# A Semantic Search Approach by Graph Matching with Negations and Inferences

Kewei Tu, Jing Lu, Haiping Zhu, Guowei Liu, and Yong Yu

Department of Computer Science and Engineering,
Shanghai JiaoTong University, Shanghai, 200030, P.R.China
tukw@sjtu.edu.cn, robert_lu@263.net, hpzhu@sjtu.edu.cn,
liugw2000@163.net, yyu@mail.sjtu.edu.cn

**Abstract.** Research on semantic search has become heated these years. In this paper we propose an approach focusing on searching for resources with descriptions. The knowledge representation we employ is based on conceptual graphs and is expressive with negation. We carry out semantic search by graph matching, which can be performed in polynomial time. Before matching we enrich the resource graphs with background knowledge by a deductive graph inference, so as to improve the search performance. The processing of negations and the graph inference method are two important contributions of this paper.

## 1 Introduction

Today more and more information has been provided in the Internet. One could find almost anything he is interested in. However, more and more time has to be spent in filtering irrelevant information when searching for that something. Complex searching techniques have been employed in today's keyword based search engines. With the goal of further improving search accuracy and efficiency, semantic search based on structured data representation has been intensively studied these years. Many semantic search techniques have been proposed, and some search engines have been developed, including OntoSeek[1], WebKB-2[2], etc.

We have also been devoted to this research area for years, focusing on searching for resources identified by descriptions. The resource or query description is often ad hoc, that is, there are often various descriptions for the same object. And in many cases the description is complicated. So precise search may not perform well. In our approach, we use a restricted but still expressive conceptual graph notation to represent the descriptions and carry out semantic search by graph matching. And before matching we enrich the resource graphs with background knowledge by deductive inference, so as to improve the search performance. We have made a primary study on semantic matching in the past. [3] introduced the matching method whose representation is non-nested with no negation. Although we now choose the clothing domain to demonstrate our approach, the method itself is domain independent and can be applied to other domains.

The rest of this paper is organized as follows. Sect.2 overviews our whole architecture for semantic search. Sect.3 describes the representation we employ. Sect.4 presents our knowledge enrichment by inference. Sect.5 introduces in detail the semantic matching. Sect.6 compares our work with related work. Sect.7 makes a conclusion in the end.

## 2 Overview of the Approach

The whole architecture of our approach is shown in Fig.1. Before performing the semantic search, we gather domain resources from the web. Then the descriptions of the resources will be extracted and converted to our representation based on conceptual graphs (using the resource converter in Fig.1). Method to convert various kinds of resources depends on the original representation of the resources. In our previous work, we have developed a system converting resources described in natural language to conceptual graphs by a machine learning approach[4]. With some improvements it can be used here as a kind of converter. After the conversion, the resource graphs will be stored into our resource repository. When the enquiry is entered, it will also be translated into our representation by converters. We use the 'entry' concept of graph, which is introduced in Sect.3, to organize and manage the resource repository and to pilot the query process.

From the view of the query handler module, the input consists of the query graph and the candidate graphs fetched from the resource repository according to the entry of the query graph, and the output is the consensus ranking of the

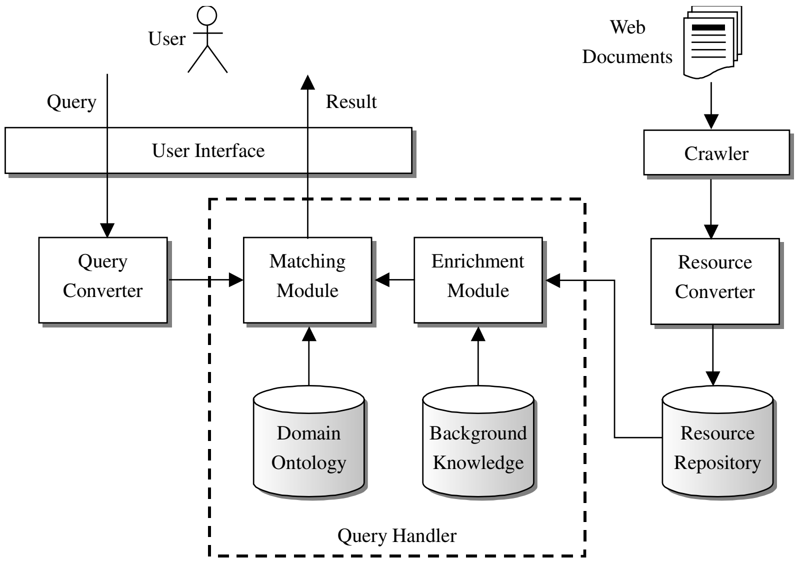


**Fig. 1.** Overview of the approach

candidates. The ordered answers out of these candidates will be returned to the user.

The query handler module contains two parts: enrichment module and matching module. When a query begins, first the enrichment module will work on the candidate resource graphs by means of inference, so as to gain more information and make the matching module perform better. A background knowledge base will provide the rules needed by the inference. Then the enriched resource graph and the query graph will be matched in the matching module, supported by domain ontology. We now use WordNet[5] as the ontology. In the future we may use more specialized domain ontology instead of it.

The details will be introduced in the following sections, including our representation, the enrichment module and the matching module.

## 3 Representation

In our approach, we adopt a subset of the classical conceptual graph notation[12]. The subset still holds enough ability to represent negative descriptions as well as assertions.

As presented in [6, 12], the conjunction of two graphs is represented simply by drawing both of them on the same sheet of paper; negation is represented by a negative proposition concept partitioning the negative proposition from the outer context. Since all Boolean operators can be reduced to negation and conjunction, they can be represented in terms of nested negative contexts. And with coreference link, universal quantifier can be represented.

To support conceptual graph matching in our approach, we made the following restrictions on conceptual graph representation:

– As each graph represents a resource object in our approach, we request that every graph has an entry concept, which is the center of the graph designating the represented object and serves as the start position in matching the graph. (See [3] for detailed discussion on the definition and feasibility of entry) The entry of a graph must be in the outermost context, i.e. it can't be in any negative context. Every concept node in the graph should be able to be accessed from the entry through directed relations and undirected coreference links. We mark the entry of a graph when the graph is generated from the description.
– Only coreference links can cross context boundaries, while arcs of relations can't, as demanded in the classical conceptual graph notation. Our restriction is for every context boundary there must be one and only one coreference link that crosses it. This restriction actually guarantees every negative context links to an outer context while avoids some intractable but rare situations such as graphs with coreference link cycles across context boundaries.

These restrictions seem reasonable, as each graph is to describe one object. In our practice, most descriptions can be represented in this way. So we believe our representation is expressive enough for semantic search.

Fig.2 is an example of the representation in our approach.

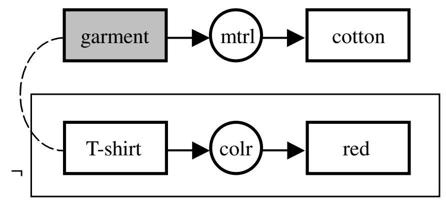


**Fig. 2.** Conceptual graph for *"a cotton garment that is not a red T-shirt"* (The grey node marks the entry of the graph)

## 4 Knowledge Enrichment by Inference

In this section, we will introduce the knowledge enrichment phase of our approach. The enrichment is implemented by inference. We will first explain why this phase is needed for semantic search. Then our deductive inference mechanism is formally presented and the enrichment procedure is shown.

### 4.1 Reasons for the Enrichment

The purpose of knowledge enrichment is to add to the resource conceptual graphs some information that is implicitly contained, so as to make more precise the semantic matching introduced in Sect.5. Properly speaking, there are two main reasons for the enrichment. One is the non-isomorphism phenomenon. The other is inferring new knowledge by using world knowledge.

Non-isomorphism means that one meaning has more than one representation. It is very common in practice. We often have many ways to express the same idea in natural language. When we represent knowledge formally, non-isomorphism still remains. We conclude several familiar kinds of non-isomorphism here:

- **POS.** There are lots of interchangeable uses of different POS forms belonging to one same word. For example, "3 button" and "3 buttoned", "A-line" and "A-lined", etc. In conceptual graph they are interpreted to different concepts with corresponding different relations.
- **Negation.** The meaning of negation can be either implicitly held within one single word or explicitly described using the negative word "not". Take the following as examples: "unlined" and "not lined", "short" and "not long", etc. They are certainly represented differently.
- **Converse Relations.** Expressions appear different when using converse relations. Say, "pocket 1 is to the left of pocket 2" and "pocket 2 is to the right of pocket 1" are differently expressed, but the converse relation pair leads to their same sense.
- **Definition.** A concept in a graph can be substituted by its definition graph without changing the meaning.

