KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS VOL. 5, NO. 7, July 2011

A Global Graph-based Approach for Transaction and QoS-aware Service Composition

Hai Liu[1], Zibin Zheng[2], Weimin Zhang[1] and Kaijun Ren[1]
[1] School of Computer, National University of Defense Technology
Chang Sha, 410073 - China
[e-mail: {hailiu, renkaijun}@nudt.edu.cn, wmzhang104@163.com]
[2] Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Hong Kong, - China
[e-mail: zbzheng@cse.cuhk.edu.hk]
[*]Corresponding author: Hai Liu

Received March 7, 2011; revised May 7, 2011; accepted June 30, 2011; published July 28, 2011

Abstract

In Web Service Composition (WSC) area, services selection aims at selecting an appropriate candidate from a set of functionally-equivalent services to execute the function of each task in an abstract WSC according to their different QoS values. In despite of many related works, few of previous studies consider transactional constraints in QoS-aware WSC, which guarantee reliable execution of Composite Web Service (CWS) that is composed by a number of unpredictable web services. In this paper, we propose a novel global selection-optimal approach in WSC by considering both transactional constraints and end-to-end QoS constraints. With this approach, we firstly identify building rules and the reduction method to build layer-based Directed Acyclic Graph (DAG) model which can model transactional relationships among candidate services. As such, the problem of solving global optimal QoS utility with transactional constraints in WSC can be regarded as a problem of solving single-source shortest path in DAG. After that, we present Graph-building algorithms and an optimal selection algorithm to explain the specific execution procedures. Finally, comprehensive experiments are conducted based on a real-world web service QoS dataset. The experimental results show that our approach has better performance over other competing selection approaches on success ratio and efficiency.

Keywords: Web service composition, QoS-aware selection, end-to-end constraints, transaction, DAG

This research was supported by the Chinese 863 project under Grant No. 2006AA01A123, No. 863-2010AA012404 and the Foundation supported by National Nature Science Foundation of China under Grant No. 60903042, No. 60736013. We express our thanks to Dr. Joyce EL Haddad for comments on our manuscript.

DOI: 10.3837/tiis.2011.07.003

1. Introduction

With the development of Internet and distributed computing technology, Service Oriented Architecture (SOA) becomes more and more prevalent. SOA integrates software components in Internet provided by different organizations. Web service is a key technical for building SOA applications [1][2]. With the increasing number of web services in the Internet, Web Service Composition (WSC) is becoming an important research problem, attracting attentions from both academia and industrial areas.

In the process of WSC, there are two key steps, i.e., abstract WSC and concrete WSC. Abstract WSC, also named functional WSC, solves problem of functional aspect for WSC. Concrete WSC, also called QoS-aware WSC, addresses the problem of optimal utility under non-functional properties. In this paper, we focus on addressing the transactional issues in concrete WSC which comprises the second indispensable step in WSC. Up to now, A lot of research work considering concrete WSC has been conducted basing on different QoS properties such as response time, execution cost, reliability, availability [3][4][5][6][7]. The key idea of the previous work is to select proper candidate for each activity in the composite web service to obtain the best QoS utility and satisfy end-to-end user QoS constraints [8]. In reality, the web service selection problem can be mapped into Multidimensional Multiple-choice Knapsack Problem (MMKP), which has been proved as a NP hard problem [9]. Hence, the time complexity of this kind of problem is exponential. To solve the problem within polynomial time, several approximation algorithms have been proposed [3][4][5][6][7][9][10][11].

Although many methods have been proposed for QoS-aware WSC, few consider the transactional constraints. Transactional characteristics ensure consistent and reliable execution of WSC. In particular, considering both transactional characteristics as well as optimal QoS utility for QoS-aware WSC is an extremely challenging research problem [12]. Due to the inherent peculiarities of web services (e.g., loosely coupled, autonomy, heterogeneous, etc.), transactional properties of the candidate web services may differ from each other. Thus, QoS-aware WSC regarding transactional properties becomes very complicated, since all transactional combinations of constituted web services are hard to satisfy the global transactional constraints. Authors in [12][13] proposed a transactional and QoS-aware services selection algorithm, which employed automation to model transactional constraints. In this selection algorithm, they firstly considered the globally transactional constraints, and then embedded a local QoS-optimal selection approach to solve the aspect of QoS. Although this algorithm considered transactional constraints, the final result on the global QoS utility may be influenced due to the local approach used in QoS aspect. How to improve the comprehensive QoS utility for QoS-aware WSC subject to globally transactional constraints is still an unsolved problem.

To address the aforementioned problem, we use a layer-based DAG model, called actual Service Candidates Graph (SCG), to define a solution for QoS-aware WSC with transactional constraints. In our model, we firstly propose building rules and the reduction method based on transactional theory of atomicity and consistency. Meanwhile, the correctness satisfying transactional constraints with these rules are proved. After that, we present a building algorithm for actual SCG. By our approach, solving optimal QoS utility with global constraints (include both end-to-end QoS constraints and transactional constraints) is translated to the problem of solving single-source shortest paths from the first activity to the last activity. For this, we propose a selection algorithm based on the MCOP algorithm [6]. Differing from the traditional MCOP algorithm, our method (1) concentrates addressing constraints not only on end-to-end QoS criteria but also on globally transactional aspect for a CWS, and (2) proposes a new cost function as a filtering standard applied to our relaxing function. Extensive experiments are conducted basing on a real-world web service dataset. The experimental results show that our approach achieves better performance on global QoS utility over other typical approaches. Meanwhile, our approach has well extensibility.

The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 introduces the composite patterns, the method of computing global QoS and utility. Section 4 proposes building rules and a reduction method for building graph models, and proves their correctness. Section 5 presents building algorithms and the optimal selection algorithm. Section 6 gives comprehensive experiments to demonstrate benefits of our approach. Finally, Section 7 concludes the paper and presents future work.

2. Related Work

2.1 QoS-aware Web Services Composition

Many research efforts have been carried out in QoS-aware web services composition . Authors in [7] proposed a method of computing QoS utility for a CWS. Liangzhao Zeng et al. [3][10] proposed several classic selection algorithms using integer programming, which are based on global allocation to services. These methods can select the best candidate services for each task in an abstract CWS according to the global QoS utility. However, the intrinsic time complexity of the integer programming algorithms are exponential. Florien Rosenberg et al. [5] proposed an global QoS optimization approach based on a constraint planning method, considering both feature constraints and end-to end QoS constraints. The Constraint Satisfaction Problem (CSP) was used to define local QoS constraints for each feature and global QoS constraints. However, this way only considered QoS constraints with quantitative features. While those qualitative constraints such as transaction of a CWS were ignored. Authors in [14] presented the way of QoS-aware WSC respecting incomplete user preferences. They made use of historical user information to amend user incomplete preferences, and then improved QoS-aware web services selection. Mohammad Alrifai et al. in [4][15] proposed two QoS-aware services selection approaches. In [4], in order to solve the problem of QoS optimal utility under each QoS dimension constraint, authors firstly translated global QoS constraints of CWS into local QoS constraints with integer programming. Afterwards, a distributed local selecting method had been applied to select the best web services satisfying these local constraints. The approach in [15] improved the approach in [4] by using skyline to effectively and efficiently select services for composition, reducing the number of candidate service to be considered. Tao Yu et al. [6][11] proposed two near-optimal services selection methods (i.e., the combinatorial model and the graph model) with end-to-end QoS constraints. The combinatorial model defines the problem as a Multiple choices Multiple dimensions *0-1* Knapsacks Problem (MMKP). The graph model defines the problem as a Multiple Constraints Optimal Path (MCOP) problem. Efficient heuristics algorithms for service processes of different composition structures are presented. Though these two methods could obtain a well result within polynomial time, transactional properties were not considered. Our approach extends the MCOP graph model to consider both QoS and qualitative properties of transactional nature for a CWS.

2.2 Transactional aspect of Web Service

In order to ensure reliability and consistency of CWS, transactional properties of CWS have attracted great attention. Due to the particular characteristics of web service transaction (e.g., long-running, heterogeneous, distributed and autonomic nature, etc.), traditional ACID transaction model can hardly be adapted for the web service environment. With development of service oriented computing, several transactional specifications for web services have been proposed, including the Business Transaction Protocol (BTP) [16], the Web Services Coordination (WS-C) [17], Web Services Transaction (WS-Tx) specifications [18][19], and Web Services Composite Application Framework (WS-CAF) [20]. Although these specifications are comprehensive, very little attention has been paid to consider the problem of transaction-based and QoS-aware WSC. In recent years, lots of work dedicate to studying transactional consistency of the whole process based on transactional properties of its components. In [21], authors proposed a unified model of consistency and atomicity theories for concurrency control of transactional processes, which consists of process-serializability, process-recoverability, process-reducibility, process correct termination. Authors in [22] proposed a web service selection framework to ensure failure atomicity of a CWS by considering transactional properties of candidate web services. In the framework, Accepted Termination States (ATS) were introduced to express the required failure atomicity for a CWS. Although transactional WSC had been implemented by this framework, global QoS optimization for CWS had not been considered. Authors in [23], evaluated the global QoS value of a CWS with transactional operators. However, they only analyzed the transactional effects on QoS for a CWS, without ensuring the optimal QoS requirement. Works [12][13] proposed a selection algorithm that satisfied users' preferences. The users' preferences are expressed as weights over QoS criteria and expressed as two risk levels to define semantically the transactional requirements. In this selection algorithm, QoS-aware selection process is embedded within the process of transactional service selection. Thus, the set of potential web services for each workflow activity is restricted by the transactional constraints for those selected previously. Although work [12][13] considers both transactional and QoS-aware WSC, it uses a local QoS optimization selection algorithm for solving the problem. Different from work [12][13], our approach employs a graph model to address the transactional aspect, and then uses a globally optimal selection approach for the QoS aspect.

