An Advanced QoS-Based Web Service Selection Approach

Wen-long ZHU\textsuperscript{1}, Bei-bei YIN\textsuperscript{1, a}, Si-qian GONG\textsuperscript{1} and Kai-Yuan CAI\textsuperscript{1}

\textsuperscript{1}School of Automation Science and Electrical Engineering, Beihang University, Beijing, China

\textsuperscript{a}yinbeibeibuaa.edu.cn

*Corresponding author

**Keywords:** Web service selection, Pareto optimal solution, Advanced fully polynomial time approximation scheme (A\textsuperscript{2}-FPTAS), Unequal local error bound

**Abstract.** QoS-based web service selection is an important aspect for achieving efficient operations for web service systems. The aim of web service selection is to select an appropriate concrete web service with the best quality of service (QoS) for each abstract web service in a workflow. One way to resolve this problem is to calculate the Pareto optimal solutions which have the better QoS values for some QoS attributes while having at least equivalent values for others. Although a lot of approaches can do that, they do not guarantee the result precision or have prohibitively large overhead. In this paper, we present an Advanced A-Fully Polynomial Time Approximation Scheme (A\textsuperscript{2}-FPTAS) by the unequal local error bound to balance the precision and the overhead. Experimental results are presented to show the efficiency of this approach.

**Introduction**

Web service technology has been increasingly important and popular in various IT applications. In order to improve the capability of web services to satisfy the expected quality of service (QoS) of users’ applications in the dynamic environment, we need an effective approach to selecting the needed services for the workflow of the application. The service selection approach needs to select an optimal concrete service from the candidate services for each abstract web service in the workflow of the application.

Since referring to multiple QoS attributes such as response time, throughput and reliability, web service selection is a quality-driven service selection (QDSS) \cite{1} and also a multi-objective optimization problem where different QoS attributes correspond to different and possibly conflicting objectives \cite{2}.

A number of approaches developed to resolve the QDSS problem can be classified in two categories. The first category is utility-based to use a single objective function which frequently has numeric weighting factors for the user’s requirements and return a single optimal (or suboptimal) solution, such as \cite{3}, \cite{4}, \cite{5}. It is hard for these approaches to set suitable weighting factors satisfying the user’s expected QoS.

The second category of approaches embraces the concept of dominance and returns a Pareto-optimal set, as proposed in \cite{2}, \cite{6}-\cite{9}. However, some approaches in this category assure the acceptable algorithm complexity but neglect the guarantee on results’ approximation precision, such as \cite{6}, \cite{7}. Conversely, other approaches neglect the prohibitively computational cost even though ensure the optimal approximation precision, such as \cite{8}, \cite{9}. In \cite{2}, it presents a fully polynomial time approximation scheme...
(A-FPTAS) algorithm which aims at the sweet spot between these two extremes. In A-FPTAS, it sets a global error bound for the solutions in the Pareto set and an equal local error bound for each abstract web service.

In this paper, we present an advanced A-FPTAS (A²-FPTAS) algorithm to resolve the QDSS problem. A²-FPTAS adopts the strategy of calculating the unequal error bound for each abstract web service. This strategy could improve the performance and the result precision of A²-FPTAS for the same global error bound comparing with A-FPTAS algorithm. The experimental results demonstrate the efficiency of A²-FPTAS algorithm.

The rest of this paper is organized as follows. Section 2 discusses the current state of the art and related work about web service selection approaches. Section 3 presents the models for our service selection approach. Section 4 describes our web service selection approach. The experimental results in section 5 demonstrate the efficiency of A²-FPTAS. Finally, conclusions and future work will be included in section 6.

**Current State of the Art and Related Work**

More than a decade, web service selection problem has attracted great interest in the research community. A number of approaches developed to resolve web service selection problem can be divided into two categories.

The first category is utility-based to use a single objective function. [1] and [10] are among the first to propose integer linear programming approach which defines a set of variables, a set of linear constraints and a linear utility function. Then, [11] extends its limitation on workflow and service models. These approaches can solve the web service selection problem optimally but have exponential time complexity. To reduce the time complexity, [3] proposes Genetic Algorithms (GAs), in which the web service selection problem is modeled as an optimization problem. Some other heuristic approaches also have been proposed. For example, [12] presents an accurate sub-swarms particle swarm optimization algorithm using sub-swarms searching grid cells to improve the accuracy of the standard PSO algorithm. [13] formalizes the web service selection problem as a multi-dimensional multi-choice 0-1 knapsack problem and as a multi-constraint optimal path problem.

