IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON MOBILE MULTIMEDIA: METHODOLOGY AND APPLICATIONS

Received December 14, 2019, accepted December 26, 2019, date of publication January 1, 2020, date of current version January 15, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2963548

Q-Graphplan: QoS-Aware Automatic Service Composition With the Extended Planning Graph

ZHAONING WANG, BO CHENG, WENKAI ZHANG, AND JUNLIANG CHEN

State Key laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Bo Cheng (chengbo@bupt.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61772479, and in part by the National Key Research and Development Program of China under Grant 2017YFB140.

ABSTRACT With the progress of web technologies, web services with abundant functionalities, such as video transmission, location, navigation, etc., are becoming more and more pervasive. Automatic web service composition aims to automatically combine selected elementary web services from a finite service set by matching the input and output parameters given an initial state and a goal state. Considering the end-to-end Quality-of-Service(QoS) of each web service, the service composition problem becomes an optimization problem to find the optimal solution. This paper maps this problem to an automatic planning problem and proposes Q-Graphplan based on the classical graphplan, an efficient planner for solving classical planning problems. First, we construct a planning graph based on the dependency relationships of the web services and extract essential heuristics according to the reachability analysis. Second, we convert this planning graph to a directed path generation graph. Finally, we extract the optimal solution from the path generation graph using a backward A* algorithm with the heuristics of the planning graph. Furthermore, our approach avoids redundancies when constructing the planning graph and improves the searching effectiveness in extracting solution. We conduct experiments on the WSC-2009 dataset to compare performance against present approaches, and the results show the efficiency and effectiveness of our proposed approach.

INDEX TERMS Service composition, AI planning, QoS-aware, heuristic search.

I. INTRODUCTION

Since the development of the internet has been accelerating in recent years, web services are becoming more and more pervasive in people's daily lives [1]. Web services are modular web-based self-described software components that implement a collection of functions or operations [2], which are made available through syntactic descriptions, such as SOAP, RESTful and Web API. Each web service associates to a set of non-functional quality of service (QoS) parameters (e.g., response time, throughput, reliability, etc.) which determine the performance of this service. Due to the variability of users' requests, a single web service hardly satisfies the users' requirements precisely.

Assuming a scenario, a user is driving a car with a smart car device in an unfamiliar city to have sightseeing. The smart car device is in the mobile network environment with abundant web services, such as video transmission, voice navigation, viewpoints recommendation, GPS, store query, music mix, etc. To make diverse web services available in the mobile network environment, plenty of innovative research work has been proposed. Work [3] proposed a creativity feature-based learning system for Internet-of-Things (IoT) services that classified the data and detect anomaly events efficiently. It effectively reduced the computation overhead and energy consumption of the IoT services and significantly improved the efficiency of neural network training to adapt to the mobile network environment better. Work [4] proposed a transmission mechanism of Internet of Vehicles that laid the foundation for multimedia services. It concurrently considered privacy protection and multimedia transmission resulting in significant improvement in the performance and security of multimedia services in the mobile network environment. Work [5] proposed a novel solution for the design and performance analysis of wireless-powered communication system. Work [6] proposed a scalable video coding sharing scheme that made video distribution more flexible in the mobile edge network. Work [7] proposed a social and smart

The associate editor coordinating the review of this manuscript and approving it for publication was Dapeng Wu.

device-to-device (D2D) network framework which provided a novel model for mobile edge cloud. Various mobile web services could be orchestrated together and invoked to dynamically satisfy the user's requirements efficiently. QoS-aware automatic service composition (ASC) is proposed to automatically combine the multiple web services as a service chain based on the input-output dependencies [8], to produce the desired output fulfilling users' goals while choosing the solution with the optimal QoS value.

Plenty of research results have been proposed about the ASC problem. Most of them focus on single-objective optimization [9]–[12]. QoS criteria include multiple categories, therefore, single-objective optimization in service composition often could not satisfy the present requirements. Consequently, multi-objective optimization of QoS values for service composition problem has become an attractive topic [13]–[15].

Among the proposed approaches, two main different models are generally used to address this issue. The first could be characterized as the static model, separately constructing an abstract service workflow and determining the atomic service via optimization algorithms to find solutions with optimal or near-optimal QoS values. This model transforms the ASC problem as an integer programming problem, an NP-hard problem [16]–[20]. This approach's search space is limited by the predefined workflow. Therefore, they can not ensure the globally optimal of the solutions. The second model is the dynamic model, dynamically constructing the service chain with the changes in QoS values. This model usually generates a dependency graph and solve the problem as a shortest path problem [1], [21]–[24]. The graphplan algorithm [25] is a frequent approach in the second model [26], [27]. In general, this approach depends on an efficient search strategy. Furthermore, it might produce redundant services that could not lead to the goal and affect the searching efficiency during the construction process of the dependency graph.

Heuristic search algorithms (e.g., A* algorithm) [28], [29] have been widely used to optimize the optimal path generation problems. The planning graph could be seen as a source of information for developing search heuristics. In this paper, we study the multi-objective optimization in QoS-aware ASC problem with the dynamic model and apply the planning graph as a dependency analyzer and the heuristic generator to solve the problem. The main contributions of this paper are as follows:

1) We combine graphplan with heuristic search and propose a new approach called Q-Graphplan as a QoS-aware ASC problem solver. Q-Graphplan considers multiple QoS criteria and dynamically chooses services with the QoS value changes other than optimizing fixed composite services. Moreover, Q-Graphplan extends the classical planning graph with an extended layer to provide more useful information.
2) We first introduce the heuristic extraction mechanism into the ASC problem-solver, which take full advantages of the planning graph and significantly increase the searching effectiveness. We present a redundancy-free algorithm to construct the planning graph which eliminates the noneffective searching space and saves the backtracking time when searching for solutions.
3) We present a new algorithm to convert the planning graph as a directed path generation graph. We use the admissible heuristics extracting from the planning graph and propose a backward A* algorithm to search optimal solution from the directed path generation graph.

This paper is organized as follows: In Section II, we introduce the existing relative literature. Section III presents some definitions and formulates the problem. Section IV presents the Q-Graphplan algorithm, our kernel proposal. Section V presents the experiment results. Section VI concludes the work look forwards the future.

II. RELATED WORK

In this section, we describe the state of art proposals from two aspects. In the first part, we focus on the QoS-aware ASC problem. In the second part, we mainly describe around graphplan and heuristic search.

As mentioned above, the static model to solve the QoS-aware ASC problem has two steps. The first step is to construct a workflow with abstract services and the second step is service selection for each abstract service. Work [30] formalized the general procedure of the static model. Work [17] proposed a mixed integer programming approach. Work [18] first selected multiple services for every abstract service class of a complex process template by a heuristic algorithm, which aims at determining a set of near-optimal service compositions. A significant number of research results have been done based on evolutionary algorithms, such as Bee Colony Optimization [31], Genetic Algorithms [19]. Work [20] studied QoS dependency on the time of the execution or the input data. The authors proposed a genetic algorithm to obtain approximations of the Pareto optimal solutions set. Work [13] proposed a QoS dependency-aware service composition considering the dependency that the QoS values of a service are correlated to other services, and when these services are selected together, their QoS values will change. In general, these approaches optimize QoS values on a static abstract service workflow, therefore, it could miss some optimal solutions with different services.

