We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

26

Semantic Matchmaking Algorithms

Umesh Bellur[1], Harin Vadodaria[2] and Amit Gupta[3]
[1]*Department of Computer Science and Engineering, Indian Institute of Technology, Bombay*
[2]*Sybase Software, Pune*
[3]*Dept. of CSE, IIT Bombay*
India

1. Introduction

The advantages of loose coupling offered by service oriented architectures (*SOA*) have made it a popular choice for today's enterprise systems. The popularity has driven standardization efforts in the areas of service advertisement and invocation and services specified using these standards are termed as Web services. A Web service is self containing, self describing application that can be deployed, published and invoked over the Internet. The *publish-find-bind* approach is the fundamental idea behind Service Oriented Architectures that web services aim to implement. The ultimate vision of *SOA* is to enable a client to automatically select an appropriate service from a pool of dynamically discovered services and invoke it without having any apriori knowledge about the service provider and the specifics of the service itself. This vision has thrown up various challenges such as - *service discovery based on an abstract query, selection of service from the discovered pool, service composition, dynamic service binding and invocation, quality of service, negotiation of service contracts and trust.*

Enhancing what has traditionally been sytanctic descriptions of services with semantics is necessary to resolve most of these issues. Once semantic descriptions are available, one needs to deal with matchmaking of these descriptions to a query. In the rest of this chapter we present concepts involved in semantic matchmaking as it applied to web services and a set of algorithms that solve the semantic matchmaking problem.

1.1 Background concepts

In this section, we present necessary background concepts essential to understand the rest of the chapter. The chapter centers around Web services although there exist several other implementation of SOA concepts such as Jini for Java.

1.1.1 Service discovery

Service discovery is the process of evaluating a query for a service and returning a set of compatible services. *WSDL* and *UDDI* are two standards used in service discovery. The Simple Object Access Protocol (SOAP) is a messaging protocol used to invoke web services and get back results asynchronously.

- The Web Services Description Language (*WSDL*) [3] is a language for description of service and contains operations supported by the service. Each operation is described

Source: Advances in Greedy Algorithms, Book edited by: Witold Bednorz,
ISBN 978-953-7619-27-5, pp. 586, November 2008, I-Tech, Vienna, Austria

by it's input and outputs. *WSDL* description of a service defines XML message format for communication with the service. A compiler at the client generates stubs based on the WSDL description for: 1) Marshaling and unmarshaling objects into SOAP messages. 2) Sending SOAP messages over communication protocol. The application is then bound with these stubs to invoke the service.

- The Universal Description and Discovery Interface [2] is a registry that contains information about different services offered by various service providers. This information is usually output as a WSDL document. UDDI provides mechanisms for: 1) Publishing a service to the registry 2) Searching a required service from the registry. The state of the art today limits storage in UDDI to Strings and searches are syntactic in nature as well.

1.1.2 Ontology

Ontology represents knowledge about a particular domain. This knowledge includes entities in the domain, their property and relationship with each other. Entities in the ontology are termed *Concepts*. A well defined syntax is required to unambiguously represent concepts of a domain. RDF[16] framework is suitable for describing an ontology. Web Ontology Language (OWL)[17] is developed on the top of RDF and is used for ontology description. Given below is a part of Entertainment ontology.

```
<owl:Class rdf:about="#Concert">
<rdfs:subClassOf rdf:resource="#Theatre"/>
</owl:Class>
<owl:Class rdf:about="#Drama">
<rdfs:subClassOf rdf:resource="#Theatre"/>
</owl:Class>
<rdf:Description rdf:resource="#Concert">
<owl:DisjointWith rdf:resource="#Drama">
</rdf:Description>
```

1.1.3 OWL-S: semantic markup for web services

Semantic web is not merely a collection of marked up content but includes (software applications packaged as) services as well. It is essential for a software agent to discover, compose, invoke and monitor web resources in order to take advantage of a service. OWL-S [1] (formerly, DAML-S) is a language for describing services which makes this possible. It uses RDF as basic framework. OWL-S is required to perform following tasks automatically.

- **Web Service Discovery:** Extract the information from the page in order to find a required service.
- **Web Service Invocation:** OWL-S along with the domain ontology specifies the invocation methods of a Web Service (e.g. necessary inputs, expected outputs).
- **Web Services Composition and Interoperation:** OWL-S provides declarative way to specify prerequisite and consequences of a service which helps software agents in composing different web services.

OWL-S provides Service Profile, Service Model and Service Grounding to represent Description, Functionality and Access Mechanism respectively.

- **Service Profile:** Service profile facilitates Service Provider to describe its service. It is up to the Service Provider how much details are given in the Service Profile. E.g. a Book selling service may also provide browsing facility but it is not necessary that it is included in Service Profile. We can categorize the information provided by Service Profile as:
  - *Provider's Information* - This may include name of the provider and contact details.
  - *Functional Description* - specifies inputs required, output generated and conditions to be set at the beginning and change in the real world after service completes its function. In short, inputs, outputs, preconditions and effects are described here.
  - *Profile Attributes* - Some parameters that service wants to specify e.g. quality guarantees, service categorization etc. They are represented by Service Parameter and Service Category.
- **Service Model:** It describes service as a process, either atomic or composite: receives and sends a single message or retains/changes state through a sequence of messages. A service can give some output and set some condition thus changing real world.
  - Inputs and output parameters are expressed as a subclass of the *parameter* class in OWL-S.
  - Preconditions and effects are modeled as logical formulas or expressions which are treated as either string literals or XML literals depending on the language used. The expression class in OWL-S specifies two separate subclasses *condition* and *effect* for precondition and effect respectively.

Often outputs and effects of the service are coupled together with a condition bounding them. E.g. service for selling software modules may have different results and effects depending on a failed or succeeded transaction.

Composite processes are more difficult to handle. They have a set of sub processes associated with a control structure. The control structure will specify the order in which different sub processes are executed. In case of composite process, client needs to send a series of messages to get the final result. Different types of control structures are: Sequence, Split, Split+Join, Any-Order, Choice, If-Then-Else, Iterate, Repeat While and Repeat Until. Data flow and parameter binding is very critical issue in case of composite process. OWL-S has adopted Consumer-Pull convention i.e. if p2 requires input which comes from p1, p2 is responsible for explicitly describing this fact.

Service Grounding: Grounding deals with the realization of services. It provides concrete details necessary to invoke the service such as message format, transfer protocol etc. OWL-S uses WSDL standard for Service grounding. WSDL provides a wrapper and can carry OWL-S message on standard network protocols. WSDL can not capture the semantic of a message while OWL-S in its own is not capable to deal with the standard transfer protocol. Both languages overlap at description of message at abstract level. Mapping from OWL-S to WSDL is done in 3 steps:

- An OWL-S atomic process corresponds to a WSDL operation.
- Inputs and outputs of OWL-S process correspond to input part and output part of WSDL messages respectively.
- Inputs and output of OWL-S process correspond to WSDL's abstract type.