The second category is Pareto-based, which embraces the concept of dominance and returns a Pareto set. This category has two subcategories. One subcategory of approaches assures the acceptable algorithm complexity but neglects the guarantee on the results’ approximation precision. [6] presents algorithms to calculate all Pareto-optimal bindings (the service skyline in their terminology) in a bottom-up fashion. [7] proposes an approach based on the notion of the skyline to effectively and efficiently select services for composition, reducing the number of candidate services to be considered. The other subcategory neglects the prohibitively computational cost even though ensures the precision. [14] uses a specific GA for multi-objective optimization. [15] uses particle swarm optimization. [9] uses multi-objective bees’ algorithms. Common to all those heuristic approaches is that they run in polynomial time but cannot give the precision guarantee. [16] proposes a top-k composite services selection method based on a preference-aware service dominance relationship. In [2], it presents a fully polynomial time approximation scheme (A-FPTAS) which could balance the sweet spot between the precision and the overhead.
Methodology
This section introduces the models used in our approach, namely, workflow and solution, quality of service, QoS range, Pareto dominance and approximate dominance.

Workflow and Solution
In a workflow, all of its abstract web services are orchestrated by workflow patterns, such as sequence (SEQ), choice (CHC), parallel (PARA) and loop (LOOP). If certain abstract services are connected by CHC (or PARA), CHC (or PARA) could be considered to be connected to another service or workflow pattern with SEQ. If abstract services are connected by LOOP, we model the loop with available types using the peeling technique [11].

In this paper, we transform the workflow to a tree. Each of abstract web services of workflow $W$ is viewed as a leaf node which has no child node in the tree and workflow patterns are treated as inner nodes which have more than one child node. This tree also has a root node which has no parent node root($W$). All of these nodes form the workflow nodes of $W$, denoted by nodes($W$). Though the workflow is transformed to a tree, the original workflow definition in a language such as BPEL, which is used for execution after a solution has been selected, would not be changed.

In runtime, the service-oriented application is instantiated as a workflow instance, in which each abstract service is operated through its corresponding concrete web service instance[1]. Thus, node $W$ has to be bound to one of its candidate web services $candidates(W)$. By $B(W)$, we refer to all possible bindings of $W$. For leaf node $W$, its binding $b$ is a tuple of $W$ and $s \in candidates(W)$, namely, $b = (W,s)$. But for the inner node, one of its bindings consists of one binding of all its child nodes. One solution is, of course, a binding of the root node.

Quality of Service
Assuming that the ordering between any two attributes in the QoS attribute set $A$ is fixed, a $|A|$-dimensional QoS value vector $q$ could differentiate web services. The specific value for attribute $a \in A$ is $q_a$ within $q$.

Function $QoS(W, b)$ is used to estimate the QoS values of node $W$ when $W$ is assigned to binding $b$. The QoS values of a leaf node equal to that of the selected concrete web service. For inner node $W$, its QoS values are the aggregation of that of all its $n$ child nodes through QoS aggregation function $QoSAF(W)$, which assigns $W$ to a vector of operators including minimum (min), maximum (max), sum (+) and product (×), that is, $QoS(W, b) = QoSAF(W)(QoS(W_1, b), \cdots, QoS(W_n, b))$. In addition, the QoS values of $W$ correspond to that of root($W$).

According to the influence of a QoS attribute over its value, QoS attributes could be divided into positive and negative ones. The positive attribute is that the larger value has the better quality, such as reliability and throughput. If the larger value of an attribute corresponds to the lower quality, such as response time and cost, this attribute is negative.

QoS Range
An QoS attribute is called bounded attribute if its value has a priori bounded value domain, for example, reliability and availability are $[0,1]$. But for other attributes, such as response
time, the upper bound of their value domain could be arbitrarily large, we call them unbounded attributes.