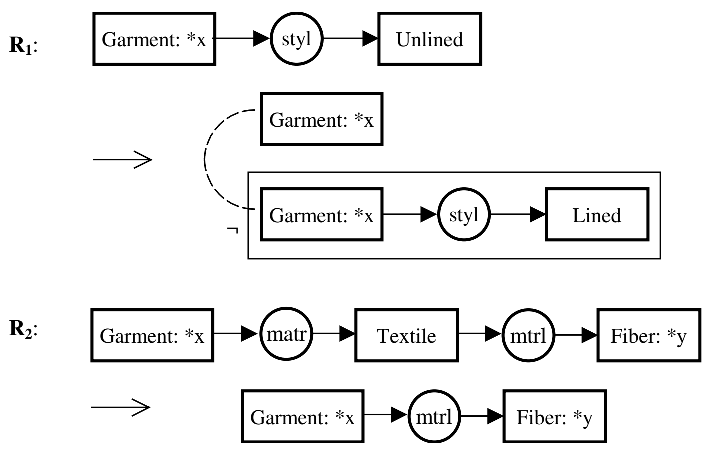
The second reason for the enrichment is to infer new knowledge. With world knowledge we can often gain new information from known descriptions. For example, if we know a garment is “stone washed” then it is “prelaundered”. New relations can also be introduced. Suppose we get the information that “shirt” is made of “denim” and “denim” is made from “cotton”, then we will most probably deduce that this “shirt” is made from “cotton”.

## 4.2 The Deductive Inference

First we define the graph rules, which represent the background knowledge.

**Definition 1.** *A rule $R$ is a graph: $G_1 \rightarrow G_2$, where $G_1$ and $G_2$ are two conceptual graphs with coreference link(s) between them. The two concepts linked by a coreference link must be of the same type and must occur in the outermost context. We can replace the coreference link by adding two identical variables as the referents of the two linked concepts. We call $G_1$ and $G_2$ the hypothesis and conclusion of the rule respectively.*

The following two graphs are rules representing two examples mentioned in Sect.4.1 (‘styl’ is the abbr. for ‘style’, ‘matr’ for ‘matter’, and ‘mtrl’ for ‘material’):



Notice that rules for non-isomorphism should use mutual implication connector instead of implication connector, that is to say, if there is a rule $R_1$ for non-isomorphism: $G_1 \rightarrow G_2$, there should be a rule $R_2 : G_2 \rightarrow G_1$.

What does a rule mean? We can represent a rule $R$ in conceptual graph informally with universal quantifier: $\forall x_1, x_2, \ldots, x_n, \neg [G_1 \neg [G_2]]$, where for every $i = 1, 2, \ldots, n$, there are two concepts in $G_1$ and $G_2$ respectively, with $x_i$ as the referent and a coreference link between them.

Now we can define the rule of inference.

**Definition 2.** ***Rule of Inference.*** *If there is a conceptual graph $G$ and a rule $R : G_1 \rightarrow G_2$, and there is a mapping $\Pi$ from $G_1$ to $G$, then we can get a new* *graph. This new graph $G'$ is derived from $G$ and $G_2$ by inserting $G_2$ to the same context as $\Pi(G_1)$ and merging each concept $c_2$ of $G_2$ with $\Pi(c_1)$. Here $c_1$ and $c_2$ denote two concepts in $G_1$ and $G_2$ linked by a coreference link, and $\Pi(G_1)$ and $\Pi(c_1)$ respectively denote the image of $G_1$ and the image concept of $c_1$ in $G$. There are some restrictions on the mapping $\Pi$ from $G_1$ to $G$. If in $G_1$ the dominant concept of $c$ is in an evenly enclosed context, then $\Pi(c)$ in $G$ must be a specialization of $c$; otherwise if the dominant concept of $c$ is in an oddly enclosed context, then $\Pi(c)$ must be a generalization of $c$.*

Here is an example illustrating the application of this rule of inference. From the ontology we know 'shirt' is a specialization of 'garment', 'denim' is a specialization of 'textile', and 'cotton' is a specialization of 'fiber'. Therefore if there are the rules $R_1$ and $R_2$ shown above and a conceptual graph $G$ (Fig.3a), then we can get a new conceptual graph $G'$(Fig.3b) by applying the rule of inference twice.

