empty. The pseudo-code of $SolutionTreeSearch(PG, r)$ is given below.

Algorithm 5: SolutionTreeSearch

Input: PG, r
Output: opt
(1) $opt \leftarrow \varnothing, ST \leftarrow \varnothing$
(2) $ST.root \leftarrow <\varnothing, O^R(r), \varnothing>$
(3) **while** $ST \neq \varnothing$

(4) $node \leftarrow HeuristicSelect(ST)$
(5) $p \leftarrow ParamWithLargestLayer(PG, node.LPList)$
(6) $AvS \leftarrow \{a \mid a \in PG \wedge p \in O^S(a)\}$
(7) **for** $\forall a \in AvS$
(8) $child \leftarrow <node.SList \cup \{a\}, node.LPList \cup I^S(a) \setminus O^S(a), node.DPList \cup O^S(a)>$
(9) **if** $EvaluateQoS(child)=false$
(10) **continue**
(11) **if** $child.LPList = \varnothing \wedge CalcProfit(child) > opt.Profit$
(12) $opt \leftarrow child$
(13) **if** $child.LPList \neq \varnothing$
(14) $Father(child) \leftarrow node$
(15) **end for**
(16) **if** $Children(node) = \varnothing$
(17) $ST \leftarrow TreePruning(ST, node)$
(18) **end while**
(19) **return** *opt*

The variable *opt* keeps the optimal solution during the iterations. Step (2) creates the root node of *ST* as the starting search point. Step (3)-(18) is a complete iteration. In step (4) a non-leaf *node* with the maximum profit is greedily selected from current *ST* for the extension in the current iteration, which is what is called the "*hill-climbing*" in terms of the profit of nodes. Step (5) identifies a parameter *p* from *LPList* of *node* such that *p* has the largest layer number in *PG*. Step 6 finds all the concrete services *AvS* in which each service could produce *p* and might be potentially extended into *node* to form a new solution. Every time step (8)-(14) are executed, each concrete service *a* in *AvS* is added to *node* and there forms a new solution *child* that locates in the next layer of *node*. Step (8) is to construct the *SList*, *LPList*, *DPList* of *child*. Step (9) checks whether *child* satisfies the QoS constraints of *r*. If the QoS constraints are violated, *child* is an illegal solution and will not be added into *ST*; otherwise, in step (11) the profit of *child* is calculated. If *child* is a leaf-node (i.e., its $LPList=\varnothing$ and it is a complete solution) and its profit is larger than *opt*, then *opt* is substituted by *child* (step (12)). If *child* is a non-leaf node (i.e., its $LPList\neq\varnothing$ and it is a partial solution), then *child* is added into *ST* as a child-node of *node* for further extension (step (13)-(14)). This implies that, if *child* is not superior to *opt*, future extension on it becomes unnecessary, and it is not added into *ST*, so that the search space in *G* is cut down. This is what is called "*branch-and bound*". After the execution of step (7)-(15), *ST* might be extended by either adding one non-leaf node under *node* or without any changes. "*No changes*" implies that all the extended solutions of *node* are checked to be infeasible or have been deleted, then *node* makes no sense and should be deleted from *ST*, so the algorithm goes to step (16)-(17) to prune *node* and its ancestors if a node satisfies the condition that it has been extended but no child nodes are kept in *ST*. The iterations continue, until the tree becomes empty, indicating that all the potential and feasible solutions have been explored, and *opt* is the final optimal solution.

To note that, in step (9) we check whether *child* satisfies the QoS constraints in *r*. As mentioned before, it is difficult to do such checking because the solutions generated during the search might be partial. Here we classify various QoS attributes into structure-unrelated (e.g., *Reliability* and *Cost*) and structure-related ones (e.g., *Execution Time*). The former refers that the calculation of their values is only related to the values of the constituent services but has nothing to do with the structure of the composite service, and the latter depends on both the QoS values of the constituent services and the structure of the composite service. If *child* is a complete solution, we check all its QoS attributes, but if it is a partial one, only the structure-unrelated QoS attributes are to be checked. If a partial solution violates some QoS constraints, it is not necessary to make further extensions on it because more extension will bring about more serious violations of the same QoS constraints. This is also what is said "*branch-and bound*" in this algorithm, too.