The QoS values of any binding of node W cannot exceed its total QoS range defined by two QoS vectors \( TL \) (lower bound) and \( TU \) (upper bound), denoted by \( TQR(W) = (TL, TU) \), that is, \( \forall a \in A, \forall b \in \mathbb{B}(W): TL^a \leq QoS^a(W, b) \leq TU^a \). The total QoS range of a bounded attribute is simplified by its priori domain. For an unbounded attribute, the total QoS range of leaf node \( L \) can be calculated by (1) for the lower bound and (2) for the upper bound; the total QoS range of inner node \( W \) is calculated out of that of its \( n \) child nodes, and its lower bound and upper bound are calculated according to (3) and (4), respectively.

\[
TQR^a_L(L) = \min_{\text{sec candidates}(L)} QoS^a(s)(1)
\]

\[
TQR^a_U(L) = \max_{\text{sec candidates}(L)} QoS^a(s)(2)
\]

\[
TQR^a_L(W) = QoSA^a(W)(TQR^a_L(W_1), \ldots, TQR^a_L(W_n))(3)
\]

\[
TQR^a_U(W) = QoSA^a(W)(TQR^a_U(W_1), \ldots, TQR^a_U(W_n))(4)
\]

In order to yield approximation precision guarantee, the critical QoS range should be calculated through total QoS range [2]. Assume that the critical range of inner node \( W \) is \( CQR(W) \) and the total QoS range of its child node \( W_i \) is \( TQR(W_i) \), TABLE 1 presents how to calculate the lower and the upper bound of the critical range for \( W_i \) (note that empty fields mean that the formula from the row above applies again). The critical range of the root node is equal to its total QoS range.

As shown in TABLE 1, different types of QoS attributes use different formulas to calculate the critical range. TABLE 2 shows the types of popular QoS attributes based on two criteria: the value domain and the set of allowed operators. The value domain is divided into three cases: 1) unitary bounded domain ([0,1]); 2) bounded domain ([0, \( c \]) where \( c \) is a positive constant); 3) unbounded domain ([0, \( \infty \) ]). The set of allowed operators is a subset of the operators \( \min, \max, + \) and \( \times \).

In order to comprehensively evaluate the different QoS attributes, we should scale QoS value vector \( q \) to the range \( R \) using function \( \text{scale}(q, R) \). For the positive attributes, scaling formula is (5), and (6) is used to scale the negative QoS attributes.

\[
\text{scale}^a(q, R) = \begin{cases} 
q^a - R_L^a & \text{if } R_L^a \neq R_U^a \\
1 & \text{if } R_L^a = R_U^a 
\end{cases}
\]

\[
\text{scale}^a(q, R) = \begin{cases} 
R_U^a - q^a & \text{if } R_L^a \neq R_U^a \\
1 & \text{if } R_L^a = R_U^a 
\end{cases}
\]
TABLE 1. Formulas for Calculating Critical Ranges.

<table>
<thead>
<tr>
<th>Type of $a$</th>
<th>$\text{QoS}^a(W)$</th>
<th>$\text{CQR}_W^a(W)$</th>
<th>$\text{TQR}_W^a(W)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2</td>
<td>(All)</td>
<td>TQR$_W^a(W)$</td>
<td>TQR$_W^a(W)$</td>
</tr>
<tr>
<td>3</td>
<td>min</td>
<td>CQR$_W^a(W)$</td>
<td>TQR$_W^a(W)$</td>
</tr>
<tr>
<td>4</td>
<td>+</td>
<td>TQR$_W^a(W)$</td>
<td>TQR$_W^a(W)$</td>
</tr>
</tbody>
</table>

TABLE 2. Classification of QoS Attributes.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value domain</th>
<th>Operators</th>
<th>QoS attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,1]</td>
<td>min, max, +</td>
<td>Reliability, availability, data quality</td>
</tr>
<tr>
<td>2</td>
<td>[0,∞]</td>
<td>min, max, +</td>
<td>Reputation</td>
</tr>
<tr>
<td>3</td>
<td>[0,∞]</td>
<td>min, +</td>
<td>Throughput</td>
</tr>
<tr>
<td>4</td>
<td>[0,∞]</td>
<td>max, +</td>
<td>Time, cost, energy consumption</td>
</tr>
</tbody>
</table>

Pareto Dominance and Approximate Dominance

For node $W$, its binding $b_1$ with QoS value vector $q_1 = \text{QoS}(W, b_1)$ is called Pareto-optimal if no other binding $b_2$ exists in $\mathbb{B}(W)$ whose QoS vector $q_2$ can dominate $q_1$: $b_2 \in \mathbb{B}(W): \text{QoS}(W, b_2) \succeq \text{QoS}(W, b_1)$. It means that $q_1$ has better (meaning a smaller (larger) value for negative (positive) attributes) QoS values than $q_2$ for certain QoS attributes and at least equivalent values for other attributes.