The dynamic model is to dynamically adjust the service sequences with the changes of the QoS values. Work [32] used CSP (Cost-Sensitive Planning) technology to find local optimal solution based on multiple QoS constraints in dynamic service composition. Work [14] found the optimal Pareto front in the dependency graph for multi-objective optimization. Work [11] proposed to generate the Pareto optimal solutions in a parallel setting. This approach fails to consider common situations when constructing the dependency graph.Work [10] first proposes a continuous query mechanism for single QoS criteria.

Work [33]'s graphplan algorithm has emerged as the fastest planner for solving classical planning problems. It is a technology using the planning graph to search plans [34]. The state of the art proposals [35] usually transform the planning graph to a DAG(directed acyclic graph) and use Dijkstra's algorithm to find the optimal solution. This approach only treats the planning graph as a dependency analyzer while wasting the heuristics contained in the planning graph. A number of heuristic functions to extract the heuristics from the planning graph have been proposed and proved effective and admissible to optimize the searching process in graphplan. Work [34] proposed HSP-R [25], a state-search planner, providing heuristic functions to extracting heuristics from the planning graph. HSP-R follows a variation of A* search algorithm, called Greedy Best First, which uses the cost function $f(S) = g(S) + h(S)$, where $g(S)$ is the accumulated cost and $h(S)$ is the heuristic value of state S. A* search has been used abundantly for low-dimensional path planning [36]. The performance of A* search are mainly influenced by the heuristic functions. Work [37] proposed a series of heuristic functions based on HSP-R and proves the usability and efficiency. Work [38] first combined graphplan and A* search with considering QoS constraints to ASC problem. Work [38], [39] all failed to deal with redundancies in constructing the planning graphs.

Q-Graphplan considers multiple QoS criteria in finding the optimal solution, constructs the redundancy-free planning graph, extracts admissible heuristic from the planning graph and use backward A* algorithm to find a optimal solution.

III. PROBLEM FORMULATION

In this section, we first describe the relative concepts of the QoS-aware ASC problem by a set of formal definitions.

A. QoS-AWARE ASC PROBLEM

Definition 1 (Elementary Web Service): An elementary web service w could be denoted as a tuple $(I(w), O(w), Q(w))$, *where* $I(w) = \{I_1(w), I_2(w)\ldots\}$ *is a set of input parameters, where* $O(w) = \{O_1(w), O_2(w)\ldots\}$ *is a set of output parameters, where* $Q(w) = \{Q_1(w), Q_2(w)\ldots\}$ *is a set of QoS values.*

Each elementary web service has one operation. A set of elementary web services consist of a web service composition. In this paper we call the elementary web service as the web service for short.

The QoS criteria described in the context of elementary web services are also used to evaluate the QoS of a set of composite web services. In the composite web services, there could exist sequential and parallel relation among different web services. Considering a set of composite web services C including n services $\{w_1, w_2, \ldots, w_n\}$, the QoS values $Q_i(C)$ are calculated according to the equations in table 1.

Some of the above criteria are negative, i.e., the lower the value, the higher the quality. The other criteria are positive, i.e.,the higher the value, the higher the quality. The QoS criteria above have different range. Therefore, we use a set of equations to scale these values to the range of [0, 1] and unified them to be negative before using them. Given a QoS criteria Q_i for an elementary web service w_j, we calculate an utility value $S_i(w_j)$. For negative criteria, such as response time and execution time, values are scaled using equation (1). For positive and non-multiplication criteria, values are scaled using equation (2). For positive and multiplication criteria, values are scaled using equation (3). Similarly, we could calculate scaled QoS values of the composite web services $S_i(C)$.

TABLE 1. QoS criteria of the composite services.

QoS Criteria	Sequential Relation	Parallel Relation
$Q_{rt}(C)$	$\sum_{i=1}^{n} Q_{rt}(w_i)$	$\max_{i=1}^{n} Q_{rt}(w_i)$
$Q_{tp}(C)$	$\min_{i=1}^{n} Q_{tp}(w_i)$	$\min_{i=1}^{n} Q_{tp}(w_i)$
$Q_{ep}(C)$	$\sum_{i=1}^{n} Q_{ep}(w_i)$	$\sum_{i=1}^{n} Q_{ep}(w_i)$
$Q_{rel}(C)$	$\prod_{i=1}^{n} Q_{rel}(w_i)$	$\prod_{i=1}^{n} Q_{rel}(w_i)$
$Q_{suc}(C)$	$\prod_{i=1}^{n} Q_{suc}(w_i)$	$\prod_{i=1}^{n} Q_{suc}(w_i)$
$Q_{ava}(C)$	$\prod_{i=1}^{n} Q_{ava}(w_i)$	$\prod_{i=1}^{n} Q_{ava}(w_i)$

rt=response time, tp=throughput, ep=execution time, rel=reliability, suc=success rate, ava=availability

$$S_i(w_j) = \begin{cases} \dfrac{Q_i(w_j) - Q_i^{min}}{Q_i^{max} - Q_i^{min}}, & Q_i^{max} - Q_i^{min} \neq 0 \\ 1, & Q_i^{max} - Q_i^{min} = 0 \end{cases} \quad (1)$$

$$S_i(w_j) = \begin{cases} \dfrac{Q_i^{max} - Q_i(w_j)}{Q_i^{max} - Q_i^{min}}, & Q_i^{max} - Q_i^{min} \neq 0 \\ 1, & Q_i^{max} - Q_i^{min} = 0 \end{cases} \quad (2)$$

$$S_i(w_j) = \begin{cases} \dfrac{\ln(Q_i^{max}) - \ln(Q_i(w_j))}{\ln(Q_i^{max}) - \ln(Q_i^{min})}, & Q_i^{max} - Q_i^{min} \neq 0 \\ 1, & Q_i^{max} - Q_i^{min} = 0 \end{cases} \quad (3)$$

Given a set of elementary web services σ including a set of web services $\{w_1, w_2, \ldots, w_n\}$, to measure the multiple QoS criteria of the composite web services, we aggregate the multiple criteria to a single criteria using equation (4).

$$S(\sigma) = \sum_i S_i(\sigma) W_i + \ln \sum_j S_j(\sigma) W_j \quad (4)$$

where $i \in \{rt, tp, ep\}, j \in \{rel, suc, ava\}$, $W_i, W_j \in [0, 1]$ and $\sum_{i,j} W = 1$.