6. Experiments And Comparative Analysis

6.1 Data Preparation

The experiments are for the three algorithms (SCA, RGA and IEA) and comparisons are made on the execution efficiency, the complexity of the generated SN, and the profit (cost-effectiveness). They are conducted on Microsoft Windows 7 (32bit), Intel® Core i3 3.10GHz processors and 2.92GB RAM. The program is implemented in Java (Eclipse 4.3+JDK1.7).

For the candidate services, we employ two web service datasets (QWS and WS-DREAM) and a self-generated dataset. 1,000 web services are extracted from the open datasets and their functionality/input/output are semantically unified. To ensure the impartiality of the results, 1,500 new services are generated by the following rules:

- 1,000 distinct data parameters are firstly selected from the parameters of the services from QWS and WS-DREAM. They are divided into 20 disjoint groups according to the similarities in their semantics;
- For each service, 1-11 parameters are randomly selected as its input, and 16-26 ones are randomly selected as its output. Either the input or output parameters of a service is selected from at most 3 different parameters groups. The number of candidate services having the same functionality and the same input/output parameters are 4 on average.
- The values of three QoS attributes (*Execution Time*, *Reliability* and *Price*) of a service range arbitrarily in [1,1000] seconds, [0.8,0.99], and [78,96335] dollars, respectively. For the *Price* (i.e., *UC*), its value is partially based on the values of the other two attributes;

in other words, a service with better performance is likely to have higher *Price*, and vice versa.

- The value of *NC* of a service ranges in [800,969000] dollars. It is not randomly generated but is based on the value of the *Price* (i.e., the *UC*). The *NC*-to-*UC* ratio ranges from [0.25,10]. This is to simulate the reality: for some real-world services, it takes much cost to investigate them and negotiate with their providers, while for some others, this cost is low.

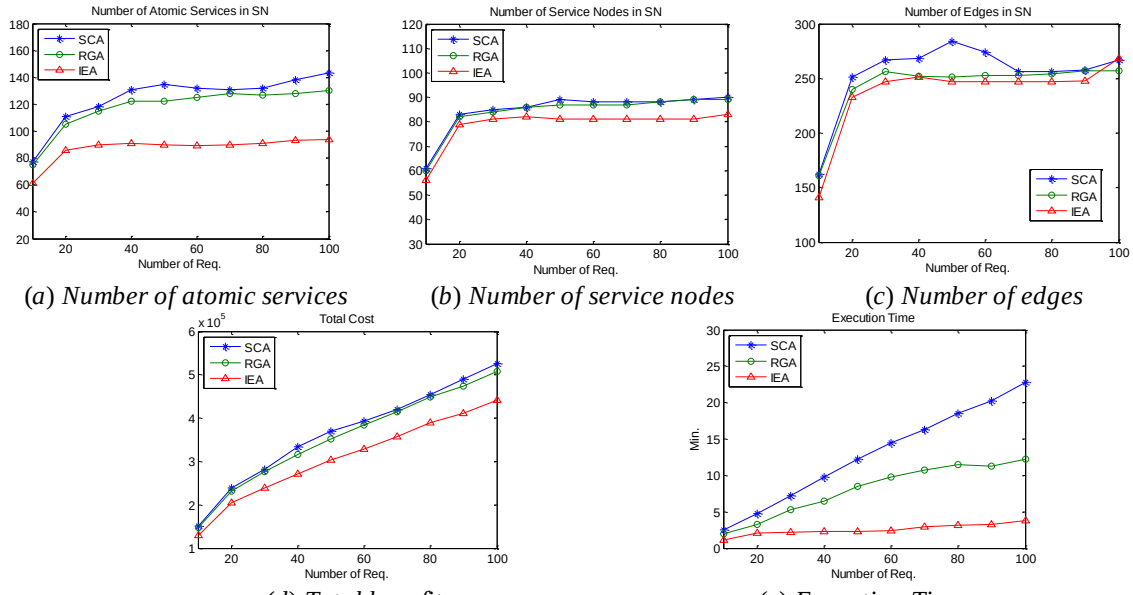
Totally 1,000 requirements are prepared. As there are not off-the-shelf requirement datasets, the requirements are generated by the rules below:

- The number of input parameters of each requirement ranges in [7,27], and the number of its output parameters is 5;
- The QoS constraints in a requirement are in terms of the *Execution Time* and *Reliability*, the value of which range in [3100,11100] seconds and [0.1,0.52], respectively. Some requirements are endowed with more strict QoS constraints, and others are with more relaxed ones.
- Concerning the WTP of a requirement, similar as the Price of a candidate service, there is also a metric called QoS/WTP Alignment Degree. The WTP of a majority of the generated requirements is aligned with the strictness of its QoS constraints (i.e., the more strict are the constraints, the larger is the WTP). A small number of requirements are with strict QoS constraints but smaller WTP, or with relaxed QoS constraints but larger WTP.

