High Performance Approach for Multi-QoS Constrained Web Services Selection

Lei Li[1,2], Jun Wei[1], and Tao Huang[1,2]

[1]Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of Science and Technology of China, Anhui Hefei, China
{lilei,wj,tao}@otcaix.iscas.ac.cn

Abstract. In general, multi-QoS constrained Web Services composition, with or without optimization, is a NP-complete problem on computational complexity that cannot be exactly solved in polynomial time. A lot of heuristics and approximation algorithms with polynomial- and pseudo-polynomial-time complexities have been designed to deal with this problem. However, they suffer from excessive computational complexities that cannot be used for service composition in runtime. In this paper, we propose a high performance approach for multi-QoS constrained Web Services selection. Firstly, a correlation model of candidate services are established in order to reduce the search space efficiently. Based on the correlation model, a heuristic algorithm is then proposed to find a feasible solution for multi-QoS constrained Web Services selection with high performance and high precision. The experimental results show that the proposed approach can achieve the expecting goal.

1 Introduction

With the integration of Web services as a business solution in many enterprise applications, the QoS presented by Web services is becoming the main concern of both service providers and consumers. Providers need to specify and guarantee the QoS in their Web services to remain competitive and achieve the highest possible revenue from their business. On the other hand, consumers expect to have a good service performance. A service composition system that can leverage, aggregate and make use of individual component's QoS information to derive the optimal QoS of the composite service is still an ongoing research problem.

Since many available Web Services provider overlapping or identical functionality, albeit with different QoS, a choice needs to be made to determine which services are to participate in a given composite service. In general, multi-QoS constrained Web Services selection, with or without optimization, is an NP-complete problem that cannot be exactly solved in polynomial time [1], [2]. Heuristics and approximation algorithms with polynomial- and pseudo-polynomial-time complexities are often used to deal with this problem. However, existing solutions suffer from excessive computational complexities, and cannot be used for dynamic service selection at runtime. The complexity of multi-QoS constrained Web Services selection problem is reflected by the following factors:

(i) the huge number of the atomic candidate services that may be available to use; (ii) the large number of QoS constrained required by user; (iii) the different possibilities of composing an individual service into a service set which can satisfy the user's demand. The above difficulties make the problem very hard to solve.

In this paper, we propose a high performance approach for multi-QoS constrained Web Services composition. The correlations of all the candidate services are collected to construct a constrained model, which can reduce the search space efficiently. By using the constrained model, we propose a heuristic algorithm to find the feasible solution with high performance and high precision. We performed experiments to evaluate the validity and efficiency of the model in the final of the paper.

The remainder of the paper is organized as follows: Section 2 provides an overview of the related works. A service correlation model is then presented in Section 3. Section 4 proposes our algorithm and experimental results are shown in Section 5. Section 6 concludes the paper and introduces our future work.

2 Related Works

QoS support for Web services is among the hot topics attracting researchers from both academia and industry. Until recently, considerable efforts have been conducted to work on QoS for Web services. Multi-QoS constrained selection is a typical problem in many other research areas.

2.1 QoS Routing

QoS routing is very similar to multi-QoS constrained Web Services selection problem. In the last ten years, large numbers of studies have been proposed to address this issue. [3], [4], [5], [6] all present their approaches to solve this problem. In essential, the QoS routing problem is to create a feasible path from a given node to the destination so that the QoS requirements of the path are satisfied and the cost of the path is minimized. [5] proposes several optimal and heuristic algorithms for QoS partitioning, which assume that all nodes in the network have full topology and cost information, and then apply approximation algorithms to realize QoS partitioning. Though QoS routing is similar to multi-QoS constrained Web Services selection problem, there still remains tremendous distinction between the two. Compare to QoS routing multi-QoS constrained Web Services selection problem is based on the workflow model and the topology is immutable, consequently we only need to select a candidate service from each task node to keep user-defined constraints satisfied and make the QoS of the selected services optimal.

