Keyword Search for Building Service-Based Systems

Qiang He, *Member*, *IEEE*, Rui Zhou, Xuyun Zhang, Yanchun Wang, Dayong Ye, Feifei Chen, John Grundy *Member, IEEE*, Yun Yang, *Senior Member*, *IEEE*

Abstract—With the fast growth of applications of service-oriented architecture (SOA) in software engineering, there has been a rapid increase in demand for building service-based systems (SBSs) by composing existing Web services. Finding appropriate component services to compose is a key step in the SBS engineering process. Existing approaches require that system engineers have detailed knowledge of SOA techniques which is often too demanding. To address this issue, we propose KS3 (**K**eyword **S**earch for **S**ervice-based **S**ystems), a novel approach that integrates and automates the system planning, service discovery and service selection operations for building SBSs based on keyword search. KS3 assists system engineers without detailed knowledge of SOA techniques in searching for component services to build SBSs by typing a few keywords that represent the tasks of the SBSs with quality constraints and optimisation goals for system quality, e.g., reliability, throughput and cost. KS3 offers a new paradigm for SBS engineering that can significantly save the time and effort during the system engineering process. We conducted large-scale experiments using a real-world Web service dataset to demonstrate the practicality, effectiveness and efficiency of KS3.

Index Terms—Service-Based System, Keyword Search, Service Composition, Web Service, Quality of Service, Cloud Computing.

1 Introduction

Service-oriented architecture (SOA) has become a major framework for building complex distributed software systems by discovering and composing loosely coupled Web services provided by different organisations [5, 49]. Executed by a system engine, e.g., a BPEL engine [7], the component services of such a service-based system (SBS) collectively realise the functionality of the SBS which is often offered as SaaS (Software-as-a-Service) in the cloud environment. The development and popularity of e-business, ecommerce, especially the pay-as-you-go business model promoted by cloud computing have fuelled the growth of Web services [21]. The statistics published by ProgrammableWeb[1], an online Web service directory, and webservices.seekda.com, a Web service search engine, both indicate a rapid growth in the number of published Web services in the past few years. The popularity of Web services and SOA enables the engineering of various SBSs that fulfil different organisations' increasingly sophisticated business needs [9].

Fig. 1 shows the engineering process for a travel booking SBS that consists of four component services performing four tasks: *flight ticket booking*, *hotel booking*, *car rental* and *insurance quote*. As depicted, the service composition process for engineering the SBS consists of three phases. The first phase is *system planning* where a system engineer determines the tasks needed to be performed to implement the functionality of the SBS, as well as the execution order of the tasks, by employing artificial intelligence techniques [24, 35, 43]. The second phase is *service discovery* where, through service registries or service search engines, the system engineer identifies a set of candidate services for each of the tasks based on the functional and semantic information on candidate services [34, 42, 44]. The third phase is *service selection* where the system engineer selects one service from each set of candidate services to fulfil the multi-dimensional constraints for system quality, e.g., reliability, throughput, cost, etc. This is a NP-complete problem often referred to as quality-aware service selection [13, 23].

Building an SBS is very complicated and has become a major obstacle to further and broader applications of SOA. Even relatively simple tools designed by SOA vendors, e.g., Oracle BPEL Process Managerand IBM Process Designer, are already too complicated for non-experts and require substantial training [1]. Thus, there has been a rapid increase in the need for an approach that allows system engineers to find services to build SBSs without detailed knowledge of the system planning, service discovery and service selection operations [6, 37].

Recently, some Web service repositories such as Pro-

- *Qiang He is with the State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China and the School of Software Engineering and Electrical Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia. Email: qhe@swin.edu.au.*
- *Rui Zhou is with the Centre for Applied Informatics, Victoria University, Melbourne, Australia. E-mail: rui.zhou@vu.edu.au.*
- *Xuyun Zhang is with the University of Auckland, Auckland, New Zealand. E-mail: xuyun.zhang@auckland.ac.nz.*
- *Yanchun Wang, Dayong Ye, Feifei Chen and Yun Yang are with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia 3122. E-mail: {yanchunwang, dye, feifeichen, yyang}@swin.edu.au.*
- *John Grundy is with Deakin University, Geelong, Victoria, Australia 3125, in the School of Information Technology,. Email: j.grundy@deakin.edu.au.*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

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

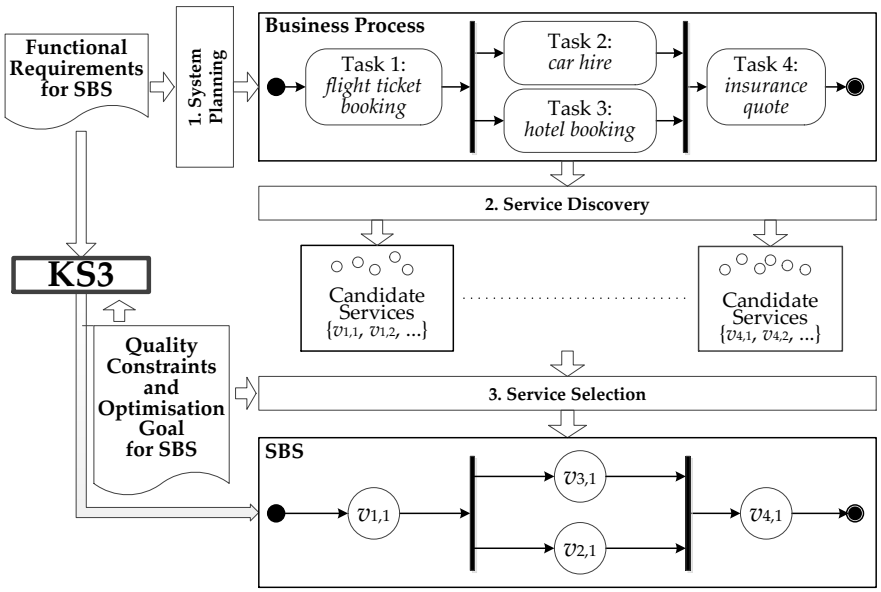

Fig.1. An example travel booking service-based system.

grammableWeb[1], Mashape[2], XMethods[3] and WebServiceList[4] emerged as simple service portals that allow users to search for Web services by keywords. Keyword search techniques have long been popularised by Web search engines like Google and Bing in locating information from Web documents. It has also been widely used to locate information from databases [10, 25, 30]. However, none of the existing keyword search techniques can be directly applied to effectively find multiple Web services for building SBSs.

In this paper, we propose KS3 (**K**eyword **S**earch for **S**ervice-based **S**ystems), a novel approach that assists system engineers in searching for Web services to build SBSs by typing only a few keywords with quality constraints and quality optimisation goals, as shown in Fig. 1. KS3 integrates and automates the system planning, service discovery and service selection operations, offering a novel paradigm for efficient SBS engineering. KS3 runs on directed data graph, where Web services are modelled as nodes connected by edges representing whether the services can be composed. Given a set of keywords that describe the tasks of an SBS, KS3 returns a subgraph of the data graph that represents the solution to the service composition for the SBS. The solution includes the services that perform the tasks of the SBS, the bridging services (if any) needed however not specified by the keywords, and the composability of those services, i.e., whether and how they can be composed.

A system engineer without detailed knowledge of SOA techniques can easily use KS3 to find the Web services needed to build the travel booking SBS depicted in Fig. 1. They only need to enter a few keywords that describe the basic tasks of the system: *flight ticket booking*, *hotel booking*, *care hire* and *insurance quote*. KS3 will take those keywords, searches its Web service library, and returns an SBS solution that specifies not only the services to use but also how they can be composed to build the SBS. Given the solution, the system engineer can deploy the SBS on a BPEL execution engine. KS3 can find an SBS solution even when the system engineer is not able to provide all the keywords for describing the SBS. For example, a system engineer enters two keywords *loan application* and *loan approval*, hoping to build a loan preapproval SBS. However, a loan preapproval process also requires a *credit check* service that succeeds the *loan application* service and precedes the *loan approval* service. The search engine can automatically identify the missing *credit check* service and provide the system engineer with a complete SBS solution. Therefore, KS3 can save system engineers a lot of time and efforts for finding the component services to build their SBSs.

The major contributions of this research are as follows:

- KS3 offers a novel paradigm for efficiently engineering SBSs by integrating and automating the system planning, service discovery and service selection operations and relieving system engineers of the detailed knowledge of corresponding SOA techniques.
- The existing network model for organising a Web service library is adapted into a data graph model, in which nodes represent Web services with keywords and quality values and directed edges represent service composability.
- Based on the data graph, constraint optimisation problem (COP) models are proposed to model and answer system engineers' queries for services for building SBSs. Three types of queries are currently supported: normal queries, constraint queries, i.e., queries with quality constraints, and optimal queries, i.e., queries with quality constraints and optimisation goals.
- Extensive experiments were conducted to evaluate the practicality, effectiveness and efficiency of KS3 using two datasets. The first dataset contains the functional information about 1496 real-world Web ser-

[1]https://www.mashape.com/
[3]http://www.xmethods.com/
[4]http://www.webservicelist.com/

vices and 2926 SBSs crawled from programmableweb.com. The second dataset is a published one which contains the functional and quality information about over 2500 real-world Web services.

The rest of this paper is organised as follows: Section 2 describes our graph model for constructing the Web service library. Section 3 formally states the research problem. Section 4 presents how KS3 models and answers keyword queries for SBS solutions based on the graph model. Section 5 evaluates the practicality, effectiveness and efficiency of KS3 with experimental results. Section 6 reviews the related work. Section 7 concludes the paper.

2 Graph Model for Web Service Library

Many approaches have been proposed in recent years for organising a Web service library using a network model. In this network model, a node represents a Web service and a directed edge between two nodes represents the service composability, i.e., whether the two corresponding Web services can be composed in the order specified by the direction of the edge. Those approaches can be grouped into two major categories: data mining based [27, 31, 38, 40] and semantics based [17, 19]. The data mining based approaches mine the service composability information, which is needed for constructing a Web services data graph, from their collaboration history. The semantics based approaches discover the service composability information by mining semantic associations and interactions between services according to well-defined ontologies. Both categories of approaches construct the Web services network offline. Once completed, the service network remains relatively stable and can be updated with minimum overheads upon certain events, e.g., new services joining or old ones leaving.

We adapt the above service network model into a data graph. KS3 is independent of the specific approach adopted for the generation of a data graph. It runs on any data graph that fulfils the simple and straightforward requirements specified by Definitions 1 and 2 below:

Definition 1. Nodes: For each Web service in the library, the data graph G has a corresponding node v. Each node in G contains one keyword k_1 that represents the function offered by the corresponding Web service. A Web service that offers multiple functions is represented by multiple nodes with the same keyword in G.

In the remainder of this paper, we will speak interchangeably of a Web service and its corresponding node in the graph, both denoted as v. Please also note that *flight ticket booking* has three terms, however is considered as one keyword, not three.

Definition 2. Edges: For each pair of composable Web services v_1 and v_2, the data graph contains an edge $e(v_1, v_2)$ between v_1 and v_2. $e(v_1, v_2)$ is directed, pointing from v_1 to v_2 if v_2 can be the succeeding node of v_1 in the composition of v_1, v_2. An edge e can be bidirectional if v_1 can also be the succeeding node of v_2 in the composition.

We use $\boldsymbol{G(V, E)}$ to denote the data graph where V is the set of nodes and E is the set of edges in G. The nodes in G are annotated with the quality values of the Web services obtained from their Service Level Agreements (SLAs), e.g., reliability and throughput, to enable quality-aware selection for service compositions - a critical and challenging problem in SBS engineering [5, 13, 23, 47, 49]. The answer to such a query is a set of Web services that collectively fulfil the functional and quality requirements for the SBS.

According to Definition 2, relevant services in the same domain are connected, either directly or indirectly, forming a connected data graph. However, a Web service library might contain Web services in different domains, e.g., *car hire* and *image processing* services, which belong to different data graphs. Thus, it is possible that a Web service library has multiple data graphs that are not connected to each other.