Based on these data, total five experiments are designed. Experiment 1 is to make comparison on the three SNP algorithms (SCA, RGA and IEA) in terms of the time complexity, the complexity of the generated SN, and the benefit/cost of the generated SN. Experiment 2 is to observe the performance fluctuation during the iterations of IEA to validate the philosophy of IEA. Experiment 3-5 are used to check the effects that three factors have on the performance of IEA, including QoS strictness of the incoming requirements, the criteria of sorting requirements before the iterative enhancement, and the ratio between NC/UC of candidate services.

6.2 Experiment 1: Performance *w.r.t.* Number of Requirements

The objective of this experiment is to compare the performance of the three SNP algorithms (SCA, RGA and IEA). Based on the same candidate services set, the algorithms execute to look for the optimal SN for different amounts of customer requirements (n=10, 20, ..., 100). There are five performance indicators in the comparison, i.e., the number of atomic services in SN, number of service nodes in SN, total cost of SN, and the execution time of algorithms.



(*a*) *Number of atomic services* (*b*) *Number of service nodes* (*c*) *Number of edges*

(*d*) *Total benefit* (*e*) *Execution Time*

Figure 4. Performance comparisons between three SNP algorithms w.r.t. number of requirements

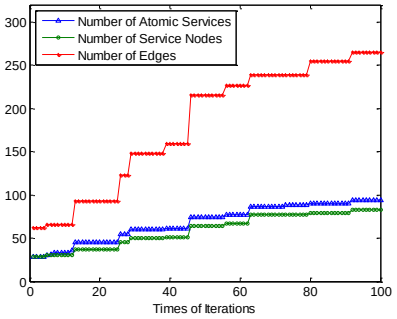
The results are shown in Figure 4. Compared with SCA and RGA, IEA shows better performance. Specifically, in terms of the same amounts of personalizes requirements, the number of service nodes, atomic services and edges in the SNs generated by IEA are all comparatively smaller than the ones by SCA and RGA, implying that IEA produces SNs with smaller sizes and complexity. This also results in lower cost. In addition, IEA exhibits better execution efficiency than SCA and RGA: the execution time of IEA increases at a very low speed due to the sorting and iterative

enhancement strategy, but the one of SCA is directly proportional to the amount of requirements. The comparison between SCA and RGA proves the advantage of requirement grouping: although the improvement degree on the size and complexity of SN are not quite large, the execution time decreases obviously. It is also seen that all the five indicators increase along with the increasing amount of customer requirements, i.e., more requirements result in larger and more complex SN with more cost, and consequentially, more execution time to plan the SN.

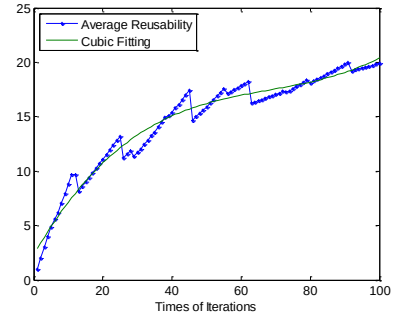
This conclusion proves that, traditional one-requirement-oriented service composition approaches are overburdened in the mass customization scenario, but our iterative enhancement strategy behaves better by generating a smaller SN at lower cost and higher efficiency.

6.3 Experiment 2: Performance Fluctuation during the Iterations of IEA

In this experiment we observe the performance fluctuations during the iterations of IEA for 100 personalized requirements. There are totally 100 times of iterations during the execution, and each iteration deals with one requirement. Figure 5 shows the results.



(*a*) *Size and complexity of the SN*



(*b*) *Average reusability of services*

Figure 5. Performance fluctuations during IEA's iterations

From Figure 5(a) we see that, the number of service nodes, atomic services and edges in the SN all increase in a step-wise manner, and the growth rate becomes increasingly slow. This is in accordance with the IEA philosophy, i.e., those requirements that row in the forefront of the queue could cover more other requirements but they are seldom mutually covered, therefore more service node and atomic services are imported to satisfy them; but for the subsequent requirements, they could be satisfied by the existing SN with a higher probability so that less services and edges are to be enhanced to the exiting SN.

