Evaluation of QoS-Based Web Service Matchmaking Algorithms

Kyriakos Kritikos and Dimitris Plexousakis
Foundation for Research and Technology Hellas, Heraklion, Greece
kritikos@ics.forth.gr and dp@ics.forth.gr

Abstract

Web Service (WS) discovery is a prerequisite for achieving WS composition and orchestration. Although a lot of research has been conducted on the functional discovery of WSs, the proposed techniques fall short when faced with the foreseen increase in the number of (potentially functionally-equivalent) WSs. The above situation can be resolved with the addition of non-functional (Quality of Service (QoS)) discovery mechanisms to WS discovery engines. QoS-based WS matchmaking algorithms have been devised for this reason. However, they are either slow - as they are based on ontology reasoners - or produce inaccurate results. Inaccuracy is caused both by the syntactic matching of QoS concepts and by wrong matchmaking metrics. In this paper, we present two Constraint Programming (CP) QoS-based WS discovery algorithms for unary constrained WS specifications that produce accurate results with good performance. We also evaluate these algorithms on matchmaking time, precision and recall in different settings in order to demonstrate their efficiency and accuracy.

1 Introduction

The success of the WS paradigm has led to a proliferation of available WSs, which are advertised in intra- or inter-organizational registries. To enable their usage and integration, these WSs have to be discovered. So far, a lot of research has been conducted on the functional discovery of WSs. Researchers finally agree in using semantics for WS description and reasoning mechanisms for WS discovery. Prototypes have already been implemented [7] with quite promising results. Unfortunately, functional discovery of WSs is not sufficient, as there may be hundreds of functionally-equivalent WSs. Users must be further assisted in selecting the appropriate WS for their needs.

The solution to the last problem is the use of QoS in WS description and discovery. QoS is a set of non-functional properties that encompass performance characteristics. As users are very concerned about the performance of WSs they use, QoS can be used for discriminating between functionally-equivalent WSs. Thus, the proposed solution comprises: a) description of the QoS aspect of WSs; b) filtering of WS functional results based on user constraints on their QoS descriptions; c) sorting the results based on user-provided weights on QoS attributes/metrics.

We have conducted a detailed study of QoS-based WS description and discovery and have come up with a set of requirements [9] for these two processes. For QoS-based WS description, we have proposed a semantically rich, extensible QoS model as one of the requirements. This requirement seems to be quite strong as none of the existing research approaches [11, 6, 14, 13, 15, 3, 2, 12, 5] satisfies it. For this reason, we have developed an extensible ontology language for QoS-based WS description called OWL-Q [8]. OWL-Q satisfies the above requirements set and complements the WS functional description language OWL-S.

There are two main approaches for QoS-based WS matchmaking: a) ontology-based [15, 12, 5]; b) CP [2] based [3, 2]. The first approach suffers from performance problems due to the use of ontology reasoners. The second approach is fast but the two research works adopting it return inaccurate results and do not provide advanced categorization of results. In addition, both approaches do not provide useful matchmaking results for over-constrained QoS demands. For these reasons, we have devised a semantic QoS metric matching algorithm [8] and two complementary CP-based and QoS-based WS discovery approaches [10] exploiting it: one with unary and one with n-ary constraints. These two approaches are accurate, provide advanced categorization of results and produce results even for over-constrained demands but they have not been evaluated yet.

The purpose of this paper is to theoretically and empirically analyze and compare the most prominent CP-based and QoS-based WS matchmaking approaches: the matchmaking algorithm of Cortés et. al. [2] and two alternative versions of our unary-constraints-based matchmaking algorithm. The fields of comparison are: time complexity, accuracy (precision and recall [4]), categorization and size of useful results. The theoretical outcome of this comparison is verified by an empirical evaluation of these three algorithms. Careful tuning of experiment parameters gives rise to different settings under which these algorithms are evaluated, resulting in many useful conclusions that validate our theoretical analysis.

The remainder of this paper is organized as follows: Section 2 analyzes the main approaches in QoS-based WS matchmaking. Section 3 analyzes and compares the three matchmaking algorithms under consideration. Section 4 describes and analyzes all details of the empirical evaluation conducted on these algorithms and discusses its main outcomes. Finally, section 5 summarizes the paper and draws directions for further research.

2 Related Work

QoS of a WS is a set of non-functional attributes that are measured by one or more QoS metrics. These metrics specify the measurement method, schedule, unit, value range and other details. A QoS specification of a WS is materialized as a set of constraints on a certain set of QoS metrics. These constraints restrict the domain of values of these metrics. A QoS offer matches a QoS demand if and only if its solution space contains combinations of metrics assignments that conform with the demand.