Definition 2 (QoS-Aware ASC Problem): A QoS-Aware ASC problem is denoted as a tuple $(W, P, R_{in}, R_{out}, Q)$, *where* $W = \{w_1, w_2\ldots\}$ *is a set of web services,* $P = \{P_1, P_2, \ldots\}$ *is set of applicable parameters,* $R_{in} = \{r_{in}^1, r_{in}^2\ldots\}$ *is a set of provided input parameters,* $R_{out} = \{r_{out}^1, r_{out}^2\ldots\}$ *is a set of expected output results, and* $Q = \{Q_1, Q_2\ldots\}$ *is a finite set of composite web services' QoS criteria.*

In the above definition, we use Q to denote QoS criteria. In the problem solving process, to simplify the calculation, we use equation (4) to scale the values denoted as $S = \{S_1, S_2, \ldots\}$.

IV. Q-GRAPHPLAN

In this section, we convert the QoS-aware ASC problem to a QoS-aware graphplan problem and describe the process of Q-Graphplan to solve the problem in detail. Generally speaking, our approach consists of three phase. The first phase is generating the planning graph and deriving heuristic

information. The second phase is converting the planning graph to a directed path generation(DPG) graph to find optimal path. The third phase is extracting the optimal solution from the LPG graph using a backward A* algorithm.

In this section we will consider a QoS-aware ASC problem $(W, P, R_{in}, R_{out}, Q)$ which is mapped to a graphplan problem (O, s_0, g) and a cost function, we will show the detailed process and algorithms of how to find the optimal web service composition solution using Q-Graphplan.

A. QoS-AWARE GRAPHPLAN PROBLEM

In this section, we define the relative concepts of the classical Graphplan, and map the QoS-aware ASC problem to a graphplan problem.

Definition 3 (Graphplan Problem): Given a finite proposition symbol set $L = \{p_1, p_2, \ldots, p_n\}$, a Graphplan problem is defined as a tuple (O, s_0, g), where:

- $O = (S, A, \gamma)$ *is a STRIPS planning domain.*
- $S = \{s_1, s_2, \ldots\}$ *is a finite or recursively enumerable set of states.*$S \subseteq 2^L$ *indicates that each state* $s \in S$ *is a subset of* L.
- $A = \{a_1, a_2, \ldots\}$ *is a finite or recursively enumerable set of actions. An action is a couple* $(pre(a), effect^+(a))$ *where* $pre(a) \subseteq S$ *denotes the preconditions and* $effect^+(a) \subseteq S$ *denotes the positive effects of the action.*
- $\gamma : S \times A \rightarrow 2^S$ *is a state transition function. For an action* a *and a state* s*, if and only if* $pre(a) \subseteq s$*, we call* a *is applicable in* s*, and* $\gamma(s, a) = s \cup effects^+(a)$.
- $s_0 \in S$ *is the initial state and* g *is the goal.*

An ASC problem could be mapped to a graphplan problem as the following rules. The proposition set L could be mapped to the parameter set P, and each parameter could be seen as a proposition p. The web service set W in the ASC problem could be mapped to the action set A in the graphplan problem. Correspondingly, each elementary web service w could be mapped to an action a. The input parameter set $I(w)$ could be mapped to the preconditions $pre(a)$, and the output parameter set $O(w)$ could be mapped to the positive effects $effect^+(a)$. Same as the graphplan problem, all the input and output parameter set in the ASC problem are positive. The input parameter set R_{in} of the ASC problem could be mapped to the initial state s_0 of the graphplan problem. Similarly, the output parameter set R_{out} are mapped to the goal set g.

Definition 4 (Planning Graph): A planning graph G_p is a directed layered graph denoted as a set of proposition layers and action layers $\langle P_0, A_1, P_1, \ldots, A_n, P_n\rangle$ where, the layer P_0 denotes the initial state s_0 of the graphplan problem, the layer A_1 denotes the applicable action set of the layer P_0, and the layer P_1 is the union set of the layer A_1's positive effects and the layer P_1.

In the ASC problem, an action layer could be mapped to a set of web services, and a proposition layer could be mapped to a set of input and output parameters.

Definition 5 (Extended Layer): An extended layer is a set of sequential hashtables $\langle H_1, H_2, \ldots, H_n\rangle$ recording the extended information of each layer in the planning graph. A hashtable H_i maps each proposition p in layer i a list of reachable action tags with a tuple $(p, ActionTagList)$. An action tag is a tuple $(a, Level)$ recording the layer number in which the proposition p is reached by action a.

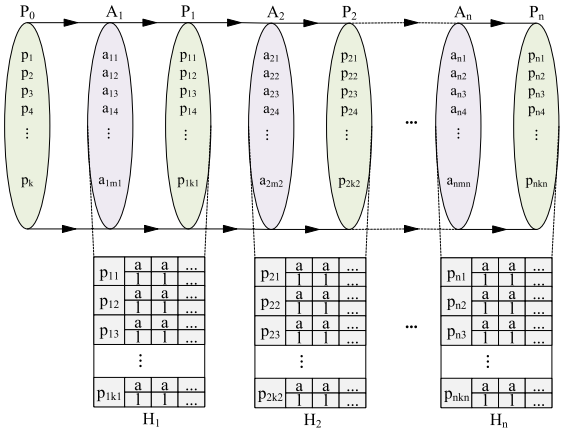

FIGURE 1. The extended planning graph.

The extended layer is an assistant layer for the classical planning graph. It indicates each proposition in the current layer in which layer by what action the proposition is reached. As is shown in figure 1, adding the hash layer, the new planning graph could be denoted as $\langle P_0, A_1, P_1, H_1 \ldots, A_n, P_n, H_n\rangle$. The hashtable in each layer could be generated when constructing the corresponding proposition and action layer. It records extended information beyond the classical planning graph.

Definition 6 (Cost Function): $cost(A)$ is a cost function for actions, $cost : A \rightarrow R$, where R is the real numbers.

Corresponding to the QoS-aware ASC problem, the cost values of the actions are the QoS values of executing the web services after scaling using equation (1)-(3).

Definition 7 (Graphplan Solution): A sequence set of actions' sets $\langle \pi_1, \pi_2, \ldots, \pi_n\rangle$ is a solution of the graphplan (O, s_0, g), if and only if each $\pi_i \in \Pi$ is independent and each action in π_i is parallel, where π_1 is applicable to s_0, π_2 is applicable to $\gamma(s_0, \pi_1)$, etc., and $g \subseteq \gamma(\ldots\gamma(\gamma(s_0, \pi_1), \pi_2)\ldots\pi_n)$.

A solution of an ASC problem could be mapped to a graphplan solution. Each action in the graphplan solution corresponds to a elementary web service in the service set. A set of parallel actions correspond to a set of web services which could be accessed concurrently. The cost of a solution could be calculated using equation (4). An optimal solution of the QoS-aware ASC problem is the solution which has the least cost. Correspondingly, finding the optimal solution of the QoS-aware ASC problem could be mapped to the problem finding the shortest executing sequence in the planning graph. We call this process QoS-aware graphplan(Q-Graphplan) in this paper.

B. CONSTRUCTING THE PLANNING GRAPH

Constructing a planning graph is to generate reachable layers according to the dependency and reachability analysis.