An example of OWL-S profile is as follows:

```
    <!--- Reference to ontology used in OWL-S --->
<owl:Ontology rdf:about="">
<owl:imports rdf:resource=
"http://www.someurl.org/ontology/hotel.owl" />
</owl:Ontology>

<!--- Description of input parameter in profile --->
<profile:hasInput  rdf:resource="#_CITY"/>

<!--- Connecting the parameter with a concept from ontology --->
<process:Input rdf:ID="_CITY">
<process:parameterType rdf:datatype=
"http://www.w3.org/2001/XMLSchema#anyURI">
http://www.someurl.org/ontology/hotel.owl#City</process:parameterType>
</process:Input>
```

2. What is semantic matchmaking?

The publish-find-bind architecture targets dynamic service invocation - i.e., the client of the service invocation has no prior knowledge of the service description and hence cannot link in pre-compiled stubs. Specification standards such as WSDL and registry standards such as UDDI facilitate the discovery process that is needed for dynamic invocation. Together UDDI and WSDL can serve the goal of automatic discovery of web services. However, the matching mechanism provided by UDDI and WSDL is no better than a simple string matching in XML. In reality automated service discovery can not be accomplished by mere string matching. For example, a simple service that takes two integers as input and produces a float as output could actually perform one of a variety of operations like interest calculation on a principal and period, average points per game given the total points and games etc. A simple syntax based matching can produce many false positives since nature of service is not captured in the service description.

In order to overcome this limitation, concept of semantics has been introduced with OWL-S. In this approach, functionality of a service is described in terms of inputs, outputs, preconditions and effects. Input and output terms of the service are expressed as concepts belonging to a set of ontologies. Use of ontology allows referring to a single concept from two or more syntactically different terms. Thus, it eliminates the limitations caused by syntactic difference between terms since matching is now possible on the basis of concepts of ontologies used to describe input and output terms.

For semantic matchmaking, if we assume that both, advertisement and query are defined in OWL-S format then an advertisement *Advt* and query *Query* match if

- For every input parameter in *Advt*, there is one input parameter in *Query*. Let $Query_{in}$ and $Advt_{in}$ represent the list of input concepts of query and the advertisement respectively. The service can correctly perform the task if all the input concepts defined in the advertisement are satisfied by the requester. Hence, matching of inputs exist if

  $$\forall c \in Advt_{in}, \exists d \in Query_{in},$$
  $$s.t.\ match(c,d) \neq Fail$$

- For every output parameter in *Query*, there is one output parameter in *Advt*. Let $Query_{out}$ and $Advt_{out}$ represent the list of output concepts of query and the advertisement

respectively. The service can be used by the requester if all the output concepts defined in the query are satisfied by the advertisement. Hence, matching of outputs exist if

$$\forall c \in Query_{out}, \exists d \in Advt_{out},$$
$$s.t.\ match(c,d) \neq Fail$$

- For every precondition in *Advt*, there is one precondition in *Query*. Let $Query_{precondition}$ and $Advt_{precondition}$ represent the list of preconditions of query and the advertisement respectively. The service can correctly perform the task if all the preconditions defined in the advertisement are satisfied by the requester. Hence, matching of preconditions exist if

$$\forall c \in Advt_{precondition}, \exists d \in Query_{precondition},$$
$$s.t.\ match(c,d) \neq Fail$$

- For every effect in *Query*, there is one effect in *Advt*. Let $Query_{effect}$ and $Advt_{effect}$ represent the list of effects of query and the advertisement respectively. The service can be used by the requester if all the effects defined in the query are satisfied by the advertisement. Hence, matching of effects exist if

$$\forall c \in Query_{effect}, \exists d \in Advt_{effect},$$
$$s.t.\ match(c,d) \neq Fail$$

2.1 An example

Let us now look at an example of how a request is matched with service advertisements. The service that is advertised is a car selling service which, when given a *Price* as input, return which car can be bought at that price. A strip-down version of advertisement is shown in Figure 2. As, is clear the input to the service are instances of the concept *Price* and the output is instances of the concept *Car*.

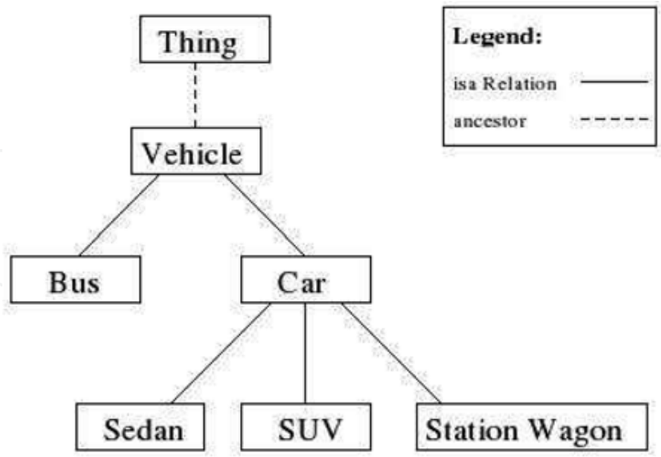


Fig. 1. A fragment of Vehicle Ontology [12]

Shown below is an example request in the same format. The request shows that the service sought should take as input instances of *Price* and should generate output as instances of *Sedan*.

Now, in the given example, for service to match with the request we need to match inputs as well as outputs. In this case, inputs match directly as they both contain the same concept *Price*. The outputs also match as *Car* provided by the service in the given ontology is a super class of the *Sedan* which is expected in the request. Hence, this will be considered as a suitable match although the score(or rank) of this match will vary accordingly with different semantic matchmaking algorithms.

```
<profile:Profile rdf:ID="CarSellingService">
  <profile:serviceName>CarSellingService</profile:serviceName>
  <profile:providedBy> ... </profile:providedBy>
  <input>
    <profile:ParameterDescription rdf:ID="Price_Input">
      <profile:parameterName>Price</profile:parameterName>
      <profile:restrictedTo rdf:resource="Concets.daml#Price"\>
    </profile:ParameterDescription>
  </input>
  <output>
    <profile:ParameterDescription rdf:ID="Car_Output">
      <profile:parameterName>Car</profile:parameterName>
      <profile:restrictedTo rdf:resource="Vehicle.daml#Car"\>
    </profile:ParameterDescription>
  </output>
</profile:Profile>
```

Fig. 2. Advertisement of a car selling service [12]

```
<profile:Profile rdf:ID="RequestSedanSellingService">
  <input>
    <profile:ParameterDescription rdf:ID="Price_Input">
      <profile:parameterName>Price</profile:parameterName>
      <profile:restrictedTo rdf:resource="Concets.daml#Price"/>
    </profile:ParameterDescription>
  </input>
  <output>
    <profile:ParameterDescription rdf:ID="Sedan_Output">
      <profile:parameterName>Sedan</profile:parameterName>
      <profile:restrictedTo rdf:resource="file:data/Vehcle.daml#Sedan"/>
    </profile:ParameterDescription>
  </output>
```

Fig. 3. Request for a car selling service service [12]

If we had an advertisement with *Sedan* as a concept, it must be ranked higher than the above advertisement as it is closer to the given request. From this example, it is clear, the match performed is a semantic match. The reason is because the fact that *Car* is a superclass of *Sedan* has been used while matching. In a syntactic matching scenario, this would result in no match as *Car* and *Sedan* are syntactically very different.