To answer different types of keyword queries (as will be detailed in Section 4), KS3 prebuilds and maintains an inverted index for a data graph G. For each keyword in G, the nodes covering the keyword are stored in this index. For example, if nodes v_1, v_8 and v_{35} cover keyword k_a, there is $V(k_a)=\{v_1, v_8, v_{35}\}$ representing the set of nodes in G that cover keyword k_a.

3 Problem Statement

Given a data graph G and a keyword query Q containing l ($l \geq 2$) keywords ($Q=\{k_1, \ldots, k_l\}$), the problem of answering the query over G consists of two steps: 1) to find an *answer tree*, denoted as $T(Q)$ in G, containing connected nodes that cover all the keywords in Q; 2) to induce the final answer based on the answer tree. Fig. 2 presents part of an example data graph G and Fig. 3 shows three answer trees, i.e., $T_1(Q)$, $T_2(Q)$ and $T_3(Q)$, from G for query $Q=\{$*flight ticket booking*, *insurance quote*$\}$. In $T_1(Q)=\{v_3, v_1, v_5\}$, node v_3 contains *flight ticket booking* and node v_5 contains *insurance quote*. From Fig. 2, we can see that v_3 and v_5 are not directly connected. However, they can be connected via v_1. Thus, v_1 is included in $T_1(Q)$ as a *bridging node* (*bridging service*), indicating that v_1, v_3 and v_5 can be composed together to perform *flight ticket booking* and *insurance quote*. Besides $T_1(Q)$, $T_2(Q)=\{v_3, v_8, v_7 v_5\}$ and $T_3(Q)=\{v_3, v_2, v_4, v_6, v_5\}$ also cover *flight ticket booking* and *insurance quote*.

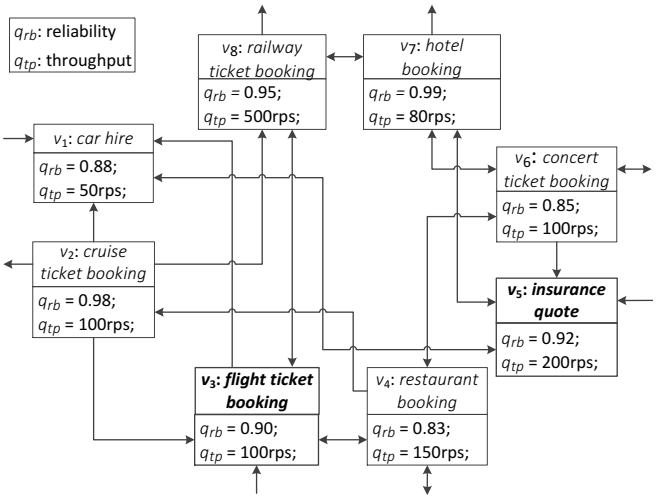

Fig. 2. Part of an example data graph

TABLE 1
QUALITY VALUES OF ANSWERS TO QUERY Q={*FLIGHT TICKET BOOKING*, *INSURANCE QUOTE*}

Answer Tree $T(Q)$	Reliability (q_{rb})	Throughput (q_{tp})
$T_1(Q)$={v_3, v_1, v_5}	0.73	50rps
$T_2(Q)$={v_3, v_8, v_7, v_5}	0.78	80rps
$T_3(Q)$={v_3, v_2, v_4, v_6, v_5}	0.57	100rps

Please note that there are other trees that cover *flight ticket booking* and *insurance quote*. However, we omit them and use only $T_1(Q)$, $T_2(Q)$ and $T_3(Q)$ as examples in the remainder of the paper. We denote $q(T(Q))$ as the quality of the SBS built based on $T(Q)$. Using the quality aggregation functions introduced in [5, 49], Table 1 presents the reliability and throughput offered by $T_1(Q)$, $T_2(Q)$ and $T_3(Q)$. Assume that the system engineer has a quality constraint for the SBS $q_{rb}(T(Q)) \geq 0.7$, i.e., at least 70% of the requests must be processed within a specified time period, $T_3(Q)$ is not a suitable solution because it does not fulfil the quality constraint. Both $T_1(Q)$ and $T_2(Q)$ fulfil this quality constraint, but $T_1(Q)$ will be selected as the answer tree because it contains the minimum number of nodes and can provide the best simplicity in the solution. Among the three answer trees, $T(Q_2)$ has the highest reliability whilst $T(Q_3)$ has the largest system throughput. Thus, $T(Q_2)$ would be the optimal solution if the system engineer's optimisation goal is maximised reliability, or $T(Q_3)$ if the system throughput needs to be maximised.

Fig. 3 demonstrates that an answer tree $T(Q)$ may contain nodes that do not cover any of the keywords in Q, and is therefore a *Steiner tree* [28], defined as follows:

DEFINITION 3. Steiner Tree. Given a graph $G=(V, N)$ and $V' \subseteq V$, T is a Steiner tree of V' in G if T is a connected subtree in G that covers all nodes in V'.

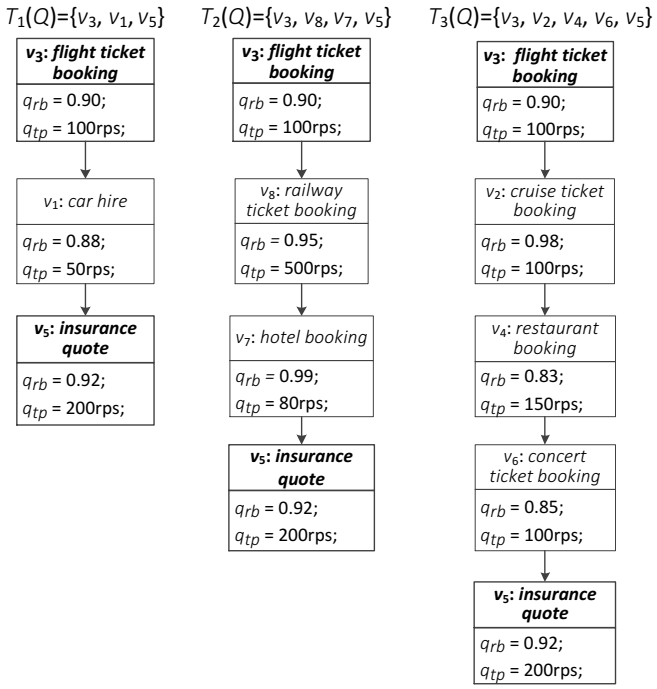

Fig. 3. Answer trees to query Q={*flight ticket booking*, *insurance quote*}

Using the inverted index introduced in Section 2, we can identify the groups of nodes in G corresponding to individual keywords in Q={k_1, …, k_l}, denoted as V_1, …, V_l where V_r ($1 \leq r \leq l$) is the set of nodes in G that cover k_r ($1 \leq r \leq l$). The problem is now to find an *exact group Steiner tree*, formally defined as follows:

DEFINITION 4. Exact Group Steiner Tree: Given a graph $G=(V, N)$ and groups V_1, …, $V_l \subseteq V$, where $V_i \cap V_j = \emptyset$, $\forall V_i, V_j$ ($0 \leq i, j \leq l$ and $i \neq j$), T is an exact group Steiner tree of V_1, …, V_l in G if T is a Steiner tree that contains exactly one node from each group V_r ($1 \leq r \leq l$).

KS3 answers three types of queries: 1) normal query; 2) constraint query; and 3) optimal query. A **normal query** aims to find an SBS solution without quality constraints. The answer tree for a normal query is an exact group Steiner tree. There are usually multiple exact group Steiner trees. KS3 aims to find the *minimum exact group Steiner tree* that answers the query with the minimum number of nodes, including *keyword nodes*, i.e., nodes that contain the keywords in the query, and *bridging nodes*, i.e., nodes that do not contain the keywords but are necessary to connect the keyword nodes. A minimum exact group Steiner tree is defined as follows:

DEFINITION 5. Minimum Exact Group Steiner Tree. Given a set of exact group Steiner trees in G, T_1, …, T_n, T_i ($0 \leq i \leq n$) is the minimum exact group Steiner tree if $|T_i| = \min(|T_1|, \ldots, |T_n|)$ where $|T_i|$ ($1 \leq i \leq n$) represents the cardinality of T_i, i.e., the number of nodes in T_i.

Take Fig. 3 for example, for query Q={*flight ticket booking*, *insurance quote*}, there are three exact group Steiner trees. $T_1(Q)$, the one with the minimum number of nodes, including two keyword nodes (v_3 and v_5) and one bridging node (v_1), is the minimum exact group Steiner tree and the answer to the query.

A **constraint query** is similar to a normal query, but with constraints for system quality, e.g., $q_{rb}(T(Q))>0.70$ and $q_{tp}(T(Q))>70$rps, i.e. the system must be able to process at least 70 requests per second. In KS3, each node v in G is annotated with the quality values of the corresponding Web service. The quality of an SBS based on an answer tree $T(Q)$ can be calculated by aggregating the quality of its component services [5, 49]. Take $T_1(Q)$ in Fig. 3 for example, its reliability can be calculated: $q_{rb}(T_1(Q)) = q_{rb}(v_5) \times q_{rb}(v_1) \times q_{rb}(v_5) = 0.90 \times 0.88 \times 0.92 = 0.73$ and its throughput: $q_{tp}(T1(Q)) = \min(q_{tp}(v_3), q_{tp}(v_1), q_{tp}(v_5)) = \min(100, 50, 200) = 50$rps. The answer tree for a constraint query is the exact group Steiner tree that fulfils all quality constraints with minimum cardinality. Assume the query Q={*flight ticket booking*, *insurance quote*} with two quality constraints: c_{rb}: $q_{rb}(T(Q))>0.70$ and c_{tp}: $q_{tp}(T(Q))>70$rps,. The only satisfactory answer trees among the three shown in Fig. 3 is $T_2(Q)$ because $T_1(Q)$ does not fulfil the constraint for system throughput, i.e., c_{tp} and $T_3(Q)$ does not fulfil the constraint for system reliability, i.e., c_{rb}.

An **optimal query** is also a constraint query, but with the objective to optimise a system quality, e.g., to maximised system reliability or throughput, instead of minimum cardinality. With each group Steiner tree representing a potential answer tree that fulfils all quality constraints, the answer to an optimal query is the *optimal exact group*

Steiner tree, formally defined as follows:

DEFINITION 6. Optimal Exact Group Steiner Tree. Given a set of exact group Steiner trees in G, T_1, …, T_n, T_j ($1 \le j \le n$) is the optimal exact group Steiner tree if $q(T_j)=\max(q(T_1), \ldots, q(T_n))$ (or $q(T_j)=\min(q(T_1), \ldots, q(T_n))$ for negative quality properties like cost) where $q(T_i)$ represents the quality offered by $T_i(Q)$.

Take Fig. 3 for example, for query Q={*flight ticket booking*, *insurance quote*} with quality constraints: $q_{rt}(T(Q))>0.70$ and $q_{tp}(T(Q))>50$rps, and an optimisation goal on system throughput, $T_2(Q)$ is the answer tree as it fulfils both quality constraints and offers the maximum system throughput.

The computation of a minimum group Steiner tree is already NP-complete [11], and is made even more complicated by the multi-dimensional constraints and the optimisation goal for system quality. KS3 models keyword queries as COPs that can be solved by applying Integer Programming (IP) techniques. Next, we discuss how different queries are modeled and answered, followed by a discussion of the answer induction.

4 Answering Keyword Queries

In this section, we first discuss how different keyword queries for SBSs are modelled and answered based on the graph model presented in Section 2. Then we describe how the final answers to keyword queries are induced. As discussed in Section 2, a Web service library may have multiple data graphs for different domains. In this research, we assume that system engineers would not search for services across different data graphs because the tasks of an SBS are usually in the same domain.

4.1 Answering Normal Queries

A normal keyword query contains a set of keywords, $Q=\{k_1, \ldots, k_l\}$. To answer a normal query Q over a data graph G, KS3 finds a minimum exact group Steiner tree $T(Q)$ that contains all the keywords in Q.