Algorithm 1 Constructing the Planning Graph

```
1: function CONSTRUCT(A, s_0, g)
2:     Candidates ← ∅, P_0 ← s_0, G_p ← ⟨P_0⟩,
3:     Label ← 0
4:     if g ⊆ s_0 or Fixedpoint(G_p) then
5:         return true
6:     end if
7:     for a ∈ A do
8:         if pre(a) ⊆ P_i then
9:             Candidates ← Candidates ∪ {a}
10:        end if
11:    end for
12:    if Expand(Candidates, A, s_0, g, Label) then
13:        add Candidates to G_p as the new action layer
14:        add s_0 to G_p as the new proposition layer
15:        generate a new hashtable for this layer
16:    end if
17:    if Construct(A, s_0, g) then
18:        return true
19:    end if
20:    return false
21: end function
```

The dependency indicates that each layer is independent to its last layer. The reachability indicates that a proposition layer is generated indicates that all the propositions in this layer is reachable.The planning graph is expanded layer by layer until the goal state or the fixed layer is arrived. In each layer, the action layer, the proposition layer and the extended layer are generated sequentially. The planning graph acts as a heuristic generator. The construction process is the process of extracting heuristics from reachability analysis.

As is shown in algorithm 1, constructing the planning graph includes two recursive processes. In line 4-6, it estimates whether the planning graph is completely constructed. Two conditions are considered, which are the goal state or the fixed point is reached. It has been proved that all the planning graph would reach a layer after which the state will not change and this layer is called a fixed point. The function $Fixedpoint(G_p)$ is used to decide if this layer is reached. Line 7-11 find an applicable action set *Candidates* for the present proposition layer P_i. Line 12-15 recursively analyze the reachability of each service and filter the redundant services in *Candidates* with the function *Expand*(), then add the remaining reachable services the new action layer. Correspondingly, the initial set s_0 is updated and added as the new proposition layer, and the new extended layer is generated. Line 17-19 recursively construct the planning graph until the terminated conditions in line 4-6 are satisfied.

C. EXPANDING THE GRAPH WITHOUT REDUNDANCIES

Expanding a planning graph is a process to establish a new layer from the existing layers. Usual approach generating a new layer is similar to a BFS(band-first-search), which means it will add all the actions applicable to the existing proposition layer to the new action layer. This will obviously cause redundancies in the action layer. A redundancy is an action that could not be expanded or whose follow-up actions could not be expanded to the final layer. To avoid this situation, we filter the candidate actions using a DFS(deep-first-search)-like backtracking to analyze the actions' reachability before adding to the new layer. This approach has been proved usable to guarantee all the involved actions are expandable.

Algorithm 2 demonstrates the detailed process of expanding the planning graph. The input includes the unprocessed applicable action set *Candidates*, the action set A, the initial set s_0, the goal set g, and a positive integer to label the recursive level *num*. Each time it gets into a new recursion, *num* will add 1. As is shown in line 6-7, *num* is an indicator identifying if it needs to scan all actions in the candidate set, since it is necessary to analyze all candidate actions' reachability in the first level, while the internal recursive levels only have to find one existing path to arrive the goal set. In line 11-14, a subset of the candidate is including all applicable actions for updated initial state s_0' is generated to get into the internal recursions. As is shown in line 15-19, once the goal set arrives, it will backtrack to the first level and keep this action in the candidate set, otherwise, the candidate will be removed. This backtracking approach is a DFS-like process which could not only analyze reachability but update the heuristics at the same time.

D. HEURISTICS OF THE PLANNING GRAPH

Our approach of extracting heuristics is based on HSP-R. It casts planning as search through the regression space of world states. The heuristic can be seen as estimating the proposition distance required to reach a state (either from the goal state or the initial state. Given a goal state g, we use $h(g)$ to denote the estimating cost from the initial state s_0 to s. The value of $h(g)$ could be calculated via an estimating function Δ representing the minimum cost from s_0 to g, which could be denoted as:

$$h(g) = \Delta(s_0, g) \tag{5}$$

$\Delta(s_0, g)$ could be determined by the minimum cost of each proposition p in the goal state g, denoted as $\Delta(s_0, p)$. The value of Δ could be calculated following the equation below:

$$\begin{cases} \Delta(s_0, p) = 0 & p \in s \\ \Delta(s_0, p) = \infty & \forall a \in A, \quad p \notin effect^+(a),\ p \notin s \\ \Delta(s_0, g) = 0 & g \subseteq s \\ Otherwise: & \\ if \quad p \in effect^+(a): & \\ \Delta(s, p) = \min\{\Delta(s_0, p), cost(a) + \max_{q \in pre(a)} \Delta(s_0, q)\} & \\ \Delta(s, g) = \max_{p \in g} \Delta(s_0, p) & \end{cases} \tag{6}$$

Equation 5 has been proved admissible, therefore, it will be used as heuristic function in the searching phase. As is

Algorithm 2 Expanding the Planning Graph and Extracting Heuristics

Require: *Candidates* is a set of services to expand, A, s_0, g, *num* is a label of the recursive level
Ensure: reachability

1: **function** EXPAND(*Candidates*, A, s_0, g, *num*)
2: $SubCan \leftarrow \emptyset$, $num \leftarrow num + 1$
3: **for** $a \in Candidates$ **do**
4: $s_0' \leftarrow s_0 \cup effect^+(a)$
5: **if** $g \subseteq s_0'$ **then** $s_0 \leftarrow s_0'$, $UpdateDelta(a)$
6: **if** $num \neq 1$ **then return** true
7: **end if**
8: **else**
9: **if** $s_0' = s_0$ **then** delete a from *Candiadates*
10: **end if**
11: **for** $a' \in A$ **do**
12: **if** $pre(a') \subseteq s_0'$ **then** add a' to *SubCan*
13: **end if**
14: **end for**
15: **if** *SubCan* $\neq \emptyset$ and
16: $Expand(SubCan, A, s_0', g, num)$ **then**
17: $s_0 \leftarrow s_0'$, $UpdateDelta(a)$
18: **else**delete a from *Candiadates*
19: **end if**
20: **end if**
21: **end for**
22: **if** *Candidates* $\neq \emptyset$ **then return** true
23: **end if**
24: **return** false
25: **end function**

shown in algorithm 2, we update the heuristic values with the function $UpdateDelta(a)$ in line 7 and 23 when finding an reachable action a. For each proposition in $effect^+(a)$, $UpdateDelta(a)$ updates the value of $\Delta(s_0, p)$ according to equation 5.

Following the rules above, after constructing the planning graph, we will obtain a complete estimating distance function $h(S)$ from initial state s_0 to all the reachable state S. In the following phase of searching solution, it will be used as a heuristic function in the backward A* searching process.

E. GRAPH CONVERSION

In this section, we will describe the process of converting the planning graph to an directed path generation(DPG) graph in detail.