2.2 Multi-QoS Constrained Web Services Selection

QoS support in Web Services plays a great role for the success of this emerging technology. Essentially, the Multi-QoS Constrained Web Services Selection is an NP-complete problem that cannot be exactly solved in polynomial time. If the QoS attributes are all multiplicative or minimal attributes, the multi-QoS constrained services selection can be solved in polynomial time [8]. Hence, in order to simplify the problem, we only discuss the additive QoS attributes in this paper. Before analyzing,let's give a formal description of the problem.

Multi-QoS Constrained Web Services Selection(MCWS). For a composite service *CS*, its structure is specified as $CS \overset{\triangle}{=} (N, E)$, where N is the set of task nodes and E is the set of edges. Each task node $n_i \in N$ has $|n_i|$ candidate services and each candidate services s_j has K additive attributes which value is denoted as w_k^j, $k{\in}[1{,}K]$. Given K constraints $\{c_k,\ k{\in}[1{,}K]\}$, the problem is to select one service from each task node and aggregate all the selected services to form a specific service set S, *c(S)* is a cost function about S, S should satisfy the following two constraints:
(i) $w_k^S \leq c_k, w_k^S = \sum_{s_i \in S} w_k^i, k \in [1, K]$
(ii) $\forall\ S^{'}$, $c(S) \leq c(S^{'})$, $S^{'}$ is also a selected service set

The above problem is known with NP-complete computational complexity. In [7], [8], the authors propose a quality driven approach to select component services during execution of a composite service. They consider multiple QoS attributes such as price, duration, reliability, take into account of global constraints, and use the integer linear programming method to solve the service selection problem, which is too complex for run time decisions. [1] defines the problem as a multi-dimension multi-choice 0-1 knapsack problem or the multi-constraint optimal path problem. [2] describes an approach for QoS-aware service composition, based on composition of the QoS attributes of the component services and on genetic algorithms. Similar to [2], [9] uses genetic algorithms to determine a set of concrete services to be bound to abstract services contained in a orchestration to meet a set of constraints and to optimize a fitness criterion on QoS attributes. Compared with linear Integer Programming, GA can deal with QoS attributes with non-linear aggregation functions. [10] proposes an approach to trigger and perform composite service replanning during execution.

These studies can solve the Multi-QoS Constrained Web Services Selection problem; however, they suffer from excessive computational complexities, which make these solutions infeasible in many scenarios. Moreover, most of these studies assume that the same service interface definition is used by all atomic service candidates for a specific service component, i.e. these studies are not concerned about the compatibility issue among services. However, whether services are compatible is a major issue in the automatic composition of Web Services.

3 Service Correlation Model

A variety of approaches have been proposed to solve multi-QoS constrained service selection problem. As we mentioned, whether services are compatible is a major issue in the automatic composition of Web Services. Because incompatibility can lead to some collaboration mistakes, if the selected services are not compatible to each other, the service set can not be a feasible solution even though its overall QoS value is optimal. Moreover, the correlation of services is very useful in the search space reduction. Therefore, the correlation of services is very useful to improve the precision and performance of the selected services. In this section, we will describe our service correlation model and illuminate how to use it to reduce the search space.

3.1 Analysis of Correlations

There are lots of researches focusing on analysis of compatibility of Web Services interface [11], [12], seldom considering the multifaceted service correlation, which may lead to mistake in some situations. For example, s_i is a service to book a ticket. s_n and s_m are Visa and Master credit card payment services respectively. Service s_i can only be paid by Master, i.e. s_i is mutually exclusive to s_n. Our approach is to specify not only the interface compatibility but also more relations between services, and utilize these relations to reduce the search space.

Borrowing from some temporal operators defined in [13], we define that operator $X(s)$ outputs the service set next to s, operator $F(s)$ outputs the service set following s in the future. The operator $comp(x, y)$ means x and y are interface compatible.

Definition 1 (Sequence Relation). If $s_j \in X(s_i)$, then s_i and s_j have the sequence relation, $seq(s_i, s_j)$.
Definition 2 (Fork Relation). If $(s_j \notin F(s_i)) \vee (s_i \notin F(s_j))$, then s_i and s_j have the fork relation, $Fork(s_i, s_j)$.
Definition 3 (Adjoined Compatibility Relation). If $seq(s_i, s_j) \wedge comp(s_i, s_j)$, then s_i and s_j have the adjoined compatibility relation, $adj\_comp(s_i, s_j)$.
Definition 4 (Mutually Exclusion Relation). If s_i has been executed, s_j should never be executed and vice verse, then s_i and s_j have the mutually exclusion relation, $MuExcl(s_i, s_j)$.

The above correlation between services must be analyzed before selecting the composite services, because these analyses can help program to avoid choosing some incompatible services. Moreover, these analyses can help program to accelerate the selection. There are also many other correlations, however in this paper we do not enumerate them all.

3.2 Service Correlation Model

Definition 5 (Incompatible Service set, ISS). Each service s_i has an incompatible service set $S_i^{\varnothing}$, which means if the service s_i has been selected, then any service in $S_i^{\varnothing}$ should not be selected at the same time. How to construct the incompatible service set will be described below.

R1: If $s_j \in X(s_i) \wedge \neg adj\_comp(s_i, s_j)$ then $S_i^{\varnothing} \leftarrow S_i^{\varnothing} \cup \{s_j\}$

Rule 1 means if s_j is the next service executed after s_i and s_i is incompatible with s_j, then s_j will be included in $S_i^{\varnothing}$.

R2: If $s_j \in F(s_i) \wedge MuExcl(s_i, s_j)$ then $S_i^{\varnothing} \leftarrow S_i^{\varnothing} \cup \{s_j\}$

Rule 2 means if s_j will be executed after s_i and s_j is mutually exclusive to s_i, then s_j will be included in $S_i^{\varnothing}$.

R3: If $Fork(s_i, s_j)$ then $S_i^{\varnothing} \leftarrow S_i^{\varnothing} \cup \{s_j\}$

Rule 3 means if s_i and s_j have fork relation, then s_j will be included in $S_i^{\varnothing}$.

Rules 1-3 are backward compatible, which guarantee the service s_j executed after s_i should be compatible to s_i. Using rules 1-3, we can construct an incompatible service set of s_i.

Assuming algorithm is considering selecting a candidate service from the task node n_j. Let $S^{\varnothing}$ be the incompatible services found in selecting round, i.e. $S^{\varnothing} = \bigcup_{1 \le i < j} S_i^{\varnothing}$. Obviously if $|S^{\varnothing}| = 0$, then every service is available to be selected. We can create the overall incompatible service set by the following rules. Assuming S_{n_i} is the service set bound to task node n_i.

R4: $\forall s \in S_{n_i}$, If $s \in S^{\varnothing}$, then set s_j as the unavailable service.

Rule 4 means that a service is unavailable service when it is in the incompatible service set.

R5: If $((\forall s \in S_{n_i}) \rightarrow (s \in S^{\varnothing}))$, then no available service can be selected.

Rule 5 means if all the services belong to task node n_i are in $S^{\varnothing}$, then the selecting program will terminate the searching of current round and begin a new round searching.

The incompatible service set $S^{\varnothing}$can be created within $O(\sum_{1 \le i < j} |n_i|)$ time . $S^{\varnothing}$ can help algorithm to reduce the search space efficiently and in the next subsection, we will analyze the efficiency of the correlation model. The efficiency of this model is determined by the size of incompatible service set, i.e. the bigger the size of incompatible service set, the more space reduction we can get.

4 The Proposed Algorithm

In this section, we present our proposed algorithm H_MCWS, which attempts to find a feasible service set subject to K additive user's constraints and minimize the cost of that service set.

4.1 Theoretical Foundation

First, we design a nonlinear cost function to evaluate the QoS value of the selected service set. The same nonlinear cost function was also used in [3] and [14] to develop algorithm for the Multiple Constrained Path problem. Assuming S is the selected service set. Consider the following cost function for S.

$$g_\lambda(S) = (\frac{w_1^S}{c_1})^\lambda + (\frac{w_2^S}{c_2})^\lambda + ... + (\frac{w_K^S}{c_K})^\lambda,\ where\ \ \lambda \ge 1 \qquad (1).$$

From the cost function, we can get the following theorems. (Other characteristics of the nonlinear function can be found in [3])

Theorem 1: If λ=1, the minimal $g_1 S$ can be found in polynomial time.

Proof: When λ=1, then the cost function is $g_1(s) = (\frac{w_1^S}{c_1}) + (\frac{w_2^S}{c_2}) + ... + (\frac{w_K^S}{c_K})$. Hence, we only need to select a service with the minimal cost from each node. Therefore, the complexity is $\sum_{n_i \in N} |n_i|,\ i.e.\ O(|S|)$. □

Theorem 2: When λ is close to ∞, it is guaranteed to find a feasible service set if one exists.

Proof: Let S be a service set that minimizes the cost function $g_{\lambda\rightarrow\infty}$. Assuming there is a feasible service set S_*. Therefore, $g_{\lambda\rightarrow\infty}(S) \leq g_{\lambda\rightarrow\infty}(S_*)$. If S is not a feasible service set, then $\exists k \in [1, K]$, $w_k^S > c_k$. When $\lambda \rightarrow \infty$, $g_{\lambda\rightarrow\infty}(S)$ is dominated by the largest term. Hence, $g_{\lambda\rightarrow\infty}(S) \rightarrow \infty$ and $g_{\lambda\rightarrow\infty}(S_*) \rightarrow 0$, i.e. $g_{\lambda\rightarrow\infty}(S) > g_{\lambda\rightarrow\infty}(S_*)$. Since this contradicts, we must have $w_k^S \leq c_k$ for each k. Therefore, S is a feasible service set. □

When we set λ=1, the algorithm can find the minimal cost in polynomial time. But unfortunately, the selected service set may not be the feasible service set. Theorem 2 can guarantee to find a feasible service set when $\lambda \rightarrow \infty$. But unfortunately, when $\lambda \geq 2$, it is impossible to provide an exact polynomial time algorithm. So, a heuristic algorithm must be proposed to solve this problem.

4.2 Proposed Algorithm

In this section, we present our algorithm H_MCWS, which attempts to find a feasible service set which satisfies all the users' constraints and simultaneously minimize the cost of that service set. H_MCWS is similar to the H_MCOP [3]. The differences of them are that H_MCWS is used to select the composite services and faster than H_MCOP. First, H_MCWS traverses all the services to eliminate the service which does not satisfy the multiplicative attributes and minimal attributes and create incompatible service set for each candidate service. This traversing process will be completed with $O(|S|)$ complexity. Second, the algorithm first finds the best service set with g_1. If the service set satisfies all the constraints, it is exactly the result and will be returned to user. Otherwise, H_MCWS finds the best temporary service set from each task node n_u to n_t. It then starts from task node n_s and discovers each task node n_u based on the minimization of $g_\lambda(S)$, where the service set S is from task node n_s to n_t and passing through task node n_u. S is determined at task node n_u by concatenating the already traveled segment from task node n_s to n_u and the estimated remaining segment from task node n_u to n_t. A pseudo code of H_MCWS is shown below. n_s represents the start task node, n_t represents the end task node and n_u represents the middle task node.

```
H_MCWS Algorithm
1. Deal_non_additive_Attributes($N$)
2. Create_ISS_Set($N$)
3. Reverse_Relax($N$,$n_t$)
4. if $\forall s_i \in S_{n_s}$, t$[s_i] > K$, then return error
5. Look_Ahead($N$)
6. if $\exists s_i \in S_{n_t}, G_k[s_i] \leq c_k, k \in [1, K], then\ return\ this\ services\ set$
7. return error
```