Note that in given example service semantics are described by input and output parameters only. In addition to these parameters, preconditions and effects can also be added to define restriction over parameter values.

3. Taxonomy of semantic matchmaking algorithms

In this section, we present the qualitative and quantitative aspects on which a semantic matchmaking algorithm can be evaluated. We then use this to compare and contrast different efforts in the area.

3.1 Qualitative aspects

Semanticmatching as compared to syntacticmatching As the term *semantic matchmaking* suggests, a semantic matchmaking algorithm should consider the meaning of concepts while performing comparisons between services and requests. It should take into account the various relations which exist between the concepts in the ontology in the process of matchmaking.

For example, in the ontology given by 1, when a request contains *Sedan*, a service with concept *Car* should be given more weightage than concept *Vehicle* as *Sedan* is closer to *Car* in the ontology. Similarly, a service with *Sedan* should be given an appropriate score(less than *Car*) when a request for *Car* is made by taking into account the fact that *Sedan* could only partially satisfy the request. Its important to note that in pure syntactic matching, this kind of reasoning is not possible as the meaning of the concepts are not considered.

False positives and negatives False positives are returned when a semantic matchmaking algorithm matches an advertisement to a given request even if it was not relevant. Analogically, a false negative is the case where a semantic matchmaking algorithm fails to match a relevant advertisement to the given request. There is a trade off between the number of false positives and false negatives returned by a matchmaking algorithm. As the algorithm becomes more flexible, the number of false positives increase and number of false negatives decrease. Therefore, its necessary to regulate the flexibility of the semantic matchmaking algorithm so as to have a balanced number of false positives and negatives. The requesting service should have some control over the flexibility of the algorithm.

Notion of Flexible matching The semantic matchmaking algorithm should promote the advertisers to be more precise in their description. It can be done by providing a degree of match for the matched advertisements. The degree of match should be higher for advertisements which are closer to the request and hence imposing penalty on advertisements which are very general. If this is not done, then all the advertisers will make advertisements as general as possible to increase their chances of match rather being specific about what they actually have.

Consider that AdOp is one of the concepts of the outputs of an advertisement Qop is one of the concepts of the outputs of a query. Four degrees of matching are:

- **Exact:** If AdOp is *an equivalent concept* to QOp, then they are labeled as *Exact match.*
- **Plug in:** If QOp is *superclass of/subsumes* AdOp, then AdOp can be plugged in place of QOp. Thus, it is marked as *Plug in match.*
- **Subsumes:** If AdOp is *superclass of/subsumes* QOp, then service may fulfill the requirements of the request since advertisements provides output in some of the subclasses of the concept defined by QOp. Thus, it is a *subsume match.*

- **Fail:** If no subsumption relation is found between QOp and AdOp, then it is declared as failure.

Soft constraints Soft constraints are constraints which should be preferably but not necessarily be satisfied. For example, if we are ordering a DVD from a web-based DVD store, we could specify that we would prefer to pay by a credit card. A semantic matchmaking algorithm should be able to take soft constraints into account while performing matches. Hence, an advertisement which satisfies a soft constraint should be given better ranking than an advertisement which does not (assuming they satisfy other constraints with same degree of match).

Preference of concepts The user should be able to specify which concepts are preferred. A semantic matchmaking algorithm should to take into account the preference of concepts as specified by the user. For example, if a user needs to book a hotel for his journey, then most important concepts for him would be the *city* and *date*. Other concepts(for e.g. prices) even if matched would be useless unless it is in the same city as mentioned by the user. Hence its important that the algorithm gives higher ranking to advertisements which match concepts of higher preferences.

User defined matching The user should have some control over the matching process. The algorithm should give the user ability to regulate various aspects such as the flexibility of the algorithm, the *quality* or *rating* which is expected for the service etc. User-defined matching helps make the matching process more suited to one's needs.

Heterogeneous ontologies The semantic matchmaking example, we discussed in previous section assumes that both the service and the request use the same shared ontology for description. However, in a truly distributed environment where services are autonomous this assumption may not hold true. Hence, an algorithm must be able to perform semantic matching across descriptions with heterogeneous ontologies.

Quality-of-Service(QoS) enabled discovery All the aspects we have discussed so far deal with how closely the advertisement matches functionally with the request. However, non-functional and QoS properties such as price, performance, throughput, reliability, availability, trust etc. are equally important while deciding whether a service is satisfactory for a given request. Hence, a semantic matchmaking algorithm must take into account the presence of such parameters while matching. User feedback must be taken into account in this framework to define QoS properties for various services.

3.2 Quantitative measures

Efficiency : As we know that the search has to be made over all possible advertisements for services. Given the current size of Web, the number of services existing on web and hence the number of advertisements will be huge. Hence, for the semantic matchmaking process to scale up to the size of web, the computational complexity of the algorithm should not be high.

Precision : Precision is defined as the number of "relevant and retrieved" advertisements over the number of "retrieved" advertisements. As the algorithm becomes more flexible in matching, the number of false positives increase and hence precision decreases. Therefore,

AdvtInput	Date,City
AdvtOutput	Theatre,Music

Table 1. Advertisement

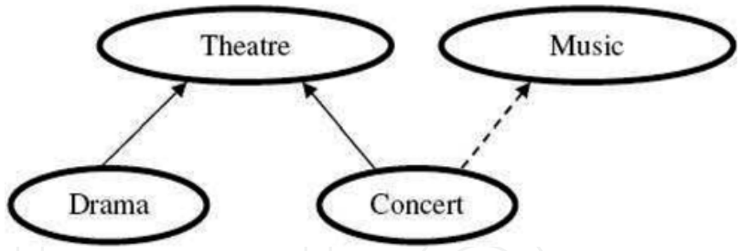


Fig. 4. A part of Entertainment Ontology

QueryInput	Date,City
QueryOutput	Concert,Drama

Table 2. Query

to obtain higher precision the semantic matchmaking algorithm has to be more rigid in matchmaking.

