Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints

TAO YU, YUE ZHANG, and KWEI-JAY LIN
University of California, Irvine

Service-Oriented Architecture (SOA) provides a flexible framework for service composition. Using standard-based protocols (such as SOAP and WSDL), composite services can be constructed by integrating atomic services developed independently. Algorithms are needed to select service components with various QoS levels according to some application-dependent performance requirements. We design a broker-based architecture to facilitate the selection of QoS-based services. The objective of service selection is to maximize an application-specific utility function under the end-to-end QoS constraints. The problem is modeled in two ways: the combinatorial model and the graph model. The combinatorial model defines the problem as a multidimension multichoice 0-1 knapsack problem (MMKP). The graph model defines the problem as a multiconstraint optimal path (MCOP) problem. Efficient heuristic algorithms for service processes of different composition structures are presented in this article and their performances are studied by simulations. We also compare the pros and cons between the two models.

Categories and Subject Descriptors: H.4.m [**Information Systems Applications**]: Miscellaneous

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: End-to-end QoS, service composition, service selection, service oriented architecture (SOA), Web services

ACM Reference Format:

Yu, T., Zhang, Y., and Lin, K.-J. 2007. Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM Trans. Web 1, 1, Article 6 (May 2007), 26 pages. DOI = 10.1145/1232722.1232728 http://doi.acm.org/10.1145/1232722.1232728

1. INTRODUCTION

Service-Oriented Architecture (SOA) is becoming a major software framework for distributed applications such as e-business and enterprise systems. Using

This paper is an extended version of a paper appearing in the 3rd International Conference on Service Oriented Computing [Yu and Lin 2005].

Tao Yu is currently employed by HP Labs China.

Corresponding author's address: K.-J. Lin, Department of Electrical Engineering and Computer Science, University of California, Irvine, CA; email: klin@uci.edu.

standard protocols (such as SOAP and WSDL), distributed applications can be dynamically and flexibly composed by integrating new and existing component services developed independently to form complex business processes and transactions. Each component service may be executed on any platform as long as its functional interface specification is properly defined in WSDL and matches that of a service request.

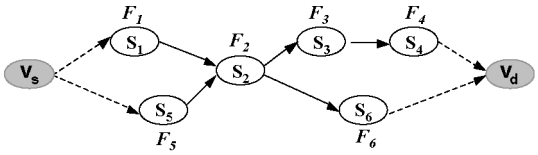
The runtime performance of a composite service is important for most practical distributed applications. For example, a health care information system may need a real-time response when a patient record is queried remotely from a hospital emergency room. Enterprise systems with online service customers need to post transaction results in real-time to reflect dynamic market conditions. End-to-end performance management is a big challenge for distributed SOA systems due to the compositional nature of Web services.

QoS for Web services refers to various nonfunctional characteristics such as response time, throughput, availability, and reliability [Menasce 2004]. Services with similar and compatible functionality may be offered at different QoS levels. Thus, to build a service process, decisions must be made to select component services at appropriate QoS levels. A QoS management subsystem can be used to ensure that requests from clients are serviced properly and achieve the best performance/cost ratio.

In this project, we design a broker-based architecture [Yu and Lin 2005], called *QBroker*, to provide end-to-end QoS management for distributed services. The main functions of QBroker include service discovery, planning, selection, and adaptation. The efficiency of QBroker is dominated by the running time of the service selection algorithm. We study service selection algorithms for applications with different composition structures. Our research models the problem in two ways: *the combinatorial model* defines the problem as a multidimension multichoice knapsack problem (MMKP) and *the graph model* defines the problem as a multiconstrained optimal path (MCOP) problem [Xiao and Boutaba 2005]. In both models, a user-defined utility function of some system parameters may be specified to optimize application-specific objectives.

We design efficient algorithms for quality-driven Web service compositions. Our model defines multiple QoS criteria and takes global constraints into account. Our technique ensures that the selected services always meet the QoS requirements. In addition, we propose heuristic algorithms to find near-optimal solutions in polynomial time which is more suitable for making runtime decisions. We also propose algorithms to handle rich composition structures including sequential, parallel, conditional, and loops of services.

The rest of this article is organized as follows. Section 2 gives the motivation for our study and the system architecture. Section 3 defines QoS models for dynamic Web service composition, the utility function definition, and the problem assumptions. Section 4 shows the algorithms for service composition for sequential flow structure. Section 5 extends the algorithms for service composition for general flow structure. Section 6 shows the performance evaluation and comparison of different algorithms. Section 7 reviews some related work. The article is concluded in Section 8.



(a) Composite service

Function	Service Class	Service Candidate	Utility	Execution Time	Price	Availability
F_1	S_1	s_{11}	212	100	50	0.95
		s_{12}	219	180	60	0.92
F_2	S_2	s_{21}	195	200	50	0.98
		s_{22}	123	160	100	0.95
		s_{23}	150	180	80	0.97
F_3	S_3	s_{31}	216	150	100	0.94
		s_{32}	160	120	80	0.99
F_4	S_4	s_{41}	150	130	60	0.93
		s_{42}	200	140	40	0.99
F_5	S_5	s_{51}	231	200	150	0.96
F_6	S_6	s_{61}	162	200	100	0.97
		s_{62}	123	180	130	0.99

(b) QoS Parameters

Fig. 1. Example of composite service process.

2. MOTIVATION AND SYSTEM ARCHITECTURE

2.1 Motivating Example

We begin by using a simple example to show the process of service composition with end-to-end QoS constraints. Figure 1(a) shows a composite service that has four possible paths: $\{F_1, F_2, F_3, F_4\}$, $\{F_1, F_2, F_6\}$, $\{F_5, F_2, F_3, F_4\}$, or $\{F_5, F_2, F_6\}$. Each function F_i may be executed by any atomic service (s_{ij}) in the service class S_i. The QoS values of each atomic service are shown in Figure 1(b). The end-to-end QoS requirements include:

—response time ≤ 600,
—cost ≤ 250,
—availability $\geq 85\%$.

The optimal selection of services, which maximized the sum of all service utilities, is $\{s_{11}, s_{21}, s_{31}, s_{42}\}$ with a total utility of 823, a response time of 590, at a cost of 240 and an availability of 86.64%.

In this simple example, there are a total of 30 candidate combinations, each with different utility and QoS values. The number of possibilities goes up exponentially with an increasing number of service classes. The complexity of service composition is due to the following factors.

(1) Many services are candidates for one functionality and they offer different QoS levels.

(2) There may exist more than one way to build a composite service.
(3) There are various performance constraints for a composite service.
(4) A service's QoS capability may be different from what is claimed by service providers.

From the previous discussion, we can see that SOA systems need to have capabilities to dynamically compose Web services with end-to-end QoS constraints, including:

—*service tracking* to identify all available services and their QoS capability for a specific functionality;
—*service planning* to create the service process for a composite service;
—*service selection* to identify the best service for each function node of a composite service process; and
—*QoS value adaptation* to update services' QoS values to their actual service delivery.

2.2 QoS Broker Architecture

We have designed a QoS service broker [Yu and Lin 2005], called *QBroker*, which acts as an external, independent broker entity that can help users construct composite services. Unlike CORBA (http://www.corba.org) where a broker is designed to be a middleware for object invocations, QBroker is not a mandatory layer between service clients and providers. The goal of QBroker is to help users select the best services for their composite service process before invocation. Actual service requests to service providers can be invoked by users directly without going through QBroker. A detailed presentation on the QBroker architecture can be found in [Yu and Lin 2005].

Figure 2 shows the steps for service composition. The following information is produced at different stages of the process.

—*Process plan*. An abstract process that defines a flow of component functions and their relationships. A process plan is designed to fulfill a user's request.
—*Function graph*. This is a functional graph which includes all process plans that can fulfill a user's request. Each node of the graph is a component function. A start node and an end node are added to the function graph.
—*Service candidate graph*. A service candidate graph is built from a functional graph and it includes a path for all combinations of function nodes that can be used to fulfill the service request. The executable composite service is created by services selection based on a service candidate graph.

3. SYSTEM MODEL AND PARAMETERS

3.1 Composition Model

Atomic services may be connected by different structures in a composite service. Figure 3 shows the six service composition structures considered in our study: *Sequential*, *AND split (fork)*, *XOR split (conditional)*, *Loop*, *AND join (Merge)* and *XOR join (Trigger)*. The AND split will trigger all nodes $S_2 \cdots S_n$ to be

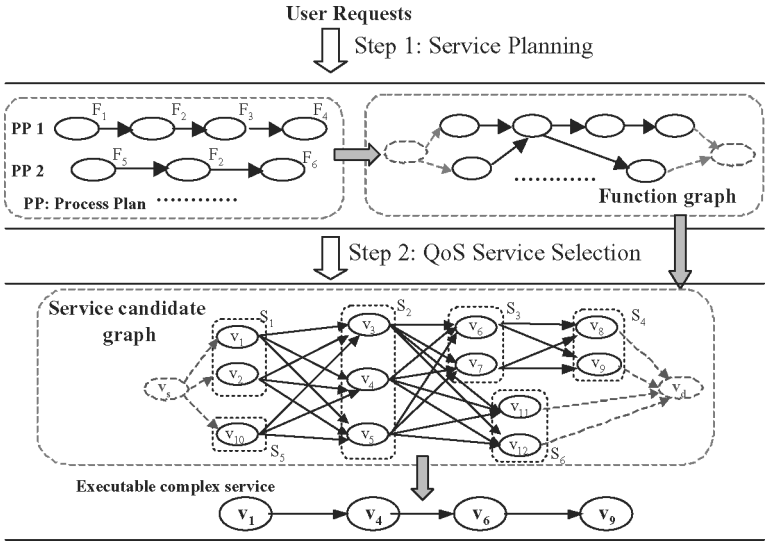


Fig. 2. QoS Web service composition.

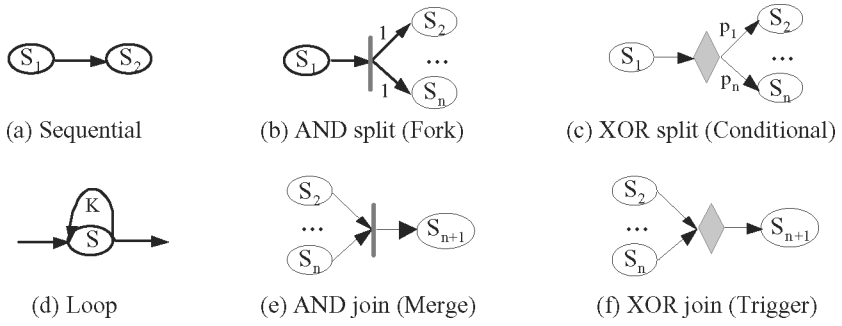


Fig. 3. Composition structure patterns.

executed in parallel after S_1 so all edges following S_1 have a probability of 1. On the other hand, the XOR split will trigger only one of the nodes $S_2 \cdots S_n$, each one with a certain probability, and the total probability is 1. The Loop structure must be defined with a maximum loop count K so that it is not executed for more than K times. Jaeger et al. [2004] presents the QoS aggregation function for these common composition patterns.

Figure 4 shows a composite service example. S_2 is followed by either S_4 or S_5 with a probability of P_1 and P_2, respectively. S_6 may be executed for at most K times. Suppose t_i and c_i are the response time and cost of service S_i, the total response time T and cost C are:

$$T = t_1 + max(t_2 + p_1 * t_4 + p_2 * t_5,\ t_3 + K * t_6) + t_7$$
$$C = c + c_2 + p_1 * c_4 + p_2 * c_5 + c_3 + K * c_6 + c_7.$$

The value of QoS parameters provided for each atomic service is either the average values ($\mathbf{q_{avg}}$) or the worst case values ($\mathbf{q_{max}}$). Using $\mathbf{q_{avg}}$ will likely

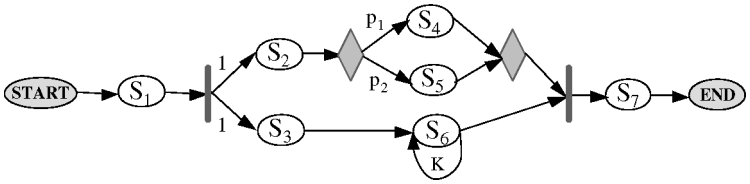


Fig. 4. Composite service example.

Table I. System Parameters and Notations

S_i	Service class i
s_{ij}	Atomic service j for service class i
$\mathcal{F}$	Utility function
$\mathbf{q_{ij}}$	QoS vector for atomic service $s_{\mathbf{ij}}$, $\mathbf{q}_{ij} = [q_{ij}^1, \ldots, q_{ij}^n]$
$\mathbf{Q_c}$	QoS requirements vector of a composite service, $\mathbf{Q_c} = [Q_c^1, \ldots, Q_c^m]$

select the most suitable services since it reflects the actual service behavior more accurately, while using $\mathbf{q_{max}}$ will select services with a stronger guarantee.

3.2 System Notations

The following notations (Table I) are used in this article.

—*Atomic service (component service) (s_{ij})*. An atomic service s_{ij} is associated with a QoS vector $\mathbf{q_{ij}} = [q_{ij}^1, \ldots, q_{ij}^n]$, with n application-level QoS parameters [Ludwig et al. 2003].

—*Service class (S_i)*. A service class is a collection of atomic services with a common functionality but different nonfunctional properties (such as QoS). A class interface parameter set (S_{in}, S_{out}) is defined for each service class.

—*Utility function* ($\mathcal{F}$). Each service candidate v has an associated utility function $\mathcal{F}_v$, which is defined by a set of system parameters including system load, cost, and/or other QoS attributes.

3.3 Utility Function

The goal of service selection algorithms is to select individual services that meet QoS constraints and also provide the best value for the user-defined utility function $\mathcal{F}$. $\mathcal{F}$ is defined by a weighted sum of system parameters including system load, cost, and other attributes. Suppose a user has m QoS constraints in a service request, $\mathbf{Q_c} = [Q_c^1, \ldots, Q_c^m]$. In our study, the utility function is defined as follows:

Definition 1 (*Utility Function*). Suppose there are x QoS attributes to be maximized and y QoS attributes to be minimized. The utility function for service s_{ij} is defined as:

$$\mathcal{F}_{ij} = \sum_{\alpha=1}^{x} w_\alpha * \left(\frac{q_{ij}^\alpha - \mu^\alpha}{\sigma^\alpha}\right) + \sum_{\beta=1}^{y} w_\beta * \left(1 - \frac{q_{ij}^\beta - \mu^\beta}{\sigma^\beta}\right),$$

where w_α and w_β are the weights for each QoS attribute ($0 < w_\alpha, w_\beta < 1$), $\sum_{\alpha=1}^{x} w_\alpha + \sum_{\beta=1}^{y} w_\beta = 1$. μ and σ are the average and standard deviation of the QoS values for all candidates in a service class.

In the utility function definition, all QoS attributes are weighted by their importance. They are also normalized by their averages and standard deviations so that the utility function will not be biased by any attribute with a large value.

3.4 Algorithms Overview and Assumptions

Given a user-defined utility function, the QoS service selection problem is to select services so that a composite service receives the largest utility $\mathcal{F}$ and meets all QoS constraints. The QoS selection problems considered in this article are divided into two categories.

(1) *Composite services with a sequential flow structure*. In this category, a composite service is constructed by a set of atomic services executed in a sequential order. Algorithms for sequential flow are discussed in Section 4.
(2) *Composite services with a general flow structure*. In this category, the relationships among atomic services of a composite service may contain sequential, parallel, conditional, or loop structures. Algorithms for general flow are discussed in Section 5.

4. SERVICE SELECTION ALGORITHMS FOR SEQUENTIAL FLOW STRUCTURE

4.1 Algorithms for the Combinatorial Model

For a composite service that has N service classes $(S_1, S_2, \ldots, S_N)$ in a process flow plan and with m QoS constraints, we map the service selection problem to a 0-1 multidimension multichoice knapsack problem (MMKP) [Martello and Toth 1987]. MMKP is defined as follows.

Definition 2 (*MMKP*). Suppose there are N object groups, each has l_i ($1 \leq i \leq N$) objects. Each object has a profit p_{ij} and requires resource $\mathbf{r}_{ij} = (r_{ij}^1, \ldots r_{ij}^m)$. The amount of resources available in the knapsack is $\mathbf{R} = (R^1, \ldots, R^m)$. MMKP is to select exactly one object from each object group to be placed in the knapsack so that the total profit is maximized while the total resources used are less than the resources available.

The QoS service selection problem is to select one service candidate from each service class to construct a composite service that meets a user's QoS constraints and maximizes the total utility. The QoS service selection problem is mapped to MMKP as follows.

—Each service class is mapped to an object group in MMKP.
—Each atomic service in a service class is mapped to an object in a group in MMKP.

—The QoS attributes of each candidate are mapped to the resources required by the object in MMKP.

—The utility a candidate produces is mapped to the profit of the object.

—A user's QoS constraints are considered as the resources available in the knapsack.

Mathematically, the service selection problem is thus formulated as follows:

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

where x_{ij} is set to 1 if atomic service j is selected for class S_i and 0 otherwise. $\mathbf{q_{ij}} = [q_{ij}^1, \ldots, q_{ij}^n]$ is the QoS resource needs of each atomic service j for class S_i; the sum of all resources q_{ij}^{α} used by all services must be less than the overall constraints $\mathbf{Q_c} = \{Q_c^{\alpha}\}, (\alpha = 1, \ldots, m)$.

The MMKP problem has been shown to be NP-complete [Martello and Toth 1987]. We may solve MMKP by finding optimal results or use heuristic algorithms to reduce the time complexity. Following, we discuss both approaches.

4.1.1 *BBLP Algorithm.* Most algorithms for finding optimal solutions for MMKP are based on the *branch-and-bound* method. [Khan 1998] presents such an algorithm, BBLP for MMKP, that finds the optimal solution by iterative generation of a search tree. Each node of the search tree represents a solution state where some service classes have selected or fixed a service while others are still free (i.e., no service selected). Every node has three state values:

(1) *fixed utility ($\mathcal{F}_f$)* produced by fixed classes $S_i \in \mathbf{C}$, $\mathcal{F}_f = \sum_{S_i \in \mathbf{C}} \mathcal{F}_{i\rho_i}$;
(2) *utility upper bound ($\mathcal{F}_b$)*: $\mathcal{F}_b = \mathcal{F}_f + \mathcal{F}_{LP}$. $\mathcal{F}_{LP}$ is computed by the linear relaxation of the subproblem, which is constructed by relaxing $x_{ij} \in \{0, 1\}$ to $0 \leq x_{ij} \leq 1$ for all free classes $S_i \notin \mathbf{C}$;
(3) *branching class (S_g)*. is the solution of $\mathcal{F}_{LP}$, the class S_i with the largest sum of x_{ij}.

BBLP iteratively performs the following steps.

(1) Find the utility upper bound $\mathcal{F}_b$ for each node in the search tree. $\mathcal{F}_b$ is computed by the linear relaxation of the whole problem.
(2) Select the node with the largest $\mathcal{F}_b$ and expand it through S_g. This is based on the hypothesis that the largest x_{ij} (closest to 1) will have the highest chance to be chosen in the final solution. This class will be fixed by adding new nodes to the search tree as the children of the current node. Each of the new nodes corresponds to select one service in S_g.

BBLP repeats the previous steps until all classes are fixed. In the search tree, the node with the largest $\mathcal{F}_f$ is the optimal solution. By tracing the tree back to the root, we can find the candidate chosen for each class.

4.1.2 *WS_HEU Algorithm.* The computation time for BBLP grows exponentially with the size of the problem. This may not be acceptable for QBrokers that need to make runtime decisions. Heuristic algorithms may be useful to find feasible solutions in polynomial time.

We use a heuristic algorithm WS_HEU to find solutions for MMKP. Figure 5 shows the flow of the WS_HEU algorithm. The notations used are shown in Table II, and the individual functions (F1 to F6) used in Figure 5 are defined in Table II. The algorithm has three main steps.

(1) Find an initial feasible solution.
For each service class S_i, WS_HEU selects a service ρ_i that has $min_j\{max_\alpha\{q_{ij}^\alpha/Q_c^\alpha\}\}$ in the class. It then checks the feasibility of the initial solution. If the solution is infeasible, the algorithm iteratively improves the solution by replacing the service ρ_i' with the largest saving of aggregated QoS among all service classes. The service replacement continues until a feasible solution is found (or else the algorithm fails).

(2) Improve the solution by feasible upgrades.
Among all classes, WS_HEU finds service ρ_i' to replace ρ_i for S_i to get a higher utility without violating the constraint requirements. The service replacement criterion is based on [Toyoda 1975] If no such service can be found, WS_HEU picks the one that maximizes $\triangle p_{ij}$.

(3) Improve the solution by infeasible upgrades followed by downgrades.
Performing only feasible upgrades may reach a local optimal in the search space. To achieve the global optimal, WS_HEU further improves the solution by using F5 to select the service that maximizes $\triangle p_{ij}'$. This replacement makes the solution infeasible. So one or more downgrades are followed by selecting the service which minimizes $\triangle p_{ij}''$. This method of upgrades followed by downgrades may increase the total utility of the solution.

WS_HEU is extended from algorithm HEU [Khan et al. 2002] which uses an initial feasible solution by always searching for the lowest utility item in each class. The search, however, is time-consuming. WS_HEU prunes out more infeasible items from each class and finds a feasible solution in a shorter time. In our simulation study, WS_HEU finds a feasible solution at the first try in most cases (more than 98% of them), while HEU conducts further revision 70% of the time. WS_HEU saves around 50% of the computation time compared to HEU.

For a composite service that has N service classes, each with l candidates and m QoS constraints, the time complexity of WS_HEU is $O(N^2(l-1)^2m)$ (same as HEU [Khan et al. 2002]), which is a polynomial function.

4.2 Algorithms for the Graph Model

We first convert the abstract function graph into a service candidate graph according to following rules:

—every service candidate is a node in the service candidate graph;

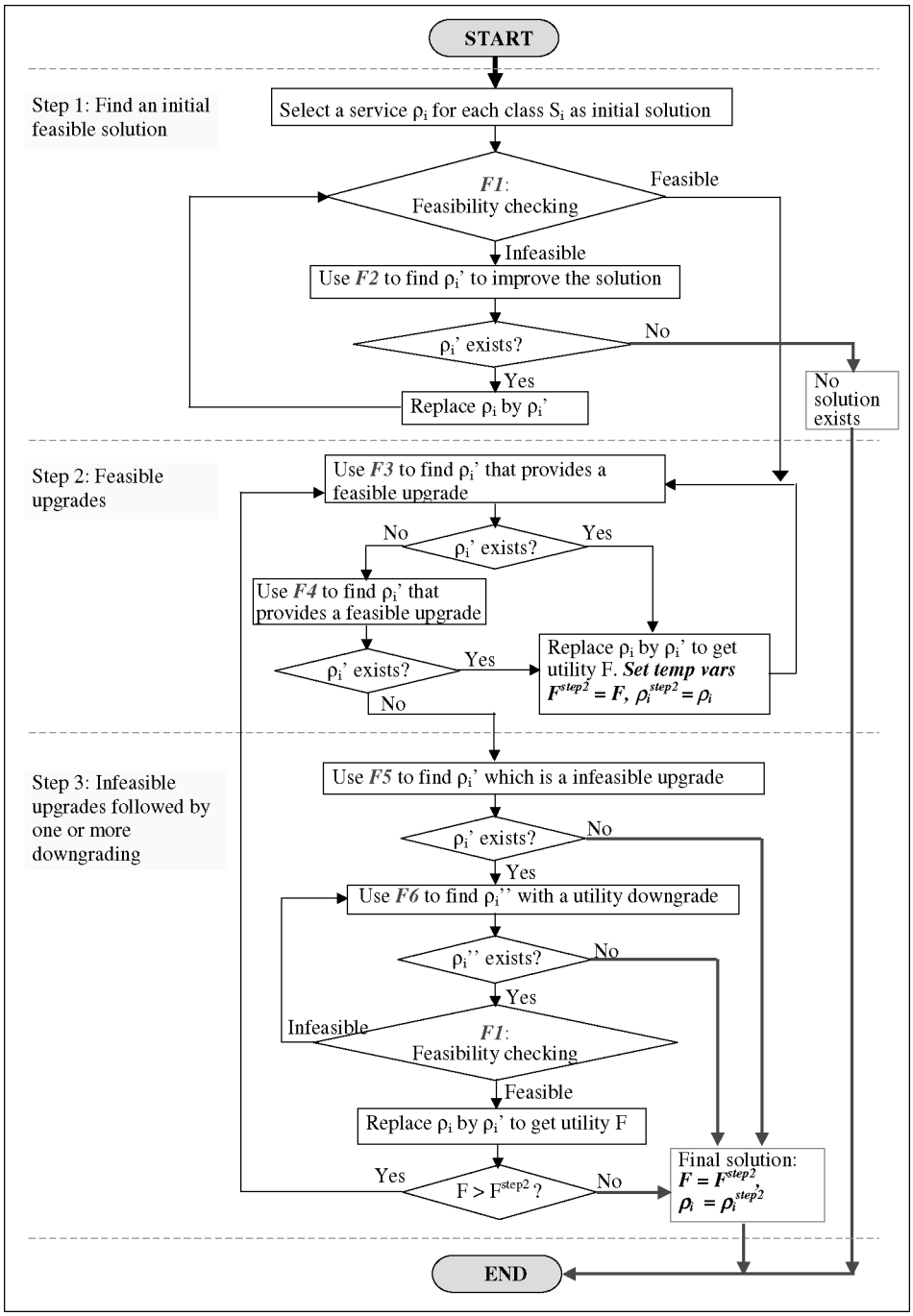


Fig. 5. Algorithm structure for WS_HEU and WFlow(*EU*).

Table II. Functions Defined in WS_HEU

Function	Definition
Notations	$\mathbf{Q} = \sum_{i=1}^{N} \mathbf{q_{i\rho_i}}$: Current aggregated QoS value; ($\mathbf{Q} = [Q^1, \ldots, Q^m]$) $f^\alpha = \frac{Q^\alpha}{Q_c^\alpha}$: feasibility factor of QoS attribute α. ≤ 1: feasible; > 1, infeasible; $\triangle a_{ij} = (\mathbf{q_{i\rho_i}} - \mathbf{q_{ij}}) \times \mathbf{Q} / \mid \mathbf{Q} \mid$: saving in aggregated QoS ; $\triangle p_{ij} = (\mathcal{F}_{i\rho_i} - \mathcal{F}_{ij}) / \triangle a_{ij}$: value gain per unit of extra QoS; $\triangle t'_{ij} = (\mathbf{q_{i\rho_i}} - \mathbf{q_{ij}}) / (\mathbf{Q_c} - \mathbf{Q}) = \sum_{\alpha=1}^{m} (q_{i\rho_i}^\alpha - q_{ij}^\alpha) / (Q_c^\alpha - Q^\alpha)$: savings in total-per-unit-QoS ; $\triangle p'_{ij} = (\mathcal{F}_{i\rho_i} - \mathcal{F}_{ij}) / \triangle t'_{ij}$: value gain per unit of extra total-per-unit-QoS; $\triangle t''_{ij} = (\mathbf{q_{i\rho_i}} - \mathbf{q_{ij}}) / (\mathbf{Q} - \mathbf{Q_c})$: losses in total-per-unit-QoS; $\triangle p''_{ij} = \triangle t''_{ij} / (\mathcal{F}_{i\rho_i} - \mathcal{F}_{ij})$: value loss per unit of extra total-per-unit-QoS;
F1	$\forall \alpha,\ Q^\alpha \leq Q_c^\alpha$
F2	$\rho'_i = max_{i,j}\{\triangle a_{ij}\}$ and satisfy: (1) $f^\beta = max_\alpha\{f^\alpha\}$, $f_{new}^\beta < f_{old}^\beta$; (2) $f_{new}^\alpha \leq f_{old}^\alpha$, if $f_{old}^\alpha > 1$ and $\alpha \neq \beta$; (3) $f_{new}^\alpha \leq 1$, if $f_{old}^\alpha \leq 1$ and $\alpha \neq \beta$; f_{old} / f_{new}: value before / after service replacement;
F3	$\rho'_i = max_{i,j}\{\triangle a_{ij}\}$, $\triangle a_{ij} > 0$
F4	$\rho'_i = max_{i,j}\{\triangle p_{ij}\}$, $\triangle a_{ij} < 0$
F5	$\rho'_i = max_{i,j}\{\triangle p'_{ij}\}$
F6	$\rho''_i = min_{i,j}\{\triangle p''_{ij}\}$

—if a link from F_i to F_j exists in the function graph, then from every service candidate for F_i to every candidate for F_j, there's a link;

—every service candidate v in the graph has a benefit value Φ and several QoS attributes;

—set network QoS attributes of the links between every two nodes;

—add a virtual source node v_s and sink node v_d. v_s is connected to all nodes without incoming links and v_d is connected to all nodes without outgoing links. The QoS attributes of these links are set to zero;

—add QoS attributes of the node to its incoming link and compute the utility of every link according to Definition 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 as shown in Figure 6.

The service selection problem is defined as follows.

Definition 4 (*QoS Service Selection Problem as MCSP*). Find a path from source v_s to sink v_d that produces the highest utility subject to the multiple QoS constraints specified by the user.

This is the well-known multiconstraint optimal path problem in the graph theory. Based on the algorithm of single-source shortest paths in directed acyclic graphs [Cormen et al. 2001], we propose the MCSP algorithm to handle the constraints requirements. The idea is to topologically sort all nodes in the service candidate graph, then visit nodes in topological order and relax it. In each node, we keep the list of paths from the source to the node meeting QoS constraints.

MCSP is shown in the Algorithms in Tables III and IV. One potential problem for MCSP is that, for every intermediate node, the number of paths in the

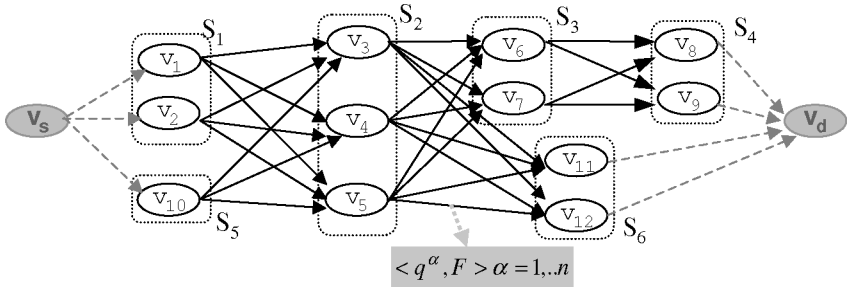


Fig. 6. Service candidate graph.

Table III. Algorithm MCSP

Algorithm MCSP ($G = (V, E)$, v_s, v_d, $\mathbf{Q}_c$)

//Definitions used in the algorithm:
v_s, v_d: source and target nodes in service candidate graph;
$\mathbf{q}(\mu, \nu)$: QoS of edge (μ, ν); $\mathbf{q}(\mu, \nu) = [q^1(\mu, \nu), \ldots, q^m(\mu, \nu)]$
$\mathcal{F}$: utility of edge (μ, ν) or path p;
$\mathbf{Q}(p)$: aggregated QoS of path p;
$\mathbf{P}(\nu)$: list of paths from v_s to node ν that satisfy QoS constraints.

```
1   Topologically sort nodes in G;
2   For each node μ, taken in topological order
3     For each ν ∈ adj[μ]
4       If (μ = v_s) then
5         Q ← q(μ, ν)
6         F ← F(μ, ν)
7         MCSP_RELAX(μ,ν,Q,F) 8        Else for each p ∈ P(μ)
9         Q ← Q(p) + q(μ, ν)
10        F ← F(p) + F(μ, ν)
11          MCSP_RELAX(μ,ν,Q,F)
12        End If
13    End For
14  End For

15  Find p*, s.t.∀p ∈ P(v_d), F(p*) ≥ F(p)
```

path list may be huge if none of them dominates each other. Thus may cause the algorithm to run very slowly and require a lot of memory spaces to store the paths. 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 heuristic algorithm is called MCSP-K. MCSP-K is different from MCSP only on the relax function where it checks the number of paths it currently keeps and removes the path with the maximum g_λ or ξ if the path number reaches K. The K-path selection criteria are based on the nonlinear cost function concept that is used to combine the multiple constraints into one [Korkmaz and Krunz 2001]. The cost function for any path p can be defined as:

$$g_\lambda(p) \triangleq \left(\frac{q^1(p)}{Q^1}\right)^\lambda + \left(\frac{q^2(p)}{Q^2}\right)^\lambda + \cdots + \left(\frac{q^m(p)}{Q^m}\right)^\lambda,$$

Table IV. Algorithm MCSP_RELAX

Procedure MCSP_RELAX (μ, ν, **Q**, $\mathcal{F}$)

```
1   If (∃α, Q^α > Q_c^α) then
2     return
3   End If
4   For each p ∈ P(ν)
5     If F(p) > F and ∀α, Q^α(p) ≤ Q^α then return
6     End If
7     If F(p) < F and ∀α, Q^α(p) ≥ Q^α then
8       remove p from P(ν)
9     End If
10  End For
11  Add path p' with attributes [μ, Q, F] to P(ν)
```

where $\lambda \geq 1$. $q^i(p)$ is the aggregated i^{th} QoS attribute for path p. As $\lambda \rightarrow \infty$, $g^*(p) \triangleq \lim_{\lambda\rightarrow\infty} g_\lambda(p)$ is equivalent to the cost function $\xi(p) \triangleq \max\{(\frac{q^1(p)}{Q^1}), (\frac{q^2(p)}{Q^2}), \ldots, (\frac{q^m(p)}{Q^m})\}$. The paths with K minimum g_λ or ξ values will be kept at each intermediate node. This ensures that MCSP-K will never prune out a feasible path if one exists.

5. SERVICE SELECTION ALGORITHMS FOR GENERAL FLOW STRUCTURE

Many real-world service processes have services that are not in strictly sequential order. They may have parallel operations to perform several services at the same time, conditional branch operations, and loops for using a service more than once in a flow.

The function graph for composite service with general composition patterns may contain complex structures among function nodes. In order to simplify the problem and construct a service candidate graph with a DAG structure, we first remove the loop operations by *unfolding* the cycles as in [Zeng et al. 2004]. A cycle is unfolded by cloning the function nodes involved in the cycle as many times as the maximal loop count.

5.1 Algorithms for the Combinatorial Model

To conduct service selection for the general flow structure using the combinatorial model, we map the problem to the 0-1 Integer Programming (0-1 IP) problem.

5.1.1 *IP Problem Formulation.* We need to identify three sets of input data: a set of service selection variables, a set of problem constraints, and an objective function. In a 0-1 IP problem, the objective function and the constraints are all linear while the selection variables are either 0 or 1. The 0-1 IP problem tries to maximize or minimize the value of the objective function by adjusting the values of the variables while meeting the constraints.

In our study, we identify two types of subgraphs.

—*Execution Route (R)*. We define an execution route as a route from start to end nodes in the function graph, which includes only one branch in each

conditional operation but all branches in parallel operations. Each execution path R_i has a probability ξ_i, which is the product of all probabilities for all conditional branches selected in the route. Therefore, $\sum_{i=1}^{K} \xi_i = 1$ if there are K execution routes in a composite service. For example, there are 2 execution routes in Figure 4:

$$\begin{aligned} R_1 &: \{S_1, S_2, S_3, S_4, S_6, S_7\} \text{ with a probability of } \xi_1; \\ R_2 &: \{S_1, S_2, S_3, S_5, S_6, S_7\} \text{ with a probability of } \xi_2; \end{aligned} \tag{2}$$

and $\xi_1 + \xi_2 = 1$.

—*Sequential Path (P)*. We define a sequential path as a path from start to end in the function graph, which includes only one branch in conditional operations and only one branch in parallel operations. So an execution route may contain more than one sequential path if there are parallel operations. For example, there are 3 sequential paths in Figure 4:

$$\begin{aligned} P_1 &: \{S_1, S_2, S_4, S_7\}; \\ P_2 &: \{S_1, S_2, S_5, S_7\}; \\ P_3 &: \{S_1, S_3, S_6, S_7\}; \end{aligned} \tag{3}$$

$P_1 \subset R_1$ and $P_2 \subset R_2$, while P_3 is a subset of both R_1 and R_2.

Now we show how to map the QoS service selection problem for the general flow structure to a 0-1 IP problem.

—*Variables definition*. The selection variables in the QoS service selection problem are x_{ij} ($0 \le i \le N, 0 \le j \le L$). If service s_{ij} is selected for class S_i, $x_{ij} = 1$; otherwise $x_{ij} = 0$.

—*Constraints definition*. There are three types of constraints.

(1) Exactly one service is selected for each service class; that is,

$$\forall i,\ 0 \le i \le N,\ x_{i1} + x_{i2} + \cdots + x_{iL} = 1. \tag{4}$$

(2) End-to-end QoS constraint for a sequential path. Every sequential path must meet the end-to-end response time constraint which guarantees that the longest branch's response time is still feasible. Suppose there are h such QoS requirements Q_c^α, $\alpha = 1, \ldots, h$ and $1 \le h \le m$. We define the following constraints for sequential path P_k:

$$\begin{gathered} \forall \alpha,\ 1 \le \alpha \le h,\ \sum_{i=1}^{N} \sum_{j=1}^{L} x_{ij} * q_{ij}^{\alpha} \le Q_c^{\alpha}, \\ \text{where } x_{ij} = 1 \text{ if } S_i \in P_k, \text{ otherwise } x_{ij} = 0. \end{gathered} \tag{5}$$

A total of h constraints are defined for each sequential path. For parallel paths on the same route, all of them must meet each of the Q_c^α.

(3) End-to-end QoS constraint for an execution route. This type of constraints applies to the sum of QoS attributes in parallel operations. For atomic services that are executed in parallel, the QoS value of the operations equals to the sum of all parallel branches. Suppose the QoS constraints are defined by Q_c^α ($\alpha = h+1, \ldots, m$). We define the following

constraints for each execution route R_k:

$$\forall \alpha,\ h+1 \le \alpha \le m,\ \sum_{i=1}^{N}\sum_{j=1}^{L} x_{ij} * q_{ij}^{\alpha} \le Q_c^{\alpha} \quad \text{where } x_{ij}=1 \text{ if } S_i \in R_k;\ \text{otherwise } x_{ij}=0; \tag{6}$$

A total of $m-h$ constraints are defined for each execution route. For parallel paths on the same route, together they must meet each of the Q_c^{α}. In other words, after all service classes are fixed, the total QoS resources used by all services must meet the QoS constraints.

Suppose the composite service has M sequential paths and K execution routes, and there are h QoS requirements with the end-to-end maximum property, the total number of constraints is $N + h * M + K * (m-h)$.

—*Objective function definition*. We have identified two objective functions:

(1) *EU.* The objective is to maximize the effective utility of all execution routes, that is,

$$max \sum_{k=1}^{K} \xi_k \mathcal{F}_k, \tag{7}$$

where ξ_k is the probability of execution route R_k, and $\mathcal{F}_k = \sum_{i=1}^{N}\sum_{j=1}^{l} x_{ij} * \mathcal{F}_{ij}$. In $\mathcal{F}_k$, $x_{ij}=1$, if $S_i \in R_k$; otherwise $x_{ij}=0$.

(2) *HP.* The objective is to maximize the utility of the route with the highest probability, that is,

$$max\ \mathcal{F}_{max},$$

where $\mathcal{F}_{max} = \sum_{i=1}^{N}\sum_{j=1}^{l} x_{ij} * \mathcal{F}_{ij}$, $x_{ij}=1$, if $S_i \in R_{max}$, otherwise $x_{ij}=0$. R_{max} is selected by $\xi_{max} = max_{k=1}^{K}\xi_k$.

5.1.2 *WS_IP Algorithm.* Once the QoS selection problem is modeled as a 0-1 IP problem, we can use well-known algorithms to find the optimal service selection. The IP problem is NP-complete [Cormen et al. 2001] and many algorithms have been studied such as the Branch-and-Bound algorithm, the Cutting Plane algorithm, and the Branch-and-Cut algorithm. Many IP solving tools have implemented these algorithms. In our project, we have designed one such algorithm, WS_IP, which uses the *lpx_intopt* routine in GLPK (http://www.gnu.org/software/glpk/) to find the optimal solution. This routine is based on the branch-and-bound method.

5.1.3 *WFlow Algorithm.* As in MMKP, the worst-case computation time for finding the optimal solution in an IP problem grows exponentially with the problem size. In order to improve the service selection performance, we present a heuristic algorithm, WFlow, to find near-optimal solutions in polynomial time.

Since there are conditional and parallel relationships among services, the overall QoS of a composite service is not the sum of its component services' QoS values. In WFlow, we use a stochastic workflow reduction algorithm (SWR) [Cardoso] to check the feasibility of a solution. SWR computes an aggregated QoS for a workflow step-by-step. At each step, a reduction rule is applied to shrink the workflow. This continues until only one atomic task is left in the

workflow, and the remaining task contains the QoS metrics corresponding to the workflow under analysis. We use the same idea to shrink the composite service until only one atomic service is left.

We have two versions of WFlow, based on different objective functions: *WFlow_EU* for the objective EU and *WFlow_HP* for the objective HP. WFlow_EU has a similar structure as WS_HEU, by finding a feasible solution at first and then trying to optimize the solution through several upgrades and downgrades. However, WFlow_EU uses different criteria to find the replacement items. The criteria used by WFlow_EU include the following.

(1) *Feasible solution.* A solution is considered feasible only if all execution routes and sequential paths can meet the corresponding constraint requirements.
(2) *Utility definition.* The utility used in the algorithm refers to the combined utility, which is defined in Equation (7).
(3) *Feasible upgrade.* An item for feasible upgrades must meet two conditions: (1) it will increase the combined utility; (2) it will not cause any execution route or sequential path to violate the constraint requirements;

WFlow_HP has a different structure from WFlow_EU. The main idea of WFlow_HP is to optimize the execution route with highest probability (R_{max}) at first but also find a feasible solution for all other routes. It has two parts. The first part uses a similar structure as WFlow_EU to find the optimal solution for R_{max}, and the second part tries to find the feasible solution for R_k, $\forall k \neq max$. WFlow_HP is different from WFlow_EU in the following aspects.

(1) *Initial solution feasibility check*. WFlow_HP only checks the feasibility of the execution route with the highest probability, while WFlow_EU will check the feasibility of all routes.
(2) *Solution optimization*. On the first pass of the optimization procedure, the algorithm only considers the feasibility of R_{max} without considering the constraint requirements for other routes. After we find the feasible solution for $R_k, k \neq max$, the optimization procedure will maintain the feasibility of all routes.
(3) *Feasible solution for* $R_k, k \neq max$. One more step is needed in WFlow_HP to find the feasible solution for other execution routes. The solution found in previous steps may need to be modified to accommodate all routes. If the utility of R_{max} has been downgraded, the algorithm will go back to Steps (2) and (3) to look for further improvements.

Similar to WS_HEU, WFlow has a polynomial complexity of $O(N^2(l-1)^2m)$.

5.2 Algorithms for the Graph Model

Figure 7 shows how a function graph in the general flow structure can be transformed to the graph model. Since loop operations can be represented by a sequential flow by unfolding the cycles, only sequential, parallel, and conditional operations are shown in the figure. A parallel or conditional operation contains a certain number of branches in the function graph. For example,

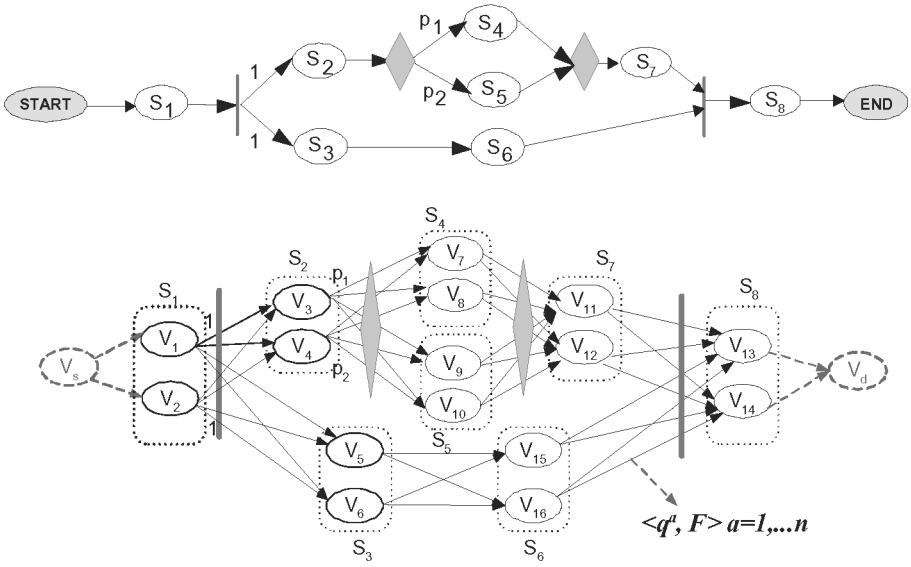


Fig. 7. Service candidate graph for general flow structure.

$\{S_2, S_4, S_7\}$ and $\{S_2, S_5, S_7\}$ are two branches for the conditional operation; $\{S_1, S_2, S_4/S_5, S_7, S_8\}$ and $\{S_1, S_3, S_6, S_8\}$ are two branches for the parallel operation.

To use MCSP and MCSP-K algorithms in a general flow structure, special considerations must be given to the following two types of services nodes:

(1) fork nodes for parallel and conditional paths, such as S_1 and S_2 in Figure 7. Since these nodes are on every branch of the parallel and conditional operations, the services selected must be the same for every branch;

(2) join nodes for parallel and conditional paths, such as S_7 and S_8 in Figure 7. Each parallel or conditional branch can be treated as an independent sequential flow. However, at join nodes, the services selected for different branches need to be merged together. Merge operations at parallel join nodes, such as merging V_{13}, V_{14} in service class S_8, are different from that at conditional join nodes, such as merging V_{11}, V_{12} in service class S_7. The operations are discussed as follows.

—Merge parallel branches.
Merging QoS Values. For parallel services, since all branches are executed simultaneously at runtime, the QoS values of all branches need to be combined. QoS parameters have two types based on their properties. For some parameters, such as cost, all branches should be added together for the overall QoS of the parallel operation. For other parameters, such as execution time, the maximum value of all branches is used as the overall QoS value for the parallel operation.
Merging Utility Values. When combining the utility of all branches, since all branches are executed concurrently, the sum of utilities is calculated as the overall utility value for parallel operations.

—Merge conditional branches.
Merging QoS Values. Since only one branch is selected for execution at runtime, as long as every branch could meet the QoS constraints, the conditional operation is guaranteed to meet the QoS constraints. Therefore, no additional QoS constraint checking is needed when merging conditional branches.

However, to decide if one path dominates another, as well as to perform MCSP-K algorithms, an operation to combine QoS values from different branches, denoted as $+_{conditional}\ (Q_1, Q_2, \ldots Q_n)$, is needed at the join node. Two types of combination are used for the objectives of *EU* and *HP*: (1) $+^{EU}_{conditional\_QoS} = \sum_{k=1}^{K} \xi_k Q_k$; (2) $+^{HP}_{conditional\_QoS} = \max\{Q_1, Q_2, \ldots, Q_n\}$.
Merging Utility Values. As defined in Equations (7) and (8), the utility values are merged depending on whether *EU* or *HP* is desired. If EU is adopted, then $+^{EU}_{conditional\_utility} = \sum_{k=1}^{K} \xi_k U_k$; otherwise if HP is adopted, $+^{HP}_{conditional\_utility} = \max\{U_1, U_2, \ldots, U_n\}$.

By including the described changes in MCSP and MCSP-K, we may perform service selections for the general flow structure. The details of the MCSP_General and MCSP-K_General algorithms for the general flow structure can be found in Yu [2006].

6. PERFORMANCE STUDY

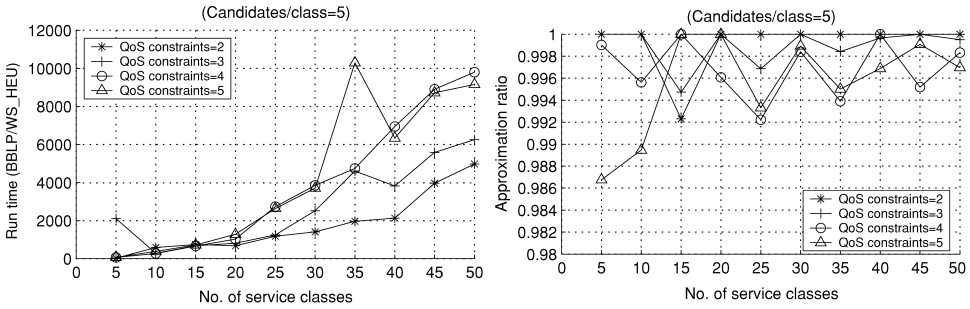
6.1 Performance Study of Algorithms for Sequential Flow Structure

We have used simulations to study the performance of the BBLP, WS_HEU, MCSP, and MCSP-K algorithms. Our study includes two parts: (1) the comparison of optimal and heuristic algorithms where we use runtime, approximation ratio (heuristic utility vs. the optimal value), and memory usage (for the graph approach) as metrics and (2) The comparison of combinatorial and graph models where we use provisioning success rate as a metric.

6.1.1 *Test Case Generation.* We use a degree-based Internet topology generator, Inet 3.0 [Winick and Jamin 2002], to generate a power-law random graph with 4,000 nodes to represent the Internet. We then randomly select $25 \sim 2{,}500$ nodes as service candidates and 2 nodes as the source and the sink. We assume an equal-degree random graph topology for the graph. The number of service classes in the process plan ranges from 5 to 50.

For simulation, 5 QoS attributes for each service q_{ij}^{α} ($\alpha = 1, 2, 3, 4, 5$) are randomly generated 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].

In our study, for each test scenario (representing different numbers of service classes and service candidates combinations), we generate 10 flow instances each with 10 sets of attributes for each instance. We then report the average value of the 100 cases.



(a) Running time (b) Utility

Fig. 8. WS_HEU vs. BBLP .

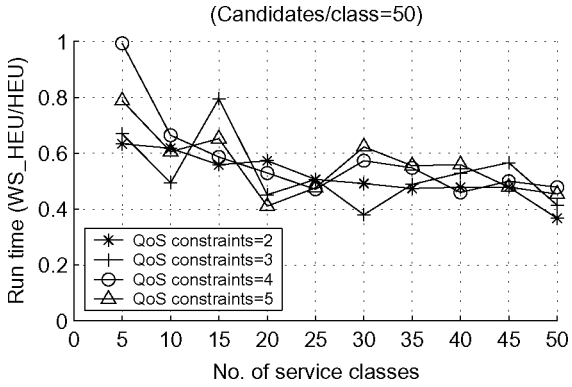


Fig. 9. WS_HEU vs. HEU .

6.1.2 *Result Analysis*

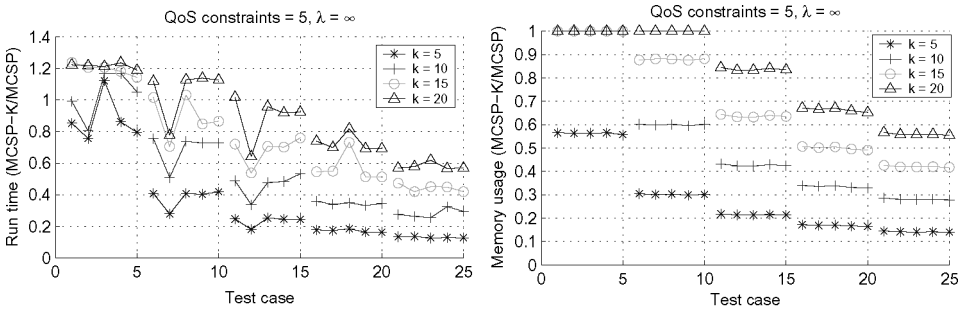
6.1.2.1 *WS_HEU vs. BBLP.* Figure 8 shows the running time and utility comparison between BBLP and WS_HEU algorithms when the number of QoS constraints range from 2 to 5. The number of service classes range from 5 to 50 with 5 candidates for each service class. We can see in Figure 8(b) that the utility generated by WS_HEU is close ($> 98.5\%$) to BBLP (i.e., the optimal result), and, in Figure 8(a), the running time is dramatically reduced (0.01% - 0.02% for 50 service classes).

6.1.2.2 *WS_HEU vs. HEU.* Figure 9 shows the running time comparison between WS_HEU and the original HEU. As discussed in Section 4.1.2, WS_HEU uses a different method to find the initial feasible solution and, as shown in Figure 9, saves 20%–70% running time while achieving the same utility. In our simulation, in 98% of the test cases feasible solutions are found on the first try in WS_HEU, while only 38% are found in HEU.

6.1.2.3 *MCSP-K vs. MCSP.* Table V shows the set-up of 25 performance runs for evaluating MCSP-K and MCSP. The performance runs are divided into 5 groups, each with the same number of service candidates (between

Table V. Test Cases for MCSP-K

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
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
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) Running time (b) Memory Usage

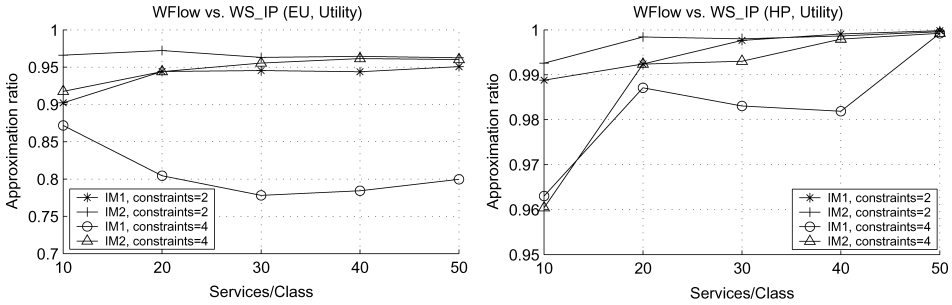
Fig. 10. Running time and memory usage comparison (MCSP-K/MCSP) .

10–50). For MCSP-K, λ(nonlinear cost) and k (number of stored paths) are set to 5, 10, 15, 20, 25, 30, ∞ and 5, 10, 15, 20, respectively. However, since $\lambda = \infty$ always has a better performance (in terms of utility) than other values, we only show $\lambda = \infty$ with different k values.

Figure 10 shows the running time and memory usage comparison of MCSP-K and MCSP under 5 QoS constraints, respectively. For both k values, MCSP-K can achieve a near-optimal performance (producing $> 90\%$ utility of MCSP). The advantage of MCSP-K is obvious as running time and space needed are significantly lower, while the performance remains nearly optimal ($> 95\%$ for $k = 10, 15, 20$). Therefore, for large process plans, using the heuristic algorithm MCSP-K can obtain a close-to-optimal solution quickly and avoid the memory growth problem.

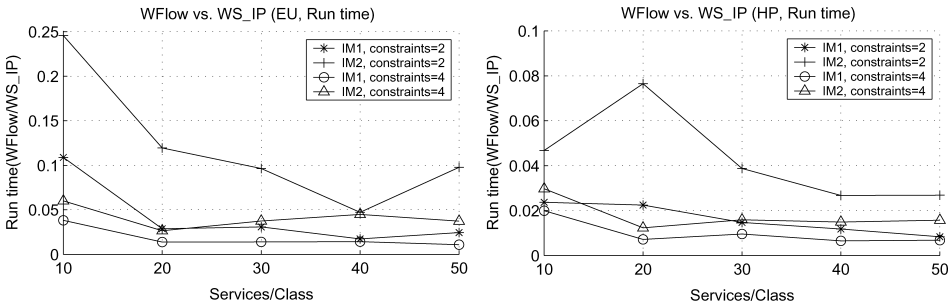
6.2 Performance Study of Algorithms for General Flow Structure

6.2.1 *Test Case Generation.* To compare the performances of WS_IP, WFlow, MCSP_General, and MCSP-K_General, we first randomly generate composite service structures with two or more execution routes. 100 structures are generated, each containing two or more composition patterns (sequential, parallel, conditional, loop) as shown in Figure 3. We then generate up to four QoS attributes and utility values for service candidates: response time, cost, availability, and reliability. Each is assigned a randomly generated value: $q_{ij}^{\alpha}(\alpha = 1, 2, 3, 4)$ with a uniform distribution between [50,200]. The utility of each service $\mathcal{F}_{ij}$ is also generated as a random value with a uniform distribution between $[1, 500]$. We also generate the utility $\mathcal{F}(\mu, \nu)$ of each link as a random value with a uniform distribution between [1,200]. Finally, to check the feasibility for each execution route, we run an implementation of the SWR algorithm [Cardoso].



(a) EU: Overall utility (b) HP: Highest probability path

Fig. 11. WFlow vs. WS_IP (approximation ratio) .



(a) Running time for EU (b) Running time for HP

Fig. 12. WFlow vs. WS_IP (Running time comparison) .

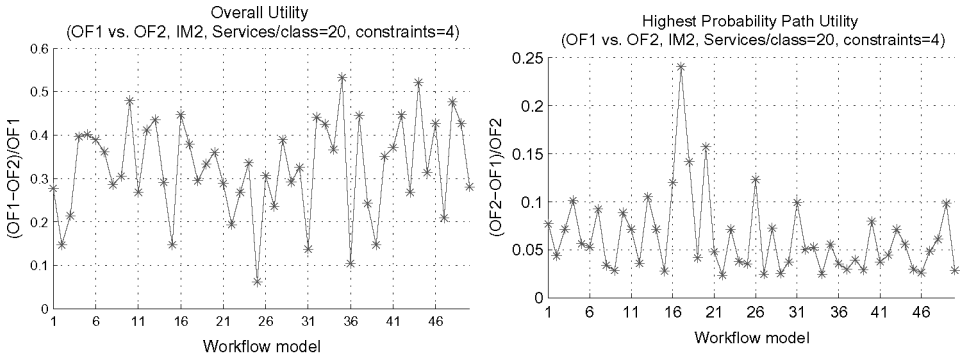
As in the previous section, we randomly generate 100 test cases and report the average value of the 100 rounds.

6.2.1.1 *WFlow vs. WS_IP.* To compare the performance of WFlow and WS_IP, we compare their runtime and approximation ratio (heuristic utility vs. the optimal utility).

We also compare the overall utility, the utility of the highest probability route achieved, and the computation time using two different methods to find the initial solution for WFlow: (IM1) Local selection where the service with the highest utility from each service class is selected and (IM2) MMKP selection where item ρ_i with $min_j\{max_\alpha\{\frac{q_{ij}^\alpha}{Q_c^\alpha}\}\}$ from each service class is selected.

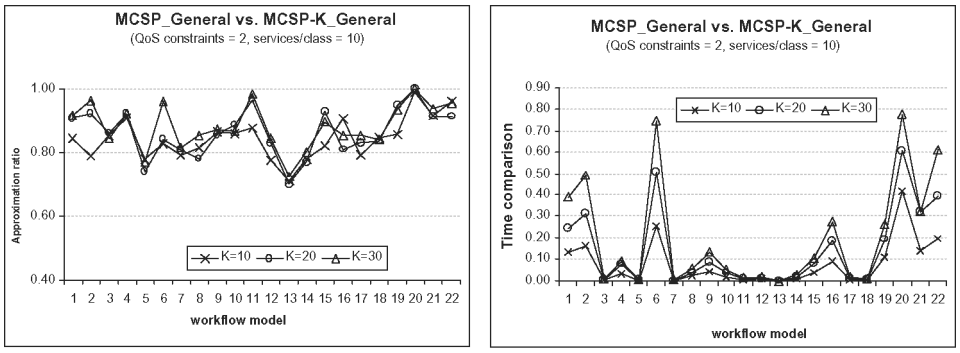
Finally, we compare the two optimization methods (objective function EU and HP) in WFlow by using the overall utility and the utility of the highest probability route generated by the algorithms.

Figures 11 and 12 present the overall utility and computation time for (a) EU optimization function, and (b) HP optimization function for the WFlow algorithm under different numbers of constraints and initial methods (IM1 and IM2). The simulation results show WFlow performs very well in all situations by achieving near-optimal results (approximation ratio $>$ 90% in most cases in Figure 11), while it only requires a small portion of the computation of WS_IP (less than 10% in most cases in Figures 12). As the number of QoS constraints increases, the advantage of using WFlow is more obvious.



(a) EU: Overall utility (b) HP: Highest probability path

Fig. 13. Optimization method comparison.



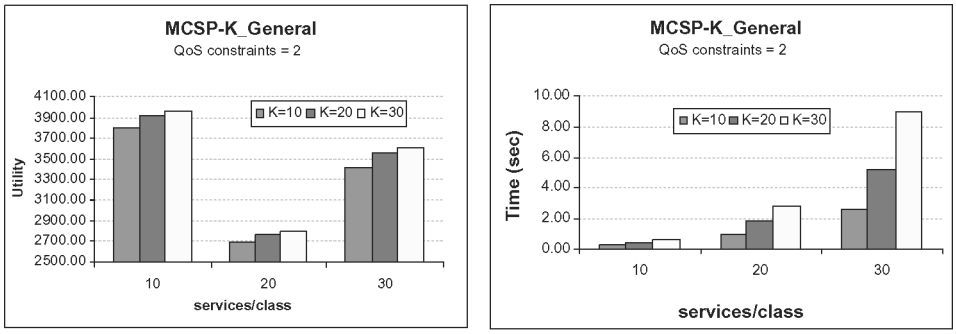
(a) Utility (b) Running time

Fig. 14. MCSP_General vs. MCSP-K_General (optimality and runtime).

Figure 11 also shows us that under the same number of QoS constraints, IM2 usually achieves a better performance than IM1, but it requires a longer computation time. For objective HP (Figure 11(b)), the performance of IM1 and IM2 are very close, and we can choose either one of them. However, For objective EU (Figure 11(a)), under 2 QoS constraints IM1 does not perform as well (approximation ratio around 80%). Therefore, in this case, we may prefer to use IM2 to find the initial feasible solution.

Figure 13(a) shows the overall utility comparison using algorithms of different objectives (EU and HP). As can be seen from the figure, the overall utility produced by EU is much higher than HP with an average gain of 32.53%. Figure 13(b) shows the utility of the highest probability path for algorithms using these two objective functions. In this case, HP achieves a higher utility than EU with an average gain of 6.32%. This study suggests that EU is a better objective in general since it improves the overall utility more significantly, while it suffers less on the highest probability path.

6.2.1.2 *MCSP_General vs. MCSP-K_General.* Figure 14(a) illustrates the approximation utility ratio of heuristic solution MCSP-K_General to optimal



(a) Utility (b) Running time

Fig. 15. MCSP-K_General (utility and running time).

solution MCSP_General. It can be seen that there is no guarantee that when K increases, a better utility will be achieved. The cause of this inconsistency is that the utility function is not used together with the cost function as the criteria to select K intermediate paths in MCSP-K algorithms. However, Figure 15(a) shows the average optimality of test cases increases with K in each group with the same service class size. It shows that MCSP-K_General does not always perform better with a larger K value.

Figure 14(b) shows the running time ratio of MCSP-K_General to MCSP_General and Figure 15(b) shows the absolute running time spent on MCSP-K_General with different K values and service class sizes. The two figures illustrate that performing MCSP-K_General can significantly reduce the running time compared to MCSP_General. It is clear that as K decreases, the time difference always increases.

7. RELATED WORK

Web service composition is an important resesrch area. One of the main approaches is the workflow-based service composition in which the composite services are defined similar to workflows that contain a set of atomic Web services with control and data flow among them [Rao 2004]. Several industry standards for service composition use this approach, such as BPEL (Web service Business Process Execution Language) [Curbera et al. 2003] and BPML (Business Process Modeling Language) [BPMI 2002]. HP's eFlow project [Casati et al. 2000] is a good example of workflow-based service composition that provides a dynamic and adaptive service composition mechanism for e-business process management. This approach is also used in our study on service composition.

QoS management in Web services is another important issue. QoS guarantee for Web services is one of the main concerns of the SLA framework [Ludwig et al. 2003]. 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. In [Zeng et al. 2004], authors present a QoS-aware middleware-supporting quality-driven Web service compositions. Two service selection approaches for constructing composite services have been proposed: local optimization and global planning. Their study shows that global planning is better than local optimization. [Agarwal et al. 2004] studies a similar approach in service selection with QoS constraints in global view. Both service selection methods are based on integer linear programming and best suited for small-size problems as its complexity increases exponentially with the increasing problem size.

Xiao and Boutaba [2005] presents an autonomic service provisioning framework for establishing QoS-assured end-to-end communication paths across administratively independent domains. The authors model the domain composition and adaptation problem as the classic k-multiconstrained optimal path (MCOP) problems and provide algorithms to solve it. However, their work doesn't address the parallel, conditional, or loop execution of services which are very common in business processes or workflows. We also adopt the MCOP model in our study as another possibility for defining service composition problems with extensions to handle parallel, conditional, and loop executions of services.

Similar to our project, earlier work by Zeng et al. [2004] defines multiple QoS criteria and takes into account of global constraints. However, service selection algorithms in this work have several issues:

(1) For service processes with more than one potential execution path (such as conditional branches), the service of a common node on different paths is optimized only for the *hot path* (the branch with the highest probability of execution). However, since actual executions many not use the hot path, some executions may not meet the QoS requirements.
(2) Another concept called the *critical path* (the path with the longest execution time) is used for making selection decisions. However, the critical path is identified by adding the worst-case performance of each service before services are selected, and thus may not reflect the actual longest path.
(3) The Integer Linear Programming (ILP) method is used to find the optimal selection. ILP has a high-time complexity and may not be suitable for systems with many services and dynamic service needs.

Our solution resolves all of the above issues by ensuring that all selected services meet the QoS requirements, and no critical path is identified before the service selection. Compared to the model in [Zeng et al. 2004], our model considers all execution routes and every branch in parallel operations. The solution (if one exists) is guaranteed to meet all QoS constraints, while in Zeng et al., may not. In addition, we propose a heuristic algorithm to find close-to-optimal solutions in polynomial time, which is more suitable for making runtime decisions. We also propose algorithms to handle rich

composition structure including sequential, parallel, conditional, and loops of services.

The success of all algorithms (ILP and heuristic algorithms) eventually depends on the correctness and the precision of the QoS data supplied. If accurate service QoS data are available, algorithms may produce excellent service selections. Otherwise, the results may deviate from the best ones. When QoS data are accurate and the problem size is small, we can use ILP to solve the problem, as proposed in Zeng et al. [2004] and Agarwal et al. [2004]. But ILP's high complexity does not work well for large systems with many service nodes and potential selections. This type of systems (precise data and large number of services) may be composed using our proposed algorithms.

8. CONCLUSIONS

We have studied the problem of service selection with multiple QoS constraints and proposed several algorithms. The algorithms are designed for two flow structures: for service processes with a sequential flow structure and for service processes with a general flow structure including loops, conditionals, and parallel operations. Both optimal and efficient heuristic algorithms have been presented. The performances of the algorithms have been studied by extensive simulations. We believe the proposed models and algorithms provide a practical solution to the end-to-end QoS guarantee on service processes.

REFERENCES

AGGARWAL, R., VERMA, K., MILLER, J., AND MILNOR, W. 2004. Constraint driven Web service composition in METEOR-S. In *Proceedings of the IEEE International Conference on Service Computing (SCC'04)*. Shanghai, China.

BPMI.org. 2002. Business Process Modeling Language (BPML), version 1.0. http://www.bpmi.org/bpml.esp.

CARDOSO, J. SWR algorithm. http://lsdis.cs.uga.edu/proj/meteor/Composition/SWR_Algorithm.htm.

CASATI, F., ILNICKI, S., JIN, L., KRISHNAMOORTHY, V., AND SHAN, M. 2000. Adaptive and dynamic service composition in eflow. Tech. Rep. HPL-200039, Software Technology Laboratory, Palo Alto, CA.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms* 2nd Ed. MIT Press.

CURBERA, F. GOLAND, Y., KLEIN, J., LEYMANN, F., ROLLER, D., THATTE, S., AND WEERAWARANA, S. 2003. Business Process Execution Language for Web Services, Version 1.1. http://www-106.ibm.com/developerworks/webservices /library/ws-bpel.

JAEGER, M. C., ROJEC-GOLDMANN, G., AND MHL, G. 2004. QoS aggregation for service composition using workflow patterns. In *Proceedings of the 8th IEEE International Conference on Enterprise Distributed Object Computing (EDOC'04)*. Monterey, CA.

KHAN, S. 1998. Quality adaptation in a multisession multimedia system: Model, algorithms and architecture. Ph.D. dissertation, Department of ECE, University of Victoria.

KHAN, S., LI, K. F., MANNING, E. G., AND AKBAR, M. 2002. Solving the knapsack problem for adaptive multimedia systems. *Studia Informatica Universalis 2*, 1, 157–178.

KORKMAZ, T. AND KRUNZ, M. 2001. Multi-constrained optimal path selection. In *Proceedings of 20th Joint Conference of IEEE Computer and Communications Societies (INFOCOM'01)*. 834–843.

LUDWIG, H., KELLER, A., DAN, A., KING, R. P., AND FRANCK, R. 2003. Web Service Level Agreement (WSLA) Language Specification. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

MARTELLO, S. AND TOTH, P. 1987. Algorithms for Knapsack problems. *Ann. Discrete Math. 31*, 70–79.

MENASCE, D. A. 2004. Composing Web services: A QoS view. *IEEE Internet Comput*.

RAO, J. 2004. Semantic Web service composition via logic-based program synthesis. PhD thesis. Department of Computer and Information Science, Norwegian University of Science and Technology.

WINICK, J. AND JAMIN, S. 2002. Inet 3.0: Internet topology generator. Tech. Rep. UM-CSE-TR-456-02 (http://irl.eecs.umich.edu/jamin/), University of Michigan.

XIAO, J. AND BOUTABA, R. 2005. QoS-aware service composition and adaptation in autonomic communication. *IEEE J. Select. Areas Comm. 23*, 12, 2344–2360.

YU, T. 2006. Quality of service (QoS) in Web services: Model, architecture and algorithms. Ph.D. thesis, University of California, Irvine, CA.

YU, T. AND LIN, K. J. 2005. A broker-based framework for QoS-aware Web service composition. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, Hong Kong, China.

ZENG, L. BENATALLAH, B., NGU, A., DUMAS, M., KALAGNANAM, J., AND CHANG, H. 2004. Quality-aware middleware for Web service composition. *IEEE Trans. Softw. Eng. 30*, 5, 311–327.

Received July 2006; revised December 2006; accepted December 2006