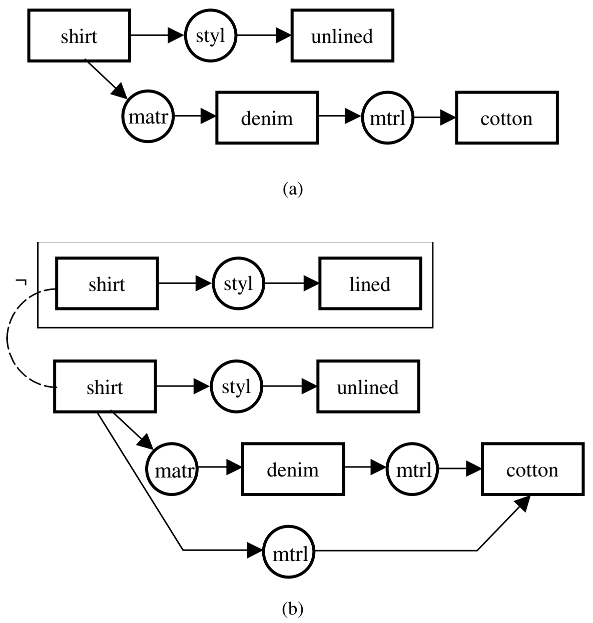


**Fig. 3.** An example of inference

This rule of inference is sound, or truth-preserving, i.e., if the graph rule and the resource graph are both true, then the result graph is also true. We now give a brief proof by using the first-order inference rules of conceptual graphs [6]. Remember that the rule $R : G_1 \rightarrow G_2$ is universally quantified. By universal instantiation, we can link the universally quantified concepts in $R$ to the corresponding ones in $G$ with coreference links. Because $G_1$ is enclosed at depth

1 when represented in conceptual graph, by erasure rule and insertion rule $G_1$ can be rewritten to $\Pi(G_1)$. By iteration rule we copy the new form of the rule to the context of $\Pi(G_1)$ in $G$. Now we have $[\neg\,[\Pi(G_1)\,\neg\,[G_2]]\ \Pi(G_1)]$ in some context $C$ of $G$. Then by applying deiteration rule and double negation rule we get $[\Pi(G_1)\,G_2]$ in $C$. The coreference links are reserved in the above procedure. So after applying coreference join rule we finally get $G'$ described in Definition 2. All the inference rules used above are sound, so our inference rule is sound too.

On the other hand, this inference is obviously not complete. But the aim of our inference is not to get everything that is true but to make the matching more precise. In fact, we only make the inference for a few steps, as described in the next section. So completeness is not necessary for our approach.

It should be noted that our inference might weaken the description of the resource conceptual graph, instead of restricting it. The simplest case is that, from the resource graph $\neg G_1$ and a rule $G_1 \rightarrow G_2$ with no universally quantified variable, we can get a new graph $\neg\,[G_1\,G_2]$, which is weaker in sense than the resource conceptual graph and can be inferred directly from the resource graph without knowing the background knowledge, i.e., the rule. To be more precise, suppose $G_1$ is the hypothesis of the rule and $G$ is the resource graph, the result conceptual graph may be weakened if $\Pi(G_1)$ is in an oddly enclosed context of $G$, and it may be restricted if $\Pi(G_1)$ is in an evenly enclosed context. As shown in the next section, this will affect our use of the deductive inference.

### 4.3 Knowledge Enrichment Procedure

In our approach, the function of knowledge enrichment is to add background knowledge to the resource graphs. In terms of model theory, this procedure is to restrict the resource representation so as to exclude those models violating the background knowledge represented by the rules. The ideal restricted resource graph is the direct conjunction of the origin graph and the rule, which unfortunately does not meet our representation requirements presented in Sect.3 in some conditions and therefore can't be processed by our matching approach. So we make a compromise between restriction and processability to use the deductive inference introduced in the last section. As we have seen, the resource graph is restricted only if the projection of the rule hypothesis is in an evenly enclosed context of the resource graph. Therefore only in that condition will we do the inference. The entire procedure is described below:

All the rules are stored in a rule base. When a resource conceptual graph needs to be enriched by inference, we will try every rule in the rule base to see whether the hypothesis of the rule and the resource conceptual graph meet the requirements of the inference rule and the forenamed constraint. If so, the deductive approach introduced above will be carried out and new information will be obtained. Before being added to the resource conceptual graph, the new information must be checked so that it won't be a copy of existing information in the resource conceptual graph. We call the last step redundancy detecting.