Recall Recall is defined as the number of ”relevant and retrieved” advertisements over the number of ”relevant” advertisements. As the algorithm becomes more flexible in matching, the number of false negatives decrease and hence recall increases. Therefore, to obtain higher recall the semantic matchmaking algorithm has to be more flexible in matchmaking.

F1 and break-even : As discussed earlier, we need to have regulated amount of flexibility in the algorithm to balance precision and recall. Since there is a trade-off between precision and recall, we can use unified measures which will give weightage to both precision and recall. For example, F1 is defined as the harmonic mean of precision and recall. Hence, when maximized, it would result in both precision and recall set to acceptable values. Similarly, break-even is the point where the precision and recall curves meet each other.

Precision and recall are very coarse-grained measures as they categorise a document into two categories :- *relevant* or *irrelevant*. Especially, in the context of semantic matchmaking where the degree of match between query and advertisement comprises of many levels, such coarse-grained measures are not the best indicators of performance of matchmaking.

We need fine-grained evaluative measures which can distinguish between documents matching with various degrees of match. [16] proposes a method based on fuzzy logic which provides fuzzy equivalents of precision and recall as measures to quantify performance of matchmaking.

These equivalents are computed in terms of two membership functions, one defined by the semantic matchmaking engine and one by the domain experts. The two membership functions are $fe : Q \times S \rightarrow [0, 1]$, and $fr : Q \times S \rightarrow [0, 1]$. *fe* is delivered by the algorithm and *fr* is calculated by the feedback of domain experts. These functions are computed using fuzzification of the degree of match performed between the advertisement and the request. The fuzzy logic equivalents of Recall(R_G) and Precision(P_G) are defined in Equations 1 and 2.

$$R_G = \frac{\sum_{S_i \epsilon S} min\{fr(R, S_i), fe(R, S_i)\}}{\sum_{S_i \epsilon S} fr(R, S_i)} \tag{1}$$

$$P_G = \frac{\sum_{S_i \epsilon S} min\{fr(R, S_i), fe(R, S_i)\}}{\sum_{S_i \epsilon S} fe(R, S_i)} \tag{2}$$

Since, theses measures are fuzzy, they take into account the values for all the advertisements and not only those documents which are relevant or returned.

4. A survey of matchmaking algorithms

In this section we take a in-depth look into existing semantic matchmaking algorithms.

4.1 Greedy approach

This algorithm was proposed by [12]. It is based on semantic matchmaking based on input and output terms.

Algorithm presented in [12] is a greedy approach for matchmaking. Algorithm tries to match every output concept of Query with one of the concepts of Advertisement. It starts from all output concepts (call it candidate list) of Query and removes a concept from candidate list as soon as it is matched with a concept from Advertisement with degree of matching > Fail.

[12] uses following scheme for degrees of matching.

- **Exact:** If AdOp is an equivalent concept to QOp, then they are labeled as Exact match. If QOp is a subclass of AdOp, then match is considered Exact under the assumption that provider agrees to provide output in every possible subclasses of AdOp.
- **Plug in:** If AdOp subsumes QOp, then AdOp can be plugged in place of QOp. Here also, it is assumed that provider agrees to provide output in some of the subclasses of AdOp.
- **Subsumes:** If QOp subsumes AdOp, then service may fulfill the requirements of the request since advertisements provides output in some of the subclasses of the concept defined by QOp.
- **Fail:** If no subsumption relation is found between QOp and AdOp, then it is declared as failure.

4.1.1 Discussion

Scheme for Degrees of matching assumes that service provider agrees to provide output in every possible subclass of the output concept. Also, Algorithm is dependent on the order in which concepts are defined in the Query. Consider following example.

As depicted in figure 4, Drama and Concert are subclass of the concept Theatre. Concert is also a sub-concept of Music via inferred relationship.

Output concept list for *Query* and *Advt* are:

- $Advt_{output}$ = Theatre, Music
- $Query_{output}$ = Concert, Drama

At first, algorithm will try to match Concert with all concepts of candidate list of $Advt_{output}$. We have,

- Theatre *is superclass of* Concert ⇒ Exact match
- Music *subsumes* Concert ⇒Plugin match

Since the algorithm uses greedy approach, it will match Concert with Theatre and removes both from respective lists. Now there is only one concept in $Advt_{output}$ and,

Match(Drama, Music) = Fail

Thus, the algorithm will return *Fail* match for *Query* and *Advt*. In reality, we can have following matching.

- Theatre *is super class of* Drama→ Exact match
- ″ Music *subsumes* Concert → *Plugin* match

Overall degree of matching for *Query* and *Advt* is *Plugin*. If we have changed order of concepts in *Query* $_{outputlist}$, we could have achieved this matching. Thus, algorithm in [12] is dependent on the order in which concepts are defined. Thus, algorithm may produce false negative results.

Consider the scenario when the output concept is not removed from the candidate list. Suppose, the advertisement is for {*Theatre*, *Cost*} and the request is for {*Drama*, *Concert*}. Here, the above algorithm would return an exact match as both *Drama* and *Concert* are immediate subclasses of *Theatre*. Hence, the requester would receive only one reservation for a *Theatre* whereas he expected two reservations for a *Drama* and a *Concert*. This would result in a false positive.

4.1.2 Quantitative analysis

For each advertisement, we have to compare each output concept of query with all the advertisement concepts and each input concept of advertisement to all the input concepts of query. Hence, the number of operations are given by

$$C = \{(Qo \times Ao) + (Qi \times Ai)\} \tag{3}$$

where Qo and Ao are the number of output concepts and Qi and Ai are the number of input concepts in query and advertisement respectively. Since, the algorithm iterates over N advertisements, the total complexity is given by

$$C = N \times \{(Qo \times Ao) + (Qi \times Ai)\} \tag{4}$$

in general, Qo,Ao,Qi and Ai are bounded by small integers. Hence, the complexity is linear in N (the number of advertisements) with small constants.

$$Complexity \approx O(N) \tag{5}$$

4.2 Bipartite graph based matching

To solve the problems mentioned in previous section, we introduce in this section another approach [5] towards semantic matchmaking which makes use of bipartite graph matching to produce a match.

The algorithm also introduces a different set of rules of match between concepts in which **PlugIn** and **Subsume** levels are interchanged in their degree of match. The assumption that if an advertiser advertises a concept, it would provide all the immediate subtypes of that concept is dropped. Hence, if the query concept is subsumed by the advertisement concept a **Subsume** is returned while if the query concept subsumes the advertisement concept **PlugIn** is returned. **PlugIn** is still ranked higher than a **Subsume** match. You can see that this scheme of matching is opposite to the one discussed in previous section.

Bipartite graph A bipartite graph is a graph in which the vertex set can be divided into two disjoint sets such that no edge of the graph lies between the two vertices in the same set.

