

# Gradual Removal of QoS Constraint Violations by Employing Recursive Bargaining Strategy for Optimizing Service Composition Execution Path

Kaijun Ren<sup>1,3</sup>, Nong Xiao<sup>1</sup>, Junqiang Song<sup>1</sup>, Chi Yang<sup>2</sup>, Min Zhu<sup>1</sup>, Jinjun Chen<sup>3</sup>

<sup>1</sup>College of Computer, National University of Defense Technology, Changsha, Hunan 410073, P.R. China

<sup>2</sup>School of Computer Science & Software Engineering, The University of Western Australia, Perth 6009, Australia

<sup>3</sup>Centre for Information Technology Research, Swinburne University of Technology, Melbourne 3122, Australia

{renkaijun; nongxiao; junqiang}@nudt.edu.cn; cyang@csse.uwa.edu.au; jchen@swin.edu.au

## Abstract

*A critical issue in service composition area is how to achieve an optimized overall end-to-end quality of service(QoS) requirements by effectively coordinating QoS constraints for individual service. However, this issue has not yet been well addressed. In this paper, we propose a novel method by employing a recursive bargaining Strategy to gradually remove QoS constraint violations for Optimizing service composition execution Path. Our method mainly exploits the hidden market competitive relationships which widely exist in real business world for developing a novel bargaining strategy. Based on this strategy, concessions can be made by service providers to offer better QoS values. By recursively using bargaining strategy, an initial execution path built by a local optimization policy for service composition, can be continually updated to be close to the optimal one by reselecting better service providers for meeting overall end-to-end QoS requirements. An experiment and evaluation have been made to demonstrate the feasibility and effectiveness of our proposed method.*

## 1. Introduction

In service oriented computing systems, a business process can be exposed as a composite service which consists of a set of logically connected sub-services. For each service in the composition, many service providers can offer the same function but may different QoS values. In service composition, when a user submits a request, overall QoS constraints called end-to-end QoS requirements, for example, time should be less than one hour, can be transmitted at the same time.

As such, how to effectively coordinate individual QoS constraints for single service to fulfill such overall QoS requirements has been a critical research issue. To address this issue, a lot of research efforts have been carried out [1-8]. However, most of these methods mainly rely on the initial QoS values advertised by service providers, but do not pay much attention to the hidden competition between service providers which usually forces service providers to dynamically adjust their service properties such as time and cost to offer better QoS. For example, in order to maximize the profits, the initial QoS values advertised by service providers are generally somewhat higher than the real ones. Hence, there exists a possibility for service providers to reduce the benefits from their initial QoS proposals if a user launches a bargaining process with them. Particularly, when a user simultaneously bargains with more than one provider, a novel strategy can be used to make service providers be aware that such competition is extremely fierce, and they possibly lose the deal if failing to make any concessions about their initial proposals. Under this pressure, some providers will make QoS adjustments to struggle for the opportunity of winning the deal. Thus, through the recursive bargaining processes, a user can gain better QoS. Unfortunately, current existing methods ignore such competition strategy on resolving overall QoS constraints in service composition. Therefore, they cannot reflect the real behavior of business process properly.

Motivated by the aforementioned problem, we propose a novel method for optimizing service composition execution path by employing the recursive bargaining strategy which can contribute to the correcting of the occurred constraint violations. In

detail, our method first uses a local optimization policy to select the locally-best service provider to construct an initial execution path. Then, the global QoS checking model is employed to determine all occurred QoS constraint violations on this path. Further, for each violation, the newly developed bargaining strategy can be recursively used to correct it. Finally, an optimized execution path can be rebuilt to meet overall QoS requirements.

The remainder of this paper is organized as follows: Section 2 gives the local optimization policy. Section 3 gives the global QoS computing and constraint checking models. After that, the recursive bargaining strategy and algorithm are given in Section 4. Section 5 gives the simulation and evaluation followed by Section 6 introducing the related work. The final section gives the conclusions and future work.

## 2. Building Initial Execution Path by Local Optimization Policy

Assuming a composition plan consists of  $n$  abstract services denoted as  $ws_i (i = 1, 2, \dots, n)$ , and each  $ws_i$  has  $l_i$  candidate service providers; for each  $ws_i$ ,  $c_{ij}$  denotes the  $j$ th service provider.  $q_{jr} (r = 1, 2, \dots, m)$  denotes QoS value of  $c_{ij}$  on the  $r$ th QoS attribute where  $m$  denotes the total QoS attributes. Thus, considering all QoS values of candidate service providers for each  $ws_i$ ,  $(q_{jr}, 1 \leq j \leq l_i, 1 \leq r \leq m)$  leads to a quality matrix  $Q_i$  where each row corresponds to a  $c_{ij}$  while each column corresponds to a quality dimension.

