Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints

Tao Yu and Kwei-Jay Lin

Dept. of Electrical Engineering and Computer Science,
University of California, Irvine, California 92697-2625, USA

Abstract. One of the promises of the service-oriented architecture (SOA) is that complex services can be composed using individual services. Individual services can be selected and integrated either statically or dynamically based on the service functionalities and performance constraints. For many distributed applications, the runtime performance (e.g. end-to-end delay, cost, reliability and availability) of complex services are very important. In our earlier work, we have studied the service selection problem for complex services with only one QoS constraint. This paper extends the service selection problem to multiple QoS constraints. The problem can be modelled in two ways: the combinatorial model and the graph model. The combinatorial model defines the problem as the multi-dimension multi-choice 0-1 knapsack problem (MMKP). The graph model defines the problem as the multi-constraint optimal path (MCOP) problem. We propose algorithms for both models and study their performances by test cases. We also compare the pros & cons between the two models.

1 Introduction

Web services present a promising technology to compose complex service applications from individual (atomic) services. Using Web services, distributed applications and enterprise business processes can be integrated by individual service components developed independently. The service components may also be upgraded or replaced dynamically at run time as system conditions change or applications' needs evolve. The enhanced service composability provides a desirable flexibility and reusability in building distributed enterprise or grid solutions. This is important for enterprise computing since the fast and dynamic construction of business processes (for supply chain or service network) is essential for companies in order to adapt their operations to dynamic market conditions. Similar needs exist in global transaction systems, health care and travel industry.

However, the composition flexibility comes at the price of increased system engineering complexity. The complexity of Web service composition includes three main factors: (1) the large number of atomic services that may be available; (2) the different possibilities of integrating atomic service components into a complex service; (3) various performance requirements (e.g. end-to-end delay,

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 130–143, 2005.

service cost, server capability) of a complex service. Web service composition thus creates a QoS engineering problem since the service selection must select the best services to compose an efficient complex service.

In recent years, it has become a common practice for service providers to offer different service levels so as to meet the needs of different customers. Companies have offered different service qualities (e.g. first class vs. coach class, gold card member vs. regular member) based on user qualifications or service costs. Similarly, although many atomic services have a similar functionality (e.g. checking market condition, making reservations, planning meetings, etc.), they differ from each other by non-functional qualities, such as service time, transaction cost, and system availability. The QoS of a Web service may be offered by different service level agreements (SLA) between service providers and clients [5].

In our study, we have proposed a broker-based framework (QCWS) for QoS-aware Web service composition [16]. In QCWS, Web service composition with QoS assurance includes two steps: *service planning* and *service selection*, which are are performed by the *Composition Manager (CM)* and *Selection Manager (SM)* in the QoS broker, respectively. In [15], we study the service selection problem for complex service with one QoS requirement. In this paper, we extend the system model to handle multiple QoS requirements. We study this problem using two different models: *the combinatorial model*, by defining the problem as a multi-dimension multi-choice 0-1 knapsack problem (MMKP) and *the graph model*, by defining the problem as a multi-constraint optimal path problem (MCOP). The objective of service selection is to maximize a user-defined utility function under the overall QoS constraints. The utility function definition may include an extended set of system parameters to achieve some user specific objective. We propose several service selection algorithms and report simulation results to compare their performances.

The rest of this paper is organized as follows. Section 2 reviews some related work. Section 3 presents the system model and assumptions for the Web service composition with QoS assurance in our study. Section 4 discusses various algorithms in both combinatorial and graph approaches, including heuristic and optimal ones. Section 5 shows the performance evaluation and comparison of different algorithms. The paper is concluded in Section 6.

2 Related Work

Web service composition has received much interest for supporting enterprise application integration. Many industry standards have been developed, such as BPEL4WS (Business Process Execution Language for Web Services) [4] and BPML (Business Process Modelling Language) [2]. Many projects have studied the Web service composition problem. The SWORD project [13] gives a simple and efficient mechanism for offline Web service composition using a rule-based expert system. SWORD is more focused on the service interoperability and no QoS issue has been addressed. The eFlow project [3] provides a dynamic and adaptive service composition mechanism for e-business process management. In

eFlow, each service node contains a search recipe, which defines the service selection rule to select a specific service for this node. The selection rule is based on local criteria and does not address the overall QoS assurance problem of the business process.

QoS guarantee for Web services is one of the main concerns of the SLA framework [5]. The framework proposes differentiated levels of Web services using automated management and service level agreements (SLAs). The service levels are differentiated based on many variables such as responsiveness, availability and performance. An initial version of the framework was released as part of the IBM Emerging Technologies Toolkit (ETTK) version 1.0 in April 2003. Although it included several SLA monitoring services to ensure a maximum level of objectivity, no end-to-end QoS management capability was implemented.