Matching A matching of a bipartite graph G = (V,E) is a subgraph G′ = (V,E′) such that no two edges e1,e2 in E′ share the same vertex.

Let the set of output concepts for query and advertisement be Q and A. We will construct a graph G = (Q+A,E) which has one vertex corresponding to each concept in query and advertisement . If there exists a degree of match between (≠ Fail) between a concept $v1$ belonging to Q and a concept $v2$ belonging to A, then we define an edge ($v1$, $v2$) with weight as the degree of match. We need a matching in which all the output concepts of Q are matched with some concept of A. If such a matching exists, we would say that the advertisement and the query match.If there exist multiple such matchings, we will choose the one which is optimal(the criterion is defined below). However, if such a matching doesn't exist the query and the advertisement doesn't match.

optimality criterion We need to select the matching which is best from the perspective of semantic match. For, this we would assign different numerical weights to edges with different degrees of match. Let us suppose, we assign minimum weight to **exact**, then **Plugin** and then **subsumes**.Let $max(wi)$ be the maximum weighted edge in the matching. An optimal matching in this case would be a complete matching with minimum $max(wi)$.

Algorithm for optimal matching Hungarian algorithm [10] computes a complete matching for a weighted bipartite graph such that sum of weights of all the edges in the matching is minimised. To adapt Hungarian algorithm to above case, where a matching with minimum value of $max(wi)$ is needed, we would assign weights according to the scheme shown as below.

Degree of Match : match(a,b)	**Weight**
Exact	w1 = 1
Plugin	$w2 = (w1 * \mid V0 \mid) + 1$
Subsume	$w3 = (w2 * \mid V0 \mid) + 1$

It can be proved that with the above weighting scheme, a matching in which Σwi is minimized is equivalent to matching in which $max(wi)$ is minimised [5].

Matchmaking Algorithm The search procedure accepts a query as input and tries to match its output concepts and input concepts with each advertisement. If there exists a match in both input and output concepts, it appends the advertisement to the result set. To match inputs as well as it outputs, it invokes Hungarian algorithm on a graph created with weights as given in above table to compute an optimal matching of the graph. The degree of match is defined by the weight of the maximum-weight edge in the matching. In the end, a list of advertisements sorted on the basis of input and output concepts is returned.

4.2.1 Discussion

The above algorithm eliminates the correctness issues with the algorithm described in the previous section. It also regulates false positives and false negatives as discussed in the example above. However, it does not allow for priority of concepts and soft constraints to be input by the user. Like the previous algorithm, the algorithm does not provide a ranking of the results. The algorithm assumes a shared ontology between the advertisements and the request. In the following sections, we would look at some algorithms which allow some of these features.

4.2.2 Quantitative analysis

Using the same notation as in Section 4.1.2, we get

The weights *wo*, *w*1 and *w*2 are computed in *O*(1) time. The weights of edges in the graph can be determined in *Qo*×*Ao* operations, by comparing all pair of concepts.
The time complexity of Hungarian algorithm is bounded by Qo^3. Hence, the total complexity of the search is bounded by:

$$C = N \times \{(Qo \times Ao) + Qo^3 + (Qi \times Ai) + Ai^3\} \quad (6)$$

Hence, we get If we assume that number of input and output concepts in the query and advertisement are small, we can approximate:

$$Complexity \approx O(N) \quad (7)$$

The complexity of above algorithm is asymptotically similar to the previous algorithm. However, the constants will be different.

4.2.3 Addition of precondition and effect matching

Original algorithm proposed by [5] was based on matching of input and output terms only. However, precondition and effect matching can also be added using same bipartite graph based technique as discussed in [6]. As discussed earlier, in OWL-S description, preconditions and effects are represented as boolean expression. Algorithm for condition matching works in two phases.

- **Parameter Compatibility:** Whether parameters used in both expressions are equivalent or not. From input-output terms matching, we obtain the mapping between terms used in query and advertisement. If every parameter used in the query's condition has an equivalent parameter (obtained from the mapping constructed during input-output term matching phase) in the advertisement's condition such that, degree of matching between two parameters > Fail Match, we have parameter compatibility between these two conditions.
- **Condition Equivalence:** This refers to structural similarity between two conditions. For our purpose, we do not need strict equivalence. If condition specified in the query contains all parameters specified in advertisement's condition AND the relation between various parameters in advertisement's condition are retained in query condition, we can flag it as condition equivalence. In other words, if condition in the query is denoted by QCondition and condition in advertisement is denoted by ACondition then what we need is,
  $QCondition \Rightarrow Acondition$
  which essentially says that, variable space in which QCondition is true is a subset of the variable space in which ACondition is true.
  This is true when we match for preconditions. The relation will be reversed when we match for effects. i.e.
  $ACondition \Rightarrow Qcondition$
  This problem is constraint satisfiability problem which NP-Complete by its nature. Some heuristics like DPLL algorithms are used to solve this problem in exponential time.

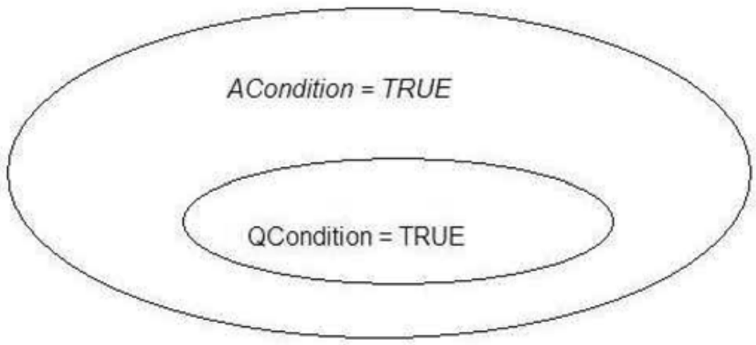


Fig. 5. Solution space for *ACondition* and *Qcondition*

4.3 Semantic matchmaking across heterogeneous ontologies

In this section, we discuss a framework for semantic matchmaking which relaxes the requirement of a single ontology and allows advertisements and requests to be expresses in different ontologies. The approach to compare concepts across ontologies uses different ways to assess the similarity of various concepts used in description of services and requests [14].

SynonymSets Synonymsets are semantically equivalent or very similar words. Hence, synonyms can be considered as the same entity. Wordnets are used to derive the synonym set of the name of the parameters. In a cross-ontology evaluation scenario these words(like person and human) are likely to refer to the same entity.

Distinguishing features of concepts Some concepts could have quite different names, while still being semantically similar. To incorporate semantics into the similarity measure in such cases we can also use some distinguishing features of concepts. We choose the properties of classes such as object properties and data type properties to perform semantic similarity tests. The assumption is that semantically similar parameters with different names are likely to have some common features or properties. The matching is performed between the properties of the two concepts.

