Conceptual Graph Matching for Semantic Search[1]

Jiwei Zhong, Haiping Zhu, Jianming Li and Yong Yu

Department of Computer Science and Engineering,
Shanghai JiaoTong University,
Shanghai, 200030, P.R.China
{zjw035, zhp036, ljm038}@mail1.sjtu.edu.cn, yyu@mail.sjtu.edu.cn

Abstract. Semantic search becomes a research hotspot. The combined use of linguistic ontologies and structured semantic matching is one of the promising ways to improve both recall and precision. In this paper, we propose an approach for semantic search by matching conceptual graphs. The detailed definitions of semantic similarities between concepts, relations and conceptual graphs are given. According to these definitions of semantic similarity, we propose our conceptual graph matching algorithm that calculates the semantic similarity. The computation complexity of this algorithm is constrained to be polynomial. A prototype of our approach is currently under development with IBM China Research Lab.

1. Introduction

Search engine techniques develop quickly in the past decade. However, the majority of traditional search engine techniques, which are based on keyword matching and link analysis [1], have inherent defects. Those engines can only retrieve documents based on the containment of keywords or the document's popularity instead of the documents' real contents. In recent years, semantic search has been brought to the front as people realized that it is insufficient to search text only by keyword matching without exploiting the hidden meaning. Thus, study of semantic search has been carried out, such as [2], [3] and [4], etc. Objects involved in semantic search range from (hyper)texts to multimedia descriptions. As shown in [3], the combined use of linguistic ontologies and structured semantic matching can improve both recall and precision. In this paper, we will propose an approach for semantic search by matching Conceptual Graphs [5] that describe the documents' contents. We will take garment domain to demonstrate our approach, though our method is domain independent and the system can be trained for various domains.

In section 1.1, we introduce the characteristics of domain specific sentences. In section 2, the whole approach is outlined by giving an overview. The exact definition of the similarity in our method and the description of our semantic matching algorithm in detail are given in section 3. In section 4, we give an evaluation of our algorithm. Finally, we compare our algorithm with related work in section 5, and draw the conclusion.

[1] This work is supported by IBM China Research Laboratory.

1.1 Domain Characteristics

The set of sentences that occur only in one given application domain is called domain specific sentences. We assume that domain specific sentences can be characterized as follows [6]:

1. Vocabulary set is limited.
2. Word usage has patterns.
3. Semantic ambiguities are rare.
4. Terms and jargon of the domain appear frequently.

Accordingly the following assumptions could be derived:

1. Sentences that have similar meanings often have similar syntactic structures and use synonyms.

2. CGs generated from sentences with similar meanings will have similar structures.

3. It is relatively easy to build a domain ontology that includes terms or jargon used in a specified domain.

4. Relations used in a specific domain are limited.

Due to assumption 1 and 2, it is natural to consider that an approach based on graph matching technique will work well in deciding the similarity between the meanings of two sentences on thematic similarity level [7].

In addition, before using CG, concept hierarchy and relation hierarchy will be constructed. In our project, WordNet [8] is employed as the main concept hierarchy, and domain ontology hierarchy will be built by hand so as to extend the WordNet to specific domain. This work is feasible considering assumption 3. At the same time, we will manually construct a limited relation hierarchy based on Sowa's thematic roles theory [9]. Though it may be simple and limited, it will work because of assumption 4.

2. Architecture Overview

The whole architecture of our approach is shown in figure 1. Before performing the semantic matching, we download and parse web pages from online garment shops. Descriptions for each garment in web pages can be extracted by hand or by other automating technique, such as wrapper induction [10]. After that, each description is converted to a CG using ALPHA system (the CG generator in figure 1). Last year, we proposed a machine learning based approach that can be trained for different domains and requires almost no manual rules to automatically generate CGs from natural language sentences [6]. ALPHA system is the prototype of this approach which had been implemented and the original results gained from it demonstrated the feasibility of the approach [11]. This makes our current work well-grounded.