3. The QoS Computation

As discussed in [7][24][25], in a CWS, atomic web services could be connected by composite patterns including sequence pattern, parallel pattern, conditional pattern and loop pattern. Since a loop pattern can be converted into a sequence pattern by unfolding loop [26], we will not discuss it in this paper. In order to compute the global QoS for a concrete CWS, we firstly employ two concepts, namely sequence path and route path which appeared in [6][10]. A sequence path denotes a path from the start task to the end task including only one branch in no matter conditional or parallel patterns. In comparison, a route path illustrates a path from the start task to the end task in an abstract CWS involving only one branch in conditional patterns but all branches in parallel patterns. According to these definitions, we use $rpl_i = (s_1, s_2, ..., s_n)$ to denote one route plan for the route path rpt where s_k denotes the web service assigned to the task t_k in rpt_i. Obviously, rpt may include multi-sequence plans if parallel patterns exist in rpt. Similar to work [12], we firstly consider four QoS criteria for each candidate web service: (1) Execution Cost (q_k^1); (2) Response Time (q_k^2); (3) Availability (q_k^3); (4)

Reliability (q_k^4). Then, for any candidate s_k of task t_k in an abstract CWS cs , we can associate s_k to a QoS vector $\mathrm{q}(s_k) = [q^1(s_k), q^2(s_k), q^3(s_k), q^4(s_k)]$. For the reason that different QoS criteria have different quantitative metric, we use Eq. (1) to normalize QoS criterion α with positive nature where $\max(q^\alpha(t_k))$ and $\min(q^\alpha(t_k))$ respectively denote the maximum and minimum values for all candidates of t_i . Analogously, Eq. (2) is used to normalize QoS criterion β with negative nature [27].

Table 1. QoS aggregation functions

QoS	Aggregation function
q^1	$q^1(rpl_i) = \sum_{s_k \in rpl_i}^{n} q^1(s_k)$
q^2	$q^2(rpl_i) = \underset{spl_j \subset rpl_i}{Max} (\sum_{s_k \in spl_j} q^2(s_k))$
q^3	$q^3(rpl_i) = \prod_{s_k \in rpl_i} q^3(s_k)$
q^4	$q^4(rpl_i) = \prod_{s_k \in rpl_i} q^4(s_k)$

Table 2. Transaction combinations for parallel pattern

s_1 / rpl_1	s_2 / rpl_2	$(s_1 / rpl_1) \| (s_2 / rpl_2)$
p	rc	p
r	r	r
r	rc	r
c	c	c
c	rc	c
rc	rc	rc
rc	p	p
rc	r	r
rc	c	c

$$q^\alpha(s_k) = \begin{cases} \dfrac{q^\alpha(s_k) - \min(q^\alpha(t_k))}{\max(q^\alpha(t_k)) - \min(q^\alpha(t_k))}, if\,(\max(q^\alpha(t_k)) - \min(q^\alpha(t_k)) \neq 0) \\ 1, \quad if\,(\max(q^\alpha(t_k)) - \min(q^\alpha(t_k)) = 0) \end{cases} \tag{1}$$

$$q^\beta(s_k) = \begin{cases} \dfrac{\max(q^\beta(t_k)) - q^\beta(s_k)}{\max(q^\beta(t_k)) - \min(q^\beta(t_k))}, if\,(\max(q^\beta(t_k)) - \min(q^\beta(t_k)) \neq 0) \\ 1, \quad if\,(\max(q^\beta(t_k)) - \min(q^\beta(t_k)) = 0) \end{cases} \tag{2}$$

By this way, the value of s_k on each QoS criterion in $\mathrm{q}(s_k)$ is ranged as [0, 1]. Additionally, the Simple Additive Weighting (SAW) [28] is introduced to support the computation of QoS utility for each candidate shown in equation group (3) where $Score(s_k)$ represents the QoS utility value of web service s_k and μ_m is the assigned weight for $q^m(s_k)$.

$$\begin{cases} Score(s_k) = \sum_{m=1}^{4} (\mu_m \times q^m(s_k)) \\ \sum_{m=1}^{4} \mu_m = 1 \end{cases} \tag{3}$$

$$\begin{cases} q^1(rpl_i) \leq q^1_{user} \\ q^2(rpl_i) \leq q^2_{user} \\ q^3(rpl_i) \geq q^3_{user} \\ q^4(rpl_i) \geq q^4_{user} \end{cases} \tag{4}$$

According to above definitions, several global QoS aggregation functions for $rpl_i = (s_1, s_2, ..., s_n)$ are given in **Table 1**, where $q^m(s_k)$ is the value of the selected service s_k for task t_k on the m^{th} QoS criterion, $m \in \{1,2,3,4\}$. More details about other QoS criteria such as reputation, data quality, compensation rate and etc. have been proposed in [3][6][25][29]. In order to check whether user's QoS constraints are satisfied for rpl_j, Eq. (4) is provided where the vector $\mathrm{q}_{user} = [q_{user}^1, q_{user}^2, q_{user}^3, q_{user}^4]$ represents user's QoS constraints.

$$F(rpl_j) = \sum_{s_k \in rpl_j} Score(s_k) \tag{5}$$

$$\begin{cases} F(cs) = \underset{rpl \in PLset(rpt_{\max})}{Max}(F(rpl)) \\ rpt_{\max} = \underset{rpt_i \in RPTset(cs)}{Max}(p(rpt_i)) \end{cases} \tag{6}$$

Our QoS utility function $F(rpl_i)$ for rpl_i can be computed by Eq. (5). Especially, we use Eq. (6) to express HP (Highest Probability) objective function for any cs based on the concept in [6] if not considering the transactional constraints.

4. Building Rules and Reduction Method

4.1 Transactional Properties of Web Services

Since web services are provided by different providers, their transactional properties may differ with each other. The main transactional properties of a web service include: retriable (r), compensatable (c), pivot (p), and the combination of retriable and compensatable (rc) [22][30]. One service is r if it is guaranteed to complete after finite execution; one service is c if it has a compensation operation that can undo its execution effect once it encounters a failure; one service is p if it has conducted successfully, its effect will be kept forever; one service is rc if it has functions of both r and c. We use TP to denote the set of all possibly transactional properties for a web service, then $TP = \{r, c, p, rc\}$. Because of the transactional consistency and atomicity requirements for global CWS, there are some constraints on transactional properties of individual candidates as defined in [12]. In the following sections, we discuss firstly possible combinations of transactional properties between two successive web services in a concrete CWS with sequence pattern. Then parallel pattern will be taken into account. However, we will not consider the conditional pattern since only one of the branches will be selected during the execution. Finally, we propose several building rules to address transactional constraint problems and present several related proofs. In the following, we use concept of route plan instead of concrete CWS since we do not consider the conditional pattern. Before discussing on transactional constraints, several functions are defined in **Table 3**.

Table 3. Function definitions

Function	Definitions
$tp(s_i)$	Transactional property of S_i.
$AC(rpl)$	Whether rpl fulfills transactional atomicity and consistency.
$\Pr eSub(s_i, rpl)$	$= \{s_1, s_2, ..., s_{i-1}\}, i > 1$, which denotes the set of web services before S_i in rpl.
$tp(rpl)$	$= (tp(s_1), tp(s_2), ..., tp(s_n))$, $rpl = (s_1, s_2, ..., s_n)$. It denotes one combination sequence of transactional properties for rpl, which consists of transactional properties of its constituted web services.

4.2 Building Rules for Sequence Pattern

We first introduce the definition of transactional atomicity and consistency [21][22]:

Def. 1 (Transactional Atomicity and Consistency): a route plan satisfies atomicity and consistency of transaction if and only if the following requirement can be fulfilled:

In an execution process of one route plan, once any web service within this route plan encounters execution failure and cannot be executed successfully, then all completed web services ahead of the failed web service must be compensatable.

Let $rpt = (t_1, t_2, ..., t_n)$ including only sequence pattern be the route path of cs with the maximum probability, $rpl_i = (s_1, s_2, ..., s_n)$ is one of route plans for rpt. For any two sequent web services s_k and s_{k+1} in rpl_i, if $AC(rpl_i) = true$, then we have following propositions:

Prop. 1: if $tp(s_k) \in \{r, p\}$ and $AC(rpl_i) = true$, then $tp(s_{k+1}) \in \{r, rc\}$.

Prop. 2: if $tp(s_k) = c$ and $AC(rpl_i) = true$,then $tp(s_{k+1}) \in \{r, p, c, rc\}$.

Prop. 3: if $tp(s_k) \in rc$ and $AC(rpl_i) = true$, then $tp(s_{k+1}) \in \{r, p, c, rc\}$.

The details on proofs for Prop.1, Prop. 2 and Prop. 3 have been presented in our previous work [31]. Due to space limitation, we don't expand them in this paper.