Semantic neighbourhoods and relations The semantic relations which exist between various classes could be used to perform semantic matchmaking. The idea is that the target concepts (i.e.which are subject of comparison) which are related to the same set of classes through similar relations, may be semantically similar. For example, semantic relations like Subclass, Disjoint With,Equivalent Classes etc. can help determine semantic similarity amongst various concepts.

To integrate the information obtained by above methods, a weighted sum of the similarity of each function component is used to compute the overall similarity.

Another similarity measure defined in terms of set theory is based on the normalisation of Tversky's model and the set-theory functions of intersection($A \cap B$) and difference (A/B). It is shown below.

$$S(a,b) = \frac{A \cap B}{A \cap B + \alpha(a,b)A/B + (1-\alpha(a,b))B/A}, for 0 \leq \alpha \leq 1 \quad (8)$$

where a and b are parameter classes

A and B corresponds to the description sets of a and b

(i.e. synonym sets,feature sets and semantic neighbourhood)

and α is a function which defines relative importance of non-common characteristics

The above mentioned similarity measures are used to compute the edge weights of edges in the bipartite graph discussed in the previous section. The function elements(concepts) are extracted from the advertisement as well as the requested profile. A bipartite graph is formed using these concepts as nodes. The edge weights are then computed using the similarity measures described above. Bipartite graph matching algorithms are then applied to produce matches and their scores which are used to generate the sorted list of relevant advertisements.

4.3.1 Discussion

The algorithm provides a way to make semantic matchmaking possible over descriptions with heterogeneous ontologies. The algorithm uses a similarity function of concepts for matching which is based on a weighted sum of synonym sets,semantic neighbourhood and distinguishing features. The algorithm however does not allow the user to input preferences of concepts. Preference of concepts would give the requester more expressive power to express their needs. In further sections, we would look into some algorithms which support preferences amongst the concepts.

4.4 Semantic matchmaking based on description-logics

We now discuss an algorithm which performs semantic matchmaking on advertisements and requests which are defined in Description Logics. Description Logics(DL) are a family of logic formalisms used for knowledge representation. They can be used to represent the knowledge of a service or an application domain in a structured and formal way which can be understood by a computer system. As we will see the algorithm discussed below provides a ranking of the matched advertisements which was not the case with the previous algorithms.

In this section, we'll discuss an algorithm for the DL of the Knowledge Representation System CLASSIC of AT&T Bell Labs [15]. The basic syntax is based on predicate logic and comprises of three kinds of descriptions.

- **concept names** concept names stand for sets of objects, such as book, room etc.
- **number restrictions** these correspond to restrictions which quantify the amount of a concept. for example, $(> 3 author)$ denotes that there should be more than three authors.
- **universal quantifications** these can be used to specify some restriction on all the objects belonging to a concept. For example, $\forall\ supplier.japanese$ implies that all the suppliers (i.e.objects belonging to the concept supplier) must be Japanese.

An advertisement (as well as a request) can be described as a conjunction of these concepts. For example, one might represent an advertisement for an apartment as

$$A = apartment \cap \forall hasRooms.roomswithTV \cap (\geq 3\ hasRooms)$$

Requests can be represented in the similar way. The matchmaking algorithm then matches the request with the candidate advertisements one by one and provides a ranking for the match.The algorithm, recursively calls itself for the parts which are universally quantified and keeps a global score which denotes the degree of match. If there happens to be a case, in which there exists a universal quantification statement for a particular concept in only one of the advertisement or request, the recursive call is made with a T (universal truth) as predicate. Hence, if a description does not mention $\forall hasRooms$, we would assume that $\forall hasRooms.T$ is present in the description. The algorithm is as follows.

4.4.1 Algorithm

The algorithm for ranking follows a recursive procedure as mentioned above. It starts with a global rank of zero for every advertisement and then increases it for every concept which differs in the advertisement and the request. Therefore, lower the rank, higher is the degree of match. The algorithm uses four rules to increase the rank which are given below :-

- Add 1 to rank for each concept name which are present in query but not in advertisement
- Add 1 to rank for each number restriction in request which can not be satisfied by the advertisement
- If for a concept $\forall R.E$, there does not exist a $\forall R.F$ in the advertisement, add to rank the answer to the recursive call with T, and E.
- If for a concept $\forall R.E$, there does exist a $\forall R.F$ in the advertisement, add to rank the answer to the recursive call with F, and E.

Total match exists when the algorithm returns 0. The above algorithm can be modified easily to provide for preference of concepts. By adding different weights for different concepts, we can penalize the match selectively according to our preferences. Thus, an important concept would cause a larger number to be added to n, hence decreasing the degree of match for advertisements which can not satisfy them. In the next section, we will see how preference of concepts can increase your expressive power in defining the query. The taxonomy can be also taken into account, while defining weights for various concepts. Hence, in the taxonomy of figure 2.1, we could say that n will be increased a larger amount if we have vehicle and SUV as compared to if we have vehicle and cars. The weights can also be learnt by the system, by providing a set of advertisements and their ranks according to human users. Hence, the system would be able to learn to distinguish between concepts which are more important by learning weights to fit given training examples of ranked advertisements.

4.4.2 Discussion

In this section we saw an algorithm which performs semantic matchmaking on advertisements and requests described using Description Logics. As we discussed, the algorithm performs an approximate matching of advertisements and requests and provides a ranking of candidate advertisements with varying degrees of match. The algorithm can also be used to take into account preference of concepts as provided by the user.

4.5 Semantic matchmaking based on ranked instance retrieval

In this section we present another method for semantic matchmaking which takes into account the preference of concepts a provided by the user [4].This algorithm uses the concept of a ranking tree to match and compare various advertisements w.r.t. a particular query.

We will take an example to describe how the preference of concepts gives you more expressive power in making the request. Suppose, the user wants to find out a service which offers DVD's for movies. Hence she could make a query like, Q1 := *OffersDVD*. However, this would provide tons of hits. To narrow down our search, she would want to provide more search criterion. Suppose she specifies that she prefers 24 hours shipping over three-day shipping and a service with shipping time more than three days is not acceptable.

In this case, writing a query like Q2 := $OffersDVD \sqcap (24HoursShipping \sqcap 3DaysShipping)$, would get unacceptable results as the proper requirement has not been expressed. To

express the correct requirement, there should be a way to annotate the concepts with preferences, thus providing a way to determine which concept is preferred. Hence, if we provide a query like Q3 :=$OffersDVD^1 \sqcap (24HoursShipping^2 \sqcap 3DaysShipping^1)$, the service with 24HoursShipping would be rated more than 3DaysShipping and hence would generate acceptable results.