The first step of the answering process is to locate nodes that contain individual keywords in Q. For each keyword k_r in Q ($1 \le k \le l$), KS3 finds the set of nodes $V(k_r)$ that contain k_r using the inverted index discussed in Section 2. Next, KS3 models the problem of answering a normal query as a constraint satisfaction problem (CSP), which consists of a finite set of variables $X=\{x_1, \ldots, x_n\}$, with domain $D=\{0, 1\}$ listing the possible values for each variable in X, and a set of constraints $C=\{c_1, c_2, \ldots, c_t\}$ over X. A solution to a CSP is an assignment of a value to each variable in X from its domain such that all constraints in C are satisfied. The CSP model of answering a normal query is formally expressed as follows.

For a $G=(V, E)$, where $V=\{v_1, \ldots, v_m\}$ and $E=\{e_1, \ldots, e_n\}$, there are two sets of 0-1 variables $X=\{x_1, \ldots, x_m\}$ and $Y=\{y_1, \ldots, y_n\}$, where $D(x_i)=\{0, 1\}$ ($i=1, \ldots, m$) and $D(y_j)=\{0, 1\}$ ($j=1, \ldots, n$), x_i and y_j being 1 if the i^{th} node and the j^{th} edge in G are selected as part of the answer tree for the query, 0 otherwise. The constraints for the CSP model are:

Keyword Constraints:

$$\sum_{v_i \in V_r} x_i = 1 \quad \forall r \in [1, \ldots, l] \tag{1}$$

where V_r is the set of nodes in G that contain keyword $k_r \in Q$ ($1 \le r \le l$). The keyword constraints ensure that exactly one node is selected from each V_r ($1 \le r \le l$) to cover each keyword in Q.

Node Constraints:

$$\sum_{e_j \in E(v_i)} y_j = 0 \quad \text{IF } x_i = 0 \quad \forall i \in [1, \ldots, m] \tag{2}$$

$$\sum_{e_j \in E(v_i)} y_j \ge 1 \quad \text{IF } x_i = 1 \quad \forall i \in [1, \ldots, m] \tag{3}$$

where $E(v_i)$ is the set of edges connected to v_i ($1 \le i \le m$). Constraint family (2) ensures that if a node is not selected, none of the edge(s) connected to it are selected. Constraint family (3) ensures that, if a node is selected, at least one edge connected to it is selected.

Edge Constraints:

$$\forall j \in [1, \ldots, n] \text{ and } v_i \in V(e_j) \text{ IF } y_j == 1 \text{ THEN } x_i = 1 \tag{4}$$

where $V(e_j)$ is the set of nodes connected to e_j. The edge constraints ensure that if an edge is selected, both nodes connected to it must be selected. When integrated into the CSP model, constraints family (4) can be transformed into:

$$x_i \ge y_j \ \forall e_j \in E \ (1 \le j \le n) \text{ and } v_i \in V(e_j) \tag{5}$$

Connectedness Constraint:

$$\sum_{j=1}^{n} y_j \ge \sum_{i=1}^{m} x_i - 1 \tag{6}$$

$$x_s - x_d + 1 \le m(1 - y_j) \quad \forall j \in [1, \ldots, n] \text{ and } x_s, x_d \in V(e_j) \tag{7}$$

The connectedness constraint guarantees that all the selected nodes and edges constitute a connected tree based on the well-known Miller–Tucker–Zemlin constraint [39].

Solving the above CSP can find the exact group Steiner tree(s) in G, each covering all the keywords in the query Q. Very often, there are many such exact group Steiner trees. Take Fig. 3 for example, there are three exact group Steiner trees for query Q={*flight ticket booking*, *insurance quote*} from G shown in Fig. 2. They all cover keywords *flight ticket booking* and *insurance quote* in the query. In fact, any spanning tree of G that contains one node from each $V(k_r)$ ($1 \le r \le l$) is an exact group Steiner tree, e.g., $\{v_1, v_2, v_3, v_5\}$, $\{v_1, v_3, v_5, v_7\}$, etc. As discussed in Section 3, KS3 identifies the minimum exact group Steiner tree, i.e., the one with the minimum number of nodes, as the answer tree for the normal query. The objective function that captures this optimisation goal is as follows:

Objective Function:

$$\text{minimise}\left(\sum_{i=1}^{m} x_i\right) \tag{8}$$

Given this objective function, the CSP turns into a COP. In a COP, each solution generated by solving the CSP is associated with a ranking value for the objective function. The solution with the optimal ranking value is the solution to the COP.

4.2 Answering Constraint Queries

A constraint query is a normal query with constraints for

system quality, e.g., $q_{rb}(T(Q))<0.7$, $q_{tp}(T(Q))>70$rps. The answer tree for a constraint query is an exact group Steiner tree that: 1) covers all keywords in Q; and 2) fulfils all quality constraints.

The quality offered by an answer tree $T(Q)$ can be calculated by using corresponding aggregation functions [5, 49]. Some quality properties of an SBS are calculated independently of its structure and dynamics, based only on the quality of its component services, e.g., reliability, throughput, cost, etc. Those quality properties are named *structure-independent system quality*. Let $T(Q)=\{v_1, \ldots, v_m\}$, formulas (9)-(11) present the functions for the calculation of system reliability, throughput and cost as examples:

$$q_{rb}(T(Q))=\prod_{i=1}^{m} q_{rb}(v_i) \quad (9)$$

$$q_{tp}(T(Q))=\min(q_{tp}(v_1), \ldots, q_{tp}(v_m)) \quad (10)$$

$$q_{cost}(T(Q))=\sum_{i=1}^{m} q_{rb}(v_i) \quad (11)$$

The calculation of some quality properties must take into account the system structure, e.g., response time. If there are multiple execution paths from the entry service to the exit service of the system, the one with maximum execution time determines the response time of the system. All existing approaches for quality-aware service composition require and presume a pre-specified and fixed structure for the target SBS [5, 13, 23, 47, 49]. The system engineers using such approaches are required to have the knowledge about system structure and business process specification. Aiming at relieving system engineers of such expert knowledge, KS3 does not require a system structure as input. Instead, KS3 helps system engineers identify a proper system structure. The system structure remains unknown to them until the solution is found. As a result, such quality properties, named *structure-dependent system quality*, cannot be integrated into the CSP model for answering a constraint query.

Besides constraints families (1), (2), (3), (5) and (6), the following constraints can be included in the CSP model for answering a constraint query to take the quality constraints into consideration:

Quality Constraints:

$$q_p(T(Q))<c_p \quad \forall p \in [1, t] \quad (12)$$

where c_p is the constraint for the p^{th} quality property of the SBS.

Similar to the COP model for answering normal queries, objective function (8) is included in the COP model for answering a constraint query to minimise the number of nodes in the answer tree. If there are multiple exact group Steiner trees with the same minimum cardinality, one of them is randomly selected as the answer tree. Here the answer tree is not necessarily a minimum exact group Steiner tree, because it does not always have the minimum cardinality among all. For example, assume a query $Q=\{$*flight ticket booking*, *insurance quotee*$\}$ with two quality constraints: $q_{rb}(T(Q))>0.50$ and $q_{tp}(T(Q))>70$rps,. Among the three exact group Steiner trees shown in Fig. 3, $T_1(Q)$ is the minimum exact group Steiner tree because it has the minimum cardinality. However, $T_1(Q)$ does not fulfil the throughput constraint. Both $T_2(Q)$ and $T_3(Q)$ fulfil all the quality constraints, but $T_2(Q)$ is the answer tree for Q because it has a lower cardinality than $T_3(Q)$.

4.3 Answering Optimal Queries

An optimal query is a constraint query with an optimisation goal for a system quality, e.g., reliability, throughput or cost. The answer tree for an optimal query Q is an optimal exact group Steiner tree that: 1) covers all the keywords in Q; 2) fulfils all quality constraints; and 3) achieves the quality optimisation goal. Similar to quality constraints, the optimisation goal can only be specified for a structure-independent system quality.

The COP model for answering an optimal query has constraint families (1), (2), (3), (5), (6), (11) and replaces objective function (8) with an objective function to optimise a system quality. Reliability, throughput, cost and system optimality are used below as examples:

Optimal System Reliability:

$$\text{maximise}(\prod_{i=1}^{n} x_i \times q_{rb}(v_i)) \quad (13)$$

Optimal System Throughput:

$$\text{maximise}(\min(x_1 \times q_{tp}(v_1), \ldots, x_m \times q_{tp}(v_m))) \quad (14)$$

Optimal System Cost:

$$\text{minimise}(\sum_{i=1}^{n} x_i \times q_{cost}(v_i)) \quad (15)$$

Optimal System Utility:

$$\text{maximise}(\sum_{i=1}^{n} x_i \times u(v_i)) \quad (16)$$

where $u(v_i)$ is the multi-objective utility of v_i calculated based on multiple dimensions of v_i's quality [13, 14].

4.4 Inducing Answers

Due to the possible parallel, selective and loop structures [22], the service composition for an SBS can be cyclic. The answer tree obtained by solving one of the COP models discussed before is acyclic and thus may miss some edges that represent the composability of the services in the tree. Such edges must be identified and included into the answer tree to induce an *answer graph* for the keyword query. Assume a normal query $Q=\{$*flight ticket booking*, *restaurant booking*, *cruise ticket booking*$\}$ over the data graph shown in Fig. 2. Fig. 4 presents the three minimum exact group Steiner trees for Q: $T_1(Q)$, $T_2(Q)$ and $T_3(Q)$. According to Fig. 2, in terms of missing edges, we can identify $e(v_4, v_2)$ from $T_1(Q)$, $e(v_2, v_3)$ from $T_2(Q)$, and $e(v_3, v_4)$ and $e(v_4, v_3)$ from $T_3(Q)$. Those missing edges must be included in the answer trees to induce the answer graph for Q. In fact, it can be observed in Fig. 4 that the answer graphs induced from $T_1(Q)$, $T_2(Q)$ and $T_3(Q)$ are the same one. Given an answer tree $T(Q)$, a naïve method for inducing the answer graph is to inspect each pair of nodes in $T(Q)$ for missing edges. The time complexity of the method is $O(n^2)$ where n is the number of nodes in $T(Q)$. To induce the answer graph more efficiently, KS3 maintains an adjacent index that records the adjacent nodes of each node in data

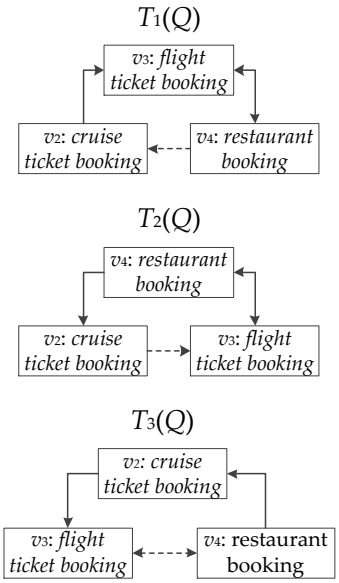

Fig. 4. Answer trees for search Q={*flight ticket booking*, *restaurant booking*, *cruise ticket booking*}

graph G. Using the adjacent index, KS3 inspects only the nodes adjacent to each node in $T(Q)$ for missing edges. The time complexity of this method is $O(kn)$ where k is the maximum number of adjacent nodes of any nodes in $T(Q)$.