Generally, different  $c_{ij}$  with different QoS values means different benefits to a user. Thus, for a given composition request with end-to-end QoS constraints, a suitable  $c_{ij}$  for each  $ws_i$  should be selected to construct an execution path for achieving the best overall QoS benefits without violating such constraints. However, with an increasing number of services and candidates in a service composition, the possibility blows up exponentially. Therefore, service selection problem for service composition is a computational-hard problem and which can be regarded as a Multiple choice Multiple dimension Knapsack Problem (MMKP) that has been proved np-hard [1, 5, 9].

In our methods, we take the local optimization policy to select the locally best  $c_{ij}$  for matching each  $ws_i$ . For this policy, the basic steps are described as follows. Due to different metric units of different QoS attributes in quality matrix  $Q_i$ , normalization is applied to uniform all different metric so that the ranking of  $c_{ij}$

will not be biased by any QoS attribute with a large value. We use the following equations (1) and (2) which use their averages to normalize different quality dimensions of  $Q_i$ , i.e. equation (1) is used for positive quality dimension and (2) for negative quality dimension.

$$q'_{jr} = q_{jr} / \left( \frac{1}{l_i} \sum_{j=1}^{l_i} q_{jr} \right) \quad (1) \quad q'_{jr} = \left( \frac{1}{l_i} \sum_{j=1}^{l_i} q_{jr} \right) / q_{jr} \quad (2)$$

After the normalization,  $Q_i$  can be transformed into another matrix  $Q'_i = (q'_{jr}, 1 \leq j \leq l_i, 1 \leq r \leq m)$ . Supposing  $ws_r, r = 1, 2, \dots, m$  to be the assigned weights for each QoS attribute  $(\sum_{r=1}^m ws_r = 1)$ , we can rank all the candidate  $c_{ij}$  by the following score formula (3).

$$Score(c_{ij}) = \sum_{r=1}^m q'_{jr} w_r \quad (3)$$

As such,  $c_{ij}$  with the highest score can be selected to best map  $ws_r$ . Further, for all  $ws_r$  included in a service composition, the corresponding  $c_{ij}$  with the highest score can be selected to construct an initial execution path.

To further explain our local optimization policy, we give a stock query example here which will also be used in the following sections as an universal example. Supposing a user is interested in the stock markets and he wants to know the Chinese price of a given stock in London stock marketplace at given time. First of all, he submits his query with overall QoS constraints that the total cost is less than 62\$, and the total response time is less than 35 milliseconds. Upon receiving the user's query, the stock service center automatically arranges a process flow to meet user's functional requirements. This process flow is composed by four abstract services ( $ws_3, ws_4, ws_2$  and  $ws_1$ ) in some inputs/outputs orders as shown in Figure 1.

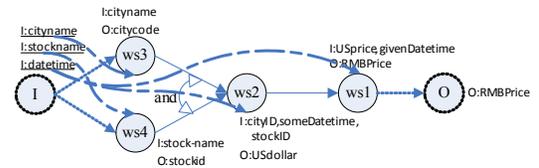


Figure 1. Composition example

Table 1 summarizes the associated service providers with different QoS values for the corresponding abstract services. With Table 1, three different quality criteria, execution cost, execution time and reputation, are taken into consideration. According to the local optimization policy, the algorithm ranks all candidate service providers for each abstract service by calculating their corresponding ranking scores, and they are respectively given in Table 1. As a result,

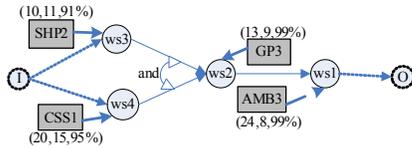
**Table 1.** The execution instances list

Ws3: citytoCode(Input:cityname,Output:citycode)					Ws4: getStockid (Input: stock-name,Output: stockid)				
Quality criteria	User's preference	Service providers			Quality criteria	User's preference	Service providers		
<i>Cost(\$)</i>	0.4	BJP1	SHP2	TJP3	<i>Cost(\$)</i>	0.4	CSS1	CQS2	GDS3
<i>Time(ms)</i>	0.3	12	10	13	<i>Time(ms)</i>	0.3	20	23	18
<i>reputation</i>	0.3	90%	91%	87%	<i>reputation</i>	0.3	15	14	17
<b>Rankscore</b>		<b>0.9934</b>	<b>1.0475</b>	<b>0.9866</b>	<b>Rankscore</b>		<b>1.0220</b>	<b>0.9746</b>	<b>1.0214</b>