Average reusability refers to the average number of requirements that each atomic service participates in. From Figure 5(b) we can see that, the average reusability keeps increasing along with more and more requirements. This is because IEA tries to utilize the previously selected services to satisfy new-incoming requirements instead of importing new ones. But to note here is that it shows some sudden drops. This implies that the incoming requirement cannot be completely covered by previous ones, so that new services have to be imported into current SN. Even so, the average reusability increases linearly on the whole (see the cubic polynomial fitting in the figure).

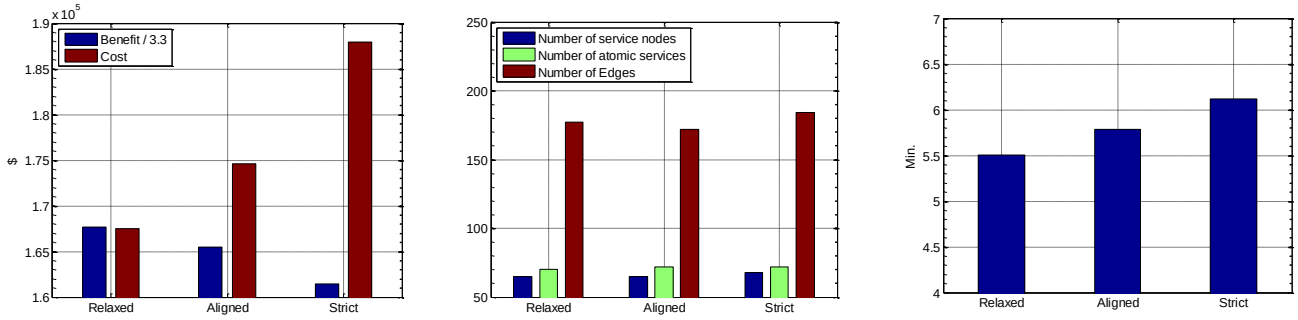
6.4 Experiment 3: Performance *w.r.t.* QoS Strictness of Requirements (QS)

Different customers will raise different requirements, not only on the functionality but also on the QoS. Some requirements have more strict QoS constraints relative to the Willing To Pay (WTP) that the corresponding customers pay, and some others have more relaxed QoS constraints. Different QoS strictness of requirements will have some impact on the efficiency and performance of IEA.

Here we use three groups of requirements, each one having 20 distinct requirements. The functionality constraints and WTP are identical, and the only difference is that the requirements in Group 1 have quite relaxed QoS constraints, the ones in Group 3 have quite strict QoS constraints, and the ones in Group 2 have aligned QoS constraints. IEA is applied to the three groups, respectively, and Figure 6 shows the comparisons between the three groups of requirements which are labeled by Relaxed, Aligned and Strict, respectively.

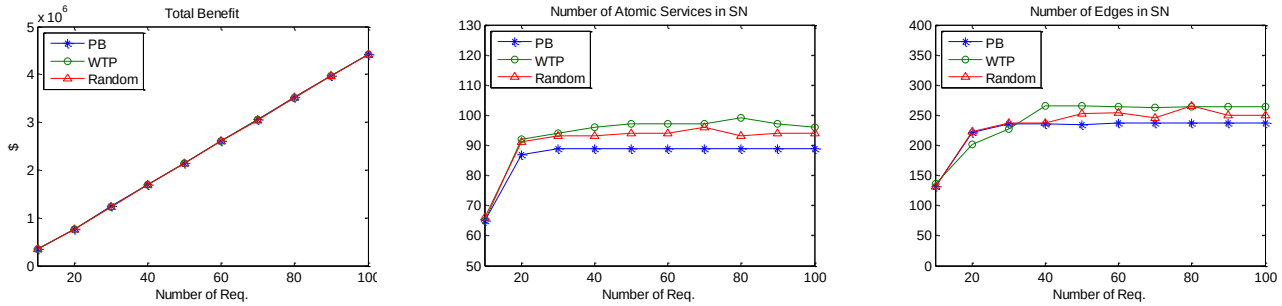
Figure 6(a) is the comparison on the benefit and cost of the generated SN for the three groups of requirements. The SN for the Relaxed group yields the maximum benefit and minimum cost, and the one for the Strict group does the opposite. The reason is obvious: requirements with more strict QoS constraints require that the SN planning algorithm selects those services with higher QoS and consequently, higher UC; therefore, the total cost is higher. Because the WTP that the customers pay is fixed, the total benefit gained from the satisfaction of these strict requirements are meanwhile lower.