KS3 relieves system engineers of having to use pre-specified system structures. The answer graph for a query indicates the services needed to perform all the tasks represented by the keywords in the query, as well as the composability of the services. However, an SBS always needs an entry service (e.g., $v_{1,1}$ in Fig. 1) and an exit service (e.g., $v_{4,1}$ in Fig. 1) [5, 13, 49, 50]. Thus, KS3 will identify an entry node and an exit node in an answer graph $G(Q)$ based on graph theory. KS3 first calculates: 1) the number of nodes in $G(Q)$without incoming edges; and 2) the number of nodes in $G(Q)$ without outgoing edges. Next, we discuss how KS3 specifies the entry node and exit node using Fig. 5 as examples:

1. If there is one node without incoming edges, it is identified as the entry node, e.g., v_1 in $G_1(Q)$. If there are multiple such nodes, a dummy node is added to $G(Q)$ as the entry node that precedes all the nodes without incoming edges, e.g., v_{et} in $G_2(Q)$.
2. If there is only one node without outgoing edges, it is specified as the exit node, e.g., v_3 in $G_1(Q)$. If there are multiple such nodes, KS3 will converge them into an added dummy exit node, e.g., v_{ex} in $G_3(Q)$.
3. If there is an exit node but no entry node, KS3 employs a unidirectional backward breadth-first algorithm to traverse $G(Q)$, starting from the exit node. It traverses $G(Q)$ backwards through only incoming edges without reversing. This algorithm still marks an edge if the target node of an edge has already been visited in order to preserve that edge which represents the composability between its source node and target node. At the end, if there is only one node whose incoming nodes are never visited, it will be specified as the entry node, e.g., v_1 in $G_4(Q)$. Otherwise, the algorithm adds a dummy node as the entry node that precedes all the nodes whose incoming nodes are never visited, e.g., v_{et} in $G_5(Q)$.
4. If there is an entry node but no exit node, KS3 employs a unidirectional forward breadth-first algorithm to traverse $G(Q)$ through outgoing edges, starting from the entry node. At the end, if there is only one node whose outgoing edges are never visited, it is specified as the exit node, e.g., v_2 in $G_6(Q)$. Otherwise, the algorithm adds a dummy exit node that succeeds all the nodes whose outgoing edges are never visited, e.g., v_{ex} in $G_7(Q)$.
5. If there is no entry or exit node, the node with the minimum number of outgoing edges is selected as the entry node, e.g., v_1 in $G_8(Q)$, and employs the unidirectional forward breadth-first algorithm to traverse $G(Q)$, starting from the entry node, to specify the exit node, e.g., v_5.

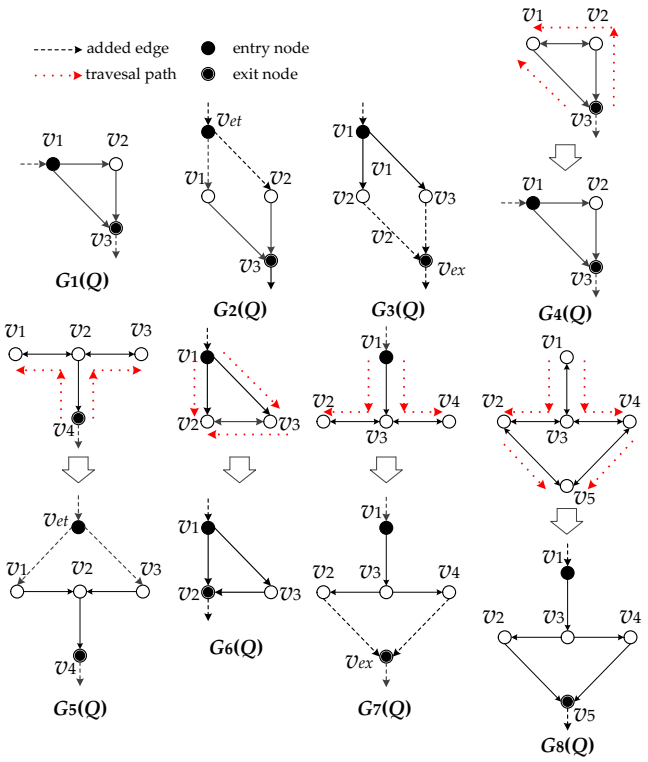

Fig. 5. Inducing answers from answer trees

The answer to a query indicates a potential system structure, which is not necessarily final. The system engineer can further adjust the system structure to fulfil their needs. For example, they can run some of the services in parallel, create selective branches, iterate certain services and even specify a different entry service or exit service. The answer graph obtained from inducing the answer tree is also returned as part of the answer for the system engineer's reference because it contains all the selectable edges that represent service composability.

5 Experimental Evaluation

This section evaluates the practicality (measured by success rate), effectiveness (measured also by success rate)

TABLE 2
THE PW DATASET

Number of Services Used by SBS	2	3	4	5	6	7	8	9	10	11-15	16-20	20+
Number of SBSs	1490	623	313	195	85	57	34	40	21	49	12	7

and efficiency (measured by computational overhead) of KS3 in answering keyword queries for building SBSs.

5.1 Prototype Implementation

We have implemented a prototype of KS3 in Java using JDK 1.6.0 and Eclipse Java IDE. It implements the mechanisms introduced in Section 4. In response to a query, it searches a given data graph and returns an answer graph that fulfills all the constraints specified in the query. Based on the answer graph, an SBS solution can be built. For solving the COPs introduced in Section 4, the prototype uses IBM CPLEX v12.2, a linear programming solver.

5.2 Experiment Setup

We have conducted three series of experiments, namely series A, B and C. Series A was conducted on the PW dataset, which contains the functional information about real-world Web services and SBSs crawled from programmableweb.com, a service portal that has been accumulating a variety of Web services and SBSs since 2005 [2, 8]. Series A demonstrates the practicality and efficiency of KS3 on a real-world Web service library and real system engineers' potential queries.

To evaluate KS3 more comprehensively, we have also conducted experiment series B and C on a publicly available and widely used dataset named QWS, which contains not only the functional but also the quality information about over 2500 real-world Web services [2]. Series B and C evaluate the effectiveness and efficiency of KS3 in comprehensive scenarios in addition to Series A. The service quality information in QWS used in series B and C enabled us to evaluate the ability of KS3 to handle constraint and optimal queries.

All experiments are conducted on a machine with Intel i5-4570 CPU 3.20GHz and 8 GB RAM, running Windows 7 x64 Enterprise.

5.2.1 Experiment Series A

The PW dataset contains the information about which of the 1496 Web services are used by each of the 2926 SBSs. Table 2 presents the relevant statistics of the PW dataset.

In this experiment series, the data graph is generated based on the information retrieved from the PW dataset. There are a total number of 1496 nodes in the data graph, each corresponding to one of the 1496 Web services in the PW dataset. Two nodes are linked in the data graph if they are both used by the same SBS with one of them succeeding or preceding the other directly. For example, suppose an SBS in the PW dataset uses three consecutive services v_1, v_2 and v_3. Two directional edges will be included in the data graph, one pointing from v_1 to v_2 and the other from v_2 to v_3, but none from v_1 to v_3. In total, there are 6899 edges in the data graph. This data graph accurately describes the composability of the 1496 real-world Web services used by the 2926 real-world SBSs that were built using those Web services.

A total of 2926 normal queries are generated, each corresponding to one of the 2926 SBSs in the PW dataset. The keywords contained in each query are obtained from the first and the last services used by the corresponding SBS. Take an SBS in the PW dataset that uses four consecutive services, BitStamp HTTP, BTC-e, CoinDesk and Mt Gox as an example, the query generated based on this SBS will contain two keywords: BitStamp HTTP and Mt Gox. In this ways, we can evaluate the ability of KS3 to identify all necessary services needed for building an SBS, given only two keywords that represent the first and last tasks of that SBS. In this experiment series, the queries do not have quality constraints or optimisation objectives, and thus are all normal queries. The ability of KS3 to handle constraint and optimal queries is evaluated in experiment series B and C. As discussed in Section 3, KS3 finds the minimum exact group Steiner tree to answer a normal query. Thus, it is possible that KS3 finds new solutions with less services than the corresponding SBSs specified in the PW dataset. Given a query that contains two keywords, a set of services identified by KS3 that is different from the ones used by the corresponding SBS as specified in the PW dataset is considered a new solution. Again, take the SBS that uses four services, BitStamp HTTP, BTC-e, CoinDesk and Mt Gox as an example, a new solution must have BitStamp HTTP and Mt Gox as the starting and ending services, with only one service in between other than BTC-e and CoinDesk. In order to find out whether KS3 can identify the solutions as specified in the PW dataset, we have changed the way KS3 answers normal queries in this experiment series. It does not stop when a new solution is found. Instead, it will continue to search for an exact Steiner tree that matches the corresponding SBS specified in the PW dataset. The success rate of answering all 2926 queries will demonstrate the practicality of KS3 as shown in Section 5.3.

In this series of experiments, we also evaluate the ability of KS3 to identify new solutions. It has a capability of offering alternative solutions to users and the success rate of finding a new solution across 2926 queries will demonstrate the practicality of KS3 from this perspective.

In addition to success rate, we measure the computational overhead, i.e., the computation time taken by KS3 to answer those queries, to evaluate the efficiency of KS3 as shown in Section 5.5. KS3 integrates and automates the system planning, service discovery and service selection operations regardless of system engineers' knowledge and experiences, achieving a similar goal as the keyword search techniques in the database community [11, 20, 29]. Inspired by [20, 29], we measure the computational overhead of KS3 needed to identify an SBS solution to evaluate the efficiency of KS3 from the software engineering perspective.

TABLE 3
EXPERIMENT CONFIGURATION (SERIES B)

Factor	Experiment Set					
	B#1	B#2	B#3	B#4	B#5	B#6
Keyword Distance	1 to 10	2	2	2	2	2
Number of Keywords in Query	2	2 to 10	2	2	2	2
Number of Quality Constraints	2	2	1 to 10	2	2	2
Graph Size (Number of Nodes)	2000	2000	2000	2000 to 20000	2000	2000
Graph Density (Number of Edges)	2000	2000	2000	4000 to 8000	4000 to 8000	2000
Stringency of Quality Constraints (Higher means Harder)	20	20	20	20	20	10 - 100

5.2.2 Experiment Series B and C

As discussed in Section 2, there are many approaches for the generation of data graphs for a Web service library. Given a set of Web services, the adoption of different approaches might result in different data graphs. In addition, different data graphs in different domains can be significantly different from many perspectives, e.g., the number of nodes and the number of edges. Thus, the observation and conclusion from experiment series A on one particular data graph (i.e., the one generated based on the PW dataset) might not be generally representative. To ensure the generality of the evaluation, in experiment set of series B and C, a random data graph is generated based on the well-known Erdős–Rényi model [18], which ensures that the nodes are randomly connected to each other in the data graph.

The relevance between the keywords in a query determines whether bridging nodes are needed to identify an SBS solution. In the data graph, directly relevant keywords are composable and hence belong to adjacent nodes. Take Fig. 1 for example, the node containing keyword *flight ticket booking* is adjacent to the one containing *car hire* and the one containing *hotel booking* in the data graph. Bridging services are needed when two keywords are not directly relevant. As discussed in Section 1, keywords *loan application* and *loan approval* are not directly relevant because they can only be connected through nodes containing keyword *credit check* in the data graph. In the experiments, we use a measurement named *keyword distance* to represent the relevance between two keywords, reflected by the number of hops they are away from each other in the data graph. Take Fig. 1 for example, the keyword distance between *flight ticket booking* and *hotel booking* is 1 and the keyword distance between *flight ticket booking* and *insurance quote* is 2. To evaluate the effectiveness and efficiency of KS3 in response to queries with both irrelevant and relevant keywords, we have conducted two series of experiments, i.e., series B (except B#2, i.e., set #2 of series B) and C, where the keyword distances are fixed at 2 and 1 respectively, *In simple words, in experiment series B, the keywords in a query never belong to adjacent nodes in the data graph; bridging nodes are always needed for an SBS solution, which however is not necessarily true in experiment series C*.

In experiment series B and C, queries are randomly generated by selecting keywords according to the prespecified keyword distance. A number of quality constraints are randomly generated for constraint and optimal queries based on the pre-specified constraint stringency. Different quality properties can be used with the corresponding quality aggregation functions to specify quality optimisation goals as discussed in Section 4.3. For the purpose of simplicity and consistency in the evaluation, we use *cost* (see formula (11) for its aggregation function). According to the quality aggregation functions, an SBS solution that contains a large number of services usually has low reliability, high cost and low system throughput. Thus, to avoid excessively large SBS solutions, we limit the maximum number of nodes to be included in an SBS solution to twice the number of keywords in the query.

To comprehensively study the impact of different factors on the effectiveness and efficiency of KS3, we vary six factors in experiment series B, as presented in Table 3, and five in series C (same as series B except with keyword distance fixed at 1). For each set of experiments, we average the results obtained from 100 runs.