Ws2: getstockPrice (Input: cityID,someDatetime,stockID,Output: USDollar)					Ws1: USStoRMB (Input: USPrice,givenDatetime,Output: RMBPrice)				
Quality criteria	User's preference	Service providers			Quality criteria	User's preference	Service providers		
<i>Cost(\$)</i>	0.4	Lp1	NP2	GP3	<i>Cost(\$)</i>	0.4	HKB1	TWB2	AMB3
<i>Time(ms)</i>	0.3	15	16	13	<i>Time(ms)</i>	0.3	21	23	24
<i>reputation</i>	0.3	8	7	9	<i>reputation</i>	0.3	11	9	8
<b>Rankscore</b>		<b>0.9911</b>	<b>1.0033</b>	<b>1.0241</b>	<b>Rankscore</b>		<b>0.9853</b>	<b>1.0012</b>	<b>1.0329</b>

concrete candidates SHP2, CSS1, GP3 and AMB3 with the highest scores for  $ws_3, ws_4, ws_2$  and  $ws_1$  respectively are selected to construct an initial execution path as shown in Figure 2.



**Figure 2.** Initial execution path by local optimization policy  
 However, the initial execution path by local optimization policy is unnecessary to guarantee all end-to-end QoS constraints. Therefore, we need to calculate the global QoS values on this path for checking what constraints have been broken.

### 3. Global QoS Computing and Violation Checking

The goal of global QoS computing model is to aggregate QoS values on each quality dimension on the composition execution path. The literatures in [1, 3, 5, 8, 10] have offered some concrete methods to aggregate QoS values for those computational QoS attributes. Table 2 shows several global QoS computing formulas where  $p$  denotes a composition path which has  $n$  abstract services, each with  $l_i$  ( $i=1,2,\dots,n$ ) candidates;  $\sum_{j=1}^{l_i} x_{ij} = 1$  guarantees that one candidate  $c_{ij}$  from all candidates of  $ws_i$  is selected to offer the corresponding function. From Table 2, the total execution price on path  $p$  equals to the sum of individual price of  $c_{ij}$ . Differently, the execution time is the maximum time among all possible sub paths [1]. More details about other QoS attributes such as reliability have been presented in [1, 3, 5, 8, 10].

**Table 2.** Global QoS Computing Model

<b>price</b>	$q_{pri}^{total}(p) = \sum_{i=1}^n \sum_{j=1}^{l_i} q_{pri}(c_{ij}) x_{ij}, \sum_{j=1}^{l_i} x_{ij} = 1, x_{ij} \in \{0, 1\}$
<b>time</b>	$q_{tim}^{total}(p) = \mathbf{Max}_{sp_m \in p} \sum_{c_{ij} \in sp_m} q_{tim}(c_{ij})$
<b>Reputation</b>	$q_{rep}^{total}(p) = \prod_{i=1}^n (\sum_{j=1}^{l_i} q_{rep}(c_{ij}) x_{ij})$

As mentioned in Section 2, the initial execution path by local optimization policy is unnecessarily the required one. Therefore, for the selected execution path, constraint checking should be conducted. We use global QoS computing models to judge whether QoS constraint violations occur. In our former example, the following equations (4), (5) and (6) are applied to check whether user's submitted QoS constraints on the execution price, time, and reputation are violated.

$$q_{pri}^{total}(p) \leq \text{price}_{user}, \quad (4) \quad q_{tim}^{total}(p) \leq \text{time}_{user}, \quad (5)$$

$$q_{rep}^{total}(p) \geq \text{reputation}_{user}, \quad (6)$$

Considering the same stock composition example, the global QoS values on the initial path in Figure 2 can be calculated by the global QoS computing formulas of Table 2. After this process, the total execution cost is 67\$, the estimated execution time is 32 milliseconds, and the total reputation is 85%. However, QoS requirements by a user are expected that the execution fee is less than 62\$, and the total response time is less than 35 milliseconds. Apparently, the total execution cost 67\$ on the selected path has exceeded the required price 62\$. Hence, price violation has occurred. In the following Sections, concrete strategies will be discussed to correct such violations.

## 4. Constraint Violation Corrections by Recursive Bargaining Strategy

Intuitively, the execution path by local optimization policy will be the best one without considering user's QoS constraints. Further, if we can correct all occurred constraint violations, the modified execution path will be a good choice to meet end-to-end QoS constraints.

### 4.1. Overall Correction Algorithm

Algorithm 1 gives the key procedures for correcting global QoS constraint violations. At the beginning, the algorithm uses the local optimization policy to produce an initial execution path called *LOIEPath* for a user-requested composition plan (step 1). Then *LOIEPath* can be temporarily regarded as a global optimal execution path called *GOIEPath* (step 2). For *GOIEPath*, the algorithm begins to correct all occurred constraint violations. First, the global QoS values on all quality dimensions need to be respectively calculated according to the global QoS computing formulas introduced in Section 3. These overall QoS values are kept in the vector called *QoSVector[]* (step 3). Then the algorithm checks all QoS constraint violations according to user's end-to-end QoS requirements, and all occurred violations are kept in the vector called *VioVector[]* (step 4). Particularly, in *VioVector[]*, these constraint items are ranked due to user's preferences. The purpose on doing this is to guarantee that more important constraint violations can be corrected in a high priority and the later violation correction cannot breach the previously-corrected constraints. Therefore, after each violation correction, the global QoS values should be recalculated for the remaining violation corrections. In step 5, the algorithm begins to process concrete constraint corrections. At first, the algorithm calculates the ideal negotiation space on each violated QoS constraint, and it equals to the difference value between global QoS value and user-required constraint value (step 6). In step 7, the algorithm begins to judge what nodes are critical nodes which have strong impacts on the corresponding constraint violation. For example, with respect to the price violation, the algorithm can rank all nodes in *GOIEPath* according to their individual price. The higher the price is, the more important impact the node has. In step 7, the vector called *keyNodeVector[]* records the ranked critical nodes. For each critical node of *keyNodeVector[]*, there is an associated service provider. Then, the algorithm relocates all other functionally-equivalent service providers, and regards them as bargaining objects stored in the vector called *equCandidateVector[]* (step 9).

From step 10 to 17, the algorithm uses a concrete bargaining strategy (discussed in the following section) to achieve a better service provider which can bring down the occurred constraint violation. In step 18, the algorithm updates the corresponding critical node on

Algorithm 1: ConstraintViolationCorrection

---

```

Inputs: abstractProcessCompositionPlan
Outputs: GOIEPath //Global Optimal Instance Execution Path
1  LOIEPath ← LocalOptimalExecutionPathSelection(abstractProcessCompositionPlan);
   //Getting an initial execution path by local optimization policy
2  GOIEPath ← LOIEPath;
3  QoSVector[] ← globalQoSComputing(LOIEPath);
   //Calculating the global QoS values according to global QoS computing formulas
4  VioVector[] ← constraintCheck(QoSVector[]);
   //Calculating the occurred constraint violations according to global constrain checking model
5  FOR i = 1 to VioVector[].size() DO
6    //For each constraint violation
   negoSpaceVector[i] ← globalQoSValue - userconstraintValue;
   //Calculating the negotiation space for the corresponding constraint violation
7    keyNodeVector[] ← keyNodesRanking(VioVector[i]);
   //Selecting and ranking critical nodes which have a big influence on the constraint violation
8    FOR j = 1 to keyNodeVector[].size() DO
9      equCandidateVector[] ← equCandidateSearch(keyNodeVector[j]);
   //Selecting all functionally-equivalent candidates which have the same associated abstract serv
10     QoSBargaining(equCandidateVector[]);
   //Launching the bargaining process among all candidate service providers
11     QoSUpdating(); //Updating QoS values after each round bargaining
12     while (!deadline() || violationRemoved())
13       {
14         rebargaining(); //Launching new round bargaining
15         QoSUpdating();
16       } //Bargaining can be done recursively until no better QoS values can be achieved
17     supplierRanking(); //Ranking all candidates according to the new bargaining results
18     supplierSelecting(); //Reselecting the suitable candidate
19     updateNegotiationSpace(negoSpaceVector[j]);
   //Updating the negotiation space for next critical node
20     GOIEPath ← executionNodeAdjusting(GOIEPath);
   //Modifying the execution path with new better service provider
21   End For
22   updatingGlobalQoSValues(GOIEPath);
   //Recalculating the changed global QoS values on the new path
23 End For
24 Return GOIEPath;

```

---

the execution path *GOIEPath*. Loop step 8 to 19 will be executed till a constraint violation can be eliminated. In step 20, the global QoS values should be recalculated on the new execution path for judging the possible existence of constraint violations. Step 5 to 21 will repeatedly be executed until all constraint violations are removed. Finally, the algorithm returns the optimized execution path.

### 4.2. Recursive Bargaining Strategy

In market fields, some competitive bargaining policies have been proposed to balance the profits between customers and business providers[11, 12]. Therefore, bargaining policy has played an important role in the business fields. As mentioned in the step 9 of algorithm 1, all candidates associated with the same critical node can be selected to be regarded as bargaining objects. Generally, in order to maximize the profits, the initial QoS values advertised by service providers are somewhat higher than the real ones. Hence, there exists a possibility for service providers to reduce the benefits from their initial QoS proposals if a user launches a bargaining process with them. Figure 3 shows our bargaining agent model. The basic strategies here let a user agent bargain simultaneously with more than one service provider (supplier), and make

suppliers compete each other to create a pressure on them for offering a better QoS values. Initially, the bargaining agent notifies candidate service providers that all of them have opportunity to offer services for the user, but the competition is extremely fierce. If they are willing to make concessions by decreasing the benefit from their initial proposals, the probability of gaining this deal will be increased. In addition, different candidate suppliers may receive different messages, which can make service providers be aware that what QoS values should be urgently updated to improve their overall ranking. The bargaining process can be executed recursively until no any concession is made in the predefined deadline or a constraint violation has been removed. However, there also exists the possibility that the constraint violation cannot be corrected even through multiple bargaining. When this occurs, it means the user's corresponding constraint limitations are too strict that no connected suppliers can meet these constraints. Therefore, the bargaining agent notifies the user to reset his constraint conditions.



Figure 3. Bargaining Agent Model

Now we introduce how the bargaining agent makes suggestions and launches the bargaining process toward candidate suppliers. Especially, we partly make use of the technique in [13] to devise our bargaining strategy. The core procedures are shown in Figure 4. First, the bargaining agent accepts the concrete constraint violation and the negotiation space (step 1). Then the bargaining agent begins to select one of critical nodes for bargaining (step 2). After all associated candidates have been located, the bargaining agent starts to rank them (step 3). The ranking algorithm still employs equation 3 to calculate the scores of all corresponding candidates by user's preferences. In step 4, the bargaining agent communicates with these suppliers and simultaneously makes some suggestions to encourage them to provide better QoS values to run up their ranks in the trading. For example, a message dialog received by a supplier could be like: "Thank you for your information. After comparing your service with others, we found your service currently at the 3rd rank in our group evaluation. If you could decrease the price by 20\$, then your service's rank will be moved to the front. Would you please consider this suggestion?" To win the service in a fierce competition business world, some of the suppliers will make concessions and their ranks are recalculated accordingly (step 5). This reranking and

bargaining process continues until there is no better QoS proposed or there is no response in a predefined deadline (step 4 and step 5). After finishing the bargaining, the agent can select the supplier with the highest rank under the condition that the new violation space is less than the original space (step 6). Notably, the replacement of the original supplier may not remove the constraint violation but only reduce the violation space. Consequently, bargaining agent expects the replacement of supplier on the next critical node can continually contribute to bring down this violation. For this aim, the bargaining agent recalculates the remaining negotiation space and sends it to next critical node for new bargaining (step 7). Thus, step from 2 to 7 are processed recursively until all of the selected critical nodes have finished the bargaining or the corresponding constraint violation has been removed. Finally, for each constraint violation, we can achieve two results: either the violation will be successfully eliminated, or user's constraint conditions are too strict that there are no combined suppliers to meet the constraint (step 8). Under this situation, the bargaining agent will send a message to notify a user for relaxing his constraint conditions on the corresponding quality dimension.

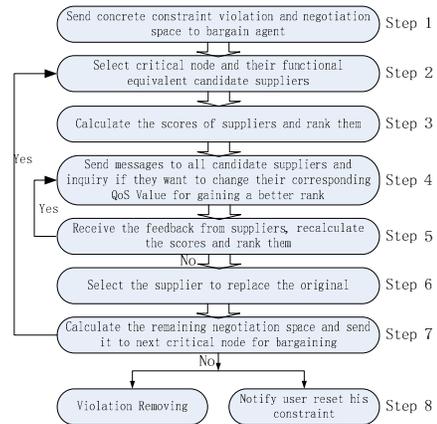


Figure 4. Bargaining procedures

### 4.3. Example Run

Through the global QoS computing and constraint checking of the universal example of Figure 2, there exists a cost violation  $5\$(67-62)$  between the initial path and the user. Hence, for cost violation, the initial negotiation space is  $[0, 5\$]$ . Then the algorithm ranks critical nodes on this path according to their importance on this violation. For example, the ranked sequence of critical nodes is  $[AMB3, CSS1, GP3, SHP2]$  by the price value. The bargaining agent now starts the bargaining process as follows. First, the bargaining agent selects critical node AMB3 for

bargaining. Further, the bargaining agent locates its functionally-equivalent service providers HKB1, TWB2, and regards them as bargaining objects. These bargaining objects are firstly ranked by equation 3 as the order: AMB3>TWB2>HKB1. In the next step, the bargaining agent begins to make suggestions and send different messages for the related providers. First, a QoS improvement factor for the providers to improve their rankings is to be found. For example, from Table 1, the improvement factor for AMB3 will be ‘price’ while the factor for HKB1 will be ‘time’ because the respective improvement of ‘price’ and ‘time’ will probably move their overall ranking to the front. The bargaining agent then sends different message to these providers through their supplier agent and asks if they can make some concessions on the specified factor item in order to improve their evaluation rank to win the deal.