The algorithm uses the following notations. $t[n_u]/t[s_i]$ represent the minimal cost of the selected services from task node n_u/service s_i to n_u. Notation $R_k[n_u]$, k∈[1,K] represents the individually accumulated link weights along the above selected services. Notation $g[n_u]$ represents the cost of a foreseen complete service set that goes from task node n_s to n_t. Notation $G_k[n_u]$, k∈[1,K] represents the individually accumulated cost of services weights from task node n_s to n_u. $c[n_u]$ represents the cost along the already selected segment of this service set from task node n_s to n_u. Deal_non_additive_Attributes(N algorithm is used to handle the non-additive QoS attributes and Create_ISS_Set(N) algorithm is used to create the incompatible service set for each candidate services. There are two directions in H_MCWS: backward to estimate the cost of the remaining segment using λ=1 and forward to find the most promising service set in terms of feasibility and optimality using λ >1. The backward algorithm and forward algorithm are shown in Reverse_Relax algorithm and Look_Ahead algorithm respectively.

```
Reverse_Relax($n_u, n_v$) Algorithm
1. t$[n_u]$=t$[n_v]$+$min_{1\leq i\leq|n_u|}\{\sum_{1\leq k\leq K}\frac{w_k^i}{c_k}\}$
2. t$[s_i]$=t$[n_v]$+c$[s_i]$ (for i=1 to $| n_u |, s_i \in n_u$)
3. $R_k[n_u]$=$R_k[n_v]$+$min_{1\leq i\leq|n_u|}\{w_k^i\}$ (for $k$=1 to $K$)
4. $R_k[s_i]$=$R_k[n_v]$+$w_k^i$ (for $k$=1 to $K$)
```

```
Look_Ahead($n_u$) Algorithm
1. for each service $s_i \in n_u$ begin
2.     if$s_i \in S^{\varnothing}$ then continue
3.     if$\lambda < \infty$ then $g[s_i] = max\{\frac{G_k[\pi_p[n_u]]+w_k^i+\pi_s[n_u]}{c_k},\ k \in [1, K]\}$
4.     $G_k[s_i] = G_k[\pi_p[n_u]] + w_k^i$ (for $k$=1 to $K$)
5.     $R_k[s_i]$=$R_k[\pi_s[n_u]]$+$w_k^i$ (for $k$=1 to $K$)
6.     $s_b = Choosing\_Best\_Service(n_u)$
7.     $c[n_u] = c[\pi_p[n_u]] + c[s_b]$
8. end
```

In the backward direction, the Reverse_Relax algorithm finds the optimal service set from every task node n_u to n_t using λ. The complexity of the backward direction is $O(| S |)$. $\pi_p[n_u]$ and $\pi_s[n_u]$ represent the predecessor and successor of task node n_u respectively. Look_Ahead algorithm is executed in the forward direction. This procedure uses the information provided by the Reverse_Relax algorithm. Look_Ahead algorithm explores the whole workflow by choosing the next services in specific task nodes based on the rule below.

```
Choosing_Best_Service($n_u$) Algorithm
1. Let $s_v$ be a virtual service in $n_u$
2. c[$s_v$]=$\infty$, $G_k[s_v] = \infty$, $R_k[s_v] = \infty$
3. for each service $s_i \in n_u$ begin
4.     if $(c[s_i] < c[s_v])\&(\forall k(G_k[s_i] + R_k[s_i]) \leq c_k), then$
5.         $s_v = s_i, c[s_v] > c[s_i], G_k[s_v] = G_k[s_i], R_k[s_v] = R_k[s_i]$
6.     else if $(c[s_i] > c[s_v])\&(\forall k(G_k[s_v] + R_k[s_v]) \leq c_k), then\ continue$
7.     else if $g[s_i] < g[s_v]$ then
8.         $s_v = s_i, c[s_v] = c[s_i], G_k[s_v] = G_k[s_i], R_k[s_v] = R_k[s_i]$
9. end
10.return $s_v$
```

The above preference rule can choose the best service from the specific task node n_u. In the end, H_MCWS returns a service set using $\lambda > 1$. As λ increases, the likelihood of finding a feasible service set also increases. When λ is close to ∞, H_MCWS can guarantee to find a feasible service set if one exists.

Lemma 1: If there are one additive QoS attributes, we can find k-minimal cost services set with the complexity $O(k| S |)$.