There are projects studying QoS-empowered service selection, such as [17] and [1]. In [17], authors propose a quality driven approach to select component services during execution of a composite service. They consider multiple QoS criteria such as price, duration, reliability and take into account of global constraints. [1] has studied a similar approach. Both of them use the integer linear programming method to solve the service selection problem, which is too complex for run time decisions.

3 System Model and Assumptions

We assume that the same service interface definition is used by all atomic service candidates for a specific service component. So we are not concerned about the compatibility issue among services and focus on the QoS service selection problem. In this study, we define the concept of *service class*. A *service class* (denoted as S) is a collection of atomic Web services with a common functionality but different non-functional properties (e.g. time, quality). A class interface parameter set (S_{in}, S_{out}) is defined for each service class. We also assume each atomic Web service (denoted as s) in the service class can provide a service according to the class interface.

Each atomic service may provide $\mathcal{L}$ different service levels; each level is associated with a QoS vector $q(s,l) = [q^1(s,l), .., q^n(s,l)]\,(1 \leq l \leq \mathcal{L})$ which contains n application-level QoS parameters such as service time, cost, reliability, availability [5]. Each service level is a candidate in the service class for service selection. Each service level also has an associated utility function $\mathcal{F}$. The utility function is defined by a set of system parameters including system load, cost and/or other QoS attributes. The system load can be considered by a benefit function ([15]). Definition 1 shows the utility function definition. Users can set the number of QoS values to be considered as well as their weights according to their requirements. In our study each user has m QoS attribute constraints in their QoS requirements: $\mathbb{Q}_c = [Q^1, .., Q^m]\,(1 \leq m \leq n)$.

Definition 1 (Utility Function). *Suppose there are α QoS values to be maximized and β QoS values to be minimized. The utility function for candidate k in a service class is defined as:*

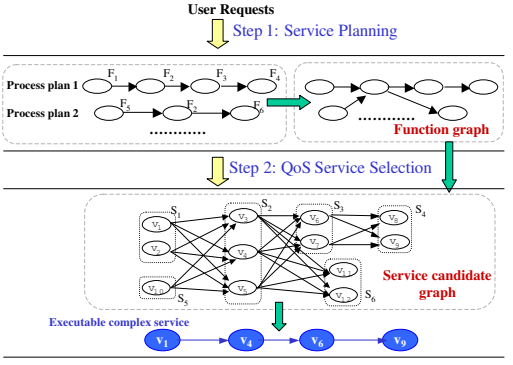


Fig. 1. QoS Web service composition model

$$\mathcal{F}(k) = \sum_{i=1}^{\alpha} w_i * (\frac{q_{ai}(k) - \mu_{ai}}{\sigma_{ai}}) + \sum_{j=1}^{\beta} w_j * (1 - \frac{q_{bj}(k) - \mu_{bj}}{\sigma_{bj}})$$

where w is the weight for each QoS parameter set by a user ($0 < w_i, w_j < 1$, $\sum_{i=1}^{\alpha} w_i + \sum_{j=1}^{\beta} w_j = 1$, $\alpha + \beta = m$). μ and σ are the average value & the standard deviation of the QoS attribute for all candidates in the service class.

In our study, the QoS Web service composition is conducted in two steps: *service planning* and *service selection,* as shown in Figure 1. For each user request, the Composition Manager in the QoS broker first matches the request with one or more process plan(s). Each of them is an abstract process that defines a flow of component functions ($\mathcal{F}$, each can be accomplished by a service class) as well as their relationships. All potential process plans together constitute a function graph. The Selection Manager then maps the function graph into a service candidate graph and constructs an executable complex service. The mapping from a user request to process plans (*step 1*) only considers the functional requirements of the user request and does not handle the QoS requirement. It performs the parametric consistency checking of service classes in order to integrate them with each other. This problem has been addressed by several research work such as [13, 6]. The mapping from a function graph to an executable complex service (*step 2)* is decided by the distributed performance of services and a user's QoS requirements. Step 2 is the focus of this paper.

The QoS attributes of the complex service are decided by the QoS attributes of its component services as well as their integration relationships, such as sequential, parallel, conditional or loop. In this paper, we only consider the sequential composition model in which the QoS attribute and the utility of the complex service is the sum of its component services' QoS attributes and utilities at the selected service level. If a QoS attribute is the product of its component QoS, such as reliability and availability, we can apply a logarithm operation to convert it into a summation relationship. For QoS attribute with convex/concave

characteristic, it may be processed by a filter operation and is not considered in this paper. Other composition models, such as parallel, conditional or loop, may be reduced or converted to the sequential model.