For effectiveness evaluation as detailed in Section 5.4, we compare three KS3 methods, namely *KS3 normal*, *KS3 constraint*, and *KS3 optimal*, that answer normal queries, constraint queries and optimal queries respectively, with their counterpart IP-based individual search methods that are assumed or adopted in most existing research on service selection for engineering SBSs [3-5, 23, 32, 47-49]. The individual search methods look up multiple Web services individually to cover the keywords in a query. When bridging services are needed, the individual search methods cannot find any SBS solutions because they are not capable of identifying bridging services. As a result, the individual search methods can find an SBS solution only in experiment set B#1 when the keyword distance is 1. *Thus, in Fig. 8 and Fig. 11, we omit the results of the individual search methods in experiment series B.* The success rate, i.e., the percentage of cases where an answer to the keyword query for an SBS solution can be found, will demonstrate the effectiveness of KS3.

For efficiency evaluation as detailed in Section 5.5, we also measure the computational overhead of KS3 in experiment series B and C, in addition to experiment series A. The computational overheads of the individual search methods are omitted in both experiment series B and C for the following reasons. The individual search methods are used only in the service selection phase. A full comparison in efficiency between the KS3 methods and the individual search methods requires the measurement of

the time and efforts saved by KS3 during all three phases during the SBS engineering process, i.e., system planning, service discovery and service selection. However, that is impossible because it is largely dependent on the system engineer's knowledge and experience.

5.3 Practicality Evaluation

Fig. 6 demonstrates KS3's success rate of answering 2926 queries corresponding to the 2926 SBSs in the PW dataset in experiment series A. It shows that KS3 can answer all the queries regardless of the number of keywords in the queries. This demonstrates that a system engineer can indeed use KS3 to identify the services needed for building any of the SBSs in the PW dataset by entering the keywords that represent the tasks of the SBS – in the experimental cases: the first and the last tasks.

Fig. 7 presents KS3's success rate of finding new solutions when answering the 2926 queries in experiment series A. In the cases of two tasks in the target SBS, KS3 cannot find any new solutions. In those cases, only two services need to be identified to answer each query. The nodes corresponding to the two services are directly linked in the data graph. No bridging nodes are needed to identify an exact group Steiner tree that covers all the keywords in the query. Thus, there is no way a new solution with less than two services can be found to answer a query in those cases. As the number of tasks in the target SBS increases from 3 to 20+, KS3's success rate increases from 0.50 to 1.0. As the number of tasks in the target SBS increases, the distance between the node containing the first keyword and the one containing the second keyword in the data graph increases as well. This increases the possibility of finding an exact group Steiner tree that has less nodes than the number of tasks in the target SBS as specified in the PW dataset.

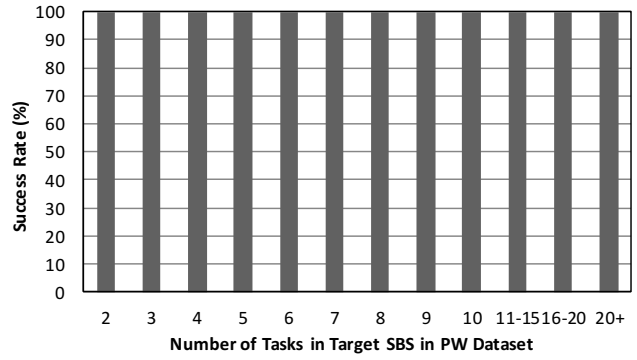

Fig. 6. Success rate of answering queries (experiment series A)

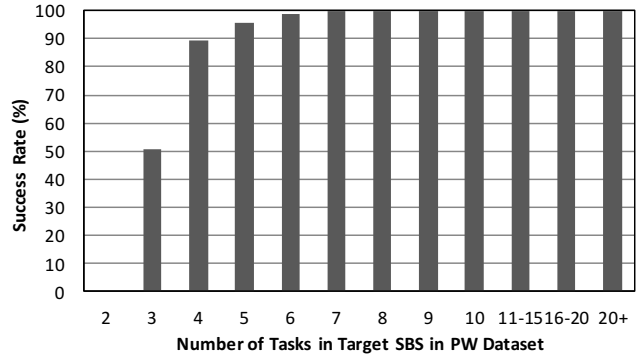

Fig. 7. Success rate of finding new solutions (experiment series A)

The results obtained from this series of experiments demonstrate that, given only a few keywords (2 in the experimental cases), KS3 can find SBS solutions as specified in the PW dataset, as well as new SBS solutions.

5.4 Effectiveness Evaluation

Fig. 8 shows the impact of different factors on the success rate of the KS3 methods in experiment series B. The KS3 normal method obtains a consistent success rate of 1.0 in all experiments because it does not consider quality constraints and quality optimisation goals. This indicates that the KS3 normal method can always find an SBS solution. *Thus, the following discussion mainly focuses on the KS3 constraint and optimal methods*. Unlike the KS3 normal method, the KS3 constraint method and optimal method are not always able to find a solution due to the quality constraints. The two methods always have the same success rate in the same set of experiments. The reason is that their success rates are determined by whether or not they can find an SBS solution that fulfils all quality constraints despite of their different optimisation goals.

Fig. 8(a) presents the results of experiment set B#1. It shows that the increase in keyword distance leads to a decrease in the success rates of the KS3 constraint and optimal methods. Even when the keywords in a query are only remotely relevant (reflected by a large keyword distance), the KS3 normal method can still identify all the bridging nodes needed for an exact group Steiner tree that covers all the keywords in the query. This observation indicates that even a system engineer only knows a few of the tasks required for the target SBS, e.g., the entry task and the exit task, KS3 can still find an SBS solution. **Fig. 8(b)** shows the results of experiment set B#2. A query with many keywords indicates that a potentially complicated SBS is in need. To build such an SBS, the probability that bridging services are needed is high. KS3 can handle such queries. However, it takes time, as shown and discussed later in Section 5.5 based on Fig. 11(b). **Fig. 8(c)** shows the success rates obtained in experiment set B#3. As the number of quality constraints increases, it becomes harder for KS3 to find an SBS solution, as indicated by the decrease in the success rates of all methods except the KS3 normal method. **Fig. 8(d)** shows the results of experiment set B#4, where the success rates decrease as the graph size increases. A bigger data graph with more nodes further distributes the keyword nodes in the data graph, directly making it harder to generate exact group Steiner trees using those keyword nodes. **Fig. 8(e)** presents the results of experiment set B#5, where the success rates increase slightly with the increase in graph density. A higher graph density means more neighbours for each node. As a result, for keyword nodes that are not directly connected, more exact group Steiner trees can be found, which increases the probability of finding one that fulfils all the quality constraints. **Fig. 8(f)** shows that in experiment set B#6, the increase in the stringency of the quality constraints largely decreases the success rates. When the stringency reaches 90 and 100, no solution can be found. This indicates that it is hard to guarantee a satisfactory

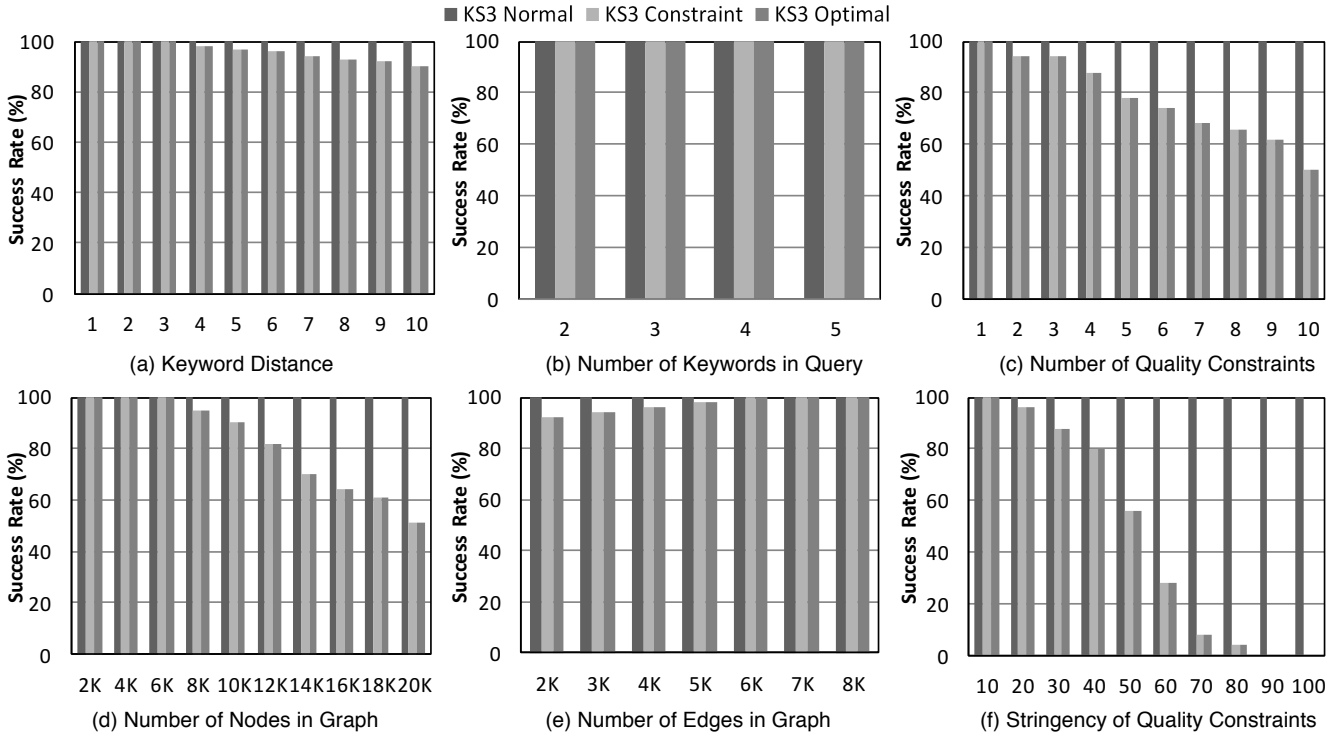

Fig. 8. Impact of Factors on success rate (experiment series B, keyword distance =2)

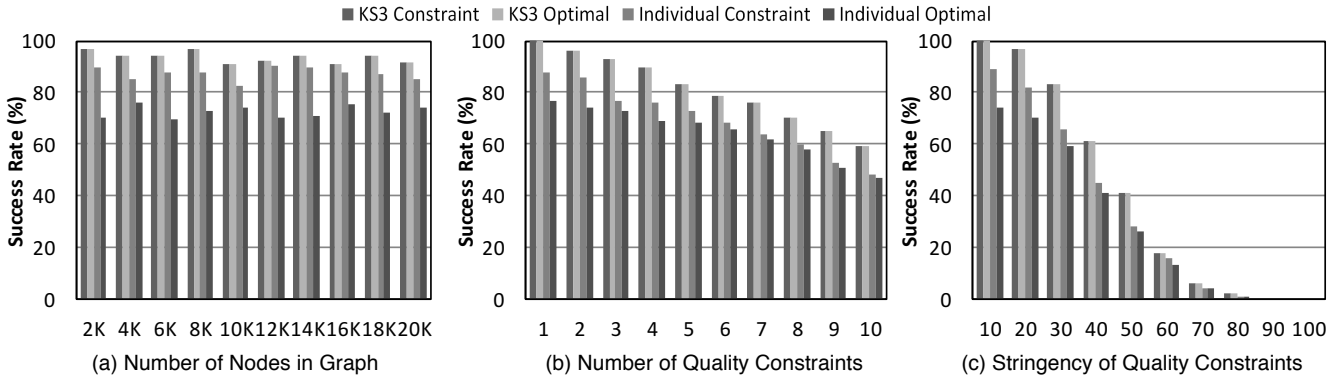

Fig. 9. Impact of factors on success rate (experiment series C, keyword distance =1)

solution when the quality constraints are stringent and bridging services are needed.

Fig. 9 compares the success rates obtained by KS3 and the individual search methods from experiment series C where the keyword distance is fixed at 1. According to Fig. 8, the three major factors that impact the success rate are the graph size, the number and stringency of the quality constraints. Thus, we only present and discuss the results of experiment sets C#3, C#4 and C#6 where the above three factors vary. In this series of experiments, the keywords in a query are directly relevant. Thus, both the KS3 normal method and the individual normal method obtain a consistent success rate of 1.0 because they do not consider quality constraints. Hence, we only present and discuss the constraint and optimal methods. According to Fig. 9(a), the graph size does not impact the success rate significantly. It is because when the keyword nodes are directly connected, the success rate depends on only the exact group Steiner trees formed by those connected keyword nodes (and possibly some of their neighbours as bridging nodes) regardless of the graph size. Fig. 9(b) and Fig. 9(c) show that the quality constraints impact the success rate in similar ways as in experiment series B. In addition, the KS3 methods always obtain higher success rates than the individual search methods because of the ability of KS3 to identify bridging nodes which allows the KS3 methods to explore more exact group Steiner trees for a satisfactory SBS solution.