There are two main categories of approaches in QoS-based WS matchmaking that differ in the way they describe QoS-based WS specifications and the way they perform the matchmaking. Unfortunately, most of these approaches provide only two categories of results: *matched* and *failed* QoS offers. However, WS requesters would like to find out more interesting features of the results. For example, they would like to know which matched QoS offers are more constrained than the QoS demand. In addition, these approaches do not offer useful results for over-constrained QoS demands. The last drawback is actually a consequence of the previous: when an over-constrained QoS demand is issued, only failed matches are returned. This is not quite useful as the WS requester does not know which QoS offers violate the constraints of the QoS demand and which not. It would be quite useful if failed matches were further categorized into two subcategories indicating the amount of failure (complete or not). Furthermore, it would be quite useful if users were informed that they could obtain matched QoS offers by relaxing a minimal number of their constraints.

The first category of approaches [15, 12, 5] uses ontologies to describe QoS-based WS specifications and reasoners to perform the matchmaking. These approaches can provide advanced and accurate categorization of results but they exhibit performance issues due to the technology they use. Moreover, they don't offer useful results for over-constrained demands. In addition, although they are capable of, they do not perform semantic QoS metric matching.

The second category of approaches uses syntactic languages to describe QoS-based WS specifications as a set of constraints and a constraint solving engine for matchmaking. Approaches of this category perform better but they exhibit low accuracy. In addition, they don't provide advanced categorization of results and no useful results at all for over-constrained demands. Moreover, as they use syntactic languages, they are not capable of performing metric matching in comparison with the other category of approaches. Last but not least, they use faulty matchmaking metrics which further reduce accuracy. Degwekar, Su and Lam [3] use a faulty matchmaking metric: a QoS offer matches a QoS demand when they have common solutions. In this way, QoS offers that may contain worse solutions than those of the demand's are returned as matching results. Cortés et. al. [2] use a stronger but also faulty matchmaking metric: a QoS offer matches a QoS demand if and only if it contains solutions that are contained in the solution space of the demand. In this way, QoS offers that perform better than what the requester expects are removed from the result set.

3 Matchmaking Algorithms

In our previous work [10], we have identified that a QoS-based WS discovery process consists of four sequential tasks: *alignment*, *matchmaking*, *optimization*, and *selection*. The alignment task aligns QoS metrics of QoS offers and demands during WS advertisement and inquiry respectively by exploiting the semantic QoS metric matching algorithm [8]. The matchmaking task matches available QoS offers with the requested QoS demand and produces categories of results. The optimization task is executed when the matchmaking task does not produce matching results i.e. the QoS demand is over-constrained. In this case, one or more of the constraints of the demand are relaxed so as to produce the required results. The selection task orders the matchmaking results by solving constraint optimization problems based on weights given by the requester.

The primary focus of this paper is on the matchmaking step. Based on the analysis of the previous section, we believe that the CP-based approach is more appropriate as it performs better and can be further improved by exploiting the QoS metric matching algorithm and by choosing a better matchmaking metric. Based on these factors, we have developed two different QoS-based WS matchmaking algorithms [10] for unary and n-ary constraints respectively. We analyze and compare the best CP-based approach of the "Related Work" section with two different versions of our unary-constraints algorithm. In this section, these three algorithms will be theoretically compared based on the following factors: execution time, accuracy, advanced categorization of results, and size of matchmaking results. The empirical evaluation of these algorithms will be analyzed in the next section.

Assumptions and notation: Our basic assumption is that the matchmaking scenario contains N QoS offers and one demand that involve a conjunctive set of unary constraints. We also assume that these QoS specifications have already been aligned with the QoS metric matching algorithm and that they eventually involve the same number of QoS metrics M. In addition, we assume that there are only numeric (real or integer) QoS metrics. So in the worst case, there will be $2 \cdot M$ constraints in each QoS specification as each QoS metric will have at most two constraints (one restricting its upper limit and another one restricting its lower limit). Finally, we assume that the average execution time of the constraint solving engine (CSE) for solving $2 \cdot M + 1$ unary constraints involving M metrics is T_M^{2M+1}.

3.1 The Algorithm of Cortés et. al.

Cortés et. al. [2] propose the concept of *conformance* for matchmaking, which is mathematically expressed by the following equivalence:

$$conformance\left(O_i, D\right) \Leftrightarrow s\left(P_i \wedge \neg P^D\right) = \mathrm{f} \qquad (1)$$

where s is a procedure returning true if the input Constraint Satisfaction Problem (CSP) [2] is satisfiable or false otherwise, and f=false. To explain, an offer O_i matches a demand D when there is no solution to the offer's CSP P_i that is not part of the solution set of the demand's CSP P^D. In the implementation of this approach, a CSE engine was used that allowed the "NOT" operator. So conformance of one QoS offer to one QoS demand was translated to satisfiability of one CSP problem containing all the constraints of the offer and the negation of all the constraints of the demand. Let us try to analyze equivalence (1):