When the service providers receive these messages from the bargaining agent, they probably reduce their initial benefits by lowering price or improving other quality to improve their ranking. The bargaining agent waits for the providers’ responses till the deadline, then recalculates, and decides whether it should continue to bargain with those providers. This bargaining process is repeated until no more providers would like to make any concessions to adjust their QoS values or the deadline has expired. Finally, the bargaining agent reevaluates the ranking of all candidates, and the service provider with higher ranking is selected to replace the original AMB3 under the condition that its current price is less than the price of AMB3. As a result, the new updated response time of HKB1 is 8ms which causes a higher ranking than AMB3. Therefore, the service provider HKB1 with fewer costs 21\$ will be selected to replace AMB3. Thus, the global price violation space is reduced by 3\$(24-21). Consequently, a new negotiation space [0, 2\$(5-3)] is sent to next critical node CSS1 for further bargaining. Finally, either the price violation is removed or user’ constraint condition is too strict to be met so that the bargaining agent notifies the user to reset his constraint condition.

## 5. Simulation and Evaluation

**Test Case Generation:** As a basis, we use a service test collection from QSQL previously built in[14] where 3500 services have been included to generate service composition flows. Additionally, candidate service providers are produced as a random value [3, 10] for being associated with corresponding abstract services. Particularly, 5 concrete QoS attributes such as price, time, and reliability, which are randomly

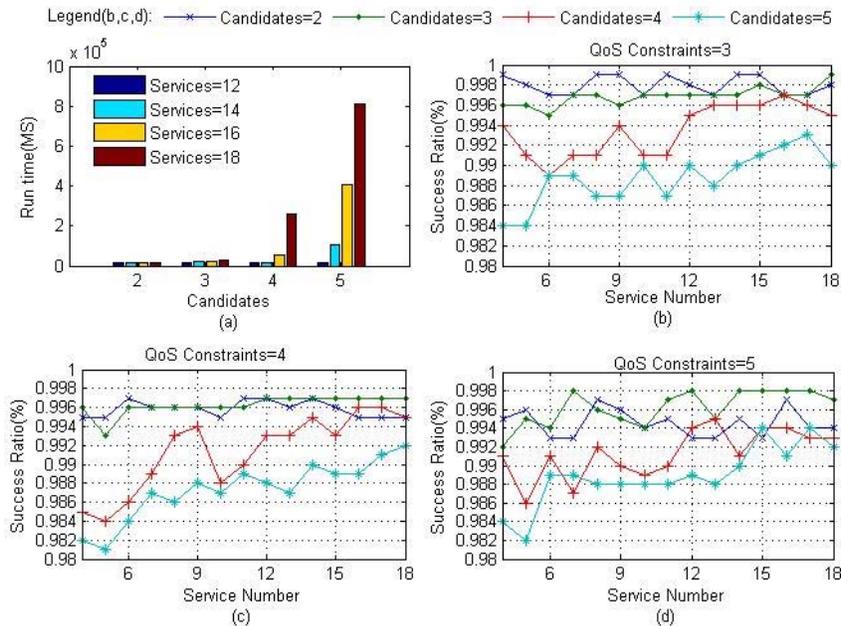
generated with a uniform distribution between [1,100], are assigned with each service provider. Simultaneously, QoS values of service providers were adjusted in a randomly specified time interval during the bargaining process.

**Performance Analysis:** We use the total scores to evaluate the overall performance of each execution path in a service composition. For a composite service that has  $n$  abstract services, each with  $l_i$  ( $i=1,2,\dots,n$ ) candidate service providers, the total score function is defined as follows:

$$F^k = \sum_{i=1}^n \sum_{j=1}^{l_i} score(C_{ij}^k) x_{ij}, \quad \sum_{j=1}^{l_i} x_{ij} = 1, \quad x_{ij} \in \{0,1\} \quad (7)$$

where,  $F^k$  denotes the total scores of the  $k$ th selected execution path;  $C_{ij}^k$  denotes such selected service provider, and the definition of  $score(C_{ij}^k)$  is the same to equation 3 of Section 2. Thus,  $F^k$  equals to the sum of all scores of individual score on the  $k$ th selected execution path. Thus, the higher the score of  $F^k$ , the better performance the  $k$ th selected execution path has.