4 Service Selection Algorithms

In this section, we present the service selection algorithms used by the QoS broker for service composition with two or more QoS constraints. There are two models to solve the problem: *the combinatorial model* and *the graph model*.

4.1 The Combinatorial Algorithm

The *Multi-choice, Multi-dimension 0-1 Knapsack Problem (MMKP)* [12] is defined as follows: Suppose there are K groups, each has $l_i\,(1 \leq i \leq K)$ items, where each item has a profit p_{ij} and requires resource $r_{ij} = (r_{ij1}, ..r_{ijm})$. The total amount of available resources in the knapsack are $R = (R_1, .., R_m)$. The objective of MMKP is to select exactly one item from each group to be included in the knapsack within the resource constraint while maximizing the total profit.

For a complex service that contains N service classes $(S_1, S_2, ..., S_N)$ in a process plan and with m QoS requirements, the service selection problem can be mapped to an MMKP as follows: (1) each service class can be viewed as a group in MMKP; (2) every candidate in a service class represents an item in a group; (3) the QoS attributes of each candidate are equivalent to the resources needed by the item; (4) the utility a candidate produces is mapped to the profit of the item; (5) a user's QoS requirements are considered as the available resources of the knapsack. The objective of service selection is to select one candidate from each service class to construct a complex service that meets a user's QoS requirements yet maximize the total utility. The problem is formulated as:

$$\begin{aligned} Max \quad & \sum_{i=1}^{N} \sum_{j \in S_i} \mathcal{F}_{ij} x_{ij} \\ Subject\ to & \sum_{i=1}^{N} \sum_{j \in S_i} q_{ij}^{\alpha} x_{ij} \leq Q^{\alpha}\ (\alpha = 1, .., m) \\ & \sum x_{ij} = 1 \\ & x_{ij} \in \{0, 1\} \quad i = 1, ..., N,\ j \in S_i \end{aligned} \tag{1}$$

The MMKP problem is NP-hard [12]. [7] proposes a branch and bound algorithm (BBLP) to find the optimal solution for MMKP. The branch and bound method uses a search tree to find a solution. A node in the search-tree represents a solution state where some classes are fixed (an item has been chosen in these classes) and some others are free (no item has been selected). A node that has free classes may be expanded to generate new nodes. BBLP has a very high

complexity and is not suitable for large size problems. The detailed description about the BBLP algorithm can be found in [7].

A heuristic algorithm (HEU) for MMKP has been presented in [8]. The idea of the algorithm is to find a feasible solution at first, then iteratively improve the solution by replacing items with a low utility with items with higher utilities in each group while keeping the solution feasible. If no such item can be found, it tries to replace items with a higher utility in a group (which makes the solution infeasible) followed by replacing items in other groups with a lower utility and less resource requirements to keep the solution feasible. This method of upgrades followed by downgrades may increase the total utility of the solution.

We modify HEU by always selecting a feasible solution (if one exists) at first without considering those infeasible ones. (In HEU, an infeasible solution may be picked at first and iterations are needed to make it feasible.) The modification can shorten the algorithm execution time. The modified algorithm WE_HEU is presented in Algorithm 1. The algorithm is used to select services for one process plan. If a user request can be matched with more than one process plans, we need to apply the algorithm to every plan and produce several executable complex services. Among them, the one with the highest utility is the final solution.

Algorithm 1. WS_HEU

Step 1: Select item ρ_i from each group i $(i = 1, 2, .., N)$, such that $\rho_i = min_j\{max_\alpha\{\frac{q_{ij}^\alpha}{Q^\alpha}\}\}$; if $\forall\alpha$, $\sum_{i=1}^{N} q_{i\rho_i}^\alpha \leq Q^\alpha$, use it as the initial feasible solution and proceed to *step 2* ; If no feasible solution exists, stop;

Step 2: Iteratively upgrade the current solution with another solution;
(a) For each item in the solution, find an item with a higher utility from the same group under resource constraint with the highest $\triangle a_{ij} = (q_{i\rho_i} - q_{ij}) \times \mathbb{C}/ \mid \mathbb{C} \mid$, where $q = [q^1, .., q^m]$, $\mathbb{C} = \sum_{i=1}^{N} q_{i\rho_i}$. $\mathbb{C}$ is the current resource usage;
(b) If no such item is found in group i, then select the item under resource constraint that maximizes the value gain per unit of extra aggregate resource: $\triangle p_{ij} = (\mathcal{F}_{i\rho_i} - \mathcal{F}_{ij})/ \triangle a_{ij}$;
(c) If no feasible upgrade is possible, go to *Step 3*;