Definition 8 (Directed Path Generation Graph): A directed path generation graph could be denoted as a tuple $G = (V, E)$. *V represents a set of graph vertexes* $\{v_{start}, v_{target}, v_1, v_2, \ldots, v_n\}$. *Each ordinary vertex* v_i *includes a set of parallel composite actions* $\{a_1, a_2, \ldots, a_n\}$. *Specially,* v_{start} *and* v_{target} *are two special vertexes which effectively records initial state* s_0 *and goal set* g. *E represents a set of directed edges* $\{e_1, e_2, \ldots, e_n\}$. *An edge* $e_i = \langle v_j, v_k \rangle$ *consists of two different vertexes from vertex* v_j *to vertex* v_k.

Algorithm 3 Planning Graph Conversion

Require: $G_p = \langle P_0, A_1, P_1, H_1, \ldots, A_n, P_n, H_n \rangle$, s_0, g, k
Ensure: $G = (V, E)$

1: **function** GRAPHCONVERSION(G_p, s_0, g)
2: DPG $G \leftarrow \emptyset$, $v_{start} \leftarrow s_0$, v_{target}, $\leftarrow g$
3: $i \leftarrow n$, $T_i \leftarrow Combination(g, H_i, k)$,
4: **while** $i \neq 1$ **do**
5: $i \leftarrow i - 1$, $T_{i-1} \leftarrow \emptyset$
6: **for** each $t \in T_i$ **do**
7: $w \leftarrow \{tag | t.Level = i\}$
8: **if** $w \neq \emptyset$ **then**
9: $T' \leftarrow GetParents(P_{i-1}, H_{i-1}, w, t, k)$
10: $T_{i-1} \leftarrow T_{i-1} \cup T'$
11: $V \leftarrow GetVertexes(T')$
12: $UpdateGraph(G, V)$
13: **else**
14: $T_{i-1} \leftarrow T_{i-1} \cup t$
15: **end if**
16: **end for**
17: **end while**
18: add v_{start} and v_{target} to G
19: **return** G
20: **end function**

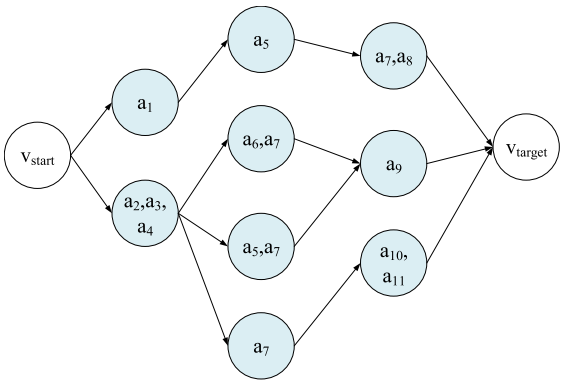

FIGURE 2. An example of the DPG graph.

As is shown in algorithm 3, the input is the planning graph G_p with the extended layer H, and the output is a DPG graph G. Line 2-3 initialize the essential variables., where G is the DPG graph to return, v_{start} and v_{target} is two terminal vertexes, i is a counter from the last layer to the first layer, and T_i is set of the action tags corresponding to the vertexes to generate. The function $Combination(g, H_i, k)$ picks action tags in H_i and divides them into different sets according to the proposition p their corresponding actions' positive effective include. Pick one action tag from each set and combine them as a tag vertex t, then return top k sorted by heuristic values of all the possible results as a tag vertex set T_i in case of the vertex explosion. The function $GetVertexes()$ extracts the action part from T_i and receive a vertex set. Line 4-19 extract vertex from the last layer to the first layer. Line 6-18 construct vertexes and edges for each tag vertex t. Line 7 finds the tag vertexes including actions in

the current layer. If these actions exist, which means the goal state could be reached via actions in current action layer, in line 8-12, the function *GetParents* constructs a new action tag set T' from the previous layer and keeps top k tags in the set, then assigns them to the tag set T_{i-1} to be used in the next circulation. The function $UpdateGraph(G, V)$ add the new vertex set and establish new edges to graph G. Otherwise, in line 13-16, the current tag is assigned to the next tag set T_{i-1}. After traversing the whole layers, the special terminal vertexes,v_{start} and v_{target}, are installed to the graph.

F. BACKWARD A* ALGORITHM

After the DPG graph is generated, a search over it must be performed. Unlikely to ordinary search approach, As is mentioned before, our search approach is based on the heuristic search strategy, A* algorithm. This algorithm takes full advantage of our extracted heuristics from the planning graph to accelerate the searching process.

The search algorithm will traverse the graph backward, from the target vertex v_{target} to the start vertex v_{start}. Searching backward is given to the features of our extracted heuristics, that the estimating distances are based on the start point of the initial state, therefore, search forward requires extra extraction calculation while searching backward could greatly decrease the calculating complexity and the searching space.

In order to perform the A* search, a set of principal concepts will be explained. The backward A* algorithm is based on classical concepts. The classical A* algorithm depends on the evaluation function $f(n) = g(n)+h(n)$, where n represents the next node on the path, $g(n)$ represents the cost of the path from the start node to n, and $h(n)$ is a heuristic function that estimates the cost of the cheapest path from n to the goal. An admissible heuristic function $h(n)$, $h(n) \leq h^*(n)$ for each node n, where $h^*(n)$ represents the actual cost from n to the goal, guarantee to find an optimal path to the goal.

In order to match the backward traverse, a backtrack function and relative concepts are imported to calculate the previous state. Given a goal g and an action a, we call action a is related to g, if and only if $g \cap effective^+(a) \neq \emptyset$. The previous state could be denoted as:

$$\gamma^{-1}(g, a) = (g - effective^+(a)) \cup pre(a) \tag{7}$$

Therefore, for a set of related actions A, the previous state could be denoted as:

$$\Gamma^{-1}(g, A) = (g - \bigcup_{a \in A} effective^+(a)) \cup \bigcup_{a \in A} pre(a) \tag{8}$$

Equation could be used to calculate the previous state of a vertex. Furthermore, the heuristic function is derived according to equation 5 which has been proved admissible. Algorithm 4 shows the process of searching the optimal solution with the backward A* algorithm. The essential input parameters include a DPG graph and its start and target vertexes. The output is an optimal solution. Line 2-4 is initializing the variables, where Π represents the solution,