To compare the performance of the execution path by our methods from the best path, some limitations such as composition length for each query have been made; thus exhaustively searching each execution path from all possibilities becomes computable and the best execution path can be picked out. In this simulation, 15 composition queries involved by different service number from 4 to 18 are generated. Additionally, for each query, we respectively limit the candidate provider number from 2 to 5. The final results include two parts: run time and success ratio. Figure 5(a) shows the time consumption trend when exhaustively searching all possible paths among 4 data sets grouped by candidate service provider number from 2 to 5. Within each set, different query including different service number from 12 to 18 is involved. As shown in Figure 5(a), whatever the view is within the sets or across the sets, the searching time tends to grow exponentially. Contrarily, our methods only detect a small bunch of paths, which will be more realistic. To demonstrate how well our proposed methods are, we give the comparison of the success ratio, which is measured by considering the ratio of the scores of the generated path by our methods and the scores of the best path. Figure 5(b) shows the average success ratio trend for 15 queries including 4 to 18 service number with 3 QoS constraints. First, we can see that the least success ratio is around 98.4%. Additionally, with the increasing number of candidate service providers from 2 to 5, the success ratio decrease slightly. Dramatically,



with the increasing service number, the success ratio tends to be increasing trend. To illustrate the effects of different QoS constraints on success ratio, constraint number from 3 to 5 is respectively applied to each query to test the overall performance. As we see in Figure 5(c) and (d) that with the increasing constraint number, the success ratio goes down slightly. However, with the increasing service number, the success ratio trends are similar to the increasing trend in Figure 5(b). To sum up, the execution path built by our methods is near to the optimal one.

## 6. Related Work

Over the last recent years, many representative works concerning QoS-based service selection for the execution of service composition have been proposed. To the best of our knowledge, we list the relevant literatures as follows. Danilo Ardagn in [1] introduces a significant modeling approach basing on his earlier work[15] for resolving web service selection problem at run time. In this method, negotiation and reoptimization techniques are exploited to identify a feasible solution of the composition problem if there are no possible solutions. However, the authors don't give the details about when and how to judge whether there exist infeasible solutions. Such a judging problem is probably another np-hard problem since searching an optimal execution path from multiple choices is np hard. Another service selection problem with multiple QoS constraints is investigated by Yu in [5]. With this method, the combinatorial-based and the graph-based techniques are employed. A drawback

concerning the proposed algorithms is that the complexity of the graph model's algorithms is not polynomial but exponential, while ours is polynomial. Swaroop in [2] proposes a dynamic and graph-theory-based service composition approach aiming at capturing the features and useful characteristics of resources in a pervasive computing environment. Although this method limits the length of composition with a predefined number to overcome a np-selection problem, the unresolved problem is that what number should be predefined, and the wrong predefined number is easy to cause to a failure of finding a successful execution process path. The problem that Liu et al. [16] attempt to solve is the provision of a framework for an open, fair, dynamic and secure framework that aims to evaluate the QoS of a vast number of web services. However, the problem of providing a concrete selection of WSs mechanisms is still unresolved. Earlier work by Zeng et al.[3] defines multiple QoS criteria and takes into account of global constraints. With this method, the Integer Linear Programming (ILP) method is used to find the optimal selection. However, ILP has a high-time complexity and may not be suitable for systems with many services and dynamic service needs. The literature in [17] presents a approach to perform the reoptimization at runtime execution when the QoS of a Web Service may have dynamic changes because of internal changes or because of workload fluctuations. This approach is useful when the execution exceptions occur. However, the authors avoid the np selection problem before resolving reoptimization. The authors in [7] employ a heuristic algorithm to solve the NP-

hard problem for syndicating web services. However, this method is only useful in a limited business field. Some work is also proposed to maximize provider's profits or user's profits [6, 18], while our work emphasize a nature balance on the profits between user and providers according to the market competition relationship.

## 7. Conclusions and Future Work

Existing typical service composition methods ignore the hidden competition relationships between service providers on resolving overall QoS constraints in service composition. Therefore, they cannot reflect the real behavior of business process properly. To address this problem, we have developed a novel method by employing a recursive bargaining strategy to gradually remove QoS constraint violations for Optimizing service composition execution Path. Based on the recursive bargaining strategy, a competition pressure can be formed to push service providers to make concessions for offering better QoS values. By applying the bargaining strategy, QoS constraint violations on the built execution path by local optimization policy can be corrected to achieve an optimized result. An experiment and evaluation have been conducted to demonstrate the feasibility and effectiveness of our proposed method. In conclusions, our method can reflect the behavior of real world business more practically and reasonably than other methods. In future, we will consider turning our method into concrete business applications.