Figure 6(b) is a comparison of the complexity of the generated SN, including the number of service nodes, the number of atomic services, and number of edges. The SN for the Strict group has the maximal complexity, while the SN for the Relaxed group has the minimal complexity, but the difference is not that big, indicating that the QS of requirements does not much affect the complexity of the generated SN.



(*a*) *Benefit and cost* (*b*) *Complexity of the SN* (*c*) *Execution time*

Figure 6. Performance comparisons among three groups of requirements having different QS



(*a*) *Total benefit* (*b*) *Number of atomic services* (*d*) *Number of edges*

Figure 7. Performance comparisons w.r.t. three requirement sorting methods

Figure 6(c) is the comparison of the execution time of IEA. It can be seen that the Strict group requires more time than the other groups. This is because the more strict QoS requirements make more potential solutions being infeasible during the ORSC search, so the *SolutionTreeSearch*(*PG*, *r*) algorithm has to make deeper search that takes more time.

6.5 Experiment 4: Performance *w.r.t.* Sorting Criteria of Requirements

In IEA, the first step is to sort the requirements in terms of the potential benefit in the descending order, so that the requirements that could bring more benefit will be dealt with preferentially. Actually, there are other sorting criteria, such as sorting by WTP of the requirements (i.e., requirement with the highest WTP is given top priority), and sorting randomly (i.e., the order by which these requirements are dealt with is not considered). This experiment tries to verify whether and how different sorting criteria have influences on the performance of the SN constructed by IEA.

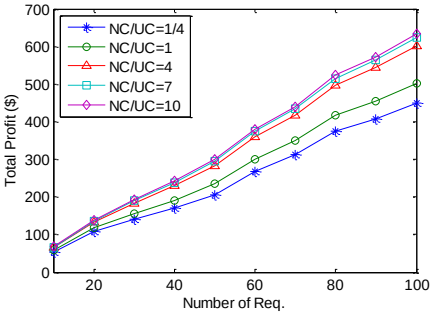
From Figure 7(a) we see that the total benefits under the three sorting criteria are almost coincident, implying that the sorting criteria will not affect the cost-effectiveness of the constructed SN. However, from Figure 7(b) and (c) we see the difference in the complexity of the constructed SN (number of service nodes, the number of atomic services, and number of edges), and in most cases, sorting by PB reduces the complexity of the SN in a certain degree. Another conclusion is that sorting by WTP will result in the highest complexity, implying that "*greedy*" is not a good sorting criterion. WTP is the direct benefit gained from the satisfaction of a requirement, while PB is the sum of both direct and potential benefits, and sorting by PB will make the algorithm select those services that might cover more other requirements, instead of only one requirement, so that the reusability of the selected services is improved. This experiment proves the effectiveness of the sorting by PB adopted in IEA.

6.6 Experiment 5: Performance *w.r.t.* Ratio between NC/UC of Candidate Services

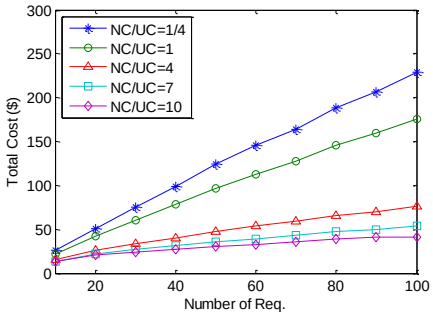
In this experiment, we try to discover whether the ratio between the negotiation cost (NC) and the usage cost (UC) of candidate web services affects the performance of the SN constructed by IEA. In terms of different amounts of requirements (10, 20, ..., 100), IEA is executed under different NC/UC settings (1/4, 1, 4, 7, and 10). To note that, the sum of NC and UC of each candidate service remains the same, e.g., NC=10, UC=40, NC/UC=1/4; NC=25, UC=25, NC/UC=1; and so on. Figure 8 shows the comparisons on the total profit and cost of the constructed SN.

From the figure we find that, with the ever increase of NC relative to UC, the benefit brought by the SN is likely to inflate and the total cost of the SN is likely to decrease. This confirms the advantage of the iterative enhancement strategy, i.e., during the execution of ORSC in each iteration, the higher NC (and correspondingly, the smaller UC) will

make the search tend to reuse those selected services because the added cost is only UC; and if UC is comparatively larger than NC, the significance of service sharing among multiple requirements becomes more trivial, so that the algorithm ties to look for new services instead of reusing existing ones, and consequently, the total cost increases and the profit decreases.



(*a*) *Total profit*



(*b*) *Total cost*

Figure 8. Performance w.r.t. ratio between NC/UC of candidate services

7. Conclusion and Future Work

The "personalization" and "heterogeneity" characteristics of services require that service providers offer different services to different customers, which is what various service composition algorithms tries to achieve. But due to the cost issue, real-world service providers usually realize partial personalization using "service product" strategy alike the products in the manufacturing domain. It is a challenging issue on the tradeoff between the higher personalization from the customer's viewpoint and the higher cost-effectiveness from the service provider's viewpoint. Research on the mass customization of services is of significance to both research and practice.

This paper uses Service Network (SN) as a tool to deal with the scenario where massive customers raise massive personalized requirements. An SN is considered as the combination of two traditional service composition approaches: the AI planning based SSC approaches (e.g., GraphPlan) aiming at the variety of connections between services, and the combinatorial optimization based QSC approaches (e.g., Skyline) aiming at the variety of global QoS performance. In SN, the corresponding means of facilitating the two varieties are "multi-source parameters" and "compound service node", respectively. The core algorithm for Service Network Planning (SNP) problem is IEA, which considers the similarities and correlations of the functionality and QoS constraints among multiple requirements, and uses an iterative enhancement strategy to continuously extend an existing SN. Compared with two traditional approaches, i.e., SCA and RGA, a set of experiments validate the superiority of IEA.

A significant future work is to consider the growth of an existing SN. There are some common features in the crowd requirements, and such features are slowly evolving, so an SN should also follow the evolution and make changes on it. Key issues include: (1) how to identify the evolution of crowd requirements and behaviors; and (2) how to map it into the structure changes of an SN.

8. Acknowledgment

Research works in this paper are supported by the National Natural Science Foundation (NSF) of China (No. 61033005, 61272187, 61472106).

9. References

Alférez, G.H., & Pelechano, V. (2011). Systematic reuse of web services through software product line engineering. *Proceedings of the 9th IEEE European Conference on Web Services*, pp. 192-199.

Alrifai, M., Risse, T., & Nejdl, W. (2012). A hybrid approach for efficient Web service composition with end-to-end QoS constraints. *ACM Transactions on the Web*, 6(2), Article No. 7.

Alrifai, M., Skoutas, D., & Risse, T. (2010). Selecting skyline services for QoS-based web service composition. *Proceedings of the 19th international conference on World Wide Web*, pp. 11-20.

Becker, J., Beverungen, D., Knackstedt, R., & Matzner, M. (2009). Configurative service engineering: a rule-based configuration approach for versatile service processes in corrective maintenance. *Proceedings of the 42nd Hawaii International Conference on System Sciences*, pp. 1-10.

Bucchiarone, A., Cappiello, C., Di Nitto, E., Kazhamiakin, R., Mazza, V., & Pistore, M. (2010). Design for adaptation of service-based applications: main issues and requirements. *Proceedings of International Conference on Service-Oriented Computing*, LNCS 6275, pp 467-476.

Business Dictionary (2014). What is mass customization? Definition and Meaning. http://www.businessdictionary.com/definition/mass-customization.html

Canfora, G., Di Penta, M., Esposito, R., & Villani, M.L. (2005). An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, pp. 1069-1075.

Cardellini, V., Casalicchio, E., Grassi, V., & Presti, F.L. (2007). Flow-based service selection for web service composition supporting multiple QoS classes. *Proceedings of the IEEE International Conference on Web Services*, pp. 743-750.

Cardoso, J., Pedrinaci, C., & De Leenheer, P. (2013). Open Semantic Service Networks: Modeling and Analysis. *Exploring Services Science*, LNBIP 143, pp. 141-154.

Chen, S., Han, Y., & Feng, Z. (2012). Social Network Analysis on the Interaction and Collaboration Behavior among Web Services. *Proceedings of AAAI Spring Symposium Series on Intelligent Web Services Meet Social Computing*, pp. 9-15.

Chen, W., Paik, I., & Huang, P.C.K. (2014). Constructing a Global Social Service Network for Better Quality of Web Service Discovery. *IEEE Transactions on Services Computing*, in press.

Danylevych, O., Karastoyanova, & D., Leymann, F. (2010). Service Networks Modelling: An SOA & BPM Standpoint. *Journal of Universal Computer Science*, 16(13), 1668-1693.

Hadaytullah, H., Koskimies, K., & Systa, T. (2009). Using Model Customization for Variability Management in Service Compositions. *Proceedings of IEEE International Conference on Web Services*, pp. 687-694.

Han, Y., Chen, S., & Feng, Z. (2014). Mining Integration Patterns of Programmable Ecosystem with Social Tags. *Journal of Grid Computing*, published online.

Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulosa, D., & Vlahavasa, I. (2013). The PORSCE II framework: Using AI planning for automated semantic web service composition. *The Knowledge Engineering Review*, 28(2), 137-156.

Hoffmann, J. (2000). A Heuristic for Domain Independent Planning and its Use in an Enforced Hill-climbing Algorithm. *Foundations of Intelligent Systems*, LNCS 1932, pp. 216-227.

Hu, B., Ma, Y., Zhang, L.J., Xing, C., Zou, J., & Xu, P. (2013). A CCRA Based Mass Customization Development for Cloud Services. *Proceedings of IEEE International Conference on Services Computing*, pp. 705-712.

Huang, K.M., Fan, Y.S., & Tan, W. (2012). An Empirical Study of Programmable Web: A Network Analysis on a Service-Mashup System. *Proceedings of IEEE 19th International Conference on Web Services*, pp. 552-559.

Hwang, J., Altmann, J., & Kim, K. (2009). The structural evolution of the Web 2.0 service network. *Online Information Review*, 33(6), 1040-1057.

Jin, H., Zou, H., Yang, F., Lin, R., & Shuai, T. (2012). A Novel Method for Optimizing Multi-User Service Selection. *Journal of Convergence Information Technology*, 7(5), 296-310.

Jung, J.J. (2011). Service chain-based business alliance formation in service-oriented architecture. *Expert Systems with Applications*, 38(3), 2206-2211.

Kannan, P.K., & Healey, J. (2011). Service customization research: A review and future directions. *The Science of Service Systems, Service Science: Research and Innovations in the Service Economy*, pp. 297-324.

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

Kuzu, M., & Cicekli, N.K. (2012). Dynamic planning approach to automated web service composition. *Applied Intelligence*, 36(1), 1-28.

Liang, Q., Wu, X., Park, E.K., Khoshgoftaar, T.M., & Chi, C.-H. (2011) Ontology-Based Business Process Customization for Composite Web Services. *IEEE Transactions on Systems, Man and Cybernetics* (*Part A*), 41(4), 717-729.

Liang, Q., Chen, S., & Feng, Z. (2013). Application of Services Relation Tracing to Automated Web Service Composition. *Applied Mathematics & Information Sciences*, 7(SI), 243-251.

Lin, S.-Y., Lin, G.-T., Chao, K.-M., & Lo, C.-C. (2012). A Cost-Effective Planning Graph Approach for Large-Scale Web Service Composition. *Mathematical Problems in Engineering*, 2012, Article ID 783476.

Liu, X., Fletcher, K.K., Kang, G., Liu, J., & Tang, M. (2011). Web Service Selection for Resolving Conflicting Service Requests. *Proceedings of the IEEE International Conference on Web Service*, pp. 387-394.

Luo, Y., Qi, Y., Hou, D., Shen, L., Chen, Y., & Zhong, X. (2011). A novel heuristic algorithm for QoS-aware end-to-end service composition. *Computer Communications*, 34(9), 1137-1144.

Maamar, Z., Hacid, H., & Huhns, M.N. (2011). Why Web Services Need Social Networks. *IEEE Internet Computing*, 15(2), 90-94.

Mohabbati, B., Asadi, M., Gašević, D., Hatala, M., & Müller, H.A. (2013). Combining service-orientation and software product line engineering: A systematic mapping study. *Information and Software Technology*, 55(11), 1845-1859.

Moon, S.K., Shu, J., Simpson, T.W., & Kumara, S. (2010). A module-based service model for mass customization: service family design. *IIE Transactions*, 43(3), 153-163.