Algorithm 4 Backward A* Search

Require: $G = (V, E)$, v_{start}, v_{target}
Ensure: $\Pi = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$
1: **function** BACKWARDASTAR(G, v_{start}, v_{target})
2: $\Pi \leftarrow \emptyset$, $Open \leftarrow \emptyset$, $Close \leftarrow \emptyset$, $CameFrom \leftarrow \emptyset$
3: $Open \leftarrow Open \cup \{v_{target}\}$,
4: $gs \leftarrow v_{target}$, $g_{neighbor} \leftarrow \infty$
5: **while** $Open \neq \emptyset$ **do**
6: $Current = \min_{i \in Open} f(\Gamma^{-1}(gs, i))$
7: $gs \leftarrow \Gamma^{-1}(gs, Current)$
8: **if** $Current = Start$ **then**
9: $PathConstruct(\Pi, CameFrom, Current)$
10: **return** Π
11: **end if**
12: add *Current* to *Close*
13: remove *Current* from *Open*
14: **for** each predecessor *pre* of *Current* **do**
15: **if** $pre \in Close$ **then continue**
16: **end if**
17: $g_{temp} \leftarrow g(gs) + cost(pre)$
18: **if** $pre \notin Open$ **then** add *pre* to *Open*
19: **else**
20: **if** $g_{temp} \geq g_{neighbor}$ **then continue**
21: **end if**
22: **end if**
23: $CameFrom(pre) \leftarrow Current$
24: $g_{neighbor} \leftarrow g_{temp}$
25: $f_{neighbor} \leftarrow g_{neighbor} + h(gs)$
26: **end for**
27: **end while**
28: **return** false
29: **end function**

Open and *Close* are two set recording vertexes to be visited and visited vertexes, *CameFrom* is a hash map recording the path that has been chosen, *gs* is the current state. Line 5-28 is the main part of the searching process. First, choose the minimum evaluation function values in *Open* as the current vertex *Current*. Line 8-11 identify if existing the circulation. If the start vertex has arrived, the function $PathConstruct(\Pi, CameFrom, Current)$ will traverse the whole set and generate the solution. Line 12-13 updates the *Open* and *Close* set. Line 14-26 traverse the predecessors of current vertex, then choose the vertex with the minimum evaluation function and update the path track map *CameFrom*, until the set *Open* stops adding new vertexes.

V. EXPERIMENT RESULTS

Our experiments were carried out on a PC with Intel(R) Core TM i3-8100, with 3.6GHz processor and 16GB RAM, under Arch Linux kernel 4.19.66-1-Manjaro and Java SE build 1.8.0_171-b11.

We applied our algorithm on a well-known benchmark of Web Services Challenge 2009(WSC-2009) [40]. WSC-2009 dataset provides five test sets, effectively

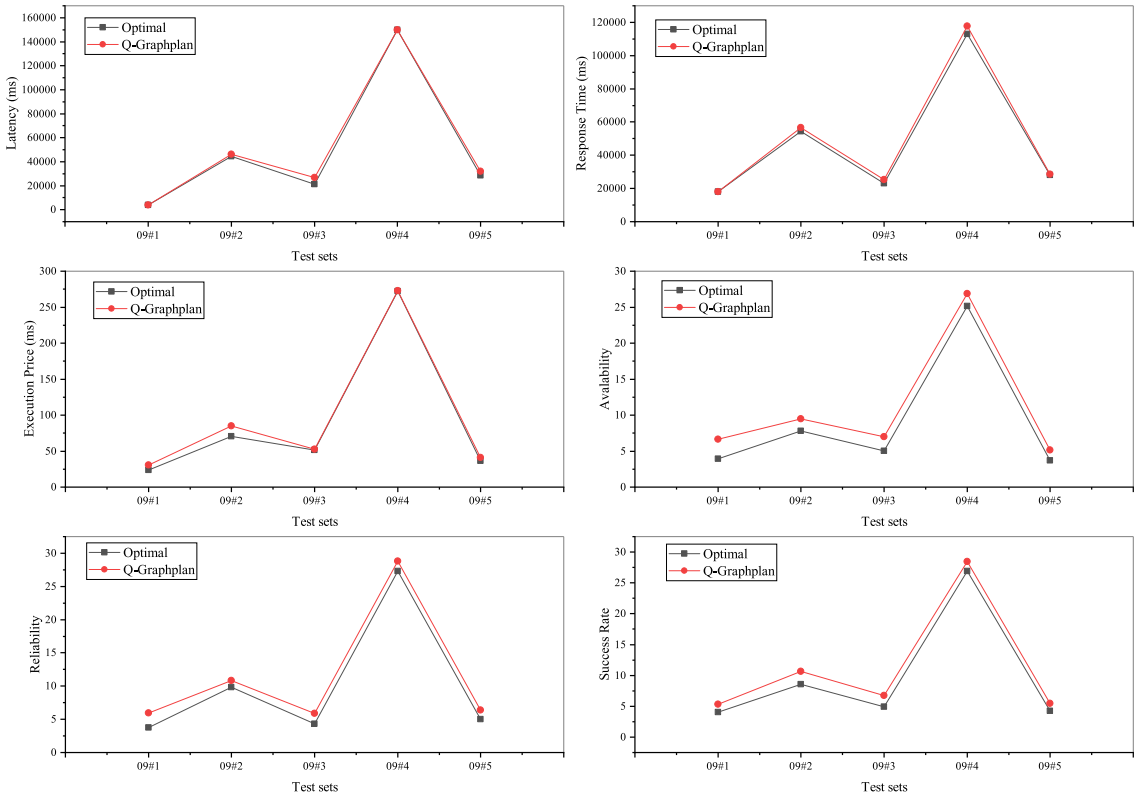

FIGURE 3. The optimality of Q-Graphplan's composition solution.

containing around 500, 4000, 8000 and 15000 services with respectively more than 1500, 10000, 15000 and 25000 data. Each test set consists of four files: a WSDL file, describing services of the registry by their inputs/outputs, hierarchically organized in terms of concepts using an ontology recorded in an OWL file, a WSLA file, storing the QoS (response time and throughput) criteria values of each service, and an XML file storing the inputs and outputs of the query associated with the service registry. Since the dataset WSC-2009 only includes data of two QoS parameters, response time and throughput, we use the monitor data from [38] which have monitored six QoS criteria (response time, execution price, latency, availability, successful rate, and reliability) from the real website data in this experiment.

At first we evaluated the optimality of the solution. In this experiment, we calculated the optimal values for the test sets in the WSC-2009 data set. Then, we set each QoS to the same weight and find solutions of test sets with Q-Graphplan. We compared the QoS values of Q-Graphplan's solution and the optimal values of each criterion. In figure 3, the red line represents the QoS values of the Q-Graphplan's composition solution and the black line represents the optimal values using single-objective optimization. To make the data more readable, we used the scaled value to demonstrate the results of the availability, the reliability and the success rate. The result showed that the Q-Graphplan could find a solution whose each QoS criterion have an approximation of the optimal value.

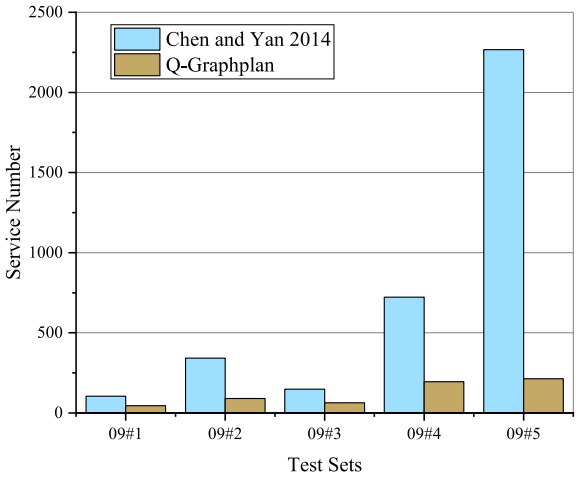

FIGURE 4. The number of services in the planning graph.