Proof: If there are only one additive QoS attributes, we can get the best candidate services from each node by $| n_i |$ comparisons. So, we can find the best services set within $\sum_{1\leq i\leq |N|} | n_i |$ comparisons, i.e. $O(|S|)$. Let w(s) represent the additive QoS value of service s. Assuming s_i is the selected service from task node n_i, $s_i \in S$ and $s_i^{'}$ represent the service with the second minimal QoS value, i.e. if $(\forall s_i^*, s_i^* \neq s_i)$, then $w(s_i^*) \geq w(s_i^{'})$. Assuming $s_j^{'}$ satisfies the following rule: if $\forall s_i, s_i^{'} \in n_i$, then $(w(s_i^{'}) - w(s_i)){\geq}(w(s_j^{'}) - w(s_j))$. Let $S^{'} = S - \{s_j\} \cup \{s_j^{'}\}$, then $S^{'}$ is the second minimal cost services set. Finding the specific service $s_j^{'}$ needs $O(|S|)$ times comparisons at the worst case. Hence, we can find 2 minimal cost services set with the complexity $O(2|S|)$. According to the same procedure, we can find k-minimal cost services set with the complexity $O(k| S |)$. □

Theorem 3: The MCWS problem can be solved by H_MCWS algorithm in time $O((k{+}1)|\mathrm{S}|)$.

Proof: The algorithm can be executed within . Similar to H_MCOP, the forward direction of H_MCWS can also be used with the k-shortest algorithm (in MCWS problem, the algorithm is k-minimal cost service set algorithm). As we proved in lemma 1, it needs time to find k-minimal cost service set. Hence, the overall complexity of H_MCWS algorithm is $O((k{+}1)| S |)$. □

Although H_MCWS is similar to H_MCOP, it is more efficient to solve the MCWS problem. Theorem 3 did not consider the influence of the correlation of services. In real practice, H_MCWS can get more performance improvement.

5 Experiments and Evaluation

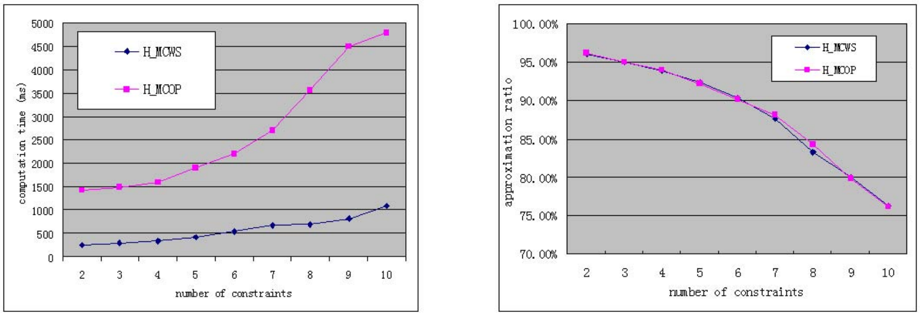
In this section, we investigate the performance of H_MCWS algorithm and compare it to the most promising algorithms selected from the ones surveyed in section 2. The simulations environments are: Pentium IV 2.8G CPU, 1024M RAM, and the operation system is Windows XP SP2. In our study, two important aspects are considered, one is computation time and the other is the excellence in approximating the optimal solution.

5.1 Comparison of H_MCWS with H_MCOP

To study the performance of H_MCWS, we randomly create a composite services structure with 20 task nodes and each node has several candidate services. We analyze the impact of i) varying the number of constraints; (ii) varying the number of candidate services. In these test groups, we did not consider the influence of the correlation of services.

(i) Analysis the impact of the number of constraints

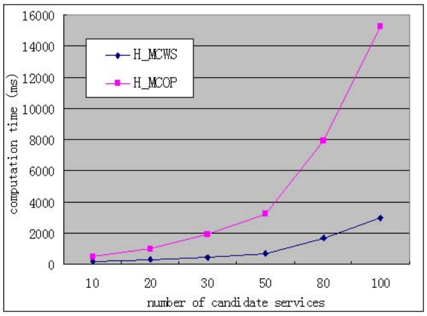
In this test case, we generate 30 candidate services for each task node, and we set the number of additive constraints from 2 to 10 and use $\lambda = 20$.