Nguyen, T., & Colman, A. (2010). A feature-oriented approach for web service customization. *Proceedings of IEEE International Conference on Web Services*, pp. 393-400.

Nguyen, T., Colman, A., & Han, J. (2011). Modeling and Managing Variability in Process-Based Service Compositions. *Service-Oriented Computing*, LNCS 7084, pp. 404-420.

Oh, S.C., Lee, D., Kumara, S.R.T. (2006). A comparative illustration of AI planning-based web services composition. *ACM SIGecom Exchanges*, 5(5), 1-10.

Peer, J. (2005). Web service composition as AI planning-a survey. *Technical Report*, University of St. Gallen.

Rao, J., & Su, X. (2005). A survey of automated web service composition methods. Semantic Web Services and Web Process Composition. *Proceedings of the 1st International Conference on Semantic Web Services and Web Process Composition*, LNCS 3387, pp. 43-54.

Seung, K.M., Simpson, T.W., Shu, J., & Kumara, S.R.T. (2011). A Platform Identification Method for Service Family Design Using a Process Model and a Clustering Method. *Mass Customization, Springer Series in Advanced Manufacturing*, pp. 151-170.

Shim, J., Han, J., Kim, J., Lee, B., Oh, J., & Wu, C. (2011). Patterns for configuration requirements of Software-as-a-Service. *Proceedings of ACM Symposium on Applied Computing*, pp. 155-161.

Sun, C., Rossing, R., Sinnema, M., Bulanov, P., Aiello, M. (2010). Modeling and managing the variability of Web service-based systems. *Journal of Systems and Software*, 83(3), 502-516.

Tao, F., Guo, H., Zhang, L, & Cheng, Y. (2012). Modelling of combinable relationship-based composition service network and the theoretical proof of its scale-free characteristics. *Enterprise Information Systems*, 6(4), 373-404.

Wang, X., Wang, Z., & Xu, X. (2012). Effective Service Composition in Large Scale Service Market: An Empirical Evidence Enhanced Approach. *International Journal of Web Service Research*, 9(1), 74-94.

Wang, Z., Xu, F., & Xu, X. (2012). A Cost-Effective Service Composition Method for Mass Customized QoS Requirements. *Proceedings of IEEE 9th International Conference on Services Computing*, pp. 194-201.

Weidmann, M., Kötter, F., Kintz, M., Schleicher, D., & Mietzner, R. (2011). Adaptive business process modeling in the internet of services (ABIS). *Proceedings of the 6th International Conference on Internet and Web Applications and Services*, pp. 29-34.

Yu, T., Zhang, Y., & Lin, K.J. (2007). Efficient algorithms for Web services selection with end-to-end QoS constraints. *ACM Transactions on the Web*, 1(1), Article No. 6.

Zeng, L., Bentallah, A., Ngu, A.H.H., Dumas, M., Kalagnanam, J., & Chang, H. (2004). QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5), 311-327.

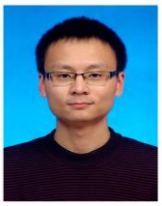
Zheng, X., & Yan, Y. (2008). An Efficient Syntactic Web Service Composition Algorithm Based on the Planning Graph Model. *Proceedings of IEEE International Conference on Web Services*, pp. 691-699.

Zou, H., Guo, J., Yang, F., & Lin, R. (2013). Selecting Web Service for Multi-user Based on Multi-QoS Prediction. *Proceedings of the IEEE International Conference on Services Computing*, pp. 551-558.

Authors

Zhongjie Wang received his PhD in 2006 from the School of Computer Science and Technology, Harbin Institute of Technology (HIT), China. He is now a professor of HIT. His main research interests include services computing, mobile and social networking services, and software architecture.

Nan Jing is now a master student in the School of Computer Science and Technology, Harbin Institute of Technology (HIT), China. Her research topic is mass customization of services.



Fei Xu got his master degree from the School of Computer Science and Technology, Harbin Institute of Technology (HIT), China in 2013. He is now a software engineer in Baidu Inc. His research interest is service networking and service composition.

Xiaofei Xu received his PhD in 1989 from the School of Computer Science and Technology, Harbin Institute of Technology. Currently, he is a professor and director of the Research Center of Intelligent Computing for Enterprises and Services (ICES). He is the assistant president of HIT. He has published four books and more than 200 papers. His current research interests include services computing, enterprise computing, data engineering and data mining.