Step 3: Upgrade the solution by using one upgrade followed by downgrades;

Step 3.1: Find a higher-utility item in any group with the highest value of $\triangle p'_{ij} = (\mathcal{F}_{i\rho_i} - \mathcal{F}_{ij})/ \triangle t'_{ij}$ and $\triangle t'_{ij} = (q_{i\rho_i} - q_{ij})/(\mathbb{Q} - \mathbb{C})$. $\mathbb{Q}_c = [Q^1, .., Q^m]$ indicates user's QoS requirements;
Step 3.2: Find a lower-utility item in any group with the highest value of $\triangle p''_{ij} = (\mathcal{F}_{i\rho_i} - \mathcal{F}_{ij})/\triangle t''_{ij}$ and $\triangle t''_{ij} = (q_{i\rho_i} - q_{ij})/(\mathbb{C} - \mathbb{Q}_c)$ while after downgrade, the total utility is still higher than achieved in *Step 2*;
Step 3.3: If an item ρ'_i is found in *Step 3.2* and ρ'_i satisfies the resource constraint, use ρ'_i to replace ρ and go back to *Step 2*. If ρ'_i is found in *Step 3.2* but violates the resource constraint, go back to *Step 3.2* for another downgrade. If no item can be found in *Step 3.2*, the algorithm stops.

To include the network performance factor in the model, we could add the network QoS attribute (such as transmission delay) to the sender service. That is, if service $a \in S_i \rightarrow b \in S_j\,(b = 1, 2, ...l)$, the corresponding network QoS attribute can be set to $q = \frac{1}{l}\sum_{b=1}^{l} q^{\alpha}(a,b)\,(\alpha = 1, ..m)$, and $q_a^{\alpha} = q_a^{\alpha} + q^{\alpha}$. The utility of every candidate in a service class can be computed according to Definition 1 after the network attributes are included.

4.2 The Graph Algorithm

Algorithms designed for the graph model can process more than one process plans at a time to find the best solution, although they have a higher complexity than the combinatorial algorithms. We first generate a candidate graph as follows: (1) Each candidate item in the service class is represented by a node in the graph, with a benefit value and several QoS attributes; (2) If service s_i is connected to service s_j, all service levels in s_i are connected to all service levels in s_j; (3) Set the network QoS attributes of every links; (4) Add a virtual source node v_s and sink node v_d. v_s is connected to all nodes without incoming link and v_d is connected to all nodes without outgoing links. The QoS attributes of these links are set to zero; (5) Add QoS attributes of the node to its incoming link and compute the utility of every link according to Def. 1.

After these steps, we have a Directed Acyclic Graph (DAG), in which every edge has a set of QoS attributes and a utility value. A service candidate graph is shown in Figure 2. The selection problem is to find a path that produces the highest utility from source v_s to sink v_d subject to the multiple constraints $\mathbb{Q}_c = [Q^1, .., Q^m]$. This is the well-known multi-constraint optimal path problem in the graph theory. Based on the CSP algorithm designed for one QoS constraint [15], we propose the MCSP algorithm to solve the MCOP problem. Same as CSP, during the execution of MCSP, each node needs to keep several paths from the source to it.

MCSP is shown in Algorithms 2 and 3. One potential problem for MCSP is that, for every intermediate node, the number of paths a node needs to keep may be huge if none of them dominates each other. That may cause the algorithm to run very slow. In order to speed up the algorithm and reduce the space needed, we modify the MCSP algorithm by keeping only K paths on each node. This

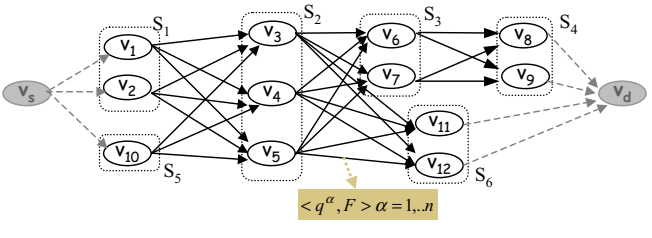


Fig. 2. Service Candidate Graph

Algorithm 2. MCSP

MCSP ($G=(V,E)$, v_s, v_d, $\mathbb{Q}_c$)
// every node ν keeps $\mathcal{L}(\nu)$paths $p(\mu,q,\mathcal{F})$ from source to it that satisfy constraints requirements

```
1 Topologically sort nodes in G;
2 for each node μ, in topological order
3    for each ν ∈ adj[μ]
4       if (μ==s) then
5          q^α:=q^α(μ,ν) ∀α = 1,2,...,m
6          F:= F(μ,ν)
7          MCSP_RELAX(μ,ν,q,F)
8       else for each p ∈ L(μ)
9          q^α=q^α(p)+q^α(μ,ν) ∀α = 1,2,...,m
10         F:= F(p) + F(μ,ν)
11         MCSP_RELAX(μ,ν,q,F)

12 p* ←∃ p* ∈ L(v_d), ∀p ∈ L(v_d), F(p*)≥F(p)
```

Algorithm 3. MCSP_RELAX

MCSP_RELAX (μ, ν, q, $\mathcal{F}$)

```
1   if (∃α, q^α > Q^α) then return;
2   for each p ∈ L(ν)
3      if F(p) > F and ∀α q^α(p) ≤ q^α then return
4      if F(p) < F and ∀α q^α ≤ q^α(p) then
5         remove p from L(ν)
6   Add (μ,q,F) to L(ν)
```

heuristic algorithm is called MCSP-K. The K-path selection criteria are based on the nonlinear cost function concept that is used to combine the multiple constraints into one [9]. The cost function for any path p can be defined as:

$$g_\lambda(p) \triangleq (\frac{q^1(p)}{Q^1})^\lambda + (\frac{q^2(p)}{Q^2})^\lambda + ... + (\frac{q^m(p)}{Q^m})^\lambda$$

where $\lambda \geq 1$. $q^i(p)$ is the aggregated i^{th} QoS attribute for path p . As $\lambda \to \infty$, $g^*(p) \triangleq \lim_{\lambda\to\infty} g_\lambda(p)$ is equivalent to the cost function

$\xi(p) \triangleq max\{(\frac{q^1(p)}{Q^1}), (\frac{q^2(p)}{Q^2}), ..., (\frac{q^m(p)}{Q^m})\}$. The paths with K minimum g_λ/ξ values will be kept at each intermediate node. This ensures that MCSP-K will never prune out a feasible path if there exists one.

Compared to MCSP, the only difference of MCSP-K lies on the relax function, in which it needs to check the number of paths it has currently and remove the path with the maximum g_λ/ξ if the maximum number K is reached. The relax function for MCSP-K is shown in Algorithm 4. MCSP-K drastically reduces the space cost and speeds up the MCSP algorithm while keeps the result close to the optimal. The simulation results and comparison of the two algorithms are shown in the next section.

Algorithm 4. MCSP-K RELAX

MCSP-K_RELAX (μ, ν, q, $\mathcal{F}$, λ)

```
1   if (∃α, q^α > Q^α) then return;
2   for each p ∈ L(ν)
3      if F(p) > F and ∀α q^α(p) ≤ q^α then return
4      if F(p) < F and ∀α q^α ≤ q^α(p) then
5         remove p from L(ν)
6   Add (μ, q, F) to L(ν)
7   if size(L(ν)) > K then
8      if λ == ∞ then
9         remove p' ∈ L(ν), ∀p ∈ L(ν), ξ(p') ≥ ξ(p)
10     else
11        remove p' ∈ L(ν), ∀p ∈ L(ν), g_λ(p') ≥ g_λ(p)
```

5 Performance Study

For systems with only one process plan connected in a sequential flow model, which contains N service classes and each class has l candidates, the worst-case time complexity of BBLP using the simplex method [11] is an exponential function 2^{Nl}. Using WS_HEU, suppose the number of QoS requirements is m, the worst case time complexity is $O(N^2(l-1)^2m)$ [8]. The worst case time complexity for MCSP is $O(Nl^2 + l^{2N-1}) = O(l^{2N-1})$ and the maximum space needed in v_d to keep all feasible paths is $O(l^N)$. For MCSP-K, the maximum time complexity is $O(Nl^2 + Kl^{N-1}) = O(Kl^{N-1})$ and the maximum space needed in v_d is $O(lK)$. Although the worst case complexity of MCSP and MCSP-K is not a polynomial function, they perform very well in practice. In this section, we study their performance by simulations.

5.1 Evaluation Methodology

We have compared the performance of BBLP, WS_HEU, MCSP and MCSP-K algorithms by extensive simulations. First, we use the degree-based Internet topology generator Inet 3.0 [14] to generate a power-law random graph topology with 4000 nodes to represent the Internet topology. Then we randomly select 25 ∼2500 (depends on different test cases) nodes as the service candidate nodes and 2 other nodes as source and sink. In our study, we assume an equal-degree random graph topology for the service candidate graph. For simplicity, we only consider one process plan with the sequential composition model. The number of service class and candidates in each service class involved in the process plan range from 5 to 50.

For our evaluation we also need to generate the service and network QoS attributes and utility. Suppose all QoS attributes have the summation properties. Five QoS attributes are considered for each service/link, each is associated with a randomly generated values: $q^k(\mu,\nu)$ $(k = 1, 2, 3, 4, 5)$ with a uniform distribution between [1,100]. We also generate the utility $\mathcal{F}(\mu, \nu)$ of each link as a random value with a uniform distribution between [1,200]. For QoS attributes of each

service candidate, a base value is first generated with a uniform distribution between [1,100]. Then an impact factor (ε) is multiplied to each service. We consider two different situations with different network impact factors: (1) large: network QoS value is comparable to services and varies; (2) small: network QoS value is less than $\frac{1}{10}$ of services. The ε is set to 2 and 200 for the two cases respectively. The utility of each service is also generated as random value with a uniform distribution between $[1, 200]$.

For the combinatorial model, we compute the average value of QoS attributes and utility for all outgoing links of a service and add to that service. For services in the first class, it also needs to add the values of link from the source to it. For the graph model, we add the QoS attributes and the utility of the service candidate to every incoming link of it. The number of user's QoS requirements ranges from 2 to 5.

Our study includes two parts: (1) *Optimal and heuristic algorithms comparison*; (2) *The comparison of combinatorial and graph models*. The metrics we measure for Part 1 include *run time*, *approximation ratio* (heuristic utility vs. the optimal value), *memory usage* (for the graph approach). The metric we use for Part 2 is the *provisioning success rate, running time and utility*. We compare two heuristic algorithms: WS_HEU and MCSP-K. A composed service provisioning is said to be successful if the generated result satisfies a user's QoS requirements. From the description of MCSP-K, we know its success rate is always 1 since it never prunes out the optimal path in all intermediate steps. But for MMKP, since the network QoS attributes are only estimated, the generated results may not meet a user's requirements.

In our study, for each test case (representing different numbers of service classes and service candidates combinations), we randomly generate 10 instances and run 10 times for each instance. We then use the average value of the 100 rounds as the result for comparison.

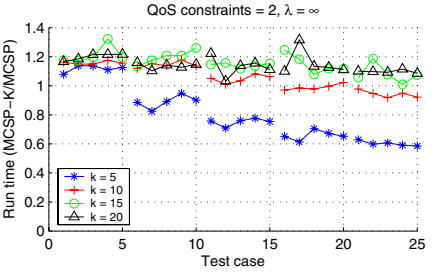
5.2 Result Analysis

For performance evaluation about the MCSP-K and MCSP algorithms, 25 test cases are used in the simulation (Table 1). The cases are divided into 5 groups; each group has the same number of candidates. For each group, we test different numbers of service classes (from 10 to 50). There are two parameters: λ is used to compute the non-linear cost for MCSP-K and k is the number of paths each intermediate node keeps. We conducted tests on $\lambda = 5, 10, 15, 20, 25, 30, \infty$ and $k = 5, 10, 15, 20$. We find that $\lambda = \infty$ always gets a better performance (in terms of utility) than other values. So here we only report the results for $\lambda = \infty$ under different k values.

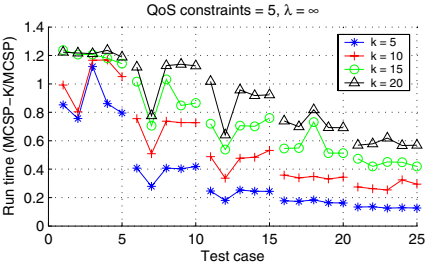
Figures 3 and 4 show the running time and memory usage comparison of MCSP-K and MCSP under 2 and 5 QoS constraints, respectively. For both k values, MCSP-K can achieve a near optimal performance (producing utility $> 90\%$ of MCSP). For the cases of 2 QoS constraints, MCSP-K is not attractive since not much space can be saved and the running time is even longer than MCSP in some cases ($k = 10, 15, 20$). The extra time is used to compute the

Table 1. Test Cases

Test Case	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Test Group	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4	5	5	5	5	5
No. Candidates	10	10	10	10	10	20	20	20	20	20	30	30	30	30	30	40	40	40	40	40	50	50	50	50	50
No. Service Class	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50	10	20	30	40	50

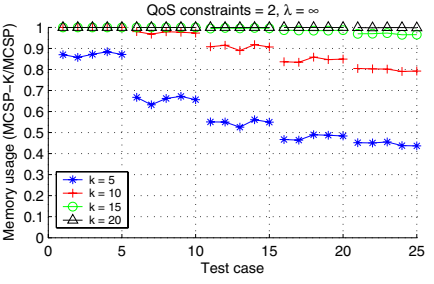


(a) QoS constraints = 2

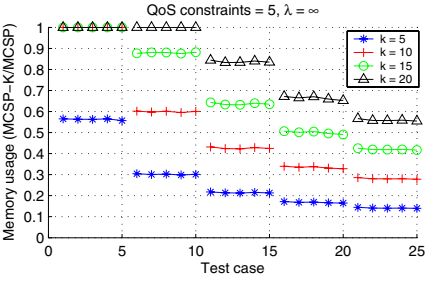


(b) QoS constraints = 5

Fig. 3. Running time comparison (MCSP-K/MCSP)



(a) QoS constraints = 2

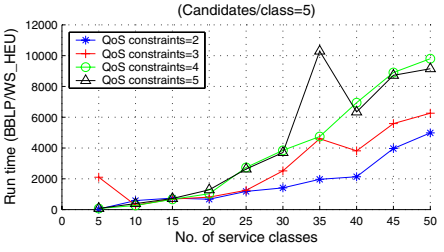


(b) QoS constraints = 5

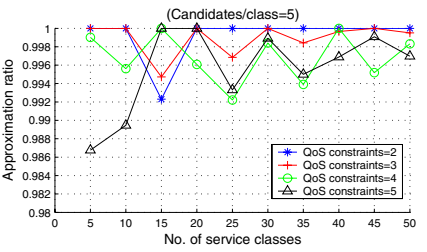
Fig. 4. Memory usage comparison (MCSP-K/MCSP)

non-linear cost and decide the paths to be pruned out. As the numbers of service classes and candidates in each class increase, MCSP-K outperforms MCSP. The advantage of MCSP-K is more obvious in the cases of 5 constraints. Both running time and space needed are significantly lower while the performance remains nearly optimal ($> 95\%$ for $k = 10, 15, 20$). So if the process plan contains a large number of service classes or there are many candidates in each service class, using the heuristic algorithm MCSP-K can get a close to optimal solution quickly and avoid the memory growth problem. $\lambda = \infty$ and $k = 10$ or 15 are the best setting for MCSP-K.

Figure 5 shows the running time and utility comparison between BBLP and WS_HEU algorithms when the number of QoS constraints is from 2 to 5, respectively. The number of service classes ranges from 5 to 50 and the number of

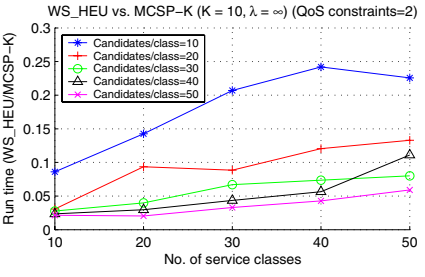


(a) Run time

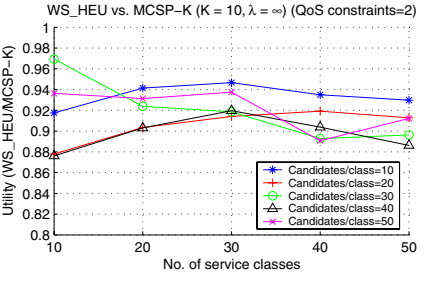


(b) Utility

Fig. 5. WS_HEU vs. BBLP



(a) Run time



(b) Utility

Fig. 6. WS_HEU vs. MCSP-K ($K = 10, \lambda = \infty$)

candidates in each service class is 5. We can see that the performance of WS_HEU is near optimal ($> 98.5\%$) while the running time is dramatically reduced.

The heuristic algorithms in both models perform very well. To see which one should be used for service composition, we test the provisioning success rate for WS_HEU under 2 and 5 constraints. (The provisioning success rate for MCSP-K is 1). When the network factor is comparable to the service, the success rate of WS_HEU is very low (0.32 for 2 constraints and 0.04 for 5 constraints). If the network impact factor is small, the success rate is high (0.96 in both cases). The reason for the low success rate lies on that MMKP does a combinatorial selection without the flow concept. Figure 6 shows the run time and utility comparison of WS_HEU and MCSP-K with $k = 10$ and $\lambda = \infty$ in the situation that the network impact factor is small. It shows that WS_HEU outperforms MCSP-K. So the combinatorial approach should be used in the situation when the network impact is small. It can also be used in the situation where the network condition for all services is uniform, such as all are on the same LAN.

From our experiments, we can see that different algorithms should be used under different system conditions. Table 2 presents a comparison of the four algorithms presented in this paper and suggests when they should be used.

Table 2. Comparison of Algorithms

	BBLP	WS_HEU	MCSP	MCSP-K
Running Time	*very slow*	*fast*	*slow*	*fast*
Memory Usage	*low*	*low*	*high*	*low*
Optimality	*optimal*	*near-optimal*	*optimal*	*near-optimal*
Network Cost	*inaccurate*	*inaccurate*	*accurate*	*accurate*
Algorithm Usage	*very small size problem, small or uniform network factor*	*large size problem, small or uniform network factor*	*small size problem, network factor is large*	*large size problem, network factor is large*

6 Conclusions

In this paper, we study the problem of complex service composition with multiple QoS constraints. Two problem models are proposed: the combinatorial model, by defining the problem as an MMKP, and the graph model, by defining the problem as an MCOP. The utility function may be defined by an extended set of system parameters, including static server information (service level), client QoS requirement (QoS constraint), dynamic server capacity (service benefit), and network factor. We have presented various algorithms, both optimal and heuristic, to compose and select services under multiple QoS constraints as well as to achieve the maximum utility. We have also compared the pros & cons between the two models and suggested their usage context. We believe the proposed models and algorithms provide a useful engineering solution to the end-to-end QoS problem for building distributed complex services.

References

1. Aggarwal, R., et al.: Constraint driven Web service composition in METEOR-S. Proc. of IEEE Conf on Service Computing (SCC'04), Shanghai, China, Sep. 2004
2. BPMI.org.: Business Process Modeling Language (BPML), Version 1.0, http://www.bpmi.org/bpml.esp, November, 2002
3. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V. and Shan, M.: Adaptive and dynamic service composition in eflow. Technical Report, HPL-200039, Software Tech Lab, March 2000
4. Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S. and Weerawarana, S.: Business Process Execution Language for Web services, Version 1.1. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel, May 2003
5. Dan, A. et al. : Web services on demand: WSLA-driven automated management, IBM Systems Journal, Vol. 43, No. 1, 2004, pp. 136-158
6. Fu, X., Shi, W., Akkerman, A. and Karamcheti, V.: CANS: Composable, Adaptive Network Services Infrastructure. Proceeding of 3rd USENIX symposium on Internet Technologies and Systems, March 2001
7. Khan, S.: Quality Adaptation in a Multisession Multimedia System: Model, Algorithms and Architecture, Ph.D. Dissertation, Department of ECE, University of Victoria, Canada, May 1998

8. Khan, S., Li, K.F., Manning, E.G. and Akbar, M.: Solving the knapsack problem for adaptive multimedia systems, Studia Informatica Universalis, Volume 2, Number 1, September 2002, pp. 157-178
9. Korkmaz, T., Krunz, M.: Multi-Constrained Optimal Path Selection. Proceeding of 20th Joint Conf. IEEE Computer & Communications (INFOCOM 2001), 2001, pp. 834-843
10. Ludwig, H., Keller, A., Dan, A., King, R.P. and Franck, R.: Web Service Level Agreement (WSLA) Language Specification, Jan. 2003, http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf
11. Maros, Istvn: Computational Techniques of the Simplex Method, Springer Publisher, December 2002
12. Martello, S. and Toth, P.: Algorithms for Knapsack Problems. Annals of Discrete Mathematics, 31:70-79, April 1987
13. Ponnekanti, S.R. and Fox, A.: Sword: A developer toolkit for Web service composition. In 11th World Wide Web Conference, Honolulu, Hawaii, May 2002
14. Winick, J. and Jamin, S.: Inet 3.0: Internet Topology Generator. Tech Report UM-CSE-TR-456-02 (http://irl.eecs.umich.edu/jamin/), University of Michigan, 2002
15. Yu, T. and Lin, K.J.: Service Selection Algorithms for Web Services with End-to-end QoS Constraints., Journal of Information Systems and E-Business Management, Volumn 3, Number 2, July 2005
16. Yu, T. and Lin, K.J.: A Broker-based Framework for QoS-Aware Web Service Composition, Proceeding of IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-05), Hong Kong, China, March 2005
17. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J. and Sheng, Q.Z.: Quality Driven Web Service Composition. Proceeding of 12th International World Wide Web Conference (WWW), 2003