**Acknowledgement:** we are grateful for the foundation support by the National "973" Research Plan Foundation of China under Grant No. 2003CB317008, NSFC60736013,863-2006AA01A106 and by Swinburne Dean's Collaborative Grants Scheme 2008-2009.

## 8. References

- [1] Danilo Ardagna Barbara Pernici. Adaptive Service Composition in Flexible Processes. *IEEE Transaction on Software Engineering*. 33(6): p. 369-383. 2007
- [2] Swaroop Kalasapur, Mohan Kumar, et al. Dynamic Service Composition in Pervasive Computing. *IEEE Transaction on Parallel and Distributed Systems*. 18(7): p. 907-918. 2007
- [3] Liangzhao Zeng, Boualem Benatallah, et al. QoS-Aware Middleware for Web Services Composition. *IEEE Transaction on Software Engineering*. 30(5): p. 311-327. 2004
- [4] Raouf Boutaba Jin Xiao. QoS-Aware Service Composition and Adaptation in Autonomic Communication. *IEEE Journal on Selected Areas in Communications*. 23(12): p. 2344-2360. 2005
- [5] Tao Yu, Yue Zhang, et al. Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints. *ACM Transactions on the Web*, Vol. 1, No. 1, Article 6, Publication date: May 2007. 1(1): p. 6:1-6:26. 2007
- [6] Dimitrios Tsesmetzis, Ioanna Roussaki, et al. Modeling and Simulation of QoS-aware Web Service Selection for Provider Profit Maximization. *Simulation*. 83(1): p. 93-106. 2007
- [7] Yi Sun, Shaoyi He, et al. Syndicating Web Services: A QoS and user-driven approach. *Decision Support Systems*. 43(1): p. 243-255. 2007
- [8] Jorge Cardoso, Amit Sheth, et al. Quality of service for workflows and web service processes. *Journal of Web Semantics*. 1(3): p. 281-308. 2004
- [9] Xiaohui Gu Klara Nahrstedt. On Composing Stream Applications in Peer-to-Peer Environments. *IEEE Transactions on Parallel and Distributed Systems*. 17(8): p. 824-837. 2006
- [10] Michael C. Jaeger, Gregor Rojec-Goldmann, et al. QoS Aggregation for Web Service Composition using Workflow Patterns. In the proceeding of the 8th IEEE International Enterprise Distributed Object Computing Conference. California, USA, September 2004: IEEE Computer Society Press, Washington, DC, USA
- [11] John Debenham. Multi-issue bargaining in an information-rich context. *Knowledge-Based Systems*. 17(2-4): p. 147-155. 2004
- [12] J. Dávila J. Eeckhout. Competitive bargaining equilibrium. *Journal of Economic Theory* 139(1): p. 269-294. 2008
- [13] Dengneng Chen, B. Jeng, et al. An agent-based model for consumer-to-business electronic commerce. *Expert Systems with Applications*. 34(1): p. 469-481. 2008
- [14] Kaijun Ren, Xiao Liu, et al. A QSQL-based Efficient Planning Algorithm for Fully-automated Service Composition in Dynamic Service Environments. In the proceeding of 2008 IEEE International Conference on Services Computing(SCC Research Track 2008). Honolulu, Hawaii, USA, July 2008: IEEE Computer Society Press, Washington, DC, USA
- [15] Danilo Ardagna Barbara Pernici. Global and Local QoS Guarantee in Web Service Selection. In the proceeding of the 3rd International Conference on Business Process Management Workshop (Lecture Notes in Computer Science, vol. 3812). Nancy,France, September 2005: Springer-Verlag: Berlin, Germany
- [16] Yutu Liu, Anne H.H. Ngu, et al. QoS Computation and Policing in Dynamic Web Service Selection. In the proceeding of the 13th International World Wide Web Conference. New York, USA, May 2004: ACM Press
- [17] G. Canfora, M. Penta, et al. QoS-Aware Replanning of Composite Web Services. In the proceeding of 2005 IEEE International Conference on Web Service. Orlando,USA, July 2005: IEEE Computer Society Press, Washington, DC, USA
- [18] L. Zhang D. Ardagna. SLA-Based Profit Optimization in Autonomic Computing Systems. In the proceeding of the 2nd International Conference on Service Oriented Computing. New York, USA, November 2004: ACM Press