Second we evaluated the effectiveness of our redundancy-free algorithm of constructing the planning graph. We compared the number of service in the planning graph with the datasets in WSC-2009. Figure 4 shows the comparison of the service number between [35]'s algorithm that uses the classical algorithm to construct the planning graph and Q-Graphplan's redundancy-free algorithm. The result indicates that Q-Graphplan effectively avoids generating redundancies in constructing the planning graph. For the test set 1 to 4, Q-Graphplan's service number decrease to the

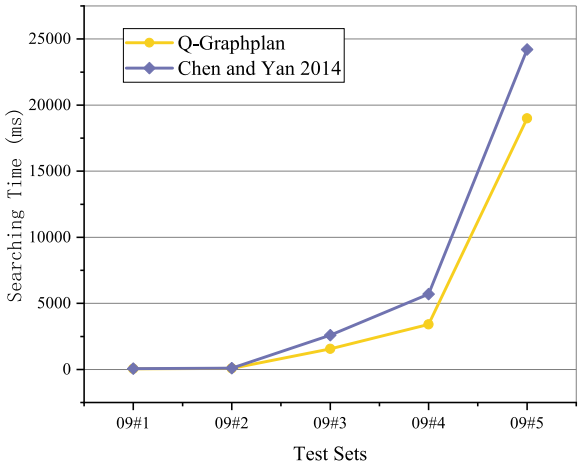

FIGURE 5. The search time of Q-Graphplan and Chen and Yan 2014.

nearly a half of the [35] in average. For the test set 5, the service number of the Q-Graphplan is only approximately one-tenth of the baseline.

Finally we evaluated the efficiency of Q-Graphplan. Our approach was compared to the algorithm in [35]. We tested the searching time on each test 100 times using two algorithms and calculate the average searching time. The result is depicted in figure 5. Due to the scope of the test set 1 and the test set 2 are relatively small, therefore the performance of two algorithms are almost same. However, with the increase the test sets' scope, our algorithm performs better. In average, our approach's searching time reduces approximately 25% of [35] according to figure 5.

VI. CONCLUSION AND FUTURE WORK

This paper proposes an approach, Q-Graphplan, to solve the QoS-aware ASC problem with multiple QoS criteria constraints using an extended version of the classical graphplan and backward A* search algorithm. We derive the heuristic from the planning graph which is used to accelerate the searching process. Experimental results on benchmarks show the effectiveness of our proposal. We believe that our work will open up a lot of new research directions in the general paradigm of automatic service composition with AI planning algorithm.

In the future, we will be focused on optimizing our algorithm to lower the complexity and increasing the efficiency. We wish to propose a more effective heuristic function of extracting the heuristics. The present problem is that out experiments are conducted under the ideal condition ignoring the dynamic states of the service set and user's requests. We wish to improve our algorithms to adapt to the actual web service environment.

REFERENCES

[1] N. Chen, N. Cardozo, and S. Clarke, "Goal–driven service composition in mobile and pervasive computing," *IEEE Trans. Serv. Comput.*, vol. 11, no. 1, pp. 49–62, Jan. 2018.

[2] J. El Hadad, M. Manouvrier, and M. Rukoz, "TQoS: Transactional and QoS–aware selection algorithm for automatic Web service composition," *IEEE Trans. Serv. Comput.*, vol. 3, no. 1, pp. 73–85, Jan. 2010.

[3] D. Wu, H. Shi, H. Wang, R. Wang, and H. Fang, "A feature–based learning system for Internet of Things applications," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1928–1937, Apr. 2019.

[4] D. Wu, L. Deng, H. Wang, K. Liu, and R. Wang, "Similarity aware safety multimedia data transmission mechanism for Internet of vehicles," *Future Gener. Comput. Syst.*, vol. 99, pp. 609–623, Oct. 2019.

[5] Z. Li, Y. Jiang, Y. Gao, L. Sang, and D. Yang, "On buffer–constrained throughput of a wireless–powered communication system," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 2, pp. 283–297, Feb. 2019.

[6] D. Wu, Q. Liu, H. Wang, D. Wu, and R. Wang, "Socially aware energy–efficient mobile edge collaboration for video distribution," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2197–2209, Oct. 2017.

[7] R. Wang, J. Yan, D. Wu, H. Wang, and Q. Yang, "Knowledge–centric edge computing based on virtualized D2D communication systems," *IEEE Commun. Mag.*, vol. 56, no. 5, pp. 32–38, May 2018.

[8] D. Zhovtobryukh, "A Petri net-based approach for automated goal–driven Web service composition," *Simulation*, vol. 83, no. 1, pp. 33–63, Jan. 2007.

[9] P. Rodriguez-Mier, M. Mucientes, and M. Lama, "Hybrid optimization algorithm for large–scale QoS–aware service composition," *IEEE Trans. Serv. Comput.*, vol. 10, no. 4, pp. 547–559, Jul. 2017.

[10] C. Lv, W. Jiang, S. Hu, J. Wang, G. Lu, and Z. Liu, "Efficient dynamic evolution of service composition," *IEEE Trans. Serv. Comput.*, vol. 11, no. 4, pp. 630–643, Jul. 2018.

[11] Y. Chen, J. Huang, C. Lin, and J. Hu, "A partial selection methodology for efficient QoS–aware service composition," *IEEE Trans. Serv. Comput.*, vol. 8, no. 3, pp. 384–397, May 2015.

[12] P. Bartalos and M. Bieliková, "Automatic dynamic Web service composition: A survey and problem formalization," *Comput. Inform.*, vol. 30, no. 4, pp. 793–827, 2011.

[13] Y. Chen, J. Huang, C. Lin, and X. Shen, "Multi–objective service composition with QoS dependencies," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 537–552, Apr. 2019.

[14] S. Chattopadhyay, A. Banerjee, and N. Banerjee, "A fast and scalable mechanism for Web service composition," *ACM Trans. Web*, vol. 11, no. 4, pp. 1–36, Aug. 2017.

[15] P. Rodriguez-Mier, M. Mucientes, and M. Lama, "Automatic Web service composition with a heuristic–based search algorithm," in *Proc. IEEE Int. Conf. Web Services (ICWS)*. Washington, DC, USA: IEEE Computer Society, Jul. 2011, pp. 81–88.

[16] I. Guidara, N. Guermouche, T. Chaari, S. Tazi, and M. Jmaiel, "Heuristic based time–aware service selection approach," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, J. A. Miller and H. Zhu, Eds. New York, NY, USA: IEEE Computer Society, Jun./Jul. 2015, pp. 65–72.

[17] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IIEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369–384, Jun. 2007.