The above method of annotating preferences, could also be used to specify soft constraints as discussed earlier. Suppose, we have a top concept T, such that every concept is an instance of type T. Now, suppose if we need to specify that we would prefer to have a service with CreditCardPayment, however its not a necessity, we could do that by writing the query as

Q4 :=$OffersDV\ D^1 \sqcap (24HoursShipping^2 \sqcap 3DaysShipping^1) \sqcap (CreditCardPayment^1 \sqcup\ T^0)$.

Similarly, we could use a bottom concept $\perp$ to denote the fact that OffersDVD is a necessary concept but should not affect the ranking.Suppose we write our query as,

Q5 :=$OffersDV\ D^1 \sqcup (24HoursShipping^2 \sqcup 3DaysShipping^1) \sqcup (CreditCardPayment^1\ \sqcup\ \perp^0)$.

In the above query, the second part of disjunction is not satisfiable and hence every hit must satisfy OffersDVD. Hence, ranking is only affected by the other concepts which could be taken into account by the matchmaking algorithm. Therefore, as we discussed allowing annotations of concepts with their preferences could give us a lot more expressive power in describing our request. It will also allow the user to specify soft constraints and constraints like in Q5.

Due to existence of disjunctive knowledge in description logics (in query Q5), a single numerical value is not sufficient for expressing rankings. Suppose, we have Q := $A^1 \sqcup (B^1 \sqcup C^2)^0$. Since, B ⊔ C has preference 0, it should not contribute to the top level rank. However, for two equal top level ranks, we should use $(B^1 \sqcup C^2)^0$ to refine the ranking. A ranking tree is appropriate for such kind of reasoning.In the following part of this section, we will discuss a ranking tree and how it can be used for matchmaking of advertisements and requests with such annotations.

4.5.1 Ranking tree

We define a ranking tree as follows:

1. for r ∈ [0 ; 1] (r) is a ranking tree.
2. let r ∈ [0 ; 1] and t1,t2 .. tn be ranking trees with n ≥ 1, then (r,t1,t2,...,tn) is a ranking tree.

for example t1:= (0,(1), (0,(1),(0))) is ranking tree.

Ordering on ranking tree Let a = (r_a, $a1$, $a2$, $a3$....an) and b = (r_b, $b1$, $b2$, $b3$.....bn) where a1,a2...an and b1,b2 ... bn are ranking trees.

Let a < b⇔$r_a < r_b$ now, a ◁ b iff

1. a < b or
2. $r_a = r_b$ and $\exists i : a_i < b_i$ and $\forall 1 \leq j \leq n : b_j \not< a_j$ or
3. $r_a = r_b$ and $\forall 1 \leq i \leq n : a_i \trianglelefteq b_i$

4.5.2 Matchmaking algorithm

Given an annotated query and an advertisement, we must evaluate the ranking tree of the advertisement. The query is represented as either a conjunction or disjunction of subqueries.

The ranking tree is evaluated by calling the routine recursively for each subquery and using the resulting ranking trees to form the top-level ranking tree. The rank of top-level rank tree is an of average user preferences of all the subqueries which are satisfied. For a negated query, the rank of top-level tree is replaced by the average of user preferences of all the concepts which are not satisfied by the query. An atomic concept belonging to the query is satisfied by advertisement if its contained in it.

Discussion The above algorithm supports most of features discussed in chapter 2.The algorithm supports preference of concepts and allows for soft constraints to be specified. In the next section, we would look at Quality-of-Service(QoS) aspects of semantic matchmaking.

4.6 QoS enabled matchmaking

A web service is a web-based interface which providing electronic description of a concrete service. The service could be of varied types from functionality of a software component (such as data backup) to a real-life business activity(such as shipping). Hence, the QoS properties of a service vary over a wide range depending upon the type of service. For example, for a network service, it could be response-time,availability etc. On the other hand, for a pizza delivery service it could the quality of food. QoS is a very important factor in deciding whether the service will be able to fulfil the demands of requester. Hence, it is very important for a semantic matchmaking algorithm to take into account QoS parameters along with the functional properties to perform a more meaningful and successful match.

To support QoS information while discovering services through matchmaking process, we need to evaluate how well a service can fulfil user's non-functional (quality) requirements based on the service's past performance. Hence, there must be an interface where the users can submit their feedback on the perceived QoS of consumed services. While discovering services, we can take into account data from the following sources [11]:-

- QoS values promised by the providers in their advertisements
- feedback on the perceived QoS submitted by the users on the interface
- reports produced by trustworthy QoS monitoring agents such as rating agencies
- QoS information provided by similar discovery components which exist over the network in a distributed setting

A complementary ontology for providing detailed QoS information have been proposed in (Semantics in service discovery and management.) Algorithms similar to what we have studied in this chapter so far, can be applied to perform matching on QoS parameters. Hence, a service would be matched on its functional properties as well as non-functional properties and QoS parameters and the information provided by both the aspects would be combined to provide a common ranking of advertisements which could be used by a requester.

5. Comparing the algorithms

In this section, we provide, a comparison table comparing the algorithms on various aspects we have presented so far.

Feature	Greedy Algo.	Bipartite Matching	Heterogeneous Ontologies	DL Algo	Ranked Instance Retrieval
IO based Matching	Yes	Yes	Yes	Yes	Yes
PE based Matching	No	Yes	No	No	No
Correctness issues	Yes	No	No	No	No
Ranking of hits	No	Yes	Yes	Yes	Yes
Soft constraints	No	No	No	No	Yes
User preferences	No	No	No	Yes(in extended version)	Yes
Heterogeneous ontologies	No	No	Yes	No	No
Computational complexity	O(N)	Within polynomial limits	High	O(N)	High

Table 3. Comparison table for different matchmaking algorithms

6. Applications of semantic matchmaking

Semantic matchmaking has been applied in a variety of contexts. It is a very importan field of research and forms a basis for service discovery and composition. Reliable and efficient algorithms for semantic matchmaking are extremely important in the new vision of web(semantic web).

The algorithm we discussed here have been used to make a combined matchmaker for DAML-S/OWL-S and UDDI. UDDI allows only keyword search based on the names, which is not enough as no inferencing or flexible matching can be performed. A matching engine which augments UDDI registries with an additional semantic layer allows for a capability based matching. It uses the ontologies published on the web. The matchmaking process allows services to be discovered on the basis of their capabilities and hence result in their interoperability and enhanced problem solving abilities with minimizing human intervention.

Preference SQL [9],a powerful extension to SQL aims for providing support for preference queries in relational databases. Its objective is similar to that of matchmaking. It aims to find approximate matches to user's queries which are based on preferences defined by him. It also performs matchmaking to find the best possible match between a request and existing data.

OWLS-MX is a semantic web services matchmaker which retrieves services for a given query. It uses both logic based semantic matching and token-based syntactic similarity measure to perform the match between a query and the services [13].