5.5 Efficiency Evaluation

Fig. 10 shows the average computation time taken by KS3 to answer different queries in experiment series A. The computation time increases from 64 milliseconds to 569 milliseconds as the number of keywords in query increases from 2 to 20+. Generally, the increase in computation time is stable. Please note that the rapid increase when the number of tasks reaches and exceeds 15 is because the results are the averages of all the cases of 16-20 tasks and

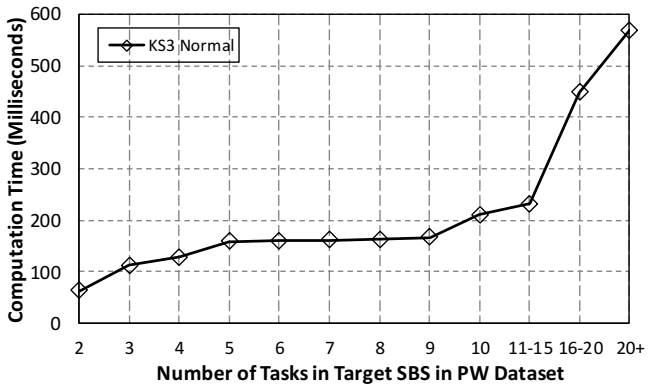

Fig. 10. Computation time in milliseconds for answering queries (experiment series A)

20+ tasks respectively. In this experiment series, there are only two keywords in a query corresponding to an SBS with a large number of tasks, which describe the first and the last tasks of the SBS. Thus, the distance between the nodes that contain the two keywords is large and KS3 needs to identify a lot of bridging nodes in the data graph. This requires KS3 to find and inspect a potentially large number of exact group Steiner trees, and thus takes KS3 more time to answer the query. However, our experimental results show that, even in the largest scenario with 29 tasks, KS3 takes only 2260 milliseconds to answer the query. This demonstrates that KS3 is fast enough in real-world applications.

Fig. 11 presents the results of experiment series B. **Fig. 11(a)** shows the impact of keyword distance on the computation time of KS3. The results demonstrate the ability of KS3 to identify the bridging nodes when the keywords in a query are not directly relevant. The KS3 normal and constraint methods can both quickly find an SBS solution with only slight increases in the computation time when the keyword distance increases. The performance of the KS3 optimal method is impacted more significantly by a large keyword distance. However, it is still capable of finding an SBS solution within a few seconds when the keyword distance is not very large. This is acceptable in most, if not all, cases. After all, not many system engineers would enter irrelevant keywords (as reflected by a large keyword distance), e.g., *car hire* and *image compression*. In fact, such keywords might not even exist in the same data graph as discussed in Section 2. An important implication learned from this set of experiments is that the quality optimisation goal is conflicting with the irrelevance between the keywords for an SBS. System engineers should not expect optimised system quality when they are not even able to identify all the tasks of the SBS. Instead, they can first enter the keywords without a quality optimisation goal. In response, the KS3 constraint method will quickly find an SBS solution that includes the keyword nodes and the bridging nodes. The engineer can then lodge an optimal query with all original keywords

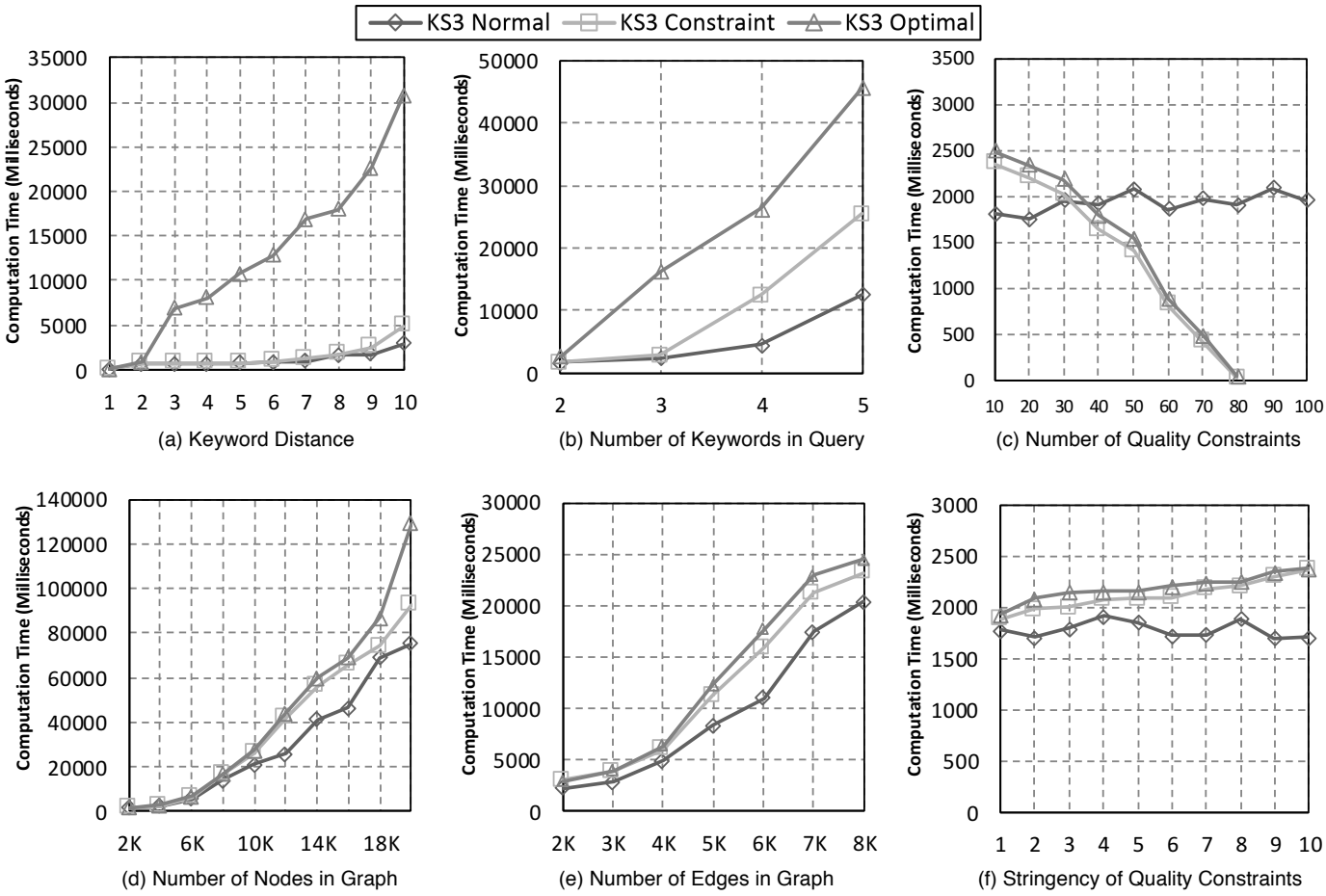

Fig. 11. Impact of factors on computation time in milliseconds (experiment series B, keyword distance = 2)

and the *bridging keywords* (the keywords in the bridging nodes). The optimal solution can be found quickly because when the keyword distance is small, the KS3 optimal method can find the optimal solution very fast, as demonstrated later in Fig. 12(b).

Fig. 11(b) shows the efficiency of the KS3 methods where different numbers of relatively irrelevant keywords are entered. It demonstrates the ability of KS3 to find an SBS solution when multiple bridging nodes are needed to connect many keyword nodes. The KS3 normal method demonstrates great performance with only a slight increase in computation time in response to the increase in the number of keywords in the query. On the other hand, the KS3 constraint and optimal methods have to explore a much larger search space. For example, to answer a query with 5 keywords, a keyword distance of 2 means that at least 9 services are needed to compose the target SBS, including 5 keyword nodes and at least 4 bridging nodes. The KS3 constraint and optimal methods need to identify and inspect the exact group Steiner trees that contain 9 nodes, as well as those that contain more than 9 nodes. The number of exact group Steiner trees to be identified and inspected is extremely large in such cases. As a result, the KS3 constraint and optimal methods need a much longer time to find the solution. This observation indicates that the system engineer may need to be notified of the potentially long waiting time when searching for an SBS solution that requires many component services.

Fig. 11(c) demonstrates that the increase in the number of quality constraints only slightly increases the computation time of the KS3 constraint and optimal methods. This indicates that KS3 can handle multiple quality constraint efficiently.

Fig. 11(d) shows how the graph size impacts the efficiency of KS3. As demonstrated, the KS3 methods take significant amounts of time (up to 129 seconds) to answer the queries on very large data graphs in extreme cases where none of the keywords are directly relevant. In a large data graph, the number of exact group Steiner trees that cover all the keyword nodes is extremely large even when the number of keywords to cover is small. The KS3 methods need to identify and inspect all those trees. As discussed in Section 3, finding minimum Steiner tree is NP-complete, and is further complicated by the quality constraints and the quality optimisation goals in the constraint and optimal queries. The extremely large search space inevitably leads to long computation time of the KS3 methods.

Fig. 11(e) shows that in a dense data graph, where each service has many neighbours, it takes a reasonable time for KS3 to find an SBS solution, within seconds in most cases. An interesting observation from comparing Fig. 11(d) with Fig. 11(e) is that the graph density does not impact the computation time of KS3 as significantly as the graph size. A higher graph density means more neighbours for each node, leading to more exact group Steiner trees to identify and inspect in response to a query. It is reflected by the increase in the computation time demonstrated in Fig. 11(e). However, the many edges connecting each node and their neighbours make it easier to identify bridging nodes for keyword nodes that are not directly relevant. It results in potentially small exact group Steiner trees, which are preferable to all KS3 methods.

Fig. 11(f) shows some interesting results. The computation times of the KS3 constraint and optimal methods decrease as the quality constraints become more stringent. After a further investigation on the results, we found out that it was because of the adoption of cost for specifying the quality constraints. Cost is a negative (lower means better) and additive quality property as presented by formula (11) in Section 4.2. An exact group Steiner tree with more nodes usually leads to a higher total cost. As a result, large Steiner trees are pruned quickly when the quality constraints are stringent. In those cases, the KS3 constraint method and optimal method either find an SBS solution quickly or fail to find one.

The results of experiment series C are shown in Fig. 12, where Fig. 12(a) is skipped on purpose for ease of comparison between Fig. 11 and Fig. 12. As demonstrated, KS3 can find an SBS solution very quickly upon queries with directly relevant keywords. It takes no more than 0.5 second in all cases. An interesting and important observation is that the impact of the number of keywords, the graph size and the graph density on the computation time of KS3 becomes negligible or rather linear, as demonstrated in Fig. 12(b), Fig. 12(d) and Fig. 12(e).

Overall, Fig. 11 and Fig. 12 demonstrate that KS3 can scale to the number of keywords in a query, the number of quality constraints, the graph size, the graph density and the stringency of the quality constraints. However, the ability of KS3 to identify bridging services comes at the price of extra computational overhead. This indicates that the system engineer should enter relevant keywords for KS3 to find an SBS solution fast. This is reasonable because in the real-world, they would not try to build SBS that performs irrelevant tasks.

5.6 Threats to Validity

Here we discuss the key threats to the validity of our evaluation of KS3.