$$\begin{aligned}
&conformance\left(O_i, D\right) \Leftrightarrow s\left(P_i^O \wedge \neg P^D\right) = \mathrm{f} \\
&\Leftrightarrow s\left(P_i^O \wedge \neg\left(c_1^D \wedge c_2^D \wedge \cdots \wedge c_{2M}^D\right)\right) = \mathrm{f} \\
&\Leftrightarrow s\left(P_i^O \wedge \left(\neg c_1^D \vee \neg c_2^D \vee \cdots \vee \neg c_{2M}^D\right)\right) = \mathrm{f} \\
&\Leftrightarrow s\left(\left(P_i^O \wedge \neg c_1^D\right) \vee \left(P_i^O \wedge \neg c_2^D\right) \vee \cdots \vee \left(P_i^O \wedge \neg c_{2M}^D\right)\right) = \mathrm{f} \\
&\Leftrightarrow \left(s\left(P_i^O \wedge \neg c_1^D\right) = \mathrm{f}\right) \wedge \left(s\left(P_i^O \wedge \neg c_2^D\right) = \mathrm{f}\right) \wedge \ldots \\
&\wedge \left(s\left(P_i^O \wedge \neg c_{2M}^D\right) = \mathrm{f}\right)
\end{aligned}$$

So, the underlying CSE translates the conformance of offer O_i to demand D to checking if all of the CSPs– constructed by offer's CSP P_i^O and the negation $\neg c_j^D$ of a demand's constraint c_j^D – are unsatisfiable. If any of the CSPs is satisfiable then offer O_i does not match with the demand D.

In the worst case, each QoS specification will contain $2 \cdot M$ unary constraints and conformance checking will be transformed to solving at most $2 \cdot M$ CSPs, each containing $2 \cdot M + 1$ constraints. In addition, this conformance checking will be performed N times, one time for each of the N QoS offers. For brevity, let us call this matchmaking algorithm '**Algorithm 1**'. Hence, Algorithm 1 will take $O\left(2 \cdot N \cdot M \cdot T_M^{2M+1}\right)$ time. This algorithm produces two types of results: **a)** *exact* results containing QoS offers conforming to the QoS demand; **b)** *fail* results containing non-conforming QoS offers. So Algorithm 1 does not provide advanced categorization of matchmaking results. In addition, this algorithm will produce only *fail* results in case of over-constrained QoS demands.

The big disadvantage of Algorithm 1 is that the *conformance* metric is faulty leading to low accuracy. The reason is that this metric excludes from the *exact* matches result set QoS offers that provide better or equal solutions with respect to the solutions of the demand. For example, suppose that provider and requester use the same metric X, measuring the QoS Property of Availability, that has as value type the set $(0.0, 1.0)$. Further assume that the WS provider's CSP has the constraint: $X \geq 0.96$ while the WS requester's CSP has the constraint: $0.95 \leq X \leq 0.999$. Based on the metric of *conformance*, the provider's offer does not match the request as it contains solutions greater than that of the request's, although these solutions are better. As accuracy is measured based on the metrics of *precision* and *recall* [4], Algorithm 1 has perfect precision (equal to 1.0) but recall less than 1.0 as it suffers from the *false negatives* effect.

3.2 Modifications and Improvements

Motivated by the previous observation, we improve the matchmaking metric of conformance as follows: an offer O_i matches a demand D when its CSP P_i has solutions that are either contained in the solution set of the demand's CSP P^D or are 'better' than the demand's solutions. A 'better solution' contains a value for a metric that is 'better' than all the values of this metric in the demand's solutions. For unary constraints of arithmetic metrics, 'better' is translated to 'greater than' for positively monotonic metrics (e.g. Availability) or to 'less than' for negatively monotonic metrics (e.g. Response Time) while 'worse' is translated to 'less than' or 'greater than' respectively.

Continuing the example of the previous paragraph, we search for those constraints of the QoS demand whose negation 'scans' the worse solutions of the QoS offer. The object of research is the constraint $0.95 \leq X$ while its negation is the constraint $0.95 > X$. Observe that if we try to solve the CSP containing the constraint $0.95 > X$ and the constraint of the QoS offer $X \geq 0.96$, we get no solution. So 'worse' solutions are not found. Conversely, the negation of the constraint $X \leq 0.999$ searches for better solutions than that of the demand's. Thus, for positively monotonic metrics X like the one of the example, we are interested in QoS demand's unary constraints of the form: $a \leq X$, as the negation of these constraints scans for worse solutions of the QoS offer under consideration. Symmetrically, for negatively monotonic metrics X, the demand's 'interesting' constraints have the following form: $X \leq b$.

Based on the analysis above, we change the metric of conformance as follows: Conformance of a QoS offer O_i and a QoS demand D is true if and only if all CSP problems P_i^O – constructed from all the constraints of the offer and the negation $\neg c_j^D$ of an 'interesting' demand's constraint c_j^D, where 'interesting' means the $a \leq X$ demand constraints for positively monotonic metrics and the $X \leq b$ demand constraints for negatively monotonic metrics– are unsatisfiable. In this way, we create a new matchmaking algorithm called '**Algorithm 2**' by altering Algorithm 1.