[9] discusses the need for semantic matchmaking in geo web services which are in growing demand these days. Geo Web services are services which provide location based information on the web. For example, an user who wants to know about hazardous objects

near her proximity would need to first find out her own location by a geo coder service. Then she could use another service to locate the all the chemical factories etc near it. With a semantic matchmaker, such services could be discovered and interoperate with each other to form more complex services.
[7] use semantic matchmaking approach for skill management of various business entities. Semantic matching is performed between buyers and sellers of skills. It is very important for knowledge intensive companies because it can be used to search for professionals who have expertise in a given area within and across companies.
These are but a few applications of matchmaking engines and algorithms that we have presented in this chapter.

7. Open issues

We currently do not have well recognized evaluation metrics for the efficiency of algorithms which would define their scalability in real world scenarios. We also need testcases and testbeds where various algorithms can be plugged in and tested against each other for various features described in previous sections.
Currently fuctional information used in semantic matchmaking is limited to IOPE. This can be extended in the following ways.

- Semantic matchmaking can involve use of contexts. Currently there is no framework defined for context identification and evaluation. In fact, context providers can be thought of web services Web service providing output in terms of contextual information. [6] has proposed an architecture for such a context aware discovery mechanism but still, inclusion of contexts needs major change in current mechanism for semantic matchmaking.
- Preconditions and effects are represented as boolean conditions and matching based on them is limited to structural similarity of expressions. However, if we can treat static and dynamic nature of parameters [6], evaluation of expression can be partially done at discovery time.

McIlraith suggested an approach for matching based on non-functional requirements of web service. We still need an improved framework to specify non-functional requirements more clearly. Also, weights of the results obtained from matching based on functional and non-functional requirements has to be set in such a way that candidate services are ranked as close as uesr's preference.
We presented an algorithm that tries to perform semantic matchmaking across heterogeneous ontologies, still the discrepancies amongst the ontologies could lead to many failed matches. We need a sophisticated mediation layer in the system which would help in translation of natural language requests to ontology based requests. We also need to find better algorithms and framework for dealing with non-functional properties of services such as trust and reliability. Such properties are very important for building semantic matchmaking systems which can be used reliably by users.

8. Concluding remarks

We have seen a variety of algorithms dealing with different aspects of semantic matchmaking. They involve matchmaking based on functional and non-functional requirements of web service. The process of semantic matchmaking assumes that semantic

information is attached to services. But, the question we need to ask is: Is this expressiveness worth the complexity of semantic matchmaking? In other words, is it possible for a semantic matchmaking system to deliver performance comparable to syntactic matching systems(like keyword search)?

9. Acknowledgments

The authors would like to acknowledge the contribution made by Amit Gupta of the Department of CSE, IIT Bombay.

10. References

[1] *OWL-S: Semantic Markup for Web Services*. http://www.w3.org/Submission/OWL-S/. Visited on 10th June,2008.

[2] *Universal Description Discovery and Integration (UDDI)*. Visited on 10th June, 2008.

[3] *Web Services Description Language (WSDL)*. http://www.w3.org/TR/wsdl. Visited on 10th June,2008.

[4] Matthias Beck and Burkhard Freitag. Semantic matchmaking using ranked instance retrieval. *SMR '06: Proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval, Co-located with VLDB*, 178 of CEUR Workshop Proceedings, 2006.

[5] Umesh Bellur and Roshan Kulkarni. Improved matchmaking algorithm for semantic web services based on bipartite graph matching. *ICWS 2007. IEEE International Conference on Web Services, 2007*, 2007.

[6] Umesh Bellur and Harin Vadodaria. On extending semantic matchmaking to include precondition and effect matching. Accepted for publication in the Proceedings of the International Conferences on Web Services, 2008, Beijing, China, 2008.

[7] Simona Colucci et al. A formal approach to ontology-based semantic match of skills descriptions. *Journal of Universal Computer Science, Volume 9, Issue 12*, pages 1437–1454, 2003.

[8] Amit Gupta. Semantic matchmaking algorithms. Technical report, Department of Computer Science and Engineering, IIT-Bombay, 2008. Seminar Report, Third Year BTech Seminar guided by Prof. Umesh Bellur.

[9] Werner Kiebling and Gerhard Kostler. Preference sql: design, implementation, experiences. *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 990–1001, 2002.

[10] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, pages 2:83–97, 1955.

[11] Fabio Porto Le-Hung Vu, Manfred Hauswirth and Karl Aberer. A search engine for qos-enabled discovery of semantic web services. *International Journal of Business Process Integration and Management 2006 - Vol. 1, No.4*, pages 244– 255, 2006.

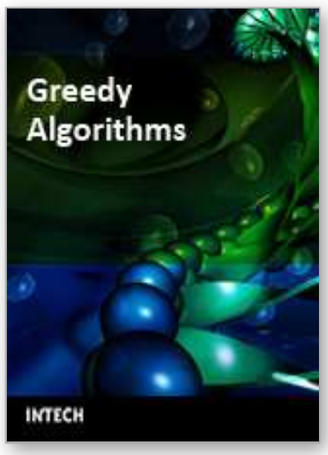
[12] T. Payne M. Paolucci, T. Kawmura and K. Sycara. Semantic matching of web service capabilities. *Springer Verlag, LNCS, Proceedings of the International Semantic Web Conference*, 2002.

[13] Benedikt Fries Matthias Klusch and Katia Sycara. Automated semantic web service discovery with owls-mx. *AAMAS '06: Proceedings of the Fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, 2006.

[14] Jiajin Le ruiqiang Guo and Dehua Chen. Matching semantic web services across heterogenous ontologies. *CIT 05: Proceedings of the Fifth international conference on computer and information technology*, 2005.

[15] Francesco M. Donini Tommaso Di Noia, Eugenio Di Sciascio and Marina Mongiello. Semantic matchmaking in a p2p electronic marketplace. *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 582–586, 2003.

[16] Christos Anagnostopoulos Vassileios Tsetsos and Stathes Hadjiefthymiades. On the evaluation of semantic web service matchmaking systems. *ECOWS '06: Proceedings of the European Conference on Web Services, IEEE Computer Society*, pages 255–264, 2006.



Greedy Algorithms

Edited by Witold Bednorz

ISBN 978-953-7619-27-5
Hard cover, 586 pages
Publisher InTech
Published online 01, November, 2008
Published in print edition November, 2008

Each chapter comprises a separate study on some optimization problem giving both an introductory look into the theory the problem comes from and some new developments invented by author(s). Usually some elementary knowledge is assumed, yet all the required facts are quoted mostly in examples, remarks or theorems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Umesh Bellur, Harin Vadodaria and Amit Gupta (2008). Semantic Matchmaking Algorithms, Greedy Algorithms, Witold Bednorz (Ed.), ISBN: 978-953-7619-27-5, InTech, Available from: http://www.intechopen.com/books/greedy_algorithms/semantic_matchmaking_algorithms

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen