Efficient versus Accurate Algorithms for Computing A Semantic Logic-based Similarity Measure

Fatma Ezzahra Gmati[1], Salem Chakhar[2], Nadia Yacoubi Ayadi[3], Afef Bahri[4], and Mark Xu[2]

[1] COSMOS, National School of Computer Science, University of Manouba, Manouba, Tunisia.
fatma.ezzahra.gmati@gmail.com
[2] Portsmouth Business School and Centre for Operational Research & Logistics, University of Portsmouth, Portsmouth, UK.
salem.chakhar@port.ac.uk; mark.xu@port.ac.uk
[3] RIADI Research Laboratory, National School of Computer Sciences, University of Manouba, Manouba, Tunisia.
nadia.yacoubi.ayadi@gmail.com
[4] MIRACL Laboratory, High School of Computing and Multimedia, University of Sfax, Sfax, Tunisia.
afef.bahri@gmail.com

Abstract. There are three types of Web services matchmakers: logic-based, non logic-based, and hybrid. Logic-based matchmakers employ the semantics of the Web services. Non-logic based matchmakers employ other approaches such as syntactic and structural matching. Hybrid matchmakers combine both approaches. This paper presents and compares two algorithms for computing a semantic logic-based similarity measure in the perspective of Web service matching. The first algorithm offers an efficient solution, while the second algorithm proposes a more accurate result. Both algorithms are evaluated using the SME2 tool. Performance evaluation shows that efficient algorithm reduces substantially the computing time while the accurate algorithm ameliorates considerably the precision of the matching process.

Keywords: Web service, Matchmaking, Similarity Measure, Degree of Match, Performance.

1 Introduction

Web service matchmaking consists in the computation of the similarity between two Web services. It is an important task in Web service discovery and Web service composition. The matchmaking process is generally based on the Web service description; several standards define this description such as WSDL, OWL-S, SAWSDL, the two later include semantics. Many matchmakers that support Web semantics have been proposed in the literature, including [1][3][4][7][9][10][12].

Three types of Web services matchmakers can be distinguished [2][11][12]: (i) logic-based matchmakers that employ the semantics of the Web services and set logic-rules and constraints in order to perform the matching; (ii) non-logic based matchmakers that rely on syntactic and structural matching techniques; and (iii) hybrid matchmakers that combine both of the above mentioned matching approaches. Logic-based techniques did not receive much attention and the authors in [9] qualify them as imprecise. However, if the logic-based techniques is considered as a component of a hybrid matchmaker, improving their performances would improve the overall performance of the matchmaker.

To the best knowledge of the authors, the first logic based matchmaker has been proposed in [10]. The authors in [1] improve [10]'s proposal by using bipartite graphs. In this paper, two logic-based algorithms for computing the similarity measure between two Web services are proposed. These algorithms can be seen as an extension of the one proposed in [10]. Both algorithms have been evaluated using the SME2, which is an open source tool for testing different semantic matchmakers. The first algorithm slightly affects the precision of [1]'s algorithm but reduces substantially its computing time. The second algorithm enriches the semantic distance values used in [1] and ameliorates considerably its precision.

The rest of this paper is organized as follows. Section 2 introduces basic definitions and shows how the similarity measure is computed. Sections 3 and 4 present the efficient and accurate algorithms, respectively. Section 5 studies the computational complexity of the algorithms and Section 6 evaluates and compares their performances. Section 7 ends the paper.

2 Similarity Measure

2.1 Basic Definitions

A semantic match between two entities frequently involves a *similarity measure*. The similarity measure quantifies the semantic distance between the two entities participating in the match. A similarity measure is defined as follows.

Definition 1 (Similarity Measure). *The similarity measure, μ, of two service attributes is a mapping that measures the semantic distance between the conceptual annotations associated with the service attributes. Mathematically, $\mu : A \times A \rightarrow V$, where A is the set of all possible attributes and V the set of possible semantic distance measures.*

There are different possible definition of set V. In [5], for instance, the mapping between two conceptual annotations may take one of the following values: Exact, Plugin, Subsumption, Container, Part-of and Disjoint.

A preferential total order may now be established on the above mentioned similarity maps.

Definition 2 (Similarity Measure Preference). *Let V be the set of all possible semantic distance values such that $V = \{v_1, v_2, \cdots, v_n\}$. Preference amongst the similarity measures is governed by the following strict order: $v_1 \succ v_2 \succ \cdots \succ v_n$, where $a \succ b$ means that a is preferred over b.*

Definition 3 (Degree of Match). *The degree of match is a function that defines a semantic distance value between two conceptual annotations. Mathematically,* $\delta : CA_1 \times CA_2 \rightarrow V$*, where* CA_1 *denotes the first conceptual annotation and* CA_2 *denotes the second conceptual annotation and* V *is the set of all possible semantic distance values.*

This generic definition of similarity measure extends the one proposed by [5]. Other matchmaking frameworks (e.g. [1][10][13]) utilize an idea similar to μ, but label it differently. The main difference between the above-cited works concerns the way the degree of match is computed. In Sections 3 and 4, two algorithms for computing the similarity measure are provided. These algorithms improve the one proposed by [1], which is presented in the rest of this section.

2.2 Computing the Similarity Measure

The computing of the similarity measure over two attributes is modeled by [1] as a *bipartite graph matching*. Let first introduce some concepts.

Definition 4 (Bipartite Graph). *A graph* $G = (V_0 + V_1, E)$ *with disjoint vertex sets* V_0 *and* V_1 *and edge set* E *is called bipartite if every edge connects a vertex of* V_0 *with a vertex of* V_1 *and there are no edge in* E *with both endpoints are in* V_0 *or in* V_1*.*

Definition 5 (Matching in a Bipartite Graph). *Let* $G = (V, E)$ *be a bipartite graph. A matching* M *of* G *is a subgraph* $G' = (V, E')$*,* $E' \subseteq E$*, such that no two edges* $e_1, e_2 \in E'$ *share the same vertex. A vertex* v *is matched if it is incident to an edge in the matching* M*.*

Let S^R be a service request and S^A be a service advertisement. The matching between $S^R.A_i$ and $S^A.A_j$ (i.e. $\mu(S^R.A_i, S^A.A_j)$) is computed as follows:

1. *Construction of the bipartite graph.* Let CA_0 and CA_1 be the sets of concepts corresponding to the attributes $S^R.A_i$ and $S^A.A_j$, respectively. These two sets are the vertex sets of the bipartite graph G. In other words, $G = (V_0 + V_1, E)$ where $V_0 = CA_0$ and $V_1 = CA_1$. Considering two concepts $a \in V_0$ and $b \in V_1$. An edge between a and b is valued with the degree of match between a and b, i.e., $\delta(a, b)$. Then, a numerical weight is assigned to every edge in the bipartite graph as given in Table 1. These weights verify the following constraints:
$$w_1 \leq w_2 \leq w_3 \leq w_4. \quad (1)$$
2. *Selection of the optimal matching.* A matching is selected only if it fulfills an injective mapping between V_0 and V_1. Figure 1 presents two matchings. The first one respects the injective mapping between V_0 and V_1 while the second does not. A bipartite graph G may contain several possible matchings. In this case, the identification of the *optimal matching* is required. According to [1], the optimal matching is the one that minimizes $\max(w_k)$ with $\max(w_k)$

Table 1. Weighting system

Degree of match	Weight of edge
`Exact`	w_1
`Plugin`	w_2
`Subsume`	w_3
`Fail`	w_4

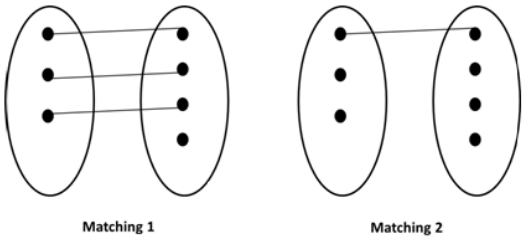

Fig. 1. Matching examples

is the maximum weighted edge in the matching. The final returned degree corresponds to the label of the edge that maximizes $\max(w_i)$ in the obtained matching.

To select the optimal matching, the authors in [1] propose the use of the Hungarian algorithm. However, the application of Hungarian algorithm using the weighing system given in Table 1 is not possible. This is because the Hungarian algorithm minimizes the sum of weights while the optimal matching is the one that minimizes the maximum weight. The authors in [1] proved that a modification in weights as in Table 2 permits the Hungarian algorithm to identify correctly the optimal matching.

Table 2. Improved Weighting System

Degree of match	Weight of edge
`Exact`	$w_1 = 1$
`Plugin`	$w_2 = (w_1 * \|V_0\|) + 1$
`Subsume`	$w_3 = (w_2 * \|V_1\|) + 1$
`Fail`	$w_4 = w_3 * 100$

The computing of the similarity measure $\mu(\cdot, \cdot)$ as described above is formalized in Algorithm 1. The function **ComputeWeights** applies the weight changes as described in Table 2. The function **HungarianAlg** implements the Hungarian Algorithm. The $w(a, b)$ in Algorithm 1 is the weight associated to edge (a, b).

The degree of match $\delta(\cdot, \cdot)$ between two conceptual annotations is established according to Algorithm 2 where: $\equiv$: Equivalence relationship; $\sqsupset_1$: Direct parent/child relationship; $\sqsupset$: Indirect parent/child relationship; and $\sqsubset$: Direct or indirect child/parent relationship.

The optimality criterion used in [1] is designed to minimize the false positives and the false negatives. In fact, minimizing the maximal weight would minimize the '`Fail`' labeled edges. However, the choice of $\max(w_i)$ as a final return value is restrictive and the risk of false negatives in the final result is higher. To avoid this problem, the consideration of both $\max(w_i)$ and $\min(w_i)$ as pertinent values in the matching is proposed.

Algorithm 1: SimilarityMeasure as defined in [1]

```
   Input  : S^R.A_i, // attribute A_i in requested service.
            S^A.A_j, // attribute A_j in advertised service.
   Output: μ(S^R.A_i, S^A.A_j)// similarity measure between S^R.A_i and S^A.A_j.
 1 Graph G ← EmptyGraph(V_0 + V_1, E);
 2 V_0 ← concepts of attribute A_i in requested service;
 3 V_1 ← concepts of attribute A_j in advertised service;
 4 (w_1, w_2, w_3, w_4) ←ComputeWeights();
 5 for (each a ∈ V_0) do
 6     for (each b ∈ V_1) do
 7         SemanticDistance ← δ(a, b);
 8         if (SemanticDistance = Exact) then
 9             w(a, b) ← w_1
10         else
11             if (SemanticDistance = Plugin) then
12                 w(a, b) ← w_2
13             else
14                 if (SemanticDistance = Subsume) then
15                     w(a, b) ← w_3
16                 else
17                     w(a, b) ← w_4
18 M ← HungarianAlg(G);
19 Let (a', b') denotes the maximal weighted edge in M;
20 FinalSemanticDistance ← δ(a', b');
21 return FinalSemanticDistance;
```

Algorithm 2: Degree of Match $\delta(\cdot,\cdot)$ as defined in [1]

```
   Input  : CA_a, // first concept.
            CA_b, // second concept.
   Output: degree of match
 1 if (CA_a ≡ CA_b) then
 2     return Exact;
 3 else if (CA_a ⊐_1 CA_b) then
 4     return Plugin;
 5 else if (CA_a ⊐ CA_b) then
 6     return Plugin;
 7 else if (CA_a ⊏ CA_b) then
 8     return Subsume;
 9 else
10     return Fail;
```

In the two next sections, two different algorithms for the computing the similarity measure are introduced. These algorithms are based on [1]'s approach. The main difference concerns the computation of the degree of match $\delta(\cdot,\cdot)$.

3 Efficient Computation of the Similarity Measure

The semantic distance values are defined similarly to [1] (see Table 1). The improvement concerns the degree of match which is now computed as in Algorithm 3 where: $\equiv$: Equivalence relationship; $\sqsubset_1$: Direct child/parent relationship; and $\sqsupset_1$: Direct parent/child relationship.

In this version of the algorithm, only direct related concepts are considered for Plugin and Subsume semantic distance values. This change affects the precision of the algorithm since it uses a small set of possible concepts. However, it reduces considerably the computing time. In fact, there is no need to use inference: only facts are parsed in the related ontology, which reduces the complexity of Algorithm 3 (as discussed in Section 5). This will necessarily improves the query response time. The proposed algorithm provides a balance between response time and precision, which is valuable in critical situations.

Algorithm 3: Degree of Match $\delta(\cdot,\cdot)$ for an efficient computation of $\mu(\cdot,\cdot)$

```
   Input  : CA_R, // first concept.
            CA_A, // second concept.
   Output: degree of match
1  if (CA_R ≡ CA_A) then
2  |  return Exact;
3  else if (CA_R ⊏_1 CA_A) then
4  |  return Plugin ;
5  else if (CAS_R ⊐_1 CA_A) then
6  |  return Subsumes;
7  else
8  |  return Fail ;
```

The computation of the similarity measure is the same as in [1] (see Algorithm 1) but the interpretation of the semantic distances is as given in the beginning of this section.

4 Accurate Computation of the Similarity Measure

In this case, six semantic distance values are defined as given in Table 3. The basic idea of this second improvement is that the precision of the algorithm increases with the number of granular values. Then, an extended weighting system is define as in Table 4. These weights verify the following constraints:

$$w_1 \leq w_2 \leq w_3 \leq w_4 \leq w_5 \leq w_6 \tag{2}$$

Table 3. Elements of the semantic distance values set V

V	Semantic distance value
v_1	`Exact`
v_2	`Plugin`
v_3	`Subsume`
v_4	`Extended-Plugin`
v_5	`Extended-Subsume`
v_6	`Fail`

Table 4. Extended weighting system

Degree of match	Weight of edge
Exact	w_1
Plugin	w_2
Subsume	w_3
Extended-Plugin	w_4
Extended-Subsume	w_5
Fail	w_6

In this version of the similarity measure, the improvement is also made over the computing of degree of match $\delta(\cdot,\cdot)$, which is now computed according to Algorithm 4 where: $\equiv$: Equivalence relationship (it does not need inference); $\sqsubset_1$: Direct child/parent relationship; $\sqsupset_1$: Direct parent/child relationship; $\sqsubset$: Indirect child/parent relationship; and $\sqsupset$: Indirect parent/child relationship.

Algorithm 4: Degree of match $\delta(\cdot,\cdot)$ for the accurate computation of $\mu(\cdot,\cdot)$

```
   Input  : CA_R, // first concept.
            CA_A, // second concept.
   Output: degree of match//
 1 if (CA_R ≡ CA_A) then
 2 |   return Exact ;
 3 else if (CA_R ⊏_1 CA_A) then
 4 |   return Plugin ;
 5 else if (CA_R ⊐_1 CA_A) then
 6 |   return Subsume ;
 7 else if (CA_R ⊏ CA_A) then
 8 |   return Extended-Plugin ;
 9 else if (CA_R ⊐ CA_A) then
10 |   return Extended-Subsume ;
11 else
12 |_  return Fail;
```

In this algorithm, the consideration of indirect concepts is performed for both Extended-Plugin and Extended-Subsume semantic distance values. Following Definition (2), the following preference order holds:

$$\text{Exact} \succ \text{Plugin} \succ \text{Subsume} \succ \text{Extended-plug-in} \succ \text{Extended-subsume} \succ \text{Fail}.$$

Direct relations (i.e. Plugin and Subsume) are preferred to indirect relations (i.e. Extended-Plugin and Extended-Subsume). To apply the Hungarian Algorithm, the weights are modified as in Table 5.

Table 5. Improved weighting system

Degree of match	Weight of edge
Exact	$w_1 = 1$
Plugin	$w_2 = (w_1 * \|V_0\|) + 1$
Subsume	$w_3 = (w_2 * \|V_1\|) + 1$
Extended-Plugin	$w_4 = (w_3 * \|V_1\|) + 1$
Extended-Subsume	$w_5 = (w_4 * \|V_1\|) + 1$
Fail	$w_6 = w_5 * 100$

The computation of the similarity measure is given in Algorithm 5. This algorithm extends the one proposed by [1] (see Algorithm 1). The function **ComputeWeights** applies the weight changes as described in Table 5. The function **HungarianAlg** implements the Hungarian algorithm. The $w(a, b)$ in Algorithm 5 is the weight associated to edge (a, b).

Algorithm 5: SimilarityMeasure

Input : $S^R.A_i$, // attribute A_i in requested service.
$S^A.A_j$, // attribute A_j in advertised service.
Output: $\mu(S^R.A_i, S^A.A_j)$// similarity measure between $S^R.A_i$ and $S^A.A_j$.

```
1  Graph G ← EmptyGraph(V0 + V1, E);
2  V0 ← concepts of attribute Ai in requested service;
3  V1 ← concepts of attribute Aj in advertised service;
4  (w1, w2, w3, w4, w5, w6) ←ComputeWeights() ();
5  for (each  a ∈ V0) do
6      for (each  b ∈ V1) do
7          SemanticDistance ←δ(a,b);
8          if (SemanticDistance = Exact) then
9              w(a, b) ← w1
10         else
11             if (SemanticDistance = Plugin) then
12                 w(a, b) ← w2
13             else
14                 if (SemanticDistance = Subsume) then
15                     w(a, b) ← w3
16                 else
17                     if (SemanticDistance = Extended-Plugin) then
18                         w(a, b) ← w4
19                     else
20                         if (SemanticDistance = Extended-Subsume) then
21                             w(a, b) ← w5
22                         else
23                             w(a, b) ← w6
24 M ←HungarianAlg (G);
25 Let (a', b') denotes the maximal weighted edge in M;
26 FinalSemanticDistance ← δ(a',b');
27 return FinalSemanticDistance;
```

5 Computational Complexity

The most expensive operation in Algorithms 1 and 5 is the computing of the degree of match. As shown in [5], inferring degree of match by ontological parse of pieces of information into facts and then utilizing commercial rule-based engines which use the fast Rete [6] pattern-matching algorithm leads to $O(|R||F||P|)$ where $|R|$ is the number of rules, $|F|$ is the number of facts, and $|P|$ is the average number of patterns in each rule. Accordingly, the complexity of Algorithms 2 and 4 is $O(|R||F||P|)$. However, the computing the degree of match according to Algorithm 3 is only $O(|F|)$ since there is not inference.

Let now discuss the algorithmic complexity of [1]'s approach. Let m be an approximation of the number of concepts for the attributes to be compared. Then: (i) the computation of weights is an operation of $O(1)$ complexity; (ii) the construction of the graph involves the comparison of every pair of concepts. It takes then $O(m^2)$ time complexity; (iii) the Hungarian Algorithm has a time complexity of $O(m^3)$; and (iv) the degree of match in Algorithm 2 is $O(|R||F||P|)$.

Generally, m is likely to take small values and it can be considered as a constant. The overall time complexity of [1]'s algorithm is than $O(1+m^2|F||R||P|+m^3) \simeq O(|F||R||P|)$. Based on the above discussion, the complexity of [1]'s algorithm will be $O(1 + m^2|F| + m^3) \simeq O(|F|)$ when the degree of similarity is computed as in Algorithm 3. The computation of the similarity measure according to Algorithm 5 is the same as [1]'s algorithm, i.e., $O(1+m^2|F||R||P|+m^3) \simeq O(|F||R||P|)$.

6 Evaluation and Comparison

6.1 Performance Analysis

To SME2 [8] tool has been used to evaluate the performance of the algorithms. The SME2 uses OWLS-TC collections to provide the matchmakers with Web service descriptions, and to compare their answers to the relevance sets of the various queries. The SME2 provides several metrics to evaluate the performance and effectiveness of a Web service matchmaker. The metrics that have been considered in this paper are: precision and recall, average precision and query response time. The definition of these metrics are given in [8].

A series of experimentations have been conducted on a Dell Inspiron 15 3735 Laptop with an Intel Core I5 processor (1.6 GHz) and 2 GB of memory. The test collection used is OWLS-TC4, which consists of 1083 Web service offers described in OWL-S 1.1 and 42 queries.

Figures 2 and 3 show the Average Precision and Recall/Precision plot of the accurate and efficient algorithms, respectively. It can be seen that the accurate algorithm outperforms efficient algorithm with respect to these two metrics. This is due to the use of logical inference, that obviously enhances the precision of the accurate algorithm. In Figure 4, however, efficient algorithm is shown to be remarkably faster than the accurate algorithm. This is due to the inference process used in accurate algorithm that consumes considerable resources.

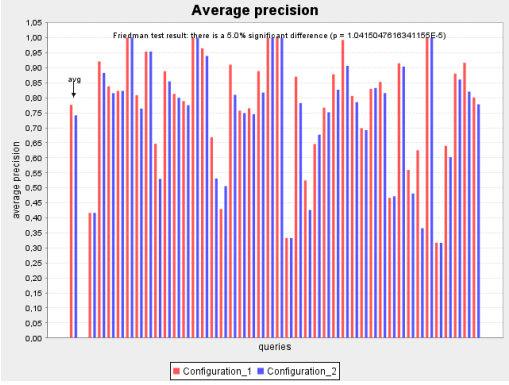

Fig. 2. Average Precision

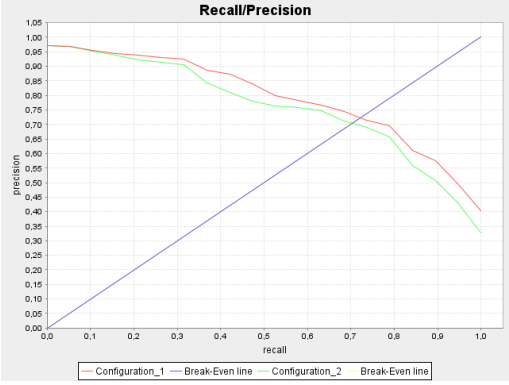

Fig. 3. Recall/Precision

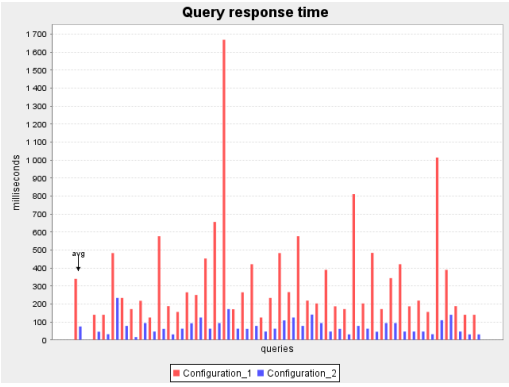

Fig. 4. Response Time

6.2 Comparative Study

Table 6 summarizes the main characteristics of the efficient and accurate algorithms and some existing algorithms proposed in [1][5][10]. The following criteria have been considered in this comparison: (i) the definition of the semantic distance set V, (ii) the computing of degree of match $\delta(\cdot,\cdot)$, (iii) the modelling technique of the matching problem, (iv) the level of precision, and (v) the level of complexity. The description of this table is straightforward.

Table 6. Comparison of similarity measure computing algorithms

Approach	Semantic distance	Degree of match $\delta(c_1, c_2)$	Modelling technique	Precision	Complexity
[5]	v_1=Exact v_2=Plug-in v_3=Subsumption v_4=Container v_5=Part-of v_6=Disjoint	Exact: $c_1 \equiv c_2$ Plug-in: $c_1 \sqsubset c_2$ Subsumption: $c_1 \sqsupset c_2$ Container: $c_1 \succ c_2$ Part-of: $c_1 \prec$ [5] c_2 Disjoint: c_1 $disj$ [6] c_2	Unspecified	High	Moderate
[10]	v_1=Exact v_2=Plugin v_3=Subsume v_4=Fail	Exact: $c_1 \sqsubseteq_1 c_2$ Plugin: $c_1 \sqsubset c_2$ Subsume: $c_1 \sqsupset c_2$ Fail: c_1 $disj$ c_2	Greedy Algorithm	Low	High
[1]	v_1=Exact v_2=Plugin v_3=Subsume v_4=Fail	Exact: $c_1 \equiv c_2$ Plugin: $c_1 \sqsubset c_2$ Subsume: $c_1 \sqsupset c_2$ Fail: c_1 $disj$ c_2	Bipartite Graph	High	Moderate
Efficient Algorithm	v_1=Exact v_2=Plugin v_3=Subsume v_4=Fail	Exact: $c_1 \equiv c_2$ Plugin: $c_1 \sqsubset_1 c_2$ Subsume: $c_1 \sqsupset_1 c_2$ Fail: c_1 $disj$ c_2	Bipartite Graph	Moderate	Low
Accurate Algorithm	v_1=Exact v_2=Plugin v_3=Subsume v_4=Extended-Plugin v_5=Extended-Subsume v_6=Fail	Exact: $c_1 \equiv c_2$ Plugin: $c_1 \sqsubset_1 c_2$ Subsume: $c_1 \sqsupset_1 c_2$ Extended-Plugin: $c_1 \sqsubset c_2$ Extended-Subsume: $c_1 \sqsupset c_2$ Fail: c_1 $disj$ c_2	Bipartite Graph	High	Moderate

7 Conclusion

Web service matchmaking is an important task in Web service discovery and Web service composition. Early non-logic based matchmakers rely on syntactic and structural matching techniques, which suffer from several shortcomings, especially the high number of false positives and false negatives. Accordingly, different logic-based matchmakers have been proposed (e.g., [1][5][9][10][13]) to overcome these shortcomings. In particular, the authors in [1] propose a bipartite graph-based algorithm that improves considerably the algorithm of [10] by reducing false positives and false negatives in the final results.

This paper presents and compares two algorithms for computing a semantic logic-based similarity measure in the perspective of Web service matching. These algorithms improve the computing of the similarity measure proposed in [1]. Both algorithms have been evaluated using the SME2 tool. The results show

that the first algorithm slightly affects the precision of [1]'s algorithm but reduces substantially its computing time, while the second algorithm enriches the semantic distance values used in [1] and ameliorates considerably its precision. In the future, more advanced performance evaluation with large datasets will be conducted.

References

1. U. Bellur and R. Kulkarni. Improved matchmaking algorithm for semantic Web services based on bipartite graph matching. In *IEEE International Conference on Web Services*, pages 86–93, Salt Lake City, Utah, USA, 9-13 July 2007.
2. M.B. Blake, L. Cabral, B. König-Ries, U. Küster, and D. Martin, editors. *Semantic Web Services, Advancement through Evaluation.* Springer, 2012.
3. S. Chakhar. Parameterized attribute and service levels semantic matchmaking framework for service composition. In *Fifth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2013)*, pages 159–165, Seville, Spain, 27 January - 1 February 2013.
4. F. Chen, C. Lu, H. Wu, and M. Li. A semantic similarity measure integrating multiple conceptual relationships for web service discovery. *Expert Systems with Applications*, 67:19–31, 2017.
5. P. Doshi, R. Goodwin, R. Akkiraju, and S. Roeder. Parameterized semantic matchmaking for workflow composition. IBM Research Report RC23133, IBM Research Division, March 2004.
6. C. Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37, 1982.
7. F.E. Gmati, N. Yacoubi Ayadi, A. Bahri, S. Chakhar, and A. Ishizaka. A framework for parameterized semantic matchmaking and ranking of web services. In *Proceedings of the 12th International Conference on Web Information Systems and Technologies*, pages 54–65, 2016.
8. M. Klusch, M. Dudev, J. Misutka, P. Kapahnke, and M. Vasileski. *SME^2 Version 2.2. User Manual.* The German Research Center for Artificial Intelligence (DFKI), Germany, 2010.
9. M. Klusch and P. Kapahnke. The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15:1–14, 2012.
10. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, pages 333–347, London, UK, UK, 2002. Springer-Verlag.
11. J. Sangers, F. Frasincar, F. Hogenboom, and V. Chepegin. Semantic web service discovery using natural language processing techniques. *Expert Systems with Applications*, 40(11):4660–4671, 2013.
12. T.G. Stavropoulos, S. Andreadis, N. Bassiliades, D. Vrakas, and I.P. Vlahavas. The tomaco hybrid matching framework for SAWSDL semantic web services. *IEEE Transactions on Services Computing*, 9(6):954–967, 2016.
13. K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The retsina mas infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48, 2003.