After the conversion, those CGs will be stored into our resource CG repository. To organize and manage the repository efficiently, we introduce the concept: 'entry' of graph. Since a CG depicts only one garment in general, it is bound to our concept hierarchy by the article it describes. The concept appearing in the CG on behalf of each garment will be recorded as the 'entry' of graph (i.e. the 'entry' for further semantic graph matching). For example, we will convert the online description "*a*

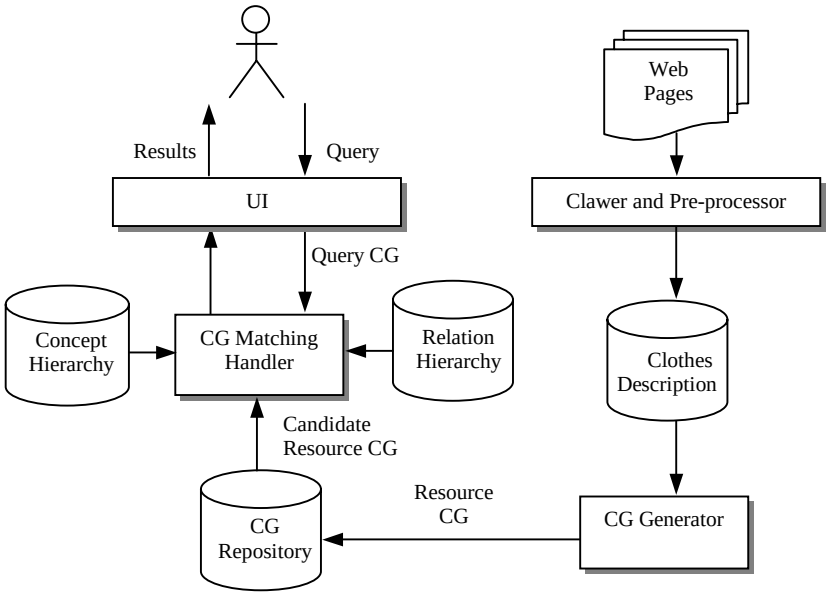


Fig. 1. Overview of the whole architecture of search engine

cotton shirt with a pocket" to the CG as figure 2. The core concept '*shirt*'[2] will be designated as the 'entry' of this graph. Other concepts like this include *dress*, *pants*, etc. The CGs in the repository will be indexed by their entries.

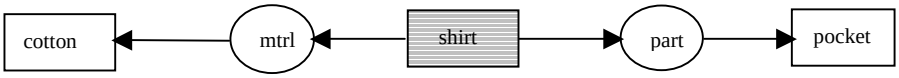


Fig. 2. CG converted from 'a cotton shirt with a pocket'

When the enquiry sentence is entered, it is translated into a query CG. Here, we also need to get entry of the query graph, i.e. the exact object that user is interested in, which will help us to retrieve the repository and get the proper candidates efficiently. Then how will we know the entry of the query graph? Within the UI (User Interface), the user is obliged to specify the 'central word' from his/her query. Afterwards the 'central word' is mapped to the 'entry' of query graph by ALPHA and sent to CG matching handler module with the query graph.

Since hyponymy of senses in WordNet could be regarded as relationship of superclass and subclass, index on CGs by WordNet makes it possible to search all categories within the user's querying object. For instance, supposing 'jersey' is subsumed by 'shirt' according to WordNet, when a user inquires about '*shirt*',

[2] The shirt is not in its word form but the concept ID in the domain ontology. We will use the same convention in the following examples.

resource CG about '*jersey*' will also be considered while matching the query although they are different in their word form.

From the view of CG matching handler module, the input consists of one query graph and one candidate graph fetched from the resource CG repository, while the output is the ranking of the candidates returned to UI. The answers out of those candidates will be returned to the user orderly. After surveying several related systems, e.g. OntoSeek, SCORE, etc., we will, in the next section, present our definition of similarity between the query graph and each candidate resource graph with the help of domain ontologies.

3. Semantic Search by Matching Conceptual Graphs

In this section, we will introduce our approach that performs the semantic search by matching CGs. We will define the similarity between CGs and give the implementation of our method.

3.1 Semantic Similarity

The measure of semantic similarity between a query CG and a resource CG is the key of our approach. Previous work in [7] defined three kinds of similarity, i.e. *surface similarity*, *structure similarity* and *thematic similarity*. Surface similarity or structure similarity is the similarity based on the matching of particular objects or relations, while thematic similarity depends on the presence of particular patterns of concepts and relations. We will focus attention on thematic similarity.

Since CG consists of concepts and relations, we will define the similarity between CGs based on the similarity between concepts and the similarity between relations.

3.1.1 Similarity between Concepts

In our method, the similarity between two concepts is obtained by the distance between them. Given two concepts c_1 and c_2, we will first calculate the distance (denoted as $d_c(c_1, c_2)$). The similarity between two concepts ($sim_c(c_1, c_2)$) is defined as $sim_c(c_1, c_2)=1-d_c(c_1, c_2)$.

The distance between two concepts is calculated by their respective positions in the concept hierarchy. Some previous work [5], [14] and [15] have studied the issue. We borrow their original thought and make some modifications to reflect our intention. In our method, every node in concept hierarchy has a value (we called it 'milestone'), which is obtained from the formula below:

$$milestone(n) = \frac{1/2}{k^{l(n)}}$$

Where k is a predefined factor larger than 1 that indicates the rate at which the value decreases along the hierarchy (currently, we set k equals 2), and $l(n)$ is the depth of the node n in hierarchy (conservatively we choose the longest path from the node to the root to measure it). For the root, $l(root)=0$.

For any two concepts in the hierarchy, they have a closest common parent. The distance between two concepts will be determined by the milestones of them and their closest common parent as follows:

$$\begin{cases} d_c(c_1, c_2) = d_c(c_1, ccp) + d_c(c_2, ccp) & ccp \text{ is the closest common parent of } c_1, c_2 \\ d_c(c, ccp) = milestone(ccp) - milestone(c) \end{cases}$$

This model stems from our thought that the differences between upper level concepts are bigger than those between lower level concepts. The model also supports our intent that the distance between 'brothers' should be longer than that between 'father' and 'son'.

In the formula of the milestone's calculation, the numerator is set to 1/2 so that the distance between the two deepest nodes taking the root as their closest common parent will be 1. That is to say, the distance between other node pairs will be within 1.

Here's an example. Suppose that we are going to find the distance between '*jersey*' and '*pullover'*. Consulting WordNet, we get the ontology segment concerning these two concepts shown in figure 3.

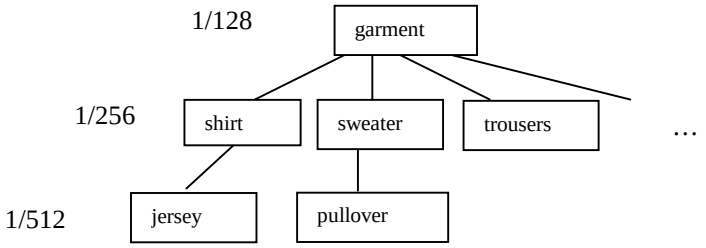


Fig. 3. Ontology segment concerning '*jersey*' and '*pullover*'

Since the closest common parent of '*jersey*' and '*pullover*' is '*garment*', the distance between these two concepts can be calculated as follows (the fractions in the diagram show the 'milestones' of certain ontology levels):

d_c(jersey, pullover)=d_c(jersey, garment)+d_c(pullover, garment)
=(1/128-1/512)+(1/128-1/512) =0.01171875

There is an exception that if the concept of a resource CG is a subclass of the concept of a query CG, the distance will be set to 0, i.e. the similarity between these two concepts will be 1. We think it is reasonable because the subclass is always a kind of superclass.

3.1.2 Similarity between Relations

Likewise, we also define the similarity between two relations as $sim_r(r_1, r_2)=1-d_r(r_1, r_2)$ and the distance between two relations is calculated by their respective positions in the relation hierarchy too. The only difference is that the relation hierarchy is constructed manually ourselves.

By adopting the similar method defined above to calculate the distance between two relations, we can compute the distance between two arbitrary relations theoretically. However, in practice, we think it worthless to assign a value to two arbitrary relations from query's perspective especially in a specific domain. Moreover

it will increase the computation complexity of our algorithm. So we simply define the similarity between two relations r_Q (the relation in query CG) and r_R (the relation in resource CG) as follows:

$$sim_r(r_Q, r_R) = 1 - d_r(r_Q, r_R) = \begin{cases} 1, & r_Q \text{ subsumes } r_R \\ 0, & \text{others} \end{cases}$$

That is to say, only when the relation in query CG is the supertype of the relation in resource CG, the similarity between these two relations is 1 and others will lead to 0. This definition is consistent with our original definition.

3.1.3 Similarity between CGs

Based on the similarity definition for concepts and relations, we can calculate the similarity between two CGs in the process of matching them. As shown in section 2, every resource CG has an entry of graph and the users are required to set the central word of query sentence which will then be mapped to the entry of query CG. These entries of CGs indicate the key concepts which the CGs describe and will be the entries of our matching algorithm. The matching process will begin with the entry and expand along the relations affiliated to it. Each relation affiliated to the entry induces a subgraph. We think the similarity between two CGs consists of the similarity between the two entries and the similarity between each subgraph pair. To reflect user's preferences on the importance of different similarity values, i.e. reflect which parts are more important and which are less important from user's view, we introduce the 'weight' on every 'entry' of graph and relations associated with it. More important part will have bigger weight value. Then where are these weights from? Within the UI, user can specify the preference information and the information will then be interpreted to the weights. The concept/relation similarity will be justified by the 'weight' in the CG matching process. However, this process is not mandatory, if user doesn't do it, the default weights value will be set and every part will be considered coordinate. Recursively, the similarity between any two subgraphs is also determined by the entries and their subgraphs according to their respective weights. The concept in the subgraph associated with the relation which induces the subgraph will serve as the entry of the subgraph. So the definition is recursive. The formula of similarity computation will be given as follows:

$$SoG(c_Q, c_R) = w(c_Q, c) \cdot sim_c(c_Q, c_R) + \max_{\substack{\text{for every} \\ \text{combination}}} \{ \sum_j w(c_Q, j) \cdot sim_r(r_Q^j, r_R^j) \cdot [SoG(c_Q^{r_Q^j}, c_R^{r_R^j})] \}$$

$$w(c_Q, c) + \sum_j w(c_Q, j) = 1 \text{ , for each subgraph with } c_Q \text{ as its entry}$$

Here, c_Q and c_R are the entries of query graph and resource graph respectively. $SoG(c_Q, c_R)$ represents the similarity between two CGs indicated by their entries.

The symbol r_Q^j (r_R^j) denotes the *jth* relation which associated with the entry c_Q (c_R) of the query (resource) CG. $c_Q^{r_Q^j}$ ($c_R^{r_R^j}$) is the entry of the subgraph which is induced by r_Q^j (r_R^j). The meaning of sim_c and sim_r are the same as the definition above. $w(c_Q, c)$ and $w(c_Q, j)$ represent the weights of the entry and the *jth* relation association with the entry respectively. To ensure the similarity between two graphs will not exceed 1, we normalize these weights. Moreover, for every graph (no matter the query CG or the resource CG), each relation associated with the entry will induce a subgraph and in theory a subgraph in query CG will be likely to mate any subgraph in resource CG. There exist many combinations among these subgraphs and we must find the best match from these candidate matches. So we choose the maximum similarity from different combinations as our result in every recursive process.

Look at the following example in figure 4. Suppose the left graph is a user query CG, and the right is a resource CG in our resource repository. *Garment* and *shirt* are entries for these two graphs respectively. Every relation induces a subgraph. For instance, *part* in the left induces a single node subgraph. *Pouch* will play the entry role when matching the subgraph. Before we can determine the best match of the two graphs, the similarity between each subgraph induced from *garment* in query CG and each from *shirt* in resource CG will be calculated respectively. Here what we need to do is to compute the similarity between concepts *pouch*, *sleeve*, and *pocket*, i.e. sim_c(pouch, pocket), sim_c(pouch, sleeve), sim_c(sleeve, pocket), and sim_c(sleeve, sleeve). As there are only two sorts of matches, the best is easily found, which is shown in figure 4 by dash lines.

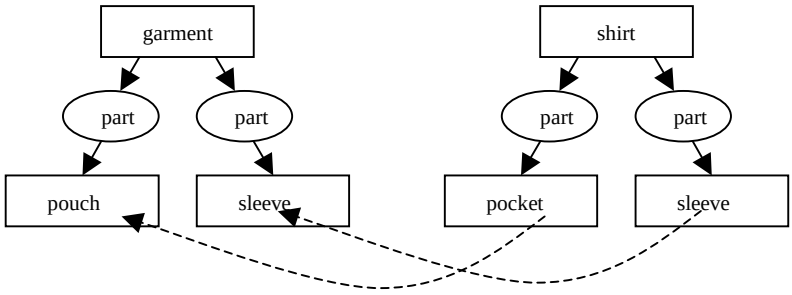


Fig. 4. The similarity between a query CG and a resource CG

3.2 Algorithm Implementation

Given a user query, the following algorithm will be performed to calculate the similarity between a resource CG and a query CG. One thing to remember is that the central word should be designated explicitly by user to indicate what s/he wants which then will be mapped to the entry of query CG.

```
1    get user query and the central word set by user.
2    parse the query and generate query CG using ALPHA.
3    get the entry of query graph E and locate it in WordNet
4    for (each resource CG indexed by E and its sub-concept in the
           domain ontology)
5    {    // the beginning of the recursive process
6     calculate the similarity between entry pair
7      For (each relation directly associated with entry in query CG)
8      {
9       For  (each  relation  directly  associated  with  entry  in
              resource CG)
10       {
11        calculate the similarity between these two relations and
          calculate the similarity between two subgraphs induced by
          the two relations recursively(line 5 to 15). Each time,
          The concept in current subgraph associated with the
          relation which induces the subgraph will serve as the
          entry of the subgraph.
12       }
13     }
14   find the best match from the above combinations of subgraphs
     using Bellman-Ford algorithm and sum up the similarity between
     entry pair and the similarity between each subgraph pair
     according to their respective weights as the similarity between
     the resource CG to the query CG
15   }    // the end of the recursive process
16   Rank the results and return answers back to user in proper order
```

Algorithm 1. Conceptual graph matching algorithm

Some details are explained as follows:

I) Every CG (query or resource) has an entry and these entries will be the entry of our matching process. We can theoretically calculate the similarity between two arbitrary CGs. But we think it is worthless in practice since the user would know exactly what s/he is interested in. So we only concern those resource CGs whose entries are the subclass of the entry of query CG.

II) Our algorithm will calculate the similarity between subgraphs recursively. But when the similarity between two relations equals 0, according to the formula defined in section 3.1.3, the calculation of two subgraphs extended by these relations is worthless. We will ignore these computations so as to make our algorithm more efficient.

III) Each relation associated with the entry induces a subgraph; these subgraphs will produce a lot of combinations. How to find the best match from these combinations is the linchpin of the complexity of our algorithm. We use the Bellman-Ford [12] algorithm to solve it. The complexity of the algorithm will be given and briefly illustrated in section 4.

IV) The query CG is dominant in our algorithm. The matching process will stop when all the relations and concepts in query CG have been checked. Relations in query graph that cannot find proper mate in resource graph will be calculated as if they are mapped to default relations in resource graph for we consider it as a kind of

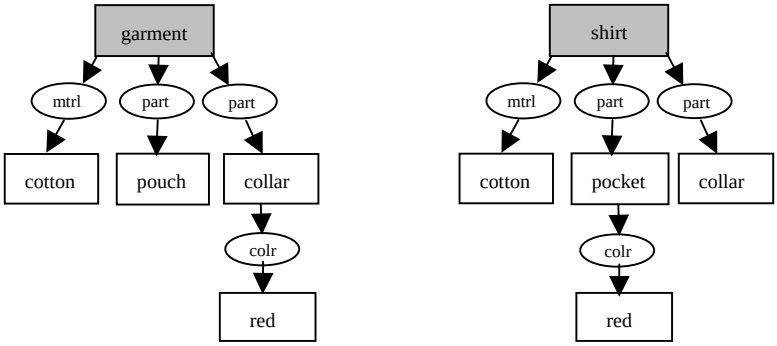
omission of default values. This rule is especially fit for those relations that represent the inherent attributes such as color, size and so on.

WordNet provides API to access senses and their hyponymy senses. After the entry is located, the above rules are activated to keep the consistency, and a recursive process is invoked to calculate the similarity between each subgraph pair and determine which is the best matching. The results will be ranked according to their similarity and displayed in an HTML page.

Here is an example to illustrate our algorithm.

Suppose the user enters the query sentence "*a cotton garment with a pouch and a red collar*" and sets the word '*garment*' as the central word. The query sentence will then be converted to the CG as (a) in figure 5 and the entry of this graph is shown as the grey node.

We locate the entry in the WordNet and find the appropriate resource CGs from our repository. Suppose one candidate resource CG is as (b) in figure 5 (the grey node represents the entry of graph; 'mtrl' is the abbr. for 'Material', while 'colr' for 'color').



(a) (b)

Fig 5. A query CG (a) and a resource CG (b)

In the following, we will calculate the similarity between the two graphs.

Firstly, we calculate the similarity between two entries. Since *shirt* is a subclass of *garment*, according to our definition, the similarity between two entries sim_c(garment, shirt)=1.

There are three relations associated with the entry in both the query CG and the resource CG. These relations will induce three subgraphs respectively and these subgraphs then produce nine subgraph pairs. We will calculate the similarity between these nine subgraph pairs. Before the calculation, we first give the values of the similarities between some concept pairs and relation pairs which will be used in the following calculations.

sim_c(pouch, pocket) =1;(since *pocket* is the subclass of *pouch*)
sim_c(pouch, collar) =0.7734;
sim_c(collar, pocket) =0.7696;
sim_r(mtrl, part) =0;
sim_r(part, mtrl) =0;

Besides, when the two relations or concepts are the same, the similarity will be 1 obviously.

$sim_r(R, R)=1$;
$sim_c(C, C)=1$;
Now, we begin to calculate the similarity between the subgraph pair as follows:
(In this example, we think these relations are coordinate and set the weights on all these three relations to 0.2. Then the weight on entry will be 0.4.)[3]

Consider the first relation (mtrl) in the query CG (which induces a subgraph with entry 'cotton')

Consider the first relation (mtrl) in the resource CG (which induces a subgraph with entry 'cotton'):

Firstly, calculate the similarity between these two relations: $sim_r(mtrl, mtrl)=1$;

Then, calculate the similarity between two subgraphs induced by these two relations.

Here, the two subgraphs are both ordinary and only contain one node. So the similarity between two subgraphs equals the similarity between two concepts.

$SoG(cotton, cotton)= sim_c(cotton, cotton)=1$

Consider the second relation (part I) in the resource CG (which induces a subgraph with entry 'pocket'):

Firstly, calculate the similarity between these two relations: $sim_r(mtrl, part)=0$;

Then, calculate the similarity between two subgraphs induced by these two relations.

Here, since the similarity between two relations is 0, according to the formula defined in section 3.1.3, the calculation of the two subgraphs is worthless and we will ignore it.

Consider the third relation (part II) in the resource CG (which induces a subgraph with entry 'collar'):

Firstly, calculate the similarity between these two relations: $sim_r(mtrl, part)=0$;

Then, calculate the similarity between two subgraphs induced by these two relations.

Here, we will ignore it because of the same reason as above.

Now, we gained three results from different pairs:

relation pair	**sim. of relations**	**sim. of subgraphs**	**weight**	**result**
mtrl-mtrl	1	1	0.2	0.2
mtrl-part I	0	/	0.2	0
mtrl-part II	0	/	0.2	0

Similarly, after considering the second relation (part I) in query CG, we can get the results as follows:

relation pair	**sim. of relations**	**sim. of subgraphs**	**weight**	**result**
part I-mtrl	0	/	0.2	0
part I-part I	1	1	0.2	0.2
part I-part II	1	0.7734	0.2	0.1547

Take notice of the relation pair part I-part I, the first part relation in query CG induces an ordinary subgraph which only contains one node 'pouch'. Though the first part relation in resource CG induces a subgraph which is not ordinary and will induce another subgraph further, the recursive process will not be invoked continuously because the query CG is dominant in our algorithm. So the similarity between these

[3] Here and the following example, the weights are all arbitrary and just an example to simplify the computation. In practice, weights will be set according to user's preferences.

two subgraphs equals the similarity between two concepts too. SoG(pouch, pocket)= sim_c(pouch, pocket)=1.

After processing the last relation (part II) in the query CG, the rest results will be gained.

relation pair	sim. of relations	sim. of subgraphs	weight	result
part II-mtrl	0	/	0.2	0
part II-part I	1	0.8618	0.2	0.1724
part II-part II	1	0.9969	0.2	0.1994

Notice that in the relation pair part II-part I, each subgraph in query and resource CG induced by these two relations is not ordinary and will induce subgraph further. In order to calculate the similarity between these two subgraphs the recursive process will be called. The work in the recursive process is similar to what we are describing. Moreover, in each graph, only one subgraph can be induced further, which makes the best match obviously. So we will not discuss the recursive process in detail again and only give the calculation here. The calculation process is as follows (in this recursive process, we set the weigh on the entry to 0.6, and the weight on relation 'colr' to 0.4):

SoG(collar, pocket)=0.6* sim_c(collar, pocket)+0.4* sim_r(colr, colr)* sim_c(red, red)
=0.6*0.7696+0.4*1*1=0.8618

As regards the relation pair part II-part II, what we need to indicate is that in the subgraph induced by the second 'part' relation in query CG, there exists another subgraph which can be induced by the 'colr' relation. But this subgraph can't find the proper mate in the corresponding subgraph induced by the second 'part' relation in resource CG to calculate. It seems that some information is missing in the resource CG. However, we don't simply handle it as mismatch. Since 'colr' represents the inherent attribute of any object in our domain, we think every collar will have this attribute. If in resource CG it isn't described explicitly, we will add a 'clor' relation automatically in program and set the concept 'color' (superclass of all concrete color such as red, blue etc.) as a default value for this attribute. Then we can calculate the similarity between these two subgraphs as alike as above (here, sim_c(red, **color**)=0.9922).

SoG(collar, collar)=0.6* sim_c(collar, collar)+0.4* sim_r(colr, **colr**)* sim_c(red, **color**)
=0.6*1+0.4*1*0.9922=0.9969

This process will be fit for other relations which represent the inherent attributes such as material, size and so on.

Up to now, we have obtained all similarities of the nine different subgraph pairs. What we need to do next is to choose the best match from the six different mate combinations. Here, Bellman-Ford algorithm will be employed to solve this problem. In this example, the best match is that mtrl-mtrl, part I-part I and part II-part II. This mate combination makes the similarity larger than any other combinations.

Finally, after finding the best match, we calculate the similarity between two graphs according to our formula. In this example, the similarity will be calculated as follows:

SoG(garment, shirt)=0.4* 1+0.2+0.2+0.1994=0.9994.

Now, we have processed one candidate CG completely. The rest of candidate graphs will be processed analogously. Eventually, the resource graphs will be ranked by their similarity and returned to user in descending order.

Currently a system implementing our method is under development with IBM China Research Lab.

4. Algorithm Evaluation

When applying graph matching algorithm, the greatest worry comes about the computation complexity, since it is well known that Maximum Subgraph Matching is a NP-complete problem. Fortunately, it can be expected in our algorithm that the computation complexity will be constrained to polynomial.

Before discussing the complexity of the algorithm, we firstly consider the effect caused by cycles in graphs to our algorithm. Since the algorithm is recursive, the cycle in graph will lead to an unending recursion and will be fatal to our algorithm. So we must eliminate the cycles in graphs before we match them. We can handle it simply by duplicating the concept in cycles. Surely, this will increase the computation complexity, especially when the cycle is very complex. Fortunately, benefiting from the domain specific characters, cycles in graphs are very rare especially in commodity domain. So we ignore it here.

In the following, we will discuss the complexity of our algorithm. Since cycles in graphs are very rare and the cycles can be eliminated simply, we will only concern the tree structure. Without losing generality, we can suppose that the query graph and the resource graph contain n arcs each and are both *l*-branch trees of *i* height, so there are more than l^i relations. We use C(*i*) to denote the time complexity of matching two trees both of *i* height. As shown in the algorithm, we will calculate the similarity between the two entries firstly (algorithm 1, line 6). We use a constant c to represent the time spent in calculating concept similarity. After this step, the time complexity is c; then we need to calculate the similarity between each subgraph pair. Since each entry will induce *l* subgraphs (line 7 and 9), we need l^2 times recursive invocations. These subgraphs are all *l*-branch trees of *i*-1 height, so in every invocation, the time complexity is C(*i*-1) (line 11). Here we ignore the time to calculate similarity between relations. After these two loops, the time complexity will be $c+l^2*C(i\text{-}1)$. Once we determine the similarity between each subgraph pair, we should find out the best match from different mate combinations (line 14). There exists *l*! combinations in these l^2 subgraph pairs, so how to handle it efficiently is important. We translate the issue into a *maximum flow* problem and execute Bellman-Ford algorithm *l* times to solve it[4], whose computation complexity is l^3, and the cumulative complexity is l^4. So the complexity can be described as follows:

$$\begin{cases} C(i+1) = l^2 C(i) + l^4 + c & i = 0,1,2,\ldots \\ C(0) = c \end{cases}$$

From the formula, we can see that C(*i*) is about l^{2i+2}. Generally, when *l* is not very small, the number of arcs n will approximate l^i, so the complexity will be $n^2 l^2$. If $l << n$, the complexity will be $O(n^2)$. For the worst case, suppose there is only 1 layer in the query graph, i.e. $l=n$, the complexity is $O(n^4)$.

Since the algorithm combines syntactic and semantic context information in the whole process, the advantages over traditional keyword match technique can easily be seen. For example, a description is about 'soft collar shirt' and another is about 'soft shirt with straight collar'. They are both selected by keyword search when using 'soft

[4] In fact, before every invocation of the B-F algorithm, some additional work will be done. We will not discuss them in details for brevity.

collar shirt' although 'soft' modifies different parts in the two sentences. While in our method, the former will be more similar to user's query through analyzing the object which 'soft' modifies. We believe that this will improve the precision. For another case, 'slim waistband' and 'narrow waistband' can be both used to depict a same kind of skirt. Some simple keyword search methods can't pick them out and some other keyword search methods that use improved techniques (such as cluster, etc) will pick them out without any other information. However, in our approach, we can not only pick them out, but also calculate the similarity between query CG and resource CG according to our ontologies and indicate the better, which is beyond the ability of traditional keyword search methods. This will improve the recall of the search.

What is likely to be suspected is that our system requires users to give the central word of graph in their queries. A query like "find a red thing" will lead to unpredictable results. In practice, however, it is reasonable to ask the user to specify the entry since s/he would know exactly what s/he is interested in. Moreover, some simple techniques, such as the 'shallow parsing' [13] could help to find out the implied 'headword'. This is beyond our discussion and will be omitted here.

Since graph matching cannot do any inference, some deeper information will not be available, which limits the utility of our approach.