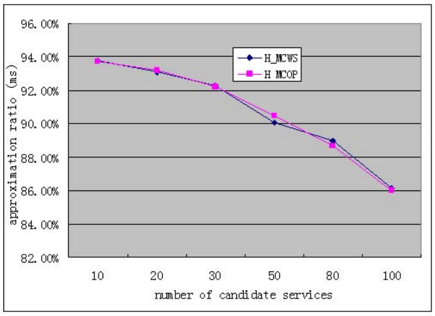
(a) computation time (b) approximation ratio

Fig. 1. Impact of the Number of Constraints

From Figure 1(a) we can find that H_MCWS performs much better than H_MCOP. Figure 1(b) demonstrates the probabilities of finding optimal service set for the two algorithms with the different number of constraints. This experiment results show H_MCWS achieves higher performance while keeping approximately the same precision comparing to H_MCOP.



(a) computation time



(b) approximation ratio

Fig. 2. Impact of the number of candidate services

(ii) Analysis of the impact of the number of candidate services

In this test case, we generate [10, 100] candidate services for each task node. The number of the additive constraints is 5 and use $\lambda = 20$.

Figure 2 depicts that with the increase of the number of candidate services, the precision and performance of the algorithm drop slightly. For H_MCOP algorithm, the complexity is $O(n\log(n)+km\log(kn)+(k^2+1)m)$, where n represents the number of task nodes and m is the number of links. In MCWS problem, the $m = \prod_{1\leq i\leq|N|} | n_i |$, which is a very huge number. For example, if there are 20 task nodes and each node has 20 candidate services, then m=20^{20}. Therefore, H_MCOP is not suitable to solve this problem directly.

5.2 Analysis of Impact of Service Correlation

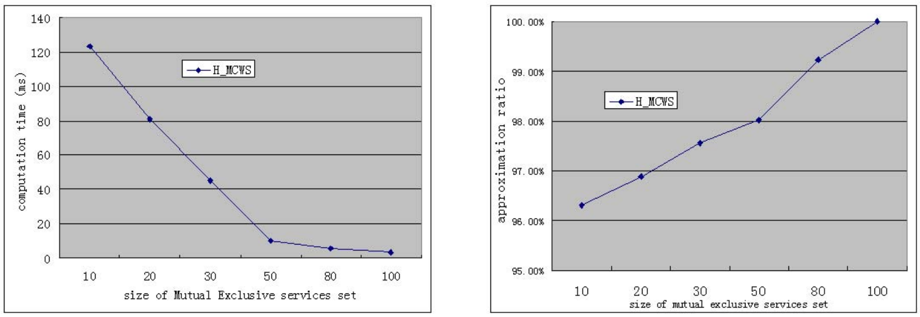
To study the impact of service correlation model, we randomly create a composite services structure with 20 task nodes and each node has 30 candidate services. Each candidate service has an ISS set and we set the size of this set from 10 to 100. The number of the additive constraints is 5 and use $\lambda = 20$.

Figure 3 shows the performance and precision of the algorithm increase dramatically with the increasing of the size of ISS. This illuminates the service correlation model proposed in this paper is effective. The above experiments demonstrate the service correlation model and H_MCWS algorithm proposed in this paper is feasible.

5.3 Evaluation and Comparison

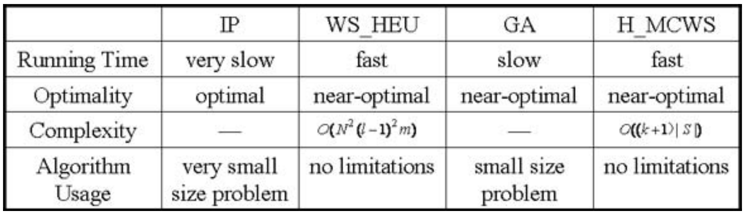
In this subsection, we analyze and compare four different algorithms: Integer Programming in [7], WS_HEU in [15], Genetic Algorithm in [9] and our approach H_MCWS. Figure 4 presents the comparison results.

From Figure 4, we can see that four different algorithm have the different properties and are suitable to the different scenarios. Integer programming is one



(a) computation time (b) approximation ratio

Fig. 3. Impact of the service correlation



	IP	WS_HEU	GA	H_MCWS
Running Time	very slow	fast	slow	fast
Optimality	optimal	near-optimal	near-optimal	near-optimal
Complexity	—	$O(N^2(l-1)^2 m)$	—	$O((k+1)\|S\|)$
Algorithm Usage	very small size problem	no limitations	small size problem	no limitations

Fig. 4. Comparison of Algorithms

of the most adopted tools to solve a QoS-aware composition problem. Integer programming can find the optimal solution, but unfortunately the running time is very slow which makes it only can be used in very small size problem. Genetic algorithm can represent a more scalable choice and are more suitable to handle generic QoS attributes. However, the genome size of GA is bound to the number of services, which makes GA slow when the number of candidate services is large. WS_HEU and H_MCWS have no limitations and can be used in every situation. H_MCWS consider the correlation of service and use it to reduce the search space. Hence, although it is near-optimal, it performs very well in practice.

6 Conclusions and Future Work

Web Services selection subject to multi-QoS constraints is an NP-complete problem. Previously proposed algorithms suffer from excessive computational complexities and are not concerned about the compatibility issue among services, which makes that these approaches can not be used in many applications. In this paper, we proposed an efficient approach for Web Services selection with multi-QoS constraints. The complexity of the algorithm is lower and the simulation results show the algorithm can find the feasible solution with high performance and high precision. We believe the proposed models and algorithms provide a useful engineering solution to multi-constrained Web Services selection problem.

User preference is an important factor in the service selection. It can help algorithm to find a more satisfying composite services for user. Moreover, user preference can help algorithm to reduce the search space. In the future, we will introduce the user preference into the selecting algorithm to further reduce the search space and gain more precision improvement.

Acknowledgments. The authors would like to thank the anonymous reviewers for their invaluable feedback. This Work is supported by the National Natural Science Foundation of China under Grant No. 60673112, the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312005 and the High-Tech Research and Development Program of China under Grand No. 2006AA01Z19B.

References

1. Yu, T., Lin, K.-J.: Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 130–143. Springer, Heidelberg (2005)
2. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: A lightweight approach for QoS-aware service composition. In: ICSOC (2004)
3. Korkmaz, T., Krunz, M.: Multi-Constrained Optimal Path Selection. In: INFOCOM 2001. Proceeding of 20th Joint Conf. IEEE Computer and Communications, pp. 834–843 (2001)
4. Wang, B., Hou, J.: Multicast routing and its QoS extension: Problems, algorithms, and Protocols. IEEE Network 14(1), 22–36 (2000)
5. Vogel, R., et al.: QoS-based routing of multimedia streams in computer networks. IEEE Journal on Selected Areas in Communications 14(7), 1235–1244 (1996)
6. Lorenz, D.H., Orda, A., Raz, D., Shavitt, Y.: Efficient QoS partition and routing of unicast and multicast. In: Proc. IEEE/IFIP IWQoS, pp. 75–83 (2000)
7. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: WWW. Proc. 12th Int'l Conf. World Wide Web (2003)
8. Liang-Zhao, Z., Boualem, B., et al.: QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering 30(5), 311–327 (2004)
9. Canfora, G., Di Penta, M., Esposito, R., et al.: An Approach for QoS-aware Service Composition based on Genetic Algorithms. In: GECCO'05 (2005)
10. Liu, Y., Ngu, A.H., Zeng, L.: QoS computation and policing in dynamic web service selection. In: WWW. Proceedings of the 13th International Conference on World Wide Web, pp. 66–73. ACM Press, New York (2004)
11. Megjahed, B., Bouguettaya, A., Elmagarmid, A.K.: Composing Web Services on the semantic Web. The VLDB Journal (2003)
12. Lamparter, S., Ankolekar, A., Grimm, S.: Preference-based Selection of Highly Configurable Web Services. In: WWW. Proceedings of International Conference on World Wide Web (2007)
13. Huth, M., Ryan, M.: Logic in Computer Science: Modeling and Reasoning about Systems, 2nd edn. Cambridge University Press, Cambridge (2004)
14. De Neve, H., Van Mieghem, P.: A multiple quality of service routing algorithm for PNNI. In: Proceedings of the ATM Workshop, pp. 324–328. IEEE Press, Los Alamitos (1998)
15. Yu, T., Zhang, Y., Lin, K.-J.: Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints. ACM Transaction on Web (May 2007)