[18] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "Qos-aware service composition in dynamic service oriented environments," in *Proc. 10th Int. Middleware Conf. Middleware ACM/IFIP/USENIX*, in Lecture Notes in Computer Science, vol. 5896, J. Bacon and B. F. Cooper, Eds. Urbana, IL, USA: Springer, Nov./Dec. 2009, pp. 123–142.

[19] Z. Zhang, S. Zheng, W. Li, Y. Tan, Z. Wu, and W. Tan, "Genetic algorithm for context–aware service composition based on context space model," in *Proc. 20th Int. Conf. Web Services*. Santa Clara, CA, USA: IEEE Computer Society, Jun./Jul. 2013, pp. 605–606.

[20] F. Wagner, A. Klein, B. Klopper, F. Ishikawa, and S. Honiden, "Multi-objective service composition with time- and input–dependent QoS," in *Proc. IEEE 19th Int. Conf. Web Services*, C. A. Goble, P. P. Chen, and J. Zhang, Eds. Honolulu, HI, USA: IEEE Computer Society, Jun. 2012, pp. 234–241.

[21] S.-C. Oh, D. Lee, and S. R. Kumara, "Effective Web service composition in diverse and large–scale service networks," *IEEE Trans. Serv. Comput.*, vol. 1, no. 1, pp. 15–32, Jan. 2008.

[22] S.-C. Oh, D. Lee, and S. R. Kumara, "Web service planner (WSPR): An effective and scalable Web service composition algorithm," *Int. J. Web Services Res.*, vol. 4, no. 1, pp. 1–22, Jan. 2007.

[23] E. Khanfir, R. B. Djmeaa, and I. Amous, "Self–adaptive goal–driven Web service composition based on context and QoS," in *Proc. 14th IEEE Int. Conf. e-Bus. Eng. (ICEBE)*, O. Hussain, L. Jiang, X. Fei, C. Lan, and K. Chao, Eds. Shanghai, China: IEEE Computer Society, Nov. 2017, pp. 201–207.

[24] S. Song and S.-W. Lee, "A goal-driven approach for adaptive service composition using planning," *Math. Comput. Model.*, vol. 58, nos. 1–2, pp. 261–273, Jul. 2013.

[25] S. Kambhampati and R. S. Nigenda, "Distance-based goal-ordering heuristics for graphplan," in *Proc. 5th Int. Conf. Artif. Intell. Planning Syst.*, S. A. Chien, S. Kambhampati, and C. A. Knoblock, Eds. Breckenridge, CO, USA: AAAI, Apr. 2000, pp. 315–322.
[26] J. Hoffmann, P. Bertoli, and M. Pistore, "Web service composition as planning, revisited: In between background theories and initial state uncertainty," in *Proc. 22nd AAAI Conf. Artif. Intell.* Vancouver, BC, Canada: AAAI Press, Jul. 2007, pp. 1013–1018.
[27] F. Lécué and A. Delteil, "Making the difference in semantic Web service composition," in *Proc. 22nd AAAI Conf. Artif. Intell.* Vancouver, BC, Canada: AAAI Press, Jul. 2007, pp. 1383–1388.
[28] S. Aine, P. P. Chakrabarti, and R. Kumar, "AWA*—A window constrained anytime heuristic search algorithm," in *Proc. 20th Int. Joint Conf. Artif. Intell. (IJCAI)*, M. M. Veloso, Ed., Hyderabad, India, Jan. 2007, pp. 2250–2255.
[29] M. Phillips, V. Narayanan, S. Aine, and M. Likhachev, "Efficient search with an ensemble of heuristics," in *Proc. 24th Int. Joint Conf. Artif. Intell. (IJCAI)*, Q. Yang and M. J. Wooldridge, Eds. Buenos Aires, Argentina: AAAI Press, Jul. 2015, pp. 784–791.
[30] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for Web services composition," *IIEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.
[31] R. Liu, Z. Wang, and X. Xu, "Parameter tuning for ABC–based service composition with end-to-end QoS constraints," in *Proc. IEEE Int. Conf. Web Services (ICWS)*. Anchorage, AK, USA: IEEE Computer Society, Jun./Jul. 2014, pp. 590–597.
[32] 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 Trans. Serv. Comput.*, vol. 7, no. 1, pp. 18–31, Jan. 2014.
[33] A. Blum and M. L. Furst, "Fast planning through planning graph analysis," in *Proc. 14th Int. Joint Conf. Artif. Intell. (IJCAI)*, vol. 2. Montreal, QC, Canada: Morgan Kaufmann, Aug. 1995, pp. 1636–1642.
[34] S. Kambhampati, E. Parker, and E. Lambrecht, "Understanding and extending graphplan," in *Proc. Recent Adv. AI Planning, 4th Eur. Conf. Planning (ECP)*, vol. 1348, in Lecture Notes in Computer Science, S. Steel and R. Alami, Eds. Toulouse, France: Springer, Sep. 1997, pp. 260–272.
[35] M. Chen and Y. Yan, "QoS-aware service composition over graphplan through graph reachability," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*. Anchorage, AK, USA: IEEE Computer Society, Jun./Jul. 2014, pp. 544–551.
[36] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
[37] X. Nguyen and S. Kambhampati, "Extracting effective and admissible state space heuristics from the planning graph," in *Proc. 17th Nat. Conf. Artif. Intell. 25th Conf. Innov. Appl. Artif. Intell.*, H. A. Kautz and B. W. Porter, Eds. Austin, TX, USA: AAAI Press, Jul./Aug. 2000, pp. 798–805.
[38] P. Rodriguez-Mier, M. Mucientes, J. C. Vidal, and M. Lama, "An optimal and complete algorithm for automatic Web service composition," *Int. J. Web Services Res.*, vol. 9, no. 2, pp. 1–20, Apr. 2012.
[39] Y. Yan and M. Chen, "Anytime QoS-aware service composition over the GraphPlan," *Service Oriented Comput. Appl.*, vol. 9, no. 1, pp. 1–19, Mar. 2015.
[40] S. Kona, A. Bansal, M. B. Blake, S. Bleul, and T. Weise, "WSC–2009: A quality of service–oriented Web services challenge," in *Proc. IEEE Conf. Commerce Enterprise Comput. (CEC)*, B. Hofreiter and H. Werthner, Eds. Vienna, Austria: IEEE Computer Society, Jul. 2009, pp. 487–490.

ZHAONING WANG is currently pursuing the Ph.D. degree in computer science and technology with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include service composition, pervasive computing, and mobile service.

BO CHENG received the Ph.D. degree in computer science from the University of Electronics Science and Technology of China, in 2006. He is currently a Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests include the Internet of Things, the mobile Internet, and services computing.

WENKAI ZHANG is currently pursuing the master's degree in computer science and technology with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include service computing, mobile service, and AI.

JUNLIANG CHEN is currently a Professor with the Beijing University of Posts and Telecommunications. His research interest is in the area of service creation technology. He was elected as a member of the Chinese Academy of Science, in 1991, and the Chinese Academy of Engineering, in 1994.