5. Related work

Semantic matching has been raised for years to improve both recall and precision of information retrieval. SCORE[4] finds first the most similar entities and then observe the correspondence of involved relationship. However, with this kind of simplification, matching on nodes is separate without the organization of subgraphs. In contrary, we try to retain subgraph structure in our matching procedure with as less cost as possible. OntoSeek [3] defines the match on isomorphism between query graph and a subgraph of resource graph where the labels of resource graph should be subsumed by the corresponding ones of query graph. The strict definition of match makes their system can't support partial matching. The assumption that user would encode the resource descriptions by hand also limits its popularization. Different from it, our method not only supports the partial matching but also introduces the weight to reflect user's preferences, which makes our method more flexible.

Some previous work have discussed the issue of semantic distance, such as [5] [14] and [15]. The basic thought is to define the distance between two concepts as the number of arcs in the shortest path between two concepts in the concept hierarchy which does not pass through the absurd type. [14] modified the definition and defined the distance as the sum of the distances from each concept to the least concept which subsumes the two given concepts. We adopt their original thought and make some modifications to make it suitable to our work.

The measurement of concept similarity was also studied before. [7] builds their similarity definition on the information interests shared by different concepts, while [16] defines the similarity between two concepts as the information content (entropy) of their closest common parent, and besides take the density in different parts of the ontology into account. The measuring of concept similarity in our approach is

different from them and is simpler. Of course, our approach is far from perfect. It needs further study based on collected experiment data in the future.

Reference

1 L.Page, S.Brin, R.Motwani, and T.Winograd.: The PageRank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998. Available at http://www-db.stanford.edu/~backrub/pageranksub.ps

2 Lum et.al.: An architecture for a multimedia DBMS supporting content search. In the Proceedings of International Conference on Computing and Information (ICCI'90), LNCS Vol.468, Springer-Verlag, 1990.

3 N. Guarino, C. Masolo, and G. Vetere.: OntoSeek: Content-Based Access to the Web. IEEE Intelligent Systems, 14(3), pp.70--80.

4 Y. A. Aslandogan, C. Thier, C. T. Yu, C. Liu, and K. R. Nair.: Design, implementation and evaluation of SCORE (a System for COntent based REtrieval of pictures). In Eleventh International Conference on Data Engineering, pages 280--287, Taipei, Taiwan, March 1995

5 J. F. Sowa.: Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley. 1984.

6 Lei Zhang and Yong Yu.: Learning to Generate CGs from Domain Specific Sentences. In proceeding of the 9th International Conference on Conceptual Structures, (ICCS2001), LNAI Vol.2120, Springer-Verlag, 2001.

7 Jonathan Poole and J. A. Campbell.: A Novel Algorithm for Matching Conceptual and Related Graphs. In G. Ellis et al eds, Conceptual Structures: Applications, Implementation and Theory, pp. 293--307, Santa Cruz, CA, USA. Springer-Verlag, LNAI 954, 1995.

8 George A.Miller.: WordNet: An On-line Lexical Database. In the International Journal of Lexicography, Vol.3, No.4, 1990.

9 John F. Sowa.: Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, 1999.

10 N. Kushmerick, Daniel S. Weld and Robert B. Doorenbos.: Wrapper Induction for Information Extraction. Intl. Joint Conference on Artificial Intelligence pp.729--737

11 Jianming Li, Lei Zhang and Yong Yu.: Learning to Generate Semantic Annotation for Domain Specific Sentences. In the Workshop on Knowledge Markup and Semantic Annotation, the First International Conference on Knowledge Capture (K-CAP 2001), Victoria B.C., Canada, Oct.2001.

12 T.H.Cormen, C.E.Leiserson and R.L.Rivest.: Introduction to Algorithms. The MIT Press, 1994.

13 W. Daelemans, S. Buchholz, and J. Veenstra.: Memory-Based Shallow Parsing. In Proceedings of EMNLP/VLC-99, pages 239-246, University of Maryland, USA, June 1999

14 Norman Foo, B. Garner, E. Tsui and A. Rao.: Semantic Distance in Conceptual Graphs. In J. Nagle and T. Nagle, editors, Fourth Annual Workshop on Conceptual Structures, 1989

15 A. Ralescu and A. Fadlalla.: The issue of semantic distance in knowledge representation with conceptual graphs. In Proceedings of Fifth Annual Workshop on Conceptual Structures, pages 141--142, 1990.

16 R. Richardson, A. F. Smeaton and J. Murphy.: Using WordNet as a Knowledge Base for Measuring Semantic Similarity between Words. In the Proceedings of AICS Conference, Trinity College, Dublin, Ireland, September 1994.