Def. 2 (Combinations Set built on Prop. 1, Prop. 2 and Prop. 3 ($RCTP(rpt)$)):

The set $RCTP(rpt) = \{tp(rpl_i) \mid tp(rpl_i)$ built based on Prop. 1, Prop. 2 and Prop.3, $i \in N^+\}$, denotes the set including all possible combination paths on transactional properties for rpt. Meanwhile, the building rules are based on Prop .1, Prop. 2 and Prop. 3.

Def. 3 (Combinations Set built on Def. 1 ($ACS(rpt)$)):

The set $ACS(rpt) = \{tp(rpl_i) \mid AC(rpl_i) = true,\ i \in N^+\}$, denotes the set including all transactional combination paths for rpt. Besides, each combination path in this set fulfilles the requirement defined in Def. 1.

Corollary 1: given any route path $rpt = (t_1, t_2, ..., t_n)$, then $ACS(rpt) \subseteq RCTP(rpt)$.

Proof: for each $tp(rpl_i) \in ACS(rpt)$, $i \in N^+$, we have $AC(rpl_i) = true$ based on Def. 3. Obviously, basing on Def. 2, we have $tp(rpl_i) \in RCTP(rpt)$, since $AC(rpl_i) = true$ can prove Prop.1, Prop. 2 and Prop. 3. It also means $ACS(rpt) \subseteq RCTP(rpt)$.

Prop. 4 (The proposed building rules): since the transactional property rc is the combination of transactional properties r and c, we divide transactional property rc into two different identifiers: rc^1 and rc^2. The two identifiers in the building rules have the same semantics as transactional property rc, but formally represent transactional property c and transaction property r, respectively. To this end, we have the following building rules for transactional combinations of route plans:

(1) Let $tp(s_k) = rc$, then $tp(s_k) = rc^1$ and $tp(s_k) = rc^2$;

(2) Let $tp(s_k) \in \{r, p\}$, then $tp(s_{k+1}) \in \{r, rc^2\}$;

(3) Let $tp(s_k) = c$, then $tp(s_{k+1}) \in \{r, c, p, rc^1\}$;

(4) In the situation that $tp(s_k) = rc^1$, then $tp(s_{k+1}) \in \{r, p, rc^1, c\}$;

(5) In the situation that $tp(s_k) = rc^2$, then $tp(s_{k+1}) \in \{r,\ rc^2\}$.

Def. 4 (Reduction Set ($RRCTP(rpt)$)):

We define the set $RRCTP(rpt) = \{tp(rpl_i) \mid tp(rpl_i)$ built based on rules of Prop. 4, rpl_i represents any possibly route plan of rpt , $i \in N^+\}$.

Corollary 2: $ACS(rpt) \subseteq RRCTP(rpt) \subseteq RCTP(rpt)$.

Proof: to prove the corollary, we firstly prove $RRCTP(rpt) \subseteq RCTP(rpt)$. If we extend the rule (2) by transforming $\{r, rc^2\}$ into $\{r, rc^1, rc^2\}$, then due to the equivalent relationship on semantics among rc^1 , rc^2 and rc , the building rules (2), (3) and (4) in Prop. 4 are equivalent to Prop. 1, Prop. 2 and Prop. 3, respectively. For this situation, the rule (5) is actually included in the rule (4). Therefore, by extending the rule (2) in Prop. 4, rules of Prop. 4 equals rules of combination among Prop. 1, Prop. 2 and Prop. 3. Thus, we can get $RRCTP(rpt) \subseteq RCTP(rpt)$ based on definitions of the two sets and the uniqueness theory of elements in a set.

Further, $ACS(rpt) \subseteq RRCTP(rpt)$ will be proved. As demonstrated above, if the extended rule (let $tp(s_k) \in \{r, p\}$, then $tp(s_{k+1}) = rc^1$) is added to Prop. 4, then $RRCTP(rpt) = RCTP(rpt)$. According to the rules (2), (4) and (5) in Prop. 4 and the same semantics between rc^1 and rc^2 , we define the set $NCTP(rpt)$ denoting the increased paths in $RCTP(rpt)$ to $RRCTP(rpt)$. Formally, $NCTP(rpt) = \{ tp(rpl_i) \mid tp(rpl_i \in RCTP(rpt)) \wedge (\exists\ s_k , s_{k+1} , s_{k+2}$ in rpl_i , $(tp(s_k) \in \{r, p\}) \wedge (tp(s_{k+1}) = rc^1) \wedge (tp(s_{k+2}) \in \{p, c\})\}$. Then, $RCTP(rpt) = RRCTP(rpt) + NCTP(rpt)$. Thus, for each $tp(rpl_i)$ in $NCTP(rpt)$, there are at least such successive web services s_k , s_{k+1} , s_{k+2} in rpl_i , which obey the rule that $tp(s_k) \in \{r, p\} \wedge tp(s_{k+1}) = rc^1 \wedge tp(s_{k+2}) \in \{p, c\}$. Obviously, if s_{k+2} encounters execution failure, then rpl_i can't be compensated due to $tp(s_k) \in \{r, p\}$. Therefore, $tp(rpl_i) \notin ACS(rpt)$, which also means that $ACS(rpt) \cap NCTP(rpt) = \Phi$. Due to the corollary 1 that $ACS(rpt) \subseteq RCTP(rpt)$, we have $ACS(rpt) \subseteq RCTP(rpt) - NCTP(rpt)$. Therefore, $ACS(rpt) \subseteq RRCTP(rpt)$.

Corollary 3: $RRCTP(rpt) \subseteq ACS(rpt)$

It can be proved by the method of mathematical induction, whose details are ignored due to lack of space.

Corollary 4: $RRCTP(rpt) = ACS(rpt)$.

Proof: it could be achieved directly by corollary 2 and corollary 3.

Evidently, corollary 4 demonstrates the correctness including both completeness and reliability for our proposed building rules to address transactional constraints. Basing on the proposed building rules, we presented a DAG model called virtual SCG (vSCG), which lays a basis for our actual SCG later. The difference is that the nodes in vSCG and actual SCG denote transactional properties and actual candidates, respectively. The details on how to build the two graph will be illustrated in our building algorithm later. **Fig. 1** gives an example of vSCG corresponding to route path $rpt = (t_1, t_2, t_3, t_4, t_5)$. For each task, there are five nodes (corresponding to r , c , p , rc^1 , rc^2). Additionally, a edge exists in two nodes if and only if the two nodes satisfy one of requirements in the proposed building rules.

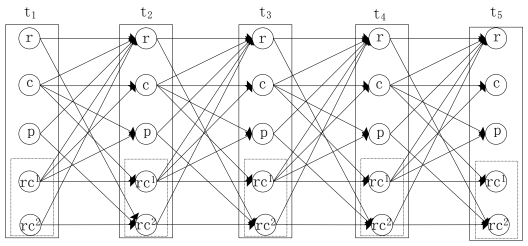


Fig. 1. The *vSCG* built based on the proposed rules

4.3 The Reduction Method

The building rules aforementioned only resolve transactional constraints for sequence pattern. To address the problem for parallel pattern, we present a reduction method to model all transactional combinations of its components into five reduction nodes. Then, we can treat each parallel pattern as a common task so as to reduce any route plan into the situation of a route plan with only sequence pattern. For any parallel pattern, the reduction method includes two-folds: 1) reduction process for each sequence branch: after building the corresponding vSCG based on the proposed building rules, we make reduction rules transform all paths in the vSCG into four reduction nodes; and 2) combination process for the parallel pattern: querying combination rules with the number of sequence branches from a database (introduced later), we can construct several mapping relationships between reduction nodes of sequence branches and reduction nodes of the parallel pattern. Besides, these mappings will enable the later selection algorithm. Definitions of the reduction nodes for any $tp(rpl_i)$ are explained:

- Reduction node r : all its constituted web services on semantics are equivalent to r .
- Reduction node c : all its constituted web services on semantics are equivalent to c .
- Reduction node rc : all its constituted web services on semantics are equivalent to rc .
- Reduction node p : 1) $AC(rpl_i) = true$; 2) it doesn't satisfy the definitions of reduction node r , reduction node c and reduction node rc .
- Then the reduction process for each sequence branch is presented after its vSCG built:
- For the path whose component nodes are only r or rc^2 , and at least include one with r , we reduce it into the reduction node r .
- For the path whose component nodes are only c or rc^1 , and at least include one with c , we reduce it into the reduction node c .
- For the path involving only component nodes rc^1 or rc^2 , we reduce it into the reduction node rc .
- Finally, we reduce other paths in vSCG into the reduction node p .

After conducting reduction process for each sequence branch, we can get the correlative relationships between the reduction nodes and transactional combination paths in its corresponding vSCG. Hence, we can use the reduction nodes instead of their correlative paths when considering the result of transactional combinations among these sequence branches within the same parallel pattern. Subsequently, the combination process is used to build correlative relationships from reduction nodes representing sequence branches to the reduction nodes of current parallel pattern. Fortunately, the combintaion rules can be searched from a rule database, which can be constructed in advance for the reason that the rules only depend on the number of sequence branches in a parallel pattern. The constructed method is based on recursively using transactional combination rules on two concurrent components in

[12][13], as demonstrated in **Table 3**. Finally, for any parallel pattern, we can get mapping relationships from the set including its reduction nodes to the set including all transactional combination paths of its sequence branches. Based on the mapping relationships, the later selection algorithm can address transactional constraints for parallel pattern.

5. The Graph-building and Selection Algorithms

In this section, based on the proposed building rules and the reduction method, we firstly present a graph-building algorithm to formulate the actual SCG for any route path. Further, a selection algorithm depending on results of the graph-building algorithm is proposed.

```
Algorithm 1 actual SCGBuilding Algorithm
 Input: abstract CWS cs, Set S /* S denotes all available candidate Web Services for the whole tasks of cs */
 Output: SCG(V, E, Q) /* actual SCG model with QoS vectors weight Q */
    GraphStruture vSCG($V_v$,$E_v$) = vSCGBuiliding (cs ); /* Formulating Transactional-based virtual SCG for cs */
    $E = \phi$; /* Initializing edges for actual SCG */
    V = {s, d}+S+$V_v$.RN; /* Initializing vertexes set for actual SCG */
    preNodeTask = s; /* Scanning vertexes from $V_v$ */
    while (preNodeTask != d) do
        if (preNodeTask == s) then
            S' = {s}; /* Initializing candidate vertexes for the preNode */
        End if
        postNodeTask=findNextTask (preNodeTask, vSCG) ; /* Solving the next task of preNode based on vSGP */
        if (postNodeTask == d) then
            for each candS∈S' do
                E=E+<candS,d>; /*Add edges from all services of S' to d */
                Q(<candS,d>)=0;
            End for
        else /*In case not the last task node d*/
            if (isRnNodes(postNodeTask) == false) then
                S''=search(S, postNodeTask); /* Get candidates for the task */
                Divide S'' into S''(r), S''(c), S''(p), S''($rc^1$), S''($rc^2$);
                if (S' == {s}) /* it denotes the first task is source node */
                    for each candS∈S'' do
                        E=E+<s,candS>;
                        Q(<s,candS>)=<candS.$q^1$, candS.$q^2$,…, candS.$q^m$>;
                    End for
                else /* Process normal tasks in vSCG.*/
                    vE=getOutEdgesByTask (preNodeTask, $E_v$); /* Find outcome edges of preNodeTask from vSCG */
                    for each ve∈vE do
                        for each candS$_1$∈S'(ve.startNode.tp) do
                            for each candS$_2$∈S'' (ve.endNode.tp) do
                                E = E+<candS1,candS2>;
                                Q(<candS$_1$,candS$_2$>)=<candS$_2$.$q^1$,…, candS$_2$.$q^m$>;
                    End for
                End if
            else
                S'' = processRnNodes (V,E, S',preNodeTask,$V_v$ , $E_v$);
                processPPbyRnNodes(V, E, postNodeTask, $V_v$ ,$E_v$);
            End if
        End if
        preNodeTask = postNodeTask;
        S' = S''; /* The pointers of task and its candidates to move forward */
    End while
    return SCG(V, E, Q);
```

Fig. 2. The actual SCGBuilding algorithm

5.1 The Graph-building Algorithm

The main Graph-building algorithm called *actual SCGBuilding* is presented in Algorithm 1 of

Fig. 2. Step 1 invokes a sub-algorithm called *vSCGBuilding* to generate a vSCG based on transactional properties, which is a layered DAG model to represent constraint relationships on transactional properties of tasks for the input parameter cs. After that, steps 2-3 conduct the initial process of the actual SCG, which adds all candidates and reduction nodes of the vSCG to the set S representing all nodes in the actual SCG. Then, the algorithm begins to add edges for the actual SCG by scanning vSCG step by step (steps 5-39). Steps 6-8 firstly assign source node s to the candidate set s' of the task pointed by *preNodeTask* if current task is source node s. Then, the function of *findNextTask (preNodeTask, vSCG)* is invoked to obtain the next task to *preNodeTask* within vSCG (step 9), meanwhile the abtained task is assigned to *postNodeTask* (that stores the succeeding task of *preNodeTask*). Further, steps 10-15 add edges and assign values of corresponding QoS weights to ***0*** when *postNodeTask* is the target node d. On the other hand, the algorithm processes the situation that *postNodeTask* is a normal task (steps 16-35). In steps 17, the algorithm invokes the function of *search(S, postNodeTask)* to get the set s'' of all candidates for *postNodeTask* from set S. Then, step 18 divides s'' into five classes based on transactional properties where the classes rc^1 and rc^2 are equivalent for the same semantics of them. In steps 20-23, the algorithm adds all edges from s to all candidates in s'' and assigns the QoS value of corresponding candidate to the weight value of each edge when s' equals to the set *{s}*. Steps 24-31 add edges and their weight values based on the vSCG model to the actual SCG model when *preNodeTask* and *postNodeTask* are normal tasks. In this case, all edges from the candidates in s' to the candidates are added to the actual SCG model (Steps 29-30). Subsequently, If *postNodeTask* is a reduction node in vSCG, the algorithm firstly adds edges from candidates in s' to the reduction nodes of *postNodeTask*, meanwhile assigns the reduction nodes to s'' for the purpose of next round processing (step 34). Afterwards, the function of *processPPbyRnNodes* is invoked to process each sequence branches by running recursive the actual SCGBuilding algorithm (step 35). Due to space limitation, the details on the function aren't expanded. Finally, The algorithm moves forward by assigning the value of *postNodeTask* to *postNodeTask* (steps 38-39) . Additionally, a new loop will be carried out. Step 41 returns our actual SCG model if the loop is ended.

Based on our proposed building rules and the reduction method, the *vSCGBuilding* algorithm to build a vSCG model is illustrated in Algorithm 2 of **Fig. 4**. In step 1, the algorithm creates a sequence structure to store tasks with sequence relationship. Steps 2-13 create virtual nodes for all tasks by scanning cs (or route plan). Specifically, steps 4-9 create virtual nodes and reduction nodes for current element e in cs when e is a AND split operation. In step 4, the function of *createReductionTask()* is invoked to create a reduction task representing current parallel pattern. Then, steps 5-8 store all the elements in current parallel pattern as a variable pb (a structure storing multi-sequence branches). Additionally, step 9 invokes a sub-algorithm called *vSCGBuildingforPP* (illustrated in Algorithm 4 of **Fig. 4**) to build a sub-vSCG model for the input parameter pb. For each sequence branch stored in pb, the main steps of *vSCGBuildingforPP* algorithm include three-folds: 1) the algorithm firstly creates a reduction task representing current sequence branch and builds a mapping relationship between the generated reduction task and the reduction task representing AND structure (parallel pattern) so as to enable our selection algorithm later (steps 3-4); then 2) it creates four reduction nodes for the generated reduction task (step 5); 3) finally, the step 7 invokes recursively the *vSCGBuilding* to build sub-vSCG model for current branch. Comparing to reduction task of a parallel pattern, reduction task of a sequence branch includes only four reduction nodes because the former will be as a common task in our selection algorithm, while the latter is just as an intermediate task used for reduction rules of parallel patterns. After executing the *vSCGBuildingforPP* algorithm, we have created a sub-vSCG for current parallel pattern. As such, we can use reduction task of current parallel pattern as a common task. Step 11, in *vSCGBuilding* algorithm, adds current reduction task rn or task e to main sequence structure sb. Subsequently, the algorithm creates five virtual nodes or reduction nodes depending on the type of the scanning element e (e.g. virtual nodes if e is a task, or reduction nodes if e is an AND join operation) (step 12). After all nodes created, the algorithm invokes a sub-algorithm called *vSCGEdgesforSP* (illustrated in Algorithm 3 of **Fig. 3**) to create edges for sb according to the proposed building rules. Finally, the algorithm returns vSCG model *vSCG(V, E)* (step 15).

Our complete graph model called vSCG can be generated. **Fig. 3** shows the corresponding vSCG for $cs = (t_1, t_2, (t_3, t_4) | t_5, t_6, t_7)$ (where the circle denotes virtual nodes, the rectangle denotes task node, and dashed rectangle denotes reduction task.). Three reduction tasks (e.g. Reduction tasks rn_1, rn_2 and rn_3 represent the parallel pattern $((t3, t4) \| t5)$, the sequence branches $(t3, t4)$ and $(t5)$ respectively.) and their reduction nodes are created. For other common tasks, corresponding virtual nodes and edges are created based on building rules.