Threats to construct validity. The main threat to the construct validity of our evaluation is comparison of success rate with the individual search methods. The individual search methods, based on Integer Programming (IP), is one of the most popular approaches to quality-aware service selection for engineering SBSs [3-5, 23, 32, 47-49], and thus is used as baseline for comparison in our evaluation. KS3 can identify the needed bridging nodes to find an exact group Steiner tree for a satisfactory SBS solution. On the other hand, the individual search methods do not identify bridging nodes, and thus cannot find a solution when bridging nodes are needed. As a result, KS3 tends to obtain better experimental results, i.e., higher success rates, than the individual search methods. Thus, the main threat to construct validity is whether the comparison with the individual search methods can properly demonstrate the effectiveness of KS3 in finding an SBS solution, especially in scenarios where bridging nodes are necessary. To minimise this threat, we fixed the keyword distance at 1 to create scenarios where bridging nodes are

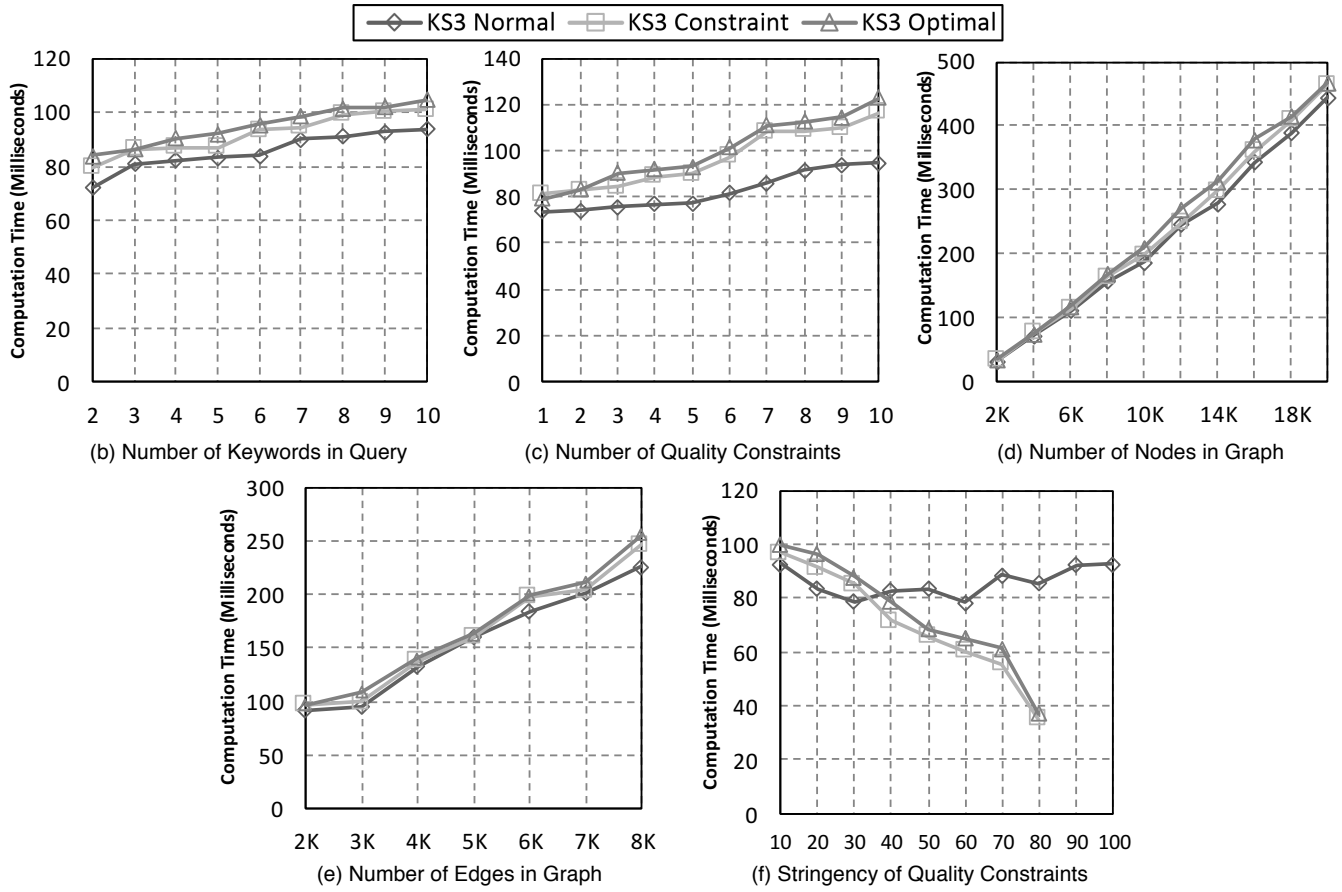

Fig. 12. Impact of factors on computation time in milliseconds (experiment series C, keyword distance = 1)

not necessary in finding an SBS solution. In such scenarios, KS3 and the individual search methods can be compared objectively. In the meantime, we changed other configuration factors, as shown in Fig. 9, to simulate different scenarios. By doing so, we could evaluate KS3 by not only the comparison with the individual search methods, but also the demonstration of how the changes in different configuration factors impact the success rate obtained by KS3.

Threats to external validity. The main threat to the external validity of our evaluation is the representativeness of the data graphs and queries created in the experiment series B and C. In experiment series B and C, we generated data graphs based on the well-known Erdős–Rényi model, using Web services in QWS [2] - a dataset widely used in research on quality-aware service selection for engineering SBSs. As discussed in Section 2, different techniques can be adopted to analyse the composability of Web services, which will lead to potentially different data graphs. Thus, the data graphs randomly generated in the experiment series B and C might not be the exact representative of the real-world Web service data graphs. This threatens the external validity of our evaluation because two keywords relevant in one data graph might not be relevant in another data graph. To minimise this threat, we used the keyword distance to control the relevance between the keywords in the generated queries. By doing so, the relevance between the query keywords is independent of the generation of the data graph. The representativeness of the randomly generated queries also threatens the external validity of our evaluation. The relevance between the query keywords in the real world is dependent on the system engineers' understanding of the functional requirements for the target SBSs. A full user study has not been performed and the presented results are specific to the data graphs and queries used in the experiments. Thus, the queries generated in experiment series B and C might not be generally or fully representative. For example, a real-world case rare but not impossible is that a system engineer is able to provide only one or two keywords for the many actually needed tasks of an SBS. To minimise this threat, we made an assumption that a system engineer can provide at least half of the needed keywords. We believe that this assumption is reasonable in most, if not all, real-world cases. In the meantime, we changed various factors related to the query keywords. Using this approach, we comprehensively evaluated KS3 by simulating query keywords provided by real-world system engineers with different levels of understanding of their target SBSs. Furthermore, the results of experiment series A can also help minimise the above threats to the external validity of our evaluation. Experiment series A was conducted based on data (crawled from programmableweb.com) about 1496 real-world Web services and 2926 real SBSs using those Web services. The data graph and queries generated in experiment series A are true representation of a real-world Web service library and system engineers' potential queries for building SBSs.

Threats to internal validity. The main threat to the internal validity of our evaluation is the comprehensiveness of our experiments. In the experiment series B and C, we simulated scenarios where six factors changed individually, as shown in Table 3. More sophisticated scenarios could have been simulated, e.g., those where two or more of those factors change at the same time. In those scenarios, the results can be predicted in general based on the results that we have obtained. For example, if the number of quality constraints and the number of nodes increase at the same time, the declining trend of the success rate of KS3 would be similar to Fig. 8(c) and Fig. 8(d). It can also be predicted that the decline would be faster because of the cumulative effect caused by the simultaneous increases in the number of quality constraints and the number of nodes. In the meantime, it can be predicted that the increase in computation time would be similar to Fig. 11(d) and Fig. 12(d), because Fig. 11(c), Fig. 11(d), Fig. 12(c) and Fig. 12(d) indicate that the impact of the increase in the number of nodes on the computation time is much more significant than the number of quality constraints.

Threats to conclusion validity. The main threat to the conclusion validity of our evaluation is the lack of statistical tests, e.g., chi-square tests. We could have conducted chi-square tests to draw conclusions when evaluating KS3. However, we ran the experiment for 100 times in each set and averaged the results each time we changed a configuration factor. This led to a large number of test cases, which tend to result in a small p-value in the chi-square tests and lower the practical significance of the test results [36]. In the largest experiment set, there were 1,000 runs. This number is not even close to the number of observation samples that concern Lin et al. in [36]. Thus, the threat to the conclusion validity due to the lack of statistical tests might be high but not significant.

6 Related Work

The service composition process for engineering an SBS consists of three phases: system planning, service discovery and service selection. There has been a large body of work on solving the problems in each of the phases individually.

In the phase of *system planning*, the system engineer determines the tasks to be performed to implement the functionality of the SBS, as well as the execution order of the tasks. Most techniques adopted in this phase to identify the tasks needed to implement an SBS are based on artificial intelligence (AI) techniques [24, 41, 43, 51]. The general idea is to model the service composition as a planning problem which can be solved using the corresponding planning problem solver. For example, in [51], the authors model the service composition problem as a cost sensitive temporally expressive (CSTE) planning problem, which is solved using a Supply Chain Planning (SCP) solver. At the end of the system planning phase, the tasks of an SBS are determined.

In the phase of *service discovery*, through service registries or service portals, the system engineer identifies a set of candidate services for each of the tasks based on the functional and semantic information of candidate services. To improve the accuracy of service matching, several semantic Web service languages have been proposed based on ontology techniques, e.g., DSD [35] and OWLS-MX [34]. These languages can semantically enrich the service description and SBS specification. The adoption of ontology automates the service matching operation that identifies the services that can perform the tasks of the SBS determined in the system planning phase. For a task, there are usually many functionally-equivalent services that can perform the task [5, 49]. Those services differ in multi-dimensional quality properties. To identify a huge number of such services, the service discovery operation must be automated. Many approaches have been proposed to address this issue [12, 16, 33]. Based on automatic service matching, these approaches adopt ontology techniques such as logical reasoning and temporal planning to automate the service discovery operation. At the end of this phase, a group of functionally-equivalent candidate services are selected for each of the tasks required for the SBS.

In the *service selection* phase, the system engineer selects one service from each set of functionally-equivalent candidate services to compose the target SBS. In this phase, the selected services must collectively fulfil the multi-dimensional quality constraints for the SBS [5, 13, 23, 47, 49], e.g., reliability, throughput, cost, etc., which is a NP-complete problem. Integer Programming (IP) is the main technique adopted in this phase. AgFlow [49] is one of the most representative approaches. Following the idea of AgFlow, many researchers have been trying to reduce the computation time for quality-aware service selection [4, 47] or to propose enhanced approaches for solving the problem in more complex environments [5, 23, 32].

There are a lot of complex techniques and approaches available for solving different problems in the above phases. A lot of time and effort are required for a system engineer to choose, learn and apply these techniques and approaches to eventually obtain an SBS solution. This has been a major obstacle to further and broader applications of SOA. An innovative approach is needed that can help system engineers find services to build SBSs fast without having to go through all the complicated phases. Some approaches have been proposed. Based on the idea of tag-based search introduced in [45], the authors of [37] propose a planning technique that explore SBS solutions by looking up services whose tags match the tags describing the SBS. For each query, the engineer needs to enter a source tag and a destination tag. The proposed planning technique will heuristically identify the possible service compositions with an entry service according to the source tag and an exit service according to the destination tag. An approach is proposed in [26] for helping system engineers navigate from the entry service to the exit service through multiple queries. There are two major limitations to this planning technique. First, each query allows only two tags, i.e., one source tag and one destination tag. Multiple tags can only be entered one by one in different queries that are processed individually until a final solution is found. Second, quality constraints cannot be speci-

fied. Thus, the planning technique is not suitable for building software systems with quality constraints.

KS3 assists a system engineer without detailed knowledge of SOA techniques in identifying the services needed for building an SBS by entering only a few keywords that represent the tasks of the SBS. By integrating and automating the system planning, service discovery and service selection operations, KS3 can significantly save the time and efforts during the SBS engineering process. Furthermore, KS3 overcomes the limitations of the approaches proposed in [26, 37]. Firstly, KS3 can handle multiple keywords (i.e., multiple system tasks) in one query. Secondly, system engineers do not have to enter the keywords in a specific order. Thirdly, KS3 allows multi-dimensional quality constraints and a quality optimisation goal to be specified for the target SBS.

7 Conclusions and Future Work

In this paper, we propose KS3, a novel approach that integrates and automates the system planning, service discovery and service selection operations for building service-based systems (SBSs). It assists system engineers without detailed knowledge of SOA techniques in identifying SBS solutions with only a few keywords that describe the tasks of the SBSs. KS3 offers a new paradigm for efficient SBSs engineering that can significantly save the time and effort during the SBS engineering process. The comprehensive experimental analysis shows the practicality, effectiveness and efficiency of KS3.