For the scaled QoS vector $q_1$ and $q_2$, $q_2$ is dominated by $q_1$ with approximation error $\varepsilon \in [0,1]$ if the value of any attribute in $q_1$ is better or at least sufficiently close to that of $q_2$: $\forall a \in A: q_2^a \leq q_1^a + \varepsilon$, denoted by $q_2 \preceq_{\varepsilon} q_1$.

A set of bindings $B_\varepsilon(W) \subseteq \mathbb{B}(W)$ approximates the Pareto set of nodes $W$ with approximation error $\varepsilon$ with regards to range $R$. It means that a binding in $\mathbb{B}(W)$ is the approximate Pareto-optimal if it approximately dominates at least one of the Pareto-optimal bindings in $B_\varepsilon(W)$ under $\varepsilon$. More formally, for each binding $b' \in \mathbb{B}(W)$, the binding $b$ is contained in $B_\varepsilon(W)$ that approximately dominates $b': \forall b' \in \mathbb{B}(W), \exists b \in B_\varepsilon(W): \text{scale}(\text{QoS}(W, b'), R) \preceq_{\varepsilon} \text{scale}(\text{QoS}(W, b), R)$. 

910
Our Web Service Selection Approach

This section has two subsections. Subsection 1 describes how to normalize a workflow to be a binary tree before applying our algorithm, and then subsection 2 describes $A^2$-FPTAS.

Workflow Normalization

Workflow normalization means that the workflow is represented as a binary tree that workflow nodes are either leaves or have exactly two child nodes. We have introduced how to transform the workflow to a tree. Here, we normalize the transformed tree to a binary tree.

In the transformed tree, if the inner node has one child node, we could simplify it with its child. For the inner node with two child nodes, we don’t make any change because it satisfies the binary tree’s properties. Nodes with more than two child nodes could be replaced by several nodes, each of which has exactly two child nodes. A sequence (or parallel) inner node of $n(n > 2)$ child nodes can be replaced by a sequence (or parallel) of any one, which is the first one usually, of the child nodes and a sequence (or parallel) inner node with other $n−1$ child nodes. A choice inner node is modelled as choosing the first branch and the other branches, then choosing the second and the remaining branches if the first branch was not chosen, etc.

Formally, the inner node $W$ with one child node $W_1$ can be replaced by a node $W'$ which has the same child nodes and aggregation function as $W_1$, i.e., $\text{childNodes}(W') = \text{childNodes}(W_1)$ and $\text{QoSaf}(W') = \text{QoSaf}(W_1)$. The inner node $W$ with $n(n > 2)$ child nodes $\text{childNodes}(W) = \langle W_1, \cdots, W_n \rangle$ can be replaced by node $W'_1$ with $\text{childNodes}(W'_1) = \langle W_1, W'_2 \rangle$, $\text{childNodes}(W'_2) = \langle W_2, W'_3 \rangle$, etc. This chain of nodes ends with node $W'_{n−1}$ where $\text{childNodes}(W'_{n−1}) = \langle W_{n−1}, W_n \rangle$, so that the new nodes ($W'_1$ to $W'_{n−1}$) are introduced to replace $W$ and have the same QoS aggregation function as $W$.

Description of $A^2$-FPTAS

Listing 1 shows the pseudo-code of function PQDSS, which performs preparatory steps for CanApprox and calls PQDSSrec to approximate the Pareto set. The input to PQDSS consists of the global error bound (GEB) $\varepsilon_g$, which could regulate the precision of the output, and the PQDSS problem $\mathcal{P} = \langle \mathcal{W}, \mathcal{S}, \mathcal{A} \rangle$, where $\mathcal{S}$ is the set of the candidate web services of all abstract web services in workflow $\mathcal{W}$. The output is an approximate Pareto set $B_{\varepsilon_g}(\mathcal{W})$ for $\mathcal{W}$ that has an approximation error at most $\varepsilon_g$.

```
1: function PQDSS(globa1P = \langle \mathcal{W}, \mathcal{S}, \mathcal{A} \rangle, \varepsilon_g)
2: m ← |nodes(\mathcal{W})|
3: globalM ← m(m + 1)/2, index ← 0
4: globalTQR ← CalcTQR(\mathcal{W})
5: globalCQR ← CalcCQR(\mathcal{W})
6: return PQDSSrec(root(\mathcal{W}))
7: end function
Listing 1  Main procedure of $A^2$-FPTAS
```

The global variables $index$ and $M$ in line 3 are used to calculate the unequal local error bound (ULEB) $\varepsilon_{ul}$ in PQDSSrec. $index$ is the visited order of a node and initialized to 0. $M$ is a constant and depends on the number of workflow nodes $m$. In line 4, $TQR$ is
calculated by function CalcTQR through (1), (2), (3) and (4). CQR is calculated by function CalcCQR in line 5, as shown in Listing 5.

1: function PQDSSrec(W)
2: res ← ∅
3: index ← index + 1
4: εul ← index · εg/M
5: if isLeaf(W) then
6: //create a binding for each candidate service
7: for all ∈ candidates(W) do
8: b ← {(W, s)}
9: q ← QoS value vector of s
10: InsertPareto(res, (b, q), εul)
11: end for
12: else
13: (W1, W2) ← childNodes(W)
14: resW1 ← PQDSSrec(W1)
15: resW2 ← PQDSSrec(W2)
16: // Combine bindings of child nodes
17: for all(b1, q1) ∈ resW1 do
18: for all(b2, q2) ∈ resW2 do
19: b ← b1 ∪ b2
20: q ← QoS AF(W)(q1, q2)
21: InsertPareto(res, (b, q), εul)
22: end for
23: end for
24: end if
25: return res
26: end function

Listing 2 Pseudo-code for function PQDSSrec.

Listing 2 shows the pseudo-code for PQDSSrec, which finds an approximate Pareto set for node W using the principle that an approximate Pareto set for an inner node is calculated by combining bindings from the approximate Pareto sets of its child nodes. The output is a set of tuples (b, q), where b denotes a binding for W and q = QoS(W, b) is the associated QoS vector. Lines 3 and 4 shows how to calculate a node’s ULEB εul, which is positive correlation with its visited order in the normalized tree. It means that the later a node is visited, the bigger εul it has, but the sum of the ULEBs of all nodes is at most εg.

If W is a leaf node (line 5), PQDSSrec creates a binding for each candidate service in candidates(W) (lines 7 to 11). Since some of these bindings could not be inserted into the approximate Pareto set res directly, we should filter out the useless bindings. Function InsertPareto could undertake this work introduced in Listing 3.

If W is an inner node, PQDSSrec calculates the approximate Pareto set for each of its two child nodes by recursive calls (lines 13 to 15). Two approximate Pareto sets, resW1 and resW2, store bindings of the first and second child nodes, respectively. Two nested for-loops (line 17 to 23) are used to check all possible bindings of W, one of which is combined by one binding from resW1 and resW2 severally. The QoS values of the new binding are aggregated by QoS values of these two bindings (line 20). We then use InsertPareto to insert this new binding into res.

1: function InsertPareto(B, (bN, qN), W, εul)
2: for all(bD, qD) ∈ Bdo
3: // Old binding can approximate new one
4: if CanApprox(qD, qN, W, εul) then
5: return
6: end if
7: end for
8: // Delete dominated old bindings
9: for all(bD, qD) ∈ Bdo
10: if qD = qD then
11: B ← B \{(bD, qD)}
12: end if
13: end for
14: // Insert new binding
15: B ← B \{(bN, qN)}
16: end procedure

Listing 3  Pseudo-code of Filtered Insert Function.

Listing 3 shows the pseudo-code of InsertPareto. It is used to insert a new binding into an approximate Pareto set and delete useless bindings in this set. Its input is a set of bindings with QoS vectors B, a new binding bN with the QoS vector qN, node W and the
local error bound $\varepsilon_{ul}$ of $W$. The function CanApprox is to examine whether one binding could approximately dominate another binding (line 4).

InsertPareto first checks whether the QoS vector of any old binding $b_o$ can approximate that of the new binding $b_N$. If so, the new binding isn’t inserted into $B$ (lines 2 to 7). If not, all bindings dominated by $b_N$ would be deleted (lines 9 to 13), and $b_N$ is inserted into $B$ in the end.

```
1: function CanApprox(q_o, q_N, W, $\varepsilon_{ul}$)
2:   $\textbf{s}_o$ $\leftarrow$ scale(q_o, $\textbf{CQR}(W)$)
3:   $\textbf{s}_N$ $\leftarrow$ scale(q_N, $\textbf{CQR}(W)$)
4:   // Compare with precision $\varepsilon_{ul}$
5:   return $\textbf{s}_N \leq \varepsilon_{ul} \textbf{s}_o$
6: end function
```

Listing 4 Pseudo-code for function CanApprox.

In Listing 4, CanApprox compares the QoS vectors scaled to the critical ranges. Its input is $\varepsilon_{ul}$, node $W$ and the QoS vectors of an old binding $q_o$ and the new binding $q_N$ of $W$. It will return true if scaled QoS vector $\textbf{s}_o$ can approximate $\textbf{s}_N$. How to calculate the critical ranges of all workflow nodes is introduced in Listing 5.

```
1: function CalcCQR(W)
2:   $\textbf{CQR}(\text{root}(W))$ $\leftarrow$ $\textbf{TQR}($root($W)$)
3:   CalcCQRdown($\text{root}(W)$)
4: end procedure
5: // Calculates $\textbf{CQR}$ for the child nodes of $W$
6: procedure CalcCQRDown(W)
7:   if isInner(W) then
8:     ($W_1, W_2$) $\leftarrow$ childNodes(W)
9:     for all $i \in \{1,2\}$ do
10:        for all $a \in A$ do
11:           $\textbf{CQR}^t(W_i)$ $\leftarrow$ Use TABLE 1
12:        end for
13:     end if
14:     end procedure
```

Listing 5 Pseudo-code for calculating Critical Ranges.

Listing 5 shows the pseudo-code for CalcCQR and CalcCQRDown. Function CalcCQR is to calculate the critical ranges for all workflow nodes, and CalcCQRDown is an auxiliary procedure used by CalcCQR. Global variable $\textbf{CQR}$ is to reserve the critical ranges of all nodes of $W$. The critical range of root$(W)$ equals to its total QoS range (line 2). CalcCQRDown is to calculate the critical ranges of two child nodes $W_1$ and $W_2$ of $W$ and to execute a recursive call to calculate the critical range for the child node $W_i$, $i \in \{1,2\}$. As to how to calculate the critical ranges of the child nodes, it has been shown in section QoS Range.

Experiments and Analysis

Experimental Setup

In this experiment, the workflow consists of $T$ ($T > 1$) abstract web services. Each abstract service is connected to another one by the sequence and could be done by $S$ candidate services. For each candidate service, we discuss its A QoS attributes. This experiment works on the computer Lenovo QiTian M4350-D007 which has 4 GB RAM 3.2GHz and Intel Core(TM) i5-3470 4-core and runs 32-Bit Windows 7. All of the referred code is implemented in pure Java.

We adopt the QWS dataset which is the QoS values of real world web services presented in [17]. QWS dataset saves the QoS values of nine QoS attributes (Availability, Reliability,
Response Time, Throughput, Success ability, Compliance, Best Practices, Latency and Document). However, we just research a part of these attributes. The QoS values of each candidate web service are assigned to one of the total 2507 QoS values in the QWS dataset randomly. If the sum of all candidate web services exceeds about 2500, it is necessary to randomly generate the vacant QoS values based on the data characteristics of QWS.

**Comparison between A-FPTAS and A²-FPTAS**

In order to adequately evaluate the performance of A-FPTAS and A²-FPTAS, we compare their running time and the number of solutions in the Pareto set in three cases.

The first case is about the varying number of abstract services. SandA are set to 50 and 4 severally. Tvaries from 10 to 80 at the interval of 10. The result is shown in TABLE 3.

The second case is about the varying number of candidate services. Tis set to 10, and A is set to 4. S increases from 200 to 2000 at the interval of 200. The result is presented in TABLE 4.

The third case is about the varying number of QoS attributes. T and S are set to 10 and 50, respectively. A is from 1 to 7 at the internal of 1. TABLE 5 shows the result of this case.