In the worst case, each QoS specification will contain $2 \cdot M$ unary constraints and conformance checking will be transformed to solving at most M CSPs that contain $2 \cdot M + 1$ constraints. So the time complexity of the matchmaking Algorithm 2 will be: $O\left(N \cdot M \cdot T_M^{2M+1}\right)$. This new matchmaking algorithm is faster than Algorithm 1. In addition, Algorithm 2 is accurate as it corrects the 'false negative' problem of Algorithm 1. However, this algorithm still does not offer advanced categorization and does not produce results for over-constrained QoS demands. This is the reason we modify Algorithm 2 so as to enable these features.

To enable advanced categorization of results, we have to find offers with better solutions than those of the demand. So we have to solve all possible CSPs ($2 \cdot M$) instead of M so as to produce the following categories of conforming offers: **a)** *super* offers promise at least one better solution than the demand's; **b)** *exact* offers contain a solution set that is a subset of the demand's solution set. For producing **exact** matching results for over-constrained demands, we have to first split non-conforming results into two categories: **a)** *partial* results that do not violate all 'interesting' QoS demand's constraints; **b)** *fail* results that violate all M 'interesting' constraints of the demand. We also have to introduce weights to the constraints. Then, if an over-constrained demand is issued, we execute a constraint relaxation task [10] that promotes some *partial* results as *exact*.

This new algorithm, called '**Algorithm 3**', is actually the algorithm proposed in [10]. In the best case, Algorithm 3 will solve $M + 1$ CSPs of M metrics and $2 \cdot M + 1$ constraints for each QoS offer. In the worst case, it will solve $2 \cdot M$ CSP problems of the same form. So its complexity is $\Omega\left(N \cdot (M+1) \cdot T_M^{2M+1}\right)$ in the best case and $O\left(2 \cdot N \cdot M \cdot T_M^{2M+1}\right)$ in the worst. As it can be seen, Algorithm 3 is the slowest of the three algorithms.

4 Evaluation

In this section, we present and analyze the results of a set of experiments carried out for testing the performance and accuracy of the QoS-based WS matchmaking algorithms.

4.1 Experimental Setup

The algorithms and their experiments were implemented in Matlab. Matlab was selected as it is faster than Java or C and offers a constraint optimization package. However, this package could not solve mixed-integer programming problems so we had to add the free MOSEK optimization toolbox for Matlab. As a result, there was a restriction on the number of CSP variables (at most 100). The tests were performed on a computer with Microsoft Windows XP Home as the operating system, a 1.8Ghz AMD Athlon microprocessor and 512 megabytes of RAM.

In each experiment a series of tests were executed. The number of these tests depended on the step of increase on one tuning parameter (e.g. a value from 10 to 50 with step 10 translates to five tests) while the other tuning parameters had specific values. This type of parameters was used for constructing a specific random input to the matchmaking algorithms. Each test was executed 15 times producing 15 values for each measured metric for every algorithm. From these 15 values, only the average or minimum was registered depending on the nature of the measured metric. So in the end of each test, every algorithm had exactly one value registered for each metric.

4.1.1 Tuning parameters

Tuning parameters were used for constructing a specific random input, common to every matchmaking algorithm at each run of a test. The tuning parameters taken into account were the following:

adscn: this parameter indicates the number of QoS offers that have to be constructed from a specific random QoS demand.

matchper: this parameter indicates the percentage of QoS offers that are conforming to the random QoS demand. In addition, it indirectly indicates the percentage of non-conforming QoS offers. There will be a total of $m = \text{round}\left(adscn \cdot matchper\right)$ matching QoS offers and a total of $nm = adscn - m$ non-matching QoS offers in the randomly constructed input.

bmper: this parameter indicates the percentage of *super* offers among all matching offers. In addition, it indirectly indicates the percentage of *exact* offers among all matching offers. There will be a total of: $b = \text{round}\left(bmper \cdot m\right)$ *super* offers and a total of $e = m - b$ *exact* offers in the randomly constructed input. This parameter also controls the *recall* accuracy metric of Algorithm 1 as it expresses the percentage of 'false negatives' that this algorithm produces.

partper: this parameter indicates the percentage of *partial* offers among all non-conforming QoS offers. It also indirectly indicates the percentage of *fail* offers among all non-conforming QoS offers. There will be a total of $p = \text{round}\,(partper \cdot nm)$ *partial* offers and a total of $f = nm - p$ *fail* offers in the randomly constructed input.

arity: this parameter indicates the arity of WS specifications' constraints. For all conducted experiments, this parameter was set to 1 so as to produce only unary constraints.

metrcn: each randomly produced QoS-based WS specification contains $metrcn$ metrics and $4 \cdot metrcn$ constraints. For every QoS specification, each metric has 2 constraints indicating the limits of its type (e.g. for an real valued metric in [1,4] we have the constraints: $x \geq 1$ and $x \leq 4$) and 2 constraints indicating the limits of the specification (e.g. for the same metric we can have $x \leq 3$ and $x \geq 2$).

realper: this parameter indicates the percentage of real valued metrics among all metrics. In addition, it indirectly indicates the percentage of integer valued metrics among all metrics. There will be $r = \text{round}\,(realper \cdot metrcn)$ real valued metrics and $i = metrcn - r$ integer valued metrics in each randomly constructed QoS-based WS specification. Each real valued metric takes just one of the three available real types randomly. These types are: $[0.0, 1.0]$, $[0.0, 100.0]$ and $[0.0, 1000.0]$.

sintper: this parameter indicates the percentage of short int valued metrics (values in [0,255]) among all metrics. In addition, it indirectly indicates the percentage of long int valued metrics (values in [0,65535]) among all metrics. There will be $si = \text{round}\,(sintper \cdot i)$ short int valued metrics and $li = i - si$ long int valued metrics in each randomly constructed QoS-based WS specification. According to Cortés et. al. [2], increasing the percentage of long integers causes matchmaking execution time to increase.

hardstat: this parameter indicates the nature of the QoS demand's constraints. If it is set to 1, then all constraints are hard (weight=2.0). If not, then some constraints are randomly selected to be hard and some to be soft (weight in $(0.0, 1.0)$). In all conducted experiments, it was set to 1.

4.1.2 Comparison metrics

Comparison metrics are actually parameters used to compare the performance of the algorithms under consideration. For our experiments, we used the following:

Matchmaking time: this parameter indicates the average execution time of a matchmaking algorithm for the 15 runs of each test.

Precision: QoS-based WS matchmaking is an instance of an information retrieval problem. There are two popular metrics for measuring the accuracy of an information system: *precision* and *recall*. If $correct$ is the set of correct results and ret is the set of returned results, then precision is expressed as: $\frac{|correct \cap ret|}{|ret|}$. Each run not only randomly creates a set of QoS offers and one demand but also creates the accurate categories of results for the purpose of our evaluation. So it creates the following sets: $super$, $exact$, $partial$ and $fail$. In each run, algorithms 1 and 2 produce the following results: $exact_i$ and $fail_i$, where i is one of 1 and 2. Thus, their precision is: $prec_i = \frac{|(super \cup exact) \cap exact_i|}{|exact_i|}$. In each run, algorithm 3 produces the following results: $super_i$, $exact_i$, $partial_i$ and $fail_i$, where i is 3. Its precision is: $prec_i = \frac{|(super \cup exact) \cap (super_i \cup exact_i)|}{|super_i \cup exact_i|}$. After the execution of 15 runs, each algorithm will have as its precision the minimum of the precision over all runs. In other words, $precision_i = \min\,(prec_i)$.

Recall: Based on the previous analysis, recall is expressed as: $\frac{|correct \cap ret|}{|correct|}$. In each run, the recall of algorithms 1 and 2 is: $rec_i = \frac{|(super \cup exact) \cap exact_i|}{|super \cup exact|}$ and the recall of algorithm 3 is: $rec_i = \frac{|(super \cup exact) \cap (super_i \cup exact_i)|}{|super \cup exact|}$. In the same manner, after 15 runs, all algorithms will have as their recall: $recall_i = \min\,(rec_i)$.

Len: this parameter indicates the average size of the matching results for the 15 runs of each test. For algorithms 1 and 2, each run has: $Len = |match_i|$. For algorithm 3 each run has: $Len = |super_i \cup match_i|$.

4.1.3 Input creation

As explained previously, each test of an experiment created a series of 15 runs. At each execution of a run, a new random input was created based on the values of the tuning parameters. Then this input was fed to the matchmaking algorithms. After the execution of these algorithms, the values of the comparison metrics for every algorithm were derived. A new random input was created for each run in order to test the matchmaking algorithms with different input elements each time. However, it must be indicated that the number of input elements remained constant at each test.

Each random input construction created one QoS demand and $adscn$ QoS offers. The QoS demand was created first having $metrcn$ QoS metrics and $4 \cdot metrcn$ unary constraints, as was explained earlier. Each metric was granted with a random value type (real, short or long int) according to the $realper$ and $sintper$ tuning parameters. In addition, the monotonicity of each metric was assigned randomly. Suppose x is the $1 \times metrcn$ array of metrics, then the QoS demand could be expressed as: $[blc_D \leq x \leq buc_D, blx \leq x \leq bux]$, where blx, bux are $1 \times metrcn$ arrays indicating the low and upper limit respectively of the value types of each metric and blc_D, buc_D are $1 \times metrcn$ arrays indicating the low and upper limit respectively of each metric.

After demand creation, $adscn$ QoS offers were created. Initially, each QoS offer was just a copy of the QoS demand. Then each QoS offer was randomly classified as *super*, *exact*, *partial* or *fail* according to the $matchper$, $bmper$ and

$partper$ tuning parameters based on the following cases:

Case 1: If offer j was classified as *super*, then a random number of metrics ($< metrcn$) was selected. If selected metric i was positively monotonic, then its limits were changed to: $blc_D(i) < blc_j(i) < buc_D(i)$ and $buc_D(i) < buc_j(i) \leq bux(i)$. Otherwise, its limits were changed to: $blx(i) \leq blc_j(i) < blc_D(i)$ and $blc_D(i) < buc_j(i) \leq buc_D(i)$. The other not selected metrics had their limits changed to: $blc_D(i) < blc_j(i) < buc_D(i)$ and $blc_j(i) \leq buc_j(i) < buc_D(i)$.

Case 2: If offer j was classified as *exact*, then all metrics had their limits changed to: $blc_D(i) < blc_j(i) < buc_D(i)$ and $blc_j(i) \leq buc_j(i) < buc_D(i)$.

Case 3: If offer j was classified as *partial*, then a random number of metrics ($< metrcn$) was selected. If selected metric i was negatively monotonic, then its limits were changed to: $blc_D(i) < blc_j(i) < buc_D(i)$ and $buc_D(i) < buc_j(i) \leq bux(i)$. Otherwise, its limits were changed to: $blx(i) \leq blc_j(i) < blc_D(i)$ and $blc_D(i) < buc_j(i) \leq buc_D(i)$. The other not selected metrics had their limits changed to: $blc_D(i) < blc_j(i) < buc_D(i)$ and $blc_j(i) \leq buc_j(i) < buc_D(i)$.

Case 4: If offer j was classified as *fail*, then for each metric i we had the following cases: If it was negatively monotonic, then its limits were changed to: $blc_D(i) < blc_j(i) < buc_D(i)$ and $buc_D(i) < buc_j(i) \leq bux(i)$. Otherwise, its limits were changed to: $blx(i) \leq blc_j(i) < blc_D(i)$ and $blc_D(i) < buc_j(i) \leq buc_D(i)$.

Thus, in the end, each offer j could be expressed as: $[blc_j \leq x \leq buc_j, blx \leq x \leq bux]$.

4.2 Results

In order to empirically evaluate the performance and accuracy of the three QoS-based WS matchmaking algorithms in different settings, a series of eight (8) experiments was conducted. Each experiment had the following values for the tuning parameters: $adscn = 10$, $matchper = 0.5$, $bmper = 0.4$, $partper = 0.5$, $arity = 1$, $metrcn = 10$, $realper = 0.5$, $sintper = 0.5$, $hardstat = 1$. Note that at each experiment a tuning parameter was increasing its value according to a specific step until a specific upper limit.

In the first conducted experiment, we increased the number of QoS offers in order to observe the performance of the matchmaking algorithms. We expected a linear increasing performance behavior for all algorithms according to the time complexity analysis of the previous section. Indeed, as can be seen in Figure 1, this is the case. In addition, Algorithm 2 was the fastest followed by Algorithms 1 and 3 in decreasing order of performance. This was also indicated by the time complexity analysis of the previous section. As far as accuracy is concerned, all algorithms had $precision = 1.0$. Recall was also 1.0 for all algorithms except Algorithm 1 that had recall equal to 0.6. This was also predicted as the percentage of *super* matches among all matches is the same as the percentage of 'false negatives' produced by Algorithm 1. For the majority of the experiments, we had $bmper = 0.4$ and $matchper! = 0.0$, so the accuracy of the algorithms does not change.

In the second experiment, we increased the number of QoS metrics in order to observe the performance of the matchmaking algorithms. We expected a linear increasing performance behavior for all algorithms according to the time complexity analysis of the previous section. Indeed, as can be seen in Figure 2, this is the case. In addition, the performance order between the algorithms did not change.

In the third experiment, we increased the percentage of matched QoS offers among the fixed-sized population of QoS offers ($adscn = 20$). As can be seen in Figure 3, Algorithm 3 exhibited a stable behavior while the rest of the algorithms exhibited a linear increasing performance behavior. There is an explanation for this kind of behavior: Algorithm 3 spends a constant amount of time in order to figure out if one QoS offer is conforming or not to the QoS demand and as long as the percentage of *super* and *partial* offers remains constant so does its whole matchmaking time. On the other hand, for each QoS offer the other algorithms spend time depending on the position of the first violating constraint in the order of the constraints. When the percentage of matching QoS offers increases, the total percentage of violating constraints decreases so the other algorithms will have to do more and more work.

In the fourth experiment, we increased the percentage of *super* matches among the fixed-sized population of QoS offers ($adscn = 20$). Concerning matchmaking time, every algorithm exhibited an almost stable behavior except from algorithm 1 that exhibited a decreasing linear behavior. The behavior of Algorithm 1 was expected. Better matching QoS offers are 'false negatives' for this algorithm so their increase causes a decrease in the number of matched QoS offers. This behavior is indicated in Figure 4. Concerning accuracy, all algorithms had precision equal to 1.0. Recall was also again 1.0 for all algorithms except Algorithm 1. As can be seen from Figure 5, recall of Algorithm 1 decreases in the same amount as $bmper$ increases. So, indeed, *better* match offers represent 'false negatives' for Algorithm 1. Thus, we can definitely express that: $rec_1 = 1 - bmper$.

In the fifth experiment, we increased the percentage of *partial* matches among the fixed-sized population of QoS offers ($adscn = 20$). As can be seen in Figure 6, Algorithm 3 had a stable behavior while the rest of the algorithms exhibited a small increasing linear behavior. This kind of behavior can be explained. As long as $matchper! = 0$, Algorithm 3 performs the same amount of work for every nonconforming QoS offer. On the other hand, the rest of the algorithms do considerably more work for *partial* results

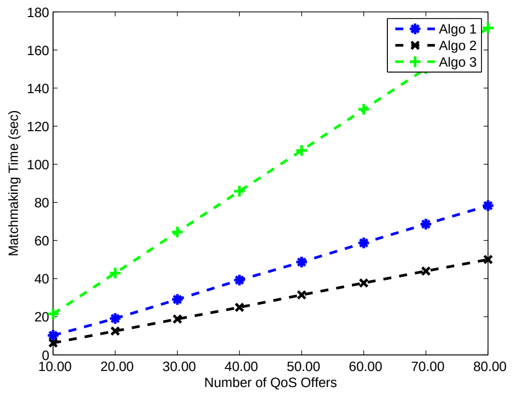

Figure 1. Plot of Matchmaking Time w.r.t the Number of QoS Offers.

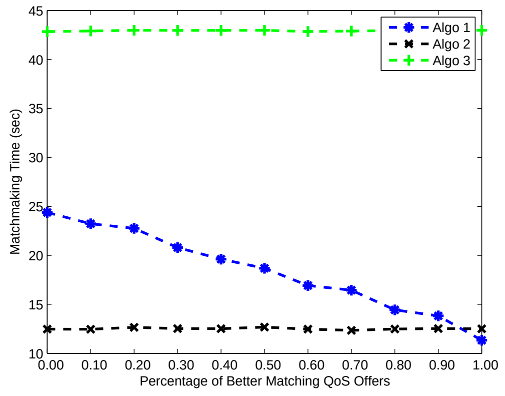

Figure 4. Plot of Matchmaking Time w.r.t the Percentage of Better Matching QoS Offers.

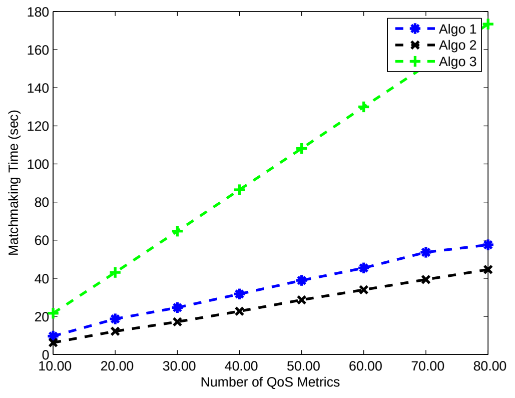

Figure 2. Plot of Matchmaking Time w.r.t the Number of QoS Metrics.

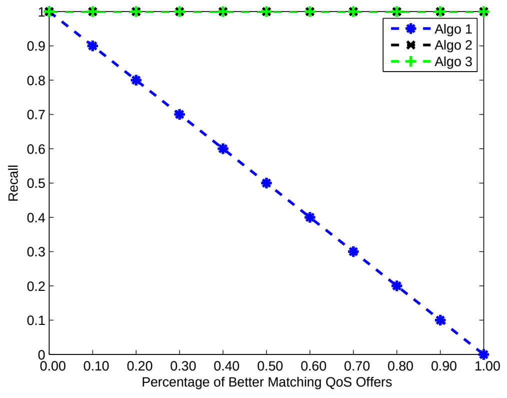

Figure 5. Plot of Recall w.r.t the Percentage of Better Matching QoS Offers.

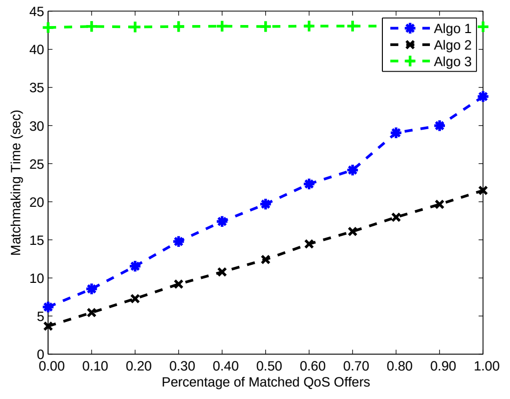

Figure 3. Plot of Matchmaking Time w.r.t the Percentage of Matched QoS Offers.

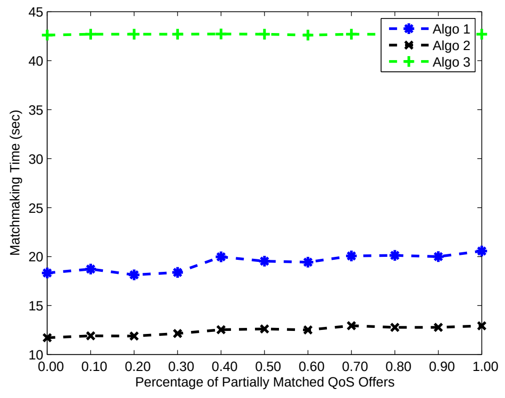

Figure 6. Plot of Matchmaking Time w.r.t the Percentage of Partially Matching QoS Offers.

than for *fail* results so when the number of *partial* results increases so does the whole matchmaking execution time.

In the sixth and seventh experiments, we had $adscn = 10$, $metrcn = 20$ and we increased the percentage of real-valued QoS metrics and short-integer-valued QoS metrics, respectively, in the QoS-based WS specifications. The results were exactly the same and similar to those of the fifth experiment. More specifically, algorithms 1, 2 and 3 had approximately a stable performance behavior. This can be explained as follows: MIP uses better techniques for feasibility checking than CP. This is the reason why this result comes in contrast to the results of the similarly conducted experiment in [2].

In the last experiment, we had $metrcn = 10$, $matchper = 0.0$ and we were increasing the number of QoS offers. We wanted to observe the performance of the algorithms when there are no matches (i.e. the QoS demand is over-constrained). Algorithm 3 exhibited a big linear increasing performance behavior while the rest of the algorithms exhibited a very small linear increasing performance behavior. This can be explained as follows: Algorithm 3 spends a specific amount of time for each non-conforming QoS offer. So as the number of non-conforming QoS offers increases, so does the matchmaking time. On the other hand, the rest of the algorithms need only to solve one to three CSPs each of the time for every QoS offer. That's why their matchmaking time increases so slowly. In this type of experiment, we cannot define accuracy. However, we can compare these algorithms on the size of conforming results. As expected, Algorithms 1 and 2 do not produce any result. On the other hand, Algorithm 3 always produces exactly one result that violates the smallest number of QoS demand's constraints. Due to page limitations, we do not include a figure showing the above results.

5 Conclusions

This paper presented a theoretical and empirical analysis of three QoS-based unary-constrained WS matchmaking algorithms: the one proposed in [2] and two different versions of the one we are proposing in [10]. The evaluation of these three algorithms validated the conclusions of the theoretical analysis. In addition, it revealed other interesting results. The two versions of our algorithm perform quite fast. However, when it comes to advanced categorization of results and return of conforming results for over-constrained QoS demands, only one performs well with the penalty of increased matchmaking time. So for increased accuracy we always pay a performance penalty.

For future work, we plan to analyze and evaluate a QoS-based WS matchmaking approach with n-ary constraints like the one we have proposed in [10]. We also plan to devise an algorithm that uses other CP techniques [1].

References

[1] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[2] A. R. Cortés, O. Martín-Díaz, A. D. Toro, and M. Toro. Improving the automatic procurement of web services using constraint programming. *Int. J. Cooperative Inf. Syst.*, 14(4):439–468, 2005.

[3] S. Degwekar, S. Y. W. Su, and H. Lam. Constraint specification and processing in web services publication and discovery. In *ICWS*, pages 210–217. IEEE Computer Society, 2004.

[4] H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In A. B. Chaudhri, M. Jeckle, E. Rahm, and R. Unland, editors, *Web, Web-Services, and Database Systems*, volume 2593 of *Lecture Notes in Computer Science*, pages 221–237. Springer, 2002.

[5] E. Giallonardo and E. Zimeo. More semantics in qos matching. In *International Conference on Service-Oriented Computing and Applications*, pages 163–171, Newport Beach, CA, USA, 2007. IEEE Computer Society.

[6] A. Keller and H. Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. Technical Report RC22456 (W0205-171), IBM, 2002.

[7] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with owls-mx. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, Hakodate, Japan, 2006. ACM Press.

[8] K. Kritikos and D. Plexousakis. Semantic qos metric matching. In *ECOWS '06: Proceedings of the European Conference on Web Services*, pages 265–274, Zurich, Switzerland, 2006. IEEE Computer Society.

[9] K. Kritikos and D. Plexousakis. Requirements for qos-based web service description and discovery. In *REFS '07: Proceedings of the 1st IEEE International Workshop on Requirements Engineering for Services*, pages 467–472, Beijing, China, 2007. IEEE Computer Society.

[10] K. Kritikos and D. Plexousakis. Semantic qos-based web service discovery algorithms. In *ECOWS '07: Proceedings of the European Conference on Web Services*, Halle, Germany, 2007. IEEE Computer Society.

[11] E. M. Maximilien and M. P. Singh. Conceptual model of web service reputation. *SIGMOD Rec.*, 31(4):36–41, 2002.

[12] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic ws-agreement partner selection. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 697–706, Edinburgh, Scotland, 2006. ACM Press.

[13] S. Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4(1):1–10, 2003.

[14] V. Tosic, B. Pagurek, and K. Patel. Wsol - a language for the formal specification of classes of service for web services. In L.-J. Zhang, editor, *ICWS*, pages 375–381. CSREA Press, 2003.

[15] C. Zhou, L.-T. Chia, and B.-S. Lee. Daml-qos ontology for web services. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, page 472, San Diego, CA, USA, 2004. IEEE Computer Society.