According to the experimental results, KS3 takes a significant amount of time to find an SBS solution in response to a query with irrelevant keywords when the number of keywords in the query or the graph size is large. Thus, it requires that system engineers have a proper understanding of the functional requirements for their SBSs. In our future work, we will address this issue by recommending relevant keywords (i.e., relevant Web services) to system engineers based on collaborative filtering techniques [50]. We will also enhance KS3 with automatic query expansion techniques [15] to handle synonymy, word inflections and polysemy.

Acknowledgment

This work is partly supported by Australian Research Council Discovery Project DP150101775. We are grateful for Jian Wang's help with crawling the information from programmableweb.com. Qiang He is the corresponding author of this paper.

References

[1] "BPM Product Analysis - A Comparison of IBM Business Process Manager and Oracle BPM," AVIO Consulting, 2013.

[2] E. Al-Masri and Q. H. Mahmoud, "Investigating Web Services on the World Wide Web," *Proc of 17th International Conference on World Wide Web (WWW 2008)*, Beijing, China, pp. 795-804, 2008.

[3] M. Alrifai and T. Risse, "Combining Global Optimization with Local Selection for Efficient QoS-Aware Service Composition," *Proc of 18th International Conference on World Wide Web (WWW 2009)*, Madrid, Spain, pp. 881-890, 2009.

[4] M. Alrifai, D. Skoutas, and T. Risse, "Selecting Skyline Services for QoS-based Web Service Composition," *Proc of 19th International Conference on World Wide Web (WWW 2010)*, Raleigh, North Carolina, USA, pp. 11-20, 2010.

[5] D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Transactions on Software Engineering (TSE),* vol. 33, no. 6, pp. 369-384, 2007.

[6] S. Balasubramaniam, G. Lewis, S. Simanta, and D. B. Smith, "Situated Software: Concepts, Motivation, Technology, and the Future," *IEEE Software,* vol. 25, no. 6, pp. 50-55, 2008.

[7] L. Baresi and S. Guinea, "Self-Supervising BPEL Processes," *IEEE Transactions on Software Engineering,* vol. 37, no. 2, pp. 247-263, 2011.

[8] A. P. Barros and M. Dumas, "The Rise of Web Service Ecosystems," *IT Professional Magazine,* vol. 8, no. 5, p. 31, 2006.

[9] D. Benslimane, S. Dustdar, and A. Sheth, "Services Mashups: The New Generation of Web Applications," *IEEE Internet Computing,* vol. 12, no. 5, pp. 13-15, 2008.

[10] S. Bergamaschi, E. Domnori, F. Guerra, R. T. Lado, and Y. Velegrakis, "Keyword Search over Relational Databases: A Metadata Approach," *Proc of 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD 2011)*, Athens, Greece, pp. 565-576, 2011.

[11] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword Searching and Browsing in Databases using BANKS," *Proc of 18th International Conference on Data Engineering (ICDE 2002)*, San Jose, CA, USA, pp. 431-440, 2002.

[12] A. Brogi, S. Corfini, and R. Popescu, "Semantics-Based Composition-Oriented Discovery of Web Services," *ACM Transactions on Internet Technology,* vol. 8, no. 4, pp. 19:1-19:39, 2008.

[13] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS Management and Optimisation in Service-Based Systems," *IEEE Transactions on Software Engineering (TSE),* vol. 37, no. 3, pp. 387-409, 2010.

[14] V. Cardellini, E. Casalicchio, V. Grassi, S. Lannucci, F. Lo Presti, and R. Mirandola, "MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems," *IEEE Transactions on Software Engineering,* vol. 38, no. 5, pp. 1138-1159, 2012.

[15] C. Carpineto and G. Romano, "A Survey of Automatic Query Expansion in Information Retrieval," *ACM Computing Surveys,* vol. 44, no. 1, pp. 1-50, 2012.

[16] G. Cassar, P. Barnaghi, and K. Moessner, "Probabilistic Matchmaking Methods for Automated Service Discovery," *IEEE Transactions on Services Computing (TSC),* vol. 7, no. 4, pp. 654-666, 2014.

[17] M. Dojchinovski, J. Kuchar, T. Vitvar, and M. Zaremba,

"Personalised Graph-Based Selection of Web APIs," *Proc of International Semantic Web Conference (ISWC2012)*, pp. 34-48, 2012.

[18] R. Durrett, *Random Rraph Dynamics*: Cambridge University Press, 2007.

[19] Z. Feng, B. Lan, Z. Zhang, and S. Chen, "A Study of Semantic Web Services Network," *The Computer Journal,* vol. 58, no. 6, pp. 1293-1305, 2015.

[20] K. Golenberg, B. Kimelfeld, and Y. Sagiv, "Keyword Proximity Search in Complex Data Graphs," *Proc of 28th ACM SIGMOD International Conference on Management of Data (SIGMOD 2008)*, pp. 927-940, 2008.

[21] L. M. V. Gonzalez, L. Rodero-Merino, C. Juan, and M. A. Lindner, "A Break in the Clouds: Towards A Cloud Definition," *Computer Communication Review,* vol. 39, no. 1, pp. 50-55, 2009.

[22] Q. He, J. Han, Y. Yang, H. Jin, J.-G. Schneider, and S. Versteeg, "Formulating Cost-Effective Monitoring Strategies for Services-based Systems," *IEEE Transactions on Software Engineering,* vol. 40, no. 5, pp. 461-482, 2014.

[23] Q. He, J. Yan, H. Jin, and Y. Yang, "Quality-Aware Service Selection for Service-based Systems Based on Iterative Multi-Attribute Combinatorial Auction," *IEEE Transactions on Software Engineering (TSE),* vol. 40, no. 2, pp. 192-215, 2014.

[24] J. Hoffmann, P. Bertoli, and M. Pistore, "Web Service Composition as Planning, Revisited: In Between Background Theories and Initial State Uncertainty," *Proc of 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, Vancouver, British Columbia, Canada, pp. 1013-1018, 2007.

[25] V. Hristidis and Y. Papakonstantinou, "DISCOVERY: Keyword Search in Relational Databases," *Proc of 28th International Conference on Very Large Data Bases (VLDB 2002)*, Hong Kong, China, pp. 670-681, 2002.

[26] G. Huang, Y. Ma, X. Liu, Y. Luo, X. Lu, and M. B. Blake, "Model-Based Automated Navigation and Composition of Complex Service Mashups," *IEEE Transactions on Services Computing (TSC),* vol. 8, no. 3, pp. 494-506, 2015.

[27] K. Huang, Y. Fan, and W. Tan, "An Empirical Study of Programmable Web: A Network Analysis on a Service-Mashup System," *Proc of 19th International Conference on Web Services (ICWS2019)*, pp. 552-559, 2012.

[28] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem* vol. 53: Elsevier, 1992.

[29] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong, "Exact Top-k Nearest Keyword Search in Large Networks," *Proc of 36th ACM SIGMOD International Conference on Management of Data (SIGMOD 2015)*, pp. 393-404, 2015.

[30] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong, "Exact Top-k Nearest Keyword Search in Large Networks," *Proc of 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD 2015)*, Melbourne, Australia, pp. 393-404, 2015.

[31] H. Kil, S.-C. Oh, E. Elmacioglu, W. Nam, and D. Lee, "Graph Theoretic Topological Analysis of Web Service Networks," *World Wide Web (WWW),* vol. 12, no. 3, pp. 321-343, 2009.

[32] A. Klein, F. Ishikawa, and S. Honiden, "Towards Network-Aware Service Composition in the Cloud," *Proc of 21st World Wide Web Conference (WWW 2012)*, Lyon, France, pp. 959-968, 2012.

[33] M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service Discovery with OWLS-MX," *Proc of 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, Hakodate, Japan, pp. 915-922, 2006.

[34] M. Klusch, B. Fries, and K. P. Sycara, "OWLS-MX: A Hybrid Semantic Web Service Matchmaker for OWL-S Services," *Journal of Web Semantics,* vol. 7, no. 2, pp. 915-922, 2009.

[35] U. Küster, B. König-Ries, M. Stern, and M. Klein, "DIANE: An Integrated Approach to Automated Service Discovery, Matchmaking and Composition," *Proc of 16th International Conference on World Wide Web (WWW 2007)*, Banff, Alberta, Canada, pp. 1033-1042, 2007.

[36] M. Lin, H. C. Lucas Jr., and G. Shmueli, "Too Big to Fail: Large Samples and the p-Value Problem," *Information Systems Research,* vol. 24, no. 4, pp. 906-917, 2013.

[37] X. Liu, Y. Ma, G. Huang, J. Zhao, H. Mei, and Y. Liu, "Data-Driven Composition for Service-Oriented Situational Web Applications," *IEEE Transactions on Services Computing (TSC),* vol. 8, no. 1, pp. 2-16, 2015.

[38] S. Lyu, J. Liu, M. Tang, G. Kang, B. Cao, and Y. Duan, "Three-Level Views of the Web Service Network: An Empirical Study Based on ProgrammableWeb," *Proc of 2014 IEEE International Congress on Big Data*, pp. 374-381, 2014.

[39] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer Programming Formulation of Traveling Salesman Problems," *Journal of the ACM (JACM),* vol. 7, no. 4, pp. 326-329, 1960.

[40] Y. Ni, Y. Fan, W. Tan, K. Huang, and J. Bi, "NCSR: Negative-Connection-Aware Service Recommendation for Large Sparse Service Network," *IEEE Transactions on Automation Science and Engineering,* vol. 13, no. 2, pp. 579-590, 2016.

[41] S.-C. Oh, D. Lee, and S. R. Kumara, "Effective Web Service Composition in Diverse and Large-Scale Service Networks," *IEEE Transactions on Services Computing (TSC),* vol. 1, no. 1, pp. 15-32, 2008.

[42] A. A. Patil, S. A. Oundhakar, A. P. Sheth, and K. Verma, "Meteor-S Web Service Annotation Framework," *Proc of 13th International Conference on World Wide Web (WWW 2004)*, New York, NY, USA, pp. 553-562, 2004.

[43] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso, "Automated Composition of Web Services by Planning at the Knowledge Level," *Proc of 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, Edinburgh, Scotland, UK, pp. 1252-1259, 2005.

[44] S. Ran, "A Model for Web Services Discovery with QoS," *SIGecom Exchanges,* vol. 4, no. 1, pp. 1-10, 2003.

[45] A. V. Riabov, E. Boillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan, "Wishful Search: Interactive Composition of Data Mashups," *Proc of 17th International Conference on World Wide Web (WWW 2008)*, Beijing, China, pp. 775-

784, 2008.

[46] J. Shang, L. Liu, and C. Wu, "WSCN: Web Service Composition Based on Complex Networks," *Proc of 2013 International Conference on Service Sciences (ICSS 2013)*, pp. 208-213, 2013.

[47] I. Trummer, B. Faltings, and W. Binder, "Multi-Objective Quality-Driven Service Selection - A Fully Polynomial Time Approximation Scheme," *IEEE Transactions on Software Engineering (TSE),* vol. 40, no. 2, pp. 167-191, 2014.

[48] F. Wagner, B. Klöpper, F. Ishikawa, and S. Honiden, "Towards Robust Service Compositions in the Context of Functionally Diverse Services," *Proc of 21st International World Wide Web Conference (WWW 2012)*, Lyon, France, pp. 969-978, 2012.

[49] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering (TSE),* vol. 30, no. 5, pp. 311-327, 2004.

[50] Z. Zheng and M. R. Lyu, "Collaborative Reliability Prediction of Service-Oriented Systems," *Proc of 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, Cape Town, South Africa, pp. 35-44, 2010.

[51] G. Zou, Q. Lu, Y. Chen, R. Huang, Y. Xu, and Y. Xiang, "QoS-Aware Dynamic Composition of Web Services Using Numerical Temporal Planning," *IEEE Transactions on Services Computing (TSC),* vol. 7, no. 1, pp. 18-31, 2014.