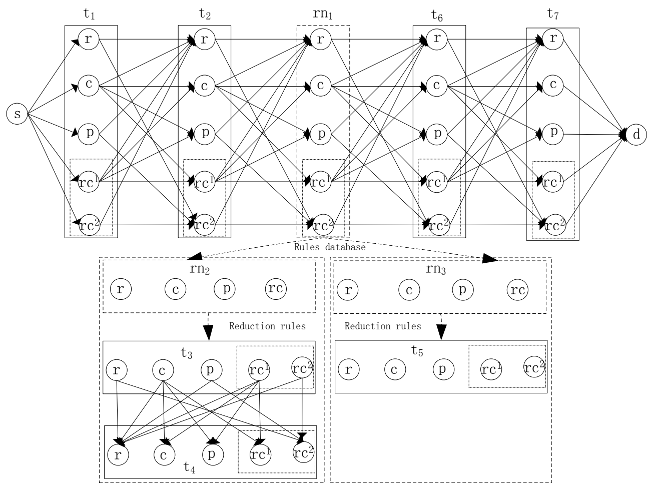


Fig. 3. The generated *vSCG* to our example

5.2 The Global Selection Algorithm

After constructing our actual SCG model, we have addressed transactional constraints. In this subsection, we present a selection algorithm based on the actual SCG. The purpose of the algorithm aims at achieving the near optimal value of QoS utility with end-to-end QoS constraints, which actually becomes one variant problem to solve the single-source shortest paths. Unfortunately, this is a well-known multi-constraints optimal-path problem in the graph theory, which is a NP-Hard problem. Our selection algorithm is based on the algorithms of *MCSP* and *MCSP_RELAX* in [6]. The difference is that our selection algorithm is built on top of our actual SCG model satisfying transactional constraints for the whole CWS, while their algorithm is based on a full connection graph model. Particularly, our algorithm presents a new cost function for the relaxing algorithm regarding both QoS constraint margin and generated global QoS utility to improve the performance on global QoS utility further.

Demonstrated in **Fig. 5**, Algorithm 5 called *QoS-optSltforTCWS* presents the selection procedure, which relaxes the best optimization problem to resolve a near optimization problem. At the beginning, the algorithm defines several initial variables (steps 1-2). Step 3 begins to search all possible intermediate paths by scanning candidate nodes, which are sorted topologically on the basis of their corresponding virtual nodes of vSCG in advance. Then, it conducts two parts. On the one hand, the situation of sequence pattern is processed (steps 4-16). In step 4, the algorithm checks whether the next of current task is a common task (step 4). In case of true, two loops are used to visit the all adjacent to nodes in *firstNodes*, and then update the utility values and the global QoS values for their corresponding paths (steps 8-9 for the situation of *firstNodes={s}*; steps 12-13 for the other.). Besides, functions *F(v)*, *F(p)* and *q(u,v)* are based on Eq. (3), Eq. (5) and the combinations of Eq. (1) and Eq. (2), respectively.

```
Algorithm 2 vSCGBuilding Algorithm
 Input: abstract CWS cs, Set V, E, Boolean mainBranch
 Output: GraphStructure vSCG(V, E) /* Virtual SCG model*/
 Global variable: Set V, E /* Their initial values are both null.*/
1     SBStructure sb = createSBStructure();
2     while (nextElement(cs, e ) != null)/*nextElement(cs, e) returns the next element of e and assigns the return value to e.*/
3         if (isAndSplit(e) == true) then
4             ReductionTask rn = createReductionTaskNode (); /* Create a reduction task node for current parallel pattern.*/
5             PBStructure pb = createPBStructure(rn);
6             while (isAndJoin (nextElement(cs, e )) != true) /* Check if current element is the end for the parallel pattern.*/
7                 pb.addParallelBranch (e);
8             End while
9             vSCGBuildingforPP (pb,V,E);
10        End if
11        sb.addSB (rn/e); /* rn/e = rn, if (isAndJoin(e) == true); otherwise, rn/e = e. */
12        V.addToTask (($C(r)$, $C(c)$, $C(p)$, $C(rc^1)$, $C(rc^2)$), rn.id /e); /* According to transactional properties{r,c,p,rc}*/
13    End while
14    vSCGEdgesForSP(sb, V, E); /* Create edges for the sequence pattern */
15    return vSCG(V, E);
```

```
Algorithm 3 vSCGEdgesforSP Algorithm
 Input: SBStructure sb, Set V, E
 Output: vSCG(V, E) /* Get virtual SCG model */
1     for i = 1 to sb.size-1 do
2         for each vnode in classes of $C_i$ do /*Create outcome edges for virtual nodes in{$C_i(r)$,$C_i(c)$,$C_i(p)$,$C_i(rc^1)$, $C_i(rc^2)$}*/
3             switch (vnode) do
4                 case $C_i(r)$: $E = E+\{<C_i(r), C_{i+1}(r)>, <C_i(r), C_{i+1}(rc^2)>\}$
5                 case $C_i(c)$: $E = E+\{<C_i(c), C_{i+1}(r)>, <C_i(c), C_{i+1}(c)>, <C_i(c), C_{i+1}(p)>, <C_i(c), C_{i+1}(rc^1)>\}$
6                 case $C_i(p)$: $E = E+\{<C_i(p), C_{i+1}(r)>, <C_i(p), C_{i+1}(rc^2)>\}$
7                 case $C_i(rc^1)$: $E = E+\{<C_i(rc1), C_{i+1}(r)>, <C_i(rc^1), C_{i+1}(c)>, <C_i(rc^1), C_{i+1}(p)>, <C_i(rc^1), C_{i+1}(rc^1)>\}$
8                 case $C_i(rc^2)$: $E = E+\{<C_i(rc^2), C_{i+1}(rc^2)>,<C_i(rc^2), C_{i+1}(r)>\}$
9         End for
10    End for
11    return vSCG(V,E);
```

```
Algorithm 4 vSCGBuildingforPP Algorithm
 Input: PBStructure pb, Set V, E
 Output: vSCG(V, E) /* vSCG model */
1     SBStructure sb = pb.nextSB ();
2     while(sb != null)
3         ReductionTask rn = createReductionTaskNode (); /* Create the reduction task node for the sequence branch sb. */
4         E.addRNLink (pb.getReductionTask, rn);
          /* Create the link between reductionTask of parallel branch and reducedTask of its sub-sequence. */
5         V.addRNToTask (($C(r)$, $C(c)$, $C(p)$,$C(rc)$), rn.id); /* Creating reduction nodes for current reduction task */
6         sb.addSBReductionTask (rn); /* Add the corresponding reduced node to this sequence branch.*/
7         vSCGBuilding (sb,V, E, false); /* Build vSCG for current sequence branch sb.*/
8     End while
9     return;
```

Fig. 4. The *vSCGBuilding*, *vSCGEdgesforSP* and *vSCGBuildingforPP* algorithms

Particularly, in order to limit the computing scale for execution of our algorithm, Algorithm 6 called *relax_basedTP* is invoked to keep just only *k*-optimal intermediate paths in each set of nodes with the same virtual node (or transactional property) (steps 10 and 14). The *relax_basedTP* algorithm, demonstrated in **Fig. 5**, includes following four-folds:

- Firstly, it checks whether each QoS criterion value of current path satisfies the global end-to-end constraint, where all the checking criteria are based on Eq. (4) (step 1);
- Then the function *searchByTP_Task* is invoked to assign current all generated paths corresponding last nodes with the same virtual node *v.vnode* to the variable *paths_tp* (step 2). Subsequently, if any path *p*, with superiority to the path of *[u,Q,F]* on the values of both global QoS criteria and utility, exists in *Paths_tp*, then we remove it (step 4);
- Otherwise, the path of *[u,Q,F]* will be added to the paths set *Paths* (steps 5- 6);
- Additionally, we add directly this path of *[u,Q,F]* to *Paths* if the number of current generated paths is less than the valve *k* (step 7). Otherwise, we instead the path of *[u,Q,F]* with the worst value of cost function in *paths_tp* (step 8).

In *relax_basedTP* algorithm, our criterion on selecting *k*-optimal paths, illustrated in Eq. (7), is improved on the cost function concept combining multiple constraints into one [6][33]. In Eq. (7), function $q^i(p)$ denotes value of the i^{th} aggregated QoS criterion for path p, two variables α and λ represent weight factors on the value of global QoS utility.

$$g(p)=[(\frac{q^1(p)}{q^1_{user}})^3+(\frac{q^2(p)}{q^2_{user}})^3+\dots+(\frac{q^m(p)}{q^m_{user}})^3]\times[\alpha\times(Score(p))^{\lambda}]^{-1} \tag{7}$$

On the other hand, Algorithm 5 conducts the selection process with the situation of parallel pattern (steps 17-29). In step 18, it invokes function *getRNsByLinks* to obtain all reduction tasks representing sub-sequence branches corresponding to the parallel pattern of reduction task *nextTask(firstNodes)*. Next, in terms of the reduction process in section 4.3, the algorithm marks each edge with color in set *{r, c, rc, rc^1, rc^2}* (corresponding to the set of reduction nodes) for each sequence branch by invoking the function *mark_color_edgesByRdRls*, whose input parameter *rnTask_SB* denotes corresponding reduction task of a sequence branch (step 20). Therefore, for each sequence branch, we can find all paths for each reduction node in a sequence branch with the help of color edges. Take **Fig. 3** as an example, we can easily find all corresponding paths for reduction node r in reduction task rn_2 if the function *mark_color_edgesByRdRls* is conducted. After that, step 21 gets all combination rules among reduction nodes (in the parallel pattern and its sequence branches) by invoking the function *searchRNTPRules*, which is used to search combination rules from a database by input parameter of the number of sub-sequence branches in current parallel pattern. Besides, the database is as mentioned in section 4.3 and can be implemented in advance. Subsequently, steps 22-29 conduct a loop, which aims to select *k*-optimal paths for each reduction node *tp* in the reduction task of current parallel pattern. Specifically, the key folds of the loop are as follows: 1) Firstly, the algorithm keeps initial *k* paths for *tp* by invoking the function *optSelect*, which selects *k*-optimal paths from the set *pre[tp]* denoting all selected paths previous to the node *tp* (step 22); 2) then steps 24-28 try to find *k*-optimal paths for *tp*. The process involves two sub-processes. Firstly, in terms of the type of *tp* and the obtained combination rules *rnTPRules*, step 24 invokes the function *getTPfromRules* to get a set *sbTPset*, which is obtained by transforming the set of combination rules relative to *tp* within *rnTPRules* into a set of reduction nodes in all sequence branches.

```
Algorithm 5 QoS-optSltforTCWS Algorithm
 Input: SCG (V, E, Q), int k  /* k denotes the relaxed variable .The SCG(V, E, Q) is the actual SCG model */
 Output: Path p   /* Return the path with near optimal QoS utitlity. */
1    Set firstNodes = {s};  /* Initialize the variable storing all nodes of current task. */
2    Node u, v; /* u denotes the current scanning node, while v denotes the adjacent node to u */
3    while (nextTask (firstNodes) != d)
4        if (isRNTask (nextTask (firstNodes)) == false) then /*nextTask(firstNodes) returns next task to firstNodes.*/
5            for each u in firstNodes do
6                for each v in adj[u] do
7                    if (u == s) then /* If the first task of current scanning edge is source node .*/
8                        Q = q(u,v); /*Update QoS Vector for each path. */
9                        F = F(v); /*The utility of current path is utility of  v. */
10                       Paths = relax_basedTP (Q,F, u,v, k);
11                   else for each p in Paths[u]
12                       Q = p.Q+ q(u,v);
13                       F = p.F+F(v);
14                       Paths = relax_basedTP(Q, F, u, v, k); /*Relax k-optimal paths in current selected paths. */
15                   End if
16           End for
17       else /* In case of parallel pattern*/
18           rnTask_SBs = getRNsByLinks (nextTask (firstNodes)); /* Get reduction tasks of the sequence branches. */
19           for each rnTask_SB in rnTask_SBs do
20               mark_color_edgesByRdRls(rnTask_SB, SCG); /*Mark edges with colors based on reduction process. */
21           rnTPRules = searchRNTPRules(nextTask(firstNodes));/*Get combination rules by current parallel pattern. */
22           for each tp in $\{p,r,c,rc^1,rc^2\}$ do /* Loop by five reduction nodes of current parallel pattern. */
23               paths_tmp[tp] = optSelect(pre[tp], k); /* Select k-optimal paths previous to the reduction node tp. */
24               sbTPset = getTPfromRules(rnTPRules, tp); /* */
25               for each sbtp in sbTPset do
26                   sb_p[sbtp] = optPtsbyColor(paths_tmp[tp], sbtp, SCG, k);
                     /*Find recursively k-optimal paths from each sequence branch*/
27               subRls = getRNTPRulesByTP(rnTPRules, tp);
28               paths[tp] = merge_paths(sb_p, subRls, k);
                     /*Merge k-optimal combination paths including all sequence branches in current parallel pattern.*/
29           End for
30       End if
31       firstNodes = getNodes(nextTask(firstNodes)); /* Get all nodes of  nextTask(firstNodes).*/
32   End while
33   Find the optimal utility p in Paths[t] s.t. adj[t] = d; /* Return a near optimal result. */
34   return p;
```

```
Algorithm 6 Relax_basedTP Algorithm
 Input: Vector Q, Real  F, Node u, v,  Int k
 Output: Paths  /* Keep paths limited less than k in each vnode. */
1    if(isSatisfyQoSConstraits(Q, $Q_c$) == false)  return Paths;
2    paths_tp = searchByTP_Task (Paths, v.vnode);
3    if  (exist p in Paths_tp that (p.F >  F ) and (p.Q  <=  Q)) ;
4        return Paths;
5    if  (exist p in Paths_tp that (p.F <  F) and (p.Q  >=  Q ))
6        repace(p, [u,Q,F], Paths[v]); return Paths;
7    if  (paths_tp.size < k)  addPath ([u,Q,F], Paths[v]);
8    else replace_or_remove (getMax_func (paths_tp), [u,Q,F]);
9        return Paths;
```

Fig. 5. The QoS-optSltforTCWS and Relax_basedTP algorithms

For expressing clearly, we still take **Fig. 3** as an example. If *tp* in rn_1 is *r*, then the combination rules relative *tp* is the set *{<r, r>, <r, rc>, <rc, r>* where each rule denotes the combination between two reduction nodes in rn_2 and rn_3 respectively. Thus, we get *sbTPset= {r, rc}*. Further, for each reduction node *sbtp* in *sbTPse*t, based on edges colored by reduction rules above, function *optPtsbyClr* is invoked to recursively compute *k*-optimal paths from all paths corresponding to *sbtp* for each sequence branch (steps 25-26). On the other hand, according to each combination rule relative *tp*, the function *merge_paths* is invoked to obtain finally *k*-optimal paths (from source node s to the reduction node *tp*) (steps 27-28). The main idea of the function *merge_paths* is, based on the combination rules of *tp* and the process of merging parallel pattern in [6] to merge the k-optimal paths from each sequence branch into the final k-optimal paths for *tp*. After that, a new loop considering another value of *tp* begins to run.

After the loop ended, step 30 moves forward *firstnodes* to store nodes of the succeeding task (step 31) so as to carry out a new *while* loop again. The *while* loop will be ended when *nextTask(firstnodes)* is the target node d. In the final, the algorithm returns one of the most optimal utility path p as a near-optimal solution (steps 33-34).

For calculating the upper bound of the worst-case computation complexity of the proposed method, we assume that there are n tasks , m candidates for each task and the relaxing variable is equivalent to k. In algorithm 2, it takes at most O($5*n$) times to create virtual nodes in steps of 2-13, meanwhile O($25*(n-1)$) iterations of adding edges to the vSCG model (algorithm 3) is required. Therefore, the total computation complexity of algorithm 2 for building the vSCG model is O($C_1 n$), C_1 denotes a constant. In algorithm 1, it's based on the vSCG model (as shown in step 1). Obviously, the main calculation of algorithm 1 focuses on steps 5-40 to add all actual nodes for the aSCG model, whose computation complexity is O(C_2*n*m^2). Here, C_2 is a constant. Therefore, the total computation complexity of algorithm 1 (building the vSCG model) is O(C_2*n*m^2). In algorithm 5, the selection algorithm has n iteration in steps 3-32 to scan the whole tasks in the considering route path. For each iteration, two layer iterations are invoked in steps 5-16, where the procedure *relax_basedTP* in algorithm 6 is invoked. The procedure *relax_basedTP* aims to keep k optimal paths for each classes in each task (Here, the class is also virtual node or transactional property), whose main calculation is O(C_3). C_3 is a constant if we implement it with the adjacent link data structure. Thus the complexity of *relax_basedTP* is O(C_3). Since we only keep k optimal paths for each classes, there are at most $5*k$ nodes in the iteration of step 5 of algorithm 5. Moreover, there are at most m edges adjacent to each currently selected path. Therefore, the total complexity of steps 5-16 is O(C_3*k*m). On the other hand, steps 18-30 in algorithm 5 are used to process the case with parallel pattern, which invoke recursively the processing code of steps 5-16 to process each sequence branch in current parallel pattern. The additional operation is the function of *merge_path*, which only depends on the number of sequence branches in current parallel pattern. Therefore, the computation complexity of steps 18-30 can be regarded nearly as the complexity of steps 5-16. Thus, the combined complexity of algorithm 5 is O($C_3*k*m*n$). Finally, based on the analysis above, we can get the total calculation complexity of the proposed method: O($C_3*k*m*n+C_2*n*m^2$), which can also be represented as O($m*n*(C_3*k+C_2*m)$).

6. Experiments and Evaluation

In order to show the efficiency and feasibility of our approach, several groups of experiments have been conducted. In our experiments, we implemented our selection approach as well as the selection approach proposed by EL Haddad et al. [12][13]. For our selection approach, implementations were conducted by setting different values for the relaxing variable *K*. The running platform under Windows XP and Java 1.6 is a Lenovo machine with Intel Core i3 3.2 GHz processors and 2GB RAM. The main QoS data on criteria of response time, reliability and availability are based on publically accessible QWS Dataset including QoS values of 2507 real web services collected by Dr. Eyhab Al-Masri in 2008 [34]. Especially, we randomly generated QoS values ranged from 50 to 1000 for cost criterion for each candidate web service since the public QWS dataset does not include the cost QoS criteria. Moreover, the transactional property values of *{r, c, rc, p}* were randomly associated with each candidate web service.

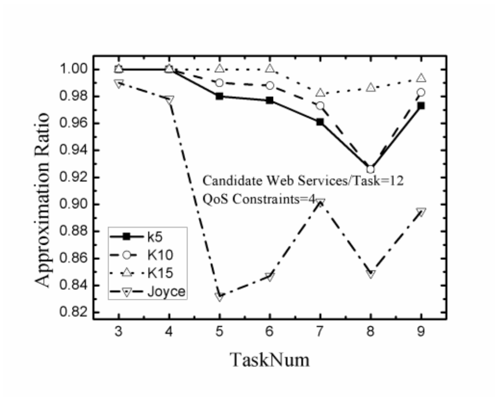
6.1 Performance Study with the Number of Tasks Varying

We first present two groups of experiments by dynamically increasing number of tasks. The first group of experiments includes two different experiments. Since the approximate ratio on the global QoS utility needs to obtain the best optimal QoS utility value which will take exponential time by exhaustive searching [10], we have made a limitation on the involved candidates by using a relatively smaller number so that exhaustive searching under exponential time in our experiments becomes posssible. Additionally, four criterion constraints were considered. **Fig. 6 (a)** plots approximation ratio of global QoS utility obtained by our selection approach and the selection approach in [12][13] to best optimal QoS utility obtained by exhaustive searching. As shown in **Fig. 6 (a)**, our approach performs better than the approach in [12][13] no matter what value of the relaxing variable *K* is assigned, since our approach is based on a global selection process while they used a local approach considering QoS aspect in order to reduce transactional constraints. Note that with the increasing task number of the route path, the approximation ratios of all approaches decrease slightly. Meanwhile, the approximation ratios are much better with larger value of variable *K*. **Fig. 6 (b)** shows the time consuming trend when task number is increased from 3 to 9. From the **Fig.6 (b)**, we can further see that our approach is slightly inferior to the approach in [12][13]. Yet, the time consuming of our approach increases linearly with the number of tasks increasing, since the complexity of our proposed method is O($m*n*(C_3*k+C_2*m)$).

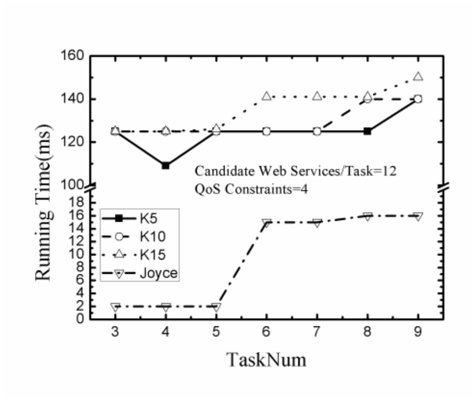
The second group of experiments was also conducted by varying the number of tasks in a route path as well as significantly increasing the number of candidates (more than 150) of each task. It aims at checking how our extensible capability will be affected by larger number of candidates and tasks. **Fig. 6 (c)** shows that, within the preconditions of larger candidate number and four criteria constraints, our approach is extensible well due to linear augment on time consuming when tasks numbers are from 2 to 16. **Fig. 6 (c)** also demonstrates that there is a little increase on computation time when value of variable *K* is changed from 5 to 100, since the complexity impacted by factor of larger candidates is dominant to the value of K .

6.2 Performance Study with the Number of Candidates Varying

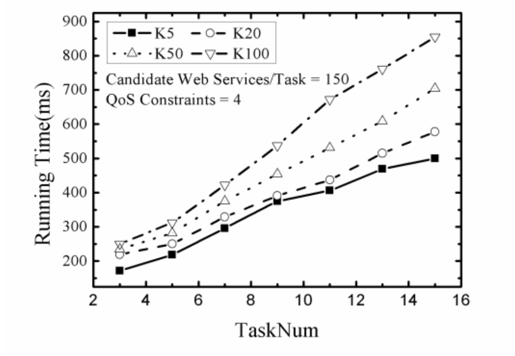
Comparing to the first group of experiments, the third group of experiments based on fixed number of tasks, contains also two experiments. This group of experiments aims to validate two facts, which are of our advantage on global QoS utility and of extensible capability with numbers of candidates increasing from 15 to 300. **Fig. 6 (d)** demonstrates that our approach owns an outstanding extensibility, sincely, within four of our approach-based implementations whose relaxing variables are *K*=5, *K*=20, *K*=50, *K*=100 respectively, their running times all increase near linearly with numbers of candidate web services enlarging for the reason that the calculation complexity of our proposed method is O($m*n*(C_3*k+C_2*m)$). From the **Fig. 6 (e)**, we can see that the global QoS utilities obtained by our approach are better than selection approach proposed by Joyce E.H. in [12][13] when candiate numbers are increased from 15 to 300. Besides, with the increasing value of variable *K*, the values of global QoS utilities obtained by our approach are increased correspondingly, this is because more possible intermediate paths can be kept so as to enable the later selection process.



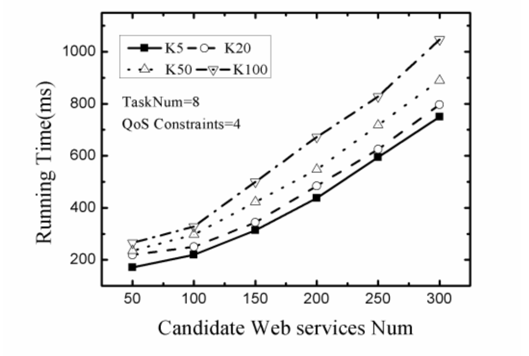
(a) Approximation ratio comparison



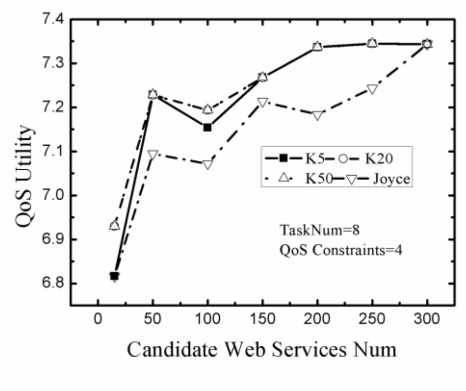
(b) Running time comparison



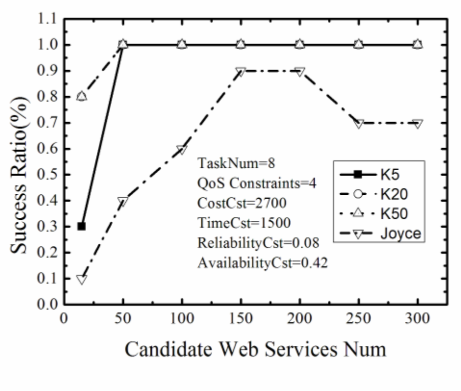
(c) Running time analysis with larger candidate web services.



(d) Running time analysis with increasing candidates number.



(e) Global QoS utility comparison



(f) Success ratio comparison.

Fig. 6. The Performance comparison

The last experiment is conducted to test the success ratio between our approach and the related work. We set the task number to be 8 and the criterion number to be 4. **Fig. 6 (f)** plots success ratios for all our approach-based implementations and the approach with the situation of risk 1[12]13] proposed by Joyce E.H with numbers of candidate web services varying from 15 to 300. We execute each test case 10 times. During different excutions, QoS values of the cost criterion are regenerated. **Fig. 6 (f)** demonstrates that our approach-based implementations are able to find the solutions, especially for the situation with larger candidates numbers. As shown in **Fig. 6 (f)**, when candiate numbers are greater than 50, the success ratios of our approaches are almost near to 1.0. The reason is that our proposed method searches the solution satisfying end-to-end constraints with a global way and employes a better criterion filtering function (as defined in Eq. 7) considering the factor of end-to-end QoS constraints, which can almost obtain a feasible solution obeying end-to-end QoS constraints.

7. Conclusion and the Future Work

Most of traditional QoS-aware WSC approaches ignore the aspect of transactional constraints to guarantee consistency and atomicity of CWS. To address this issue, in this paper, we have proposed a new QoS-aware service selection approach for transactional CWS. Firstly, we have presented two building algorithms to formulate a graph model called actual SCG, which can address the problem of transactional atomicity consistency for CWS. Secondly, several proofs on transactional theories have been presented to show the correctness of our graph model to resolve transactional constraints. After that, we transformed the problem of QoS optimal-utility selection with transactional constraints into the problem of solving single-source shortest paths, and proposed an improved near-optimal service selection algorithm to search the path with optimal QoS utility. Finally, extensive experiments have been conducted to validate efficiency and feasibility of our approach. The experimental results have demonstrated the advantages of our approach.

In our future work, a more complex situation such as the nested transaction in WSC will be considered. Besides, based the proposed approach in this paper, a new selection approach combining risk factor will be studied. Particularly, we will apply the proposed approach to solve a practice problem on ensemble prediction application in scientific computing arena.

References

[1] Sangyoon Oh, Mehmet Aktas and Geoffrey C. Fox, "Mobile Web Service Architecture Using Context-store," *KSII Transactions on Internet and Information Systems*, vol. 4, no. 5, pp. 836-858, 2010. Article (CrossRef Link).

[2] Kwanghoon Kim and Ilkyeun Ra, "e-Lollapalooza: A Process-Driven e-Business Service Integration System for e-Logistics Services," *KSII Transactions on Internet and Information Systems*, vol. 1, no. 1, pp. 33-51, 2007. Article (CrossRef Link).

[3] Liangzhao Zeng, Boualem Benatallah and Marlon Dumas, "Quality Driven Web Services Composition," *in Proc. of the Int. World Wide Web Conf.* , pp.411-421, May 20-24, 2003. Article (CrossRefLink).

[4] Mohammad Alrifai and Thomas Risse, "Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition," *in Proc. of the Int. World Wide Web Conf.*, pp. 881-890, April 20-24, 2009.Article (CrossRef Link).

[5] Florian Rosenberg, Predrag Celikovic, Anton Michlmayr, Philipp Leitner and Schahram Dustdar, "An End-to-End Approach for QoS-Aware Service Composition," *in Proc. of the IEEE Int. Enterprise Distributed Object Computing Conference*, pp. 151-160, October 11-14, 2009. Article (CrossRefLink).

[6] Tao Yu, Yue Zhang and KWei-Jay Lin, "Efficient Algorithms for Web Services Selection with

End-to-End QoS Constraints," *ACM Transactions on Web*, vol. 1, no.1, pp. 1-26, 2007. Article (CrossRefLink).

[7] Daniel A.Menasce and George Mason, "Composing Web Services: A QoS View," *IEEE Internet Computing*, vol. 8, no. 6 pp. 89-91, 2004. Article (CrossRefLink).

[8] Zibin Zheng and Michael R. Lyu, "A QoS-aware fault tolerant middleware for dependable service composition," *in Proc. of the IEEE/IFIP Int. Conf. on Dependable Systems & Networks*, 2009, pp. 239-248. Article (CrossRefLink).

[9] Md Mostofa Akbar, M.Sohel Rahman, M. Kaykobad, E.G. Manning and G.C. Shoja, "Solving the Multidimensional Multiple-choice Knapsack Problem by constructing convex hulls," *Computers & Operations Research*, vol. 33, no. 2006, pp. 1259-1273, 2006.

[10] Liangzhao Zeng, B. Benatallah and A.H.H. Ngu, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311-327, 2004. Article (CrossRef Link).

[11] Y. Tao, "Service Selection Algorithms for Web Services with End-to-End QoS Constraints," *in Proc. of the IEEE International Conf. on E-Commerce Technology*, pp. 129-136, July 6-9, 2004. Article (CrossRefLink).

[12] Joyce EL Haddad, Maude Manouvrier and Marta Rukoz, "TQoS:Transactional and QoS-aware selection algorithm for automatic Web service composition," *IEEE Transactions on Service Computing*, vol. 99, no. (preprints), pp.73-85, 2010. Article (CrossRefLink).

[13] Joyce EL Haddad, Maude Manouvrier and Marta Rukoz, " QoS-driven Selection of Web Services for Transactional Composition," *in Proc. of the Int. Conf. Web Services*, pp. 653-660, September 23-26, 2008. Article (CrossRefLink).

[14] Hongbing Wang, Shizhi Sha and Xuan Zhou, "Web Service Selection with Incomplete or Inconsistent User Preferences," *in Proc. of the 7th Int. Conf. on Service-Oriented Computing*, pp. 83-89, November 24-27, 2009. Article (CrossRefLink).

[15] Mohammad Alrifai, D. Skoutas and T. Risse, "Selecting Skyline Services for QoS-based Web Service Composition," in Proc. of 19th Int. Conf. *World Wide Web*, pp. 11-20, April 26-30, 2010. Article (CrossRefLink).

[16] A. Ceponkus, et al. , "Business transaction protocol version 1.0," available: *http://www.oasis-open.org/committees/business-transactions*, accessed on December 2010.

[17] E. Newcomer, et al. , "Web services coordination(WS-Coordination) version 1.1," available: *http://docs.osis-open/ws-tx/wstx-wscoor-1.1-spec-os.pdf*, accessed on December 2010.

[18] E. Newcomer, et al. , "Web services business activity framework (WS-BusinessActivity) version 1.1," available: http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os.pdf, accessed on December 2010.

[19] E. Newcomer, et al. , "Web services atomic transaction(WS-AtomicTransaction) Version 1.1," av-ailable: http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os.pdf, accessed on December 2010.

[20] D. Bunting, et al. , "Web services composite application framework(WS-CAF)," available: http://www.oasis-open.org/committees/ws-caf/, accessed on December 2010.

[21] Heiko Schuldt, Gustavo Alonso, C. Beeri and H. Schek, "atomicity and isolation for transactional processes," *ACM Transactions on database systems*, vol. 27, no. 1, pp. 1-52, March, 2002. Article (CrossRef Link).

[22] Sami Bhiri, Olivier Perrin and Claude Godart, "Ensuring Required Failure Atomicity of Composite Web Services," *in Proc. of the 14th Int. World Wide Web Conf.*, pp. 138-147, May 10-14, 2005. Article (CrossRefLink).

[23] An Liu, Liusheng Huang and Qing Li, "QoS-Aware Web Services Composition Using Transactional Composition Operator," *in Proc. of Int. Conf. on Web-Age Information Management*, pp. 217-228, Jule 17-19, 2006.Article (CrossRef Link).

[24] J. Cardoso, et al., "Quality of Service for Workflows and Web Service Processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 2004, pp. 281-308, 2004. Article (CrossRefLink).

[25] Michael C.Jaeger, Gregor Rojec-Goldmann and Gero Muehl, "QoS Aggregation for Web Service Composition using Workflow Patterns," *in Proc. of Enterprise Distributed Object computing*

Conf. , pp. 149-159, September 20-24, 2004.

[26] W. M. P. v. d. Aalst, et al., "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5-51, July, 2003. Article (CrossRef Link).

[27] Kaijun Ren and Jinjun Chen, "Optimizing execution path of scientific workflow by gradual removal of QoS constraint violations in Reverse Order," *Concurr. Comput. : Pract. Exper.*, vol. 21, no. 11, pp. 2033-2051, November, 2009. Article (CrossRefLink).

[28] C. L. Hwang and K. Yoon, "Multiple attribute decision making: methods and applications," *Lecture notes in economics and mathematical systems*, vol. 186, Springer-Verlag, March, 1981. Article (CrossRefLink).

[29] Michael C.Jaeger, et al., "QoS Aggregation in Web Service Compositions," *in Proc. of the 2005 IEEE Int. Conf. on e-Technology, e-Eommerce , e-Service*, pp. 181-185, March 29-April 1, 2005. Article (CrossRefLink).

[30] S. Mehrotra, et al., "A Transaction Model for Multidatabase Systems," *Lecture Notes in Computer Science*, vol. 1124, no. 1996, pp. 862-865, 1996. Article (CrossRefLink).

[31] Hai Liu, Weimin Zhang, Kaijun Ren and Zhuxi Zhang, "A Novel Selection Approach for Transactional Web Services Composition," *in Proc. of theNinth International Conference on Grid and Cloud Computing*, pp. 450-456, November 1-5, 2010. Article (CrossRefLink).

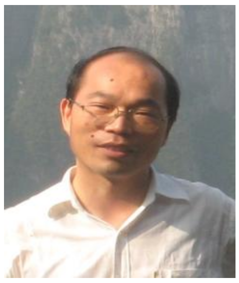
[32] Hai Liu, Weimin Zhang, Kaijun Ren, Cancan Liu and Zhuxi Zhang, "A Risk-Driven Selection Approach for Transactional Web Service Composition," *in Proc. of the Eighth Int. Conf. on Grid and Cooperative Computing*, pp. 391-397, August 27-29, 2009. Article (CrossRefLink).

[33] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," *in Proc. of the 20th Joint Conference of IEEE Computer and Communications Societies*, pp. 834-843, April 22-26, 2001. Article (CrossRefLink).

[34] E. Al-Masri and Q. H. Mahmoud, "Investigating web services on the World Wide Web," *in Proc. of the 17th Int. World Wide Web Conf.* , pp. 795-804, April 21-25, 2008. Article (CrossRefLink).

Hai Liu received his M.S. degree (2006) in School of Computer from National University of Defense Technology, Changsha, China, respectively. He is currently a Ph.D Candidate in School of Computer from National University of Defense Technology. His main research interests include transactional and QoS-aware Web Services composition, concurrently transactional control in Web Service arena and Grid and Cloud computing

Zibin Zheng received his B.Eng. degree and M.Phil. degree in Computer Science from the Sun Yat-sen University, Guangzhou, China, in 2005 and 2007, respectively. He received his Ph.D. degree from the department of Computer Science and Engineering, The Chinese University of Hong Kong in 2010. He received ACM SIGSOFT Distinguish Paper Award at ICSE'2010, Best Student Paper Award at ICWS'2010, and IBM Ph.D. Fellowship Award 2010-2011. He served as program committee member of IEEE CLOUD'2009, CLOUDCOMPUTING'2010, CLOUDCOMPU- TING'2011, CGC'2011, and MCCTA'2011. He also served as reviewer for international journals and conferences, e.g., TSE,TPDS, TSC, IJCCBS, IJBPIM, IJWGS, JSS, JSW, WWW, DSN, KDD, WSDM, CloudCom, ICEBE, SCC, ISAS, QSIC, etc. His research interests include service computing, cloud computing, and software reliability engineering



Weimin Zhang is a professor in School of Computer from National University of Defense Technology, Changsha, China. He received the M.S degree and the PH.D degree in School of Computer from National University of Defense Technology. He has published more than thirty papers now. His research interests include Service Oriented Computing, Grid Computing, Workflow, High performance computing and parallel computing.

Kaijun Ren received the BS degree (1998) in Applied Mathematics, both MS degree (2003) and Ph.D degree (2008) in Computer Science all from the National University of Defense Technology, China. He is currently an associate professor in the School of Computer of the National University of Defense Technology. His main research interests include service oriented computing (service discovery, service composition, QoS-based composition optimal execution), e-science/e-research, workflow management, cloud and high performance computing . He is a member of the IEEE and ACM.