<table>
<thead>
<tr>
<th>Nr. Abstract Services</th>
<th>Time (ms)</th>
<th>Nr. Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A(0.1)</td>
<td>A²(0.1)</td>
</tr>
<tr>
<td>10</td>
<td>1248</td>
<td>359</td>
</tr>
<tr>
<td>20</td>
<td>23993</td>
<td>4461</td>
</tr>
<tr>
<td>30</td>
<td>93694</td>
<td>34445</td>
</tr>
<tr>
<td>40</td>
<td>366507</td>
<td>117469</td>
</tr>
<tr>
<td>50</td>
<td>1995478</td>
<td>410968</td>
</tr>
<tr>
<td>60</td>
<td>2912103</td>
<td>932850</td>
</tr>
<tr>
<td>70</td>
<td>3623278</td>
<td>1393800</td>
</tr>
<tr>
<td>80</td>
<td>8277234</td>
<td>1394377</td>
</tr>
</tbody>
</table>

Note: 1) A means A-FPTAS; 2) A² is A²-FPTAS; 3) A(0.1) means A-FPTAS runs with GEB 0.1.

<table>
<thead>
<tr>
<th>Nr. Services</th>
<th>Time (ms)</th>
<th>Nr. Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A(0.1)</td>
<td>A²(0.1)</td>
</tr>
<tr>
<td>200</td>
<td>2948</td>
<td>499</td>
</tr>
<tr>
<td>400</td>
<td>2823</td>
<td>640</td>
</tr>
<tr>
<td>600</td>
<td>2964</td>
<td>593</td>
</tr>
<tr>
<td>800</td>
<td>3245</td>
<td>608</td>
</tr>
<tr>
<td>1000</td>
<td>2449</td>
<td>593</td>
</tr>
<tr>
<td>1200</td>
<td>2559</td>
<td>577</td>
</tr>
<tr>
<td>1400</td>
<td>2246</td>
<td>514</td>
</tr>
<tr>
<td>1600</td>
<td>1950</td>
<td>483</td>
</tr>
<tr>
<td>1800</td>
<td>1934</td>
<td>453</td>
</tr>
<tr>
<td>2000</td>
<td>1857</td>
<td>452</td>
</tr>
</tbody>
</table>

By comparing the columns 3 and 5, columns 7 and 9 in TABLE 3, TABLE 4 and TABLE 5, we could also find the same conclusions in [2], i.e., the bigger GEB is, the
shorter running time is, and the smaller number of solutions is. TABLE 4 also shows that
the number of candidate services has almost no influence on the running time and size of
the approximate Pareto set. TABLE 5 displays that the number of attributes has a
significant effect on the running time and size of the approximate Pareto set.

<table>
<thead>
<tr>
<th>Nr. Attrs</th>
<th>Time (ms)</th>
<th>Nr. Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A(0.1)</td>
<td>A(0.25)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>22</td>
</tr>
<tr>
<td>4</td>
<td>500</td>
<td>205</td>
</tr>
<tr>
<td>5</td>
<td>78533</td>
<td>11393</td>
</tr>
<tr>
<td>6</td>
<td>1362488</td>
<td>95909</td>
</tr>
<tr>
<td>7</td>
<td>51884947</td>
<td>3084656</td>
</tr>
</tbody>
</table>

Note: Attrs is short for QoS attributes.

Moreover, for TABLE 3, we find that the running time of A-FPTAS is reduced by at
least 65% than A²-FPTAS by comparing columns 2(A(0.1)) and 3(A²(0.1)). The number of
solutions of A-FPTAS is reduced by at least 50% than A²-FPTAS by comparing columns 6
(A(0.1)) and 7(A²(0.1)). We can also draw the same conclusion according to the
comparison between columns 4 and 5 and columns 8 and 9 in TABLE 3. For TABLE 4 and
TABLE 5, we also find that both the running time and the number of solutions of
A²-AFPTAS are smaller than A-FPTAS at the same GEB.

Conversely, if the approximate Pareto sets of both A²-FPTAS and A-FPTAS have
the same number of solutions, the GEB of A²-FPTAS can be smaller, so that A²-FPTAS has a
higher precision than A-FPTAS.

In conclusion, the performance of A²-FPTAS is better than A-FPTAS.

Conclusion and Future Work

In this paper, we present the A²-FPTAS algorithm to address the QoS-based web service
selection. It adopts the strategy of using the unequal local error bound to regulate the
precision of solutions in the Pareto set.

Through the comparisons in the aspects of QoS attribute, number of abstract web
services and concrete web services, the experiments demonstrate that the unequal local
error bound is efficient to improve the performance of A²-FPTAS comparing with
A-FPTAS.

The approach proposed in this paper just calculates the Pareto set, and then it cannot
select a suitable solution for the user according to his requirements. When multiple users
request the same workflow with different requirements, we should select appropriate
solution for each user based on their own requirements in the future.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (Grant
Number 61402027).
References