Such a process will be performed iteratively until no new information could be added or the limit of iteration times is reached. This limit is tunable according to the time and storage limitation. It also prevents the iteration from endlessly going on because of the inference loop, which is caused by such kind of rules that after applying one or more of them serially we will get some non-redundant information which happens to be the hypothesis of the first rule. Notice that such loop can't be prevented by the redundancy-detecting step because the new information gained is actually new.

## 5 Graph Matching with Negations

This section will explain our matching approach on graphs with negations. We have made a primary study on semantic matching in the past [3]. But with introduction of negations, there are some critical changes in the matching approach. Also there are some other improvements. We will first illustrate how to convert graphs to trees before matching. Then the matching similarity is defined. At last the algorithm is given and evaluated.

### 5.1 Conversion of the Graphs

Notice that in our approach every graph has an entry which marks the concept described by the whole graph. The entry can't be placed in any negated context and can access every concept of the graph. So we can convert a graph to a tree with the entry as the root. Then matching can be performed on trees instead of graphs.

The conversion is performed by using the entry as the root of the tree and then extending the tree in directed relations and undirected coreference links. Fig.4 is a simple example illustrating the process.

First, notice that the node "button" is duplicated in the resulted tree. This is because there are two possible paths from the entry to the node "button". So the result tree may have more nodes than the original graph. The same node can't be added twice to one path of the tree, so as to break the cycle in the graph.

Second, in the result tree there are two types of links between the concept nodes. One is relation links (solid arrows), which have the same meaning as in a conceptual graph. The other (dashed arrows) is used to represent the coreference links in the original graph. For each group of coreferenced nodes in the graph, there will be a "dummy" node in the result tree, and these nodes will be linked to that dummy node with dashed arrows. This dummy node can be viewed as the individual thing all these coreferenced concept nodes represent. Actually each concept node in the graph without coreference link can also be converted in the tree a concept node with its dummy node linked to it.

Third, the negation symbol on the context "T-shirt colored red" in the original graph is converted to the negation symbol on the node "T-shirt" only. But in fact the negation symbol in the tree means not the negation on the concept

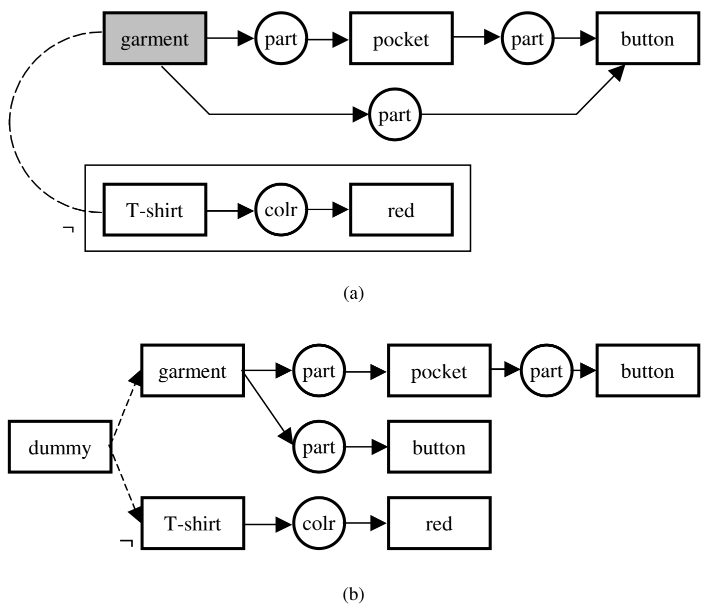


**Fig. 4.** An example of converting a graph (a) into a tree (b) (The grey node of graph (a) marks the entry. ‘colr’ is the abbr. for ‘color’.)

node “T-shirt”, but the negation on the whole subtree rooted on that node. As in Sect.3 we have made restrictions on negations and coreference links in our graphs, this conversion is always sound.

## 5.2 Similarity Definition

Now we can define the similarity between the query graph and the resource graph. The similarity between graphs is based on the similarity between concepts and relations, which is mainly supported by the domain ontology. As the concept and relation similarity definition used here is the same as in our previous work [3], here we will only focus on the similarity definition between graphs.

In the following discussion, we will use the function $SIM_C(C_Q, C_R)$ to represent the similarity between two concepts, the function $SIM_R(R_Q, R_R)$ to represent the similarity between two relations, and the function $SIM_T(T_Q, T_R)$ to represent the similarity between two subtrees. The subscript $Q$ or $R$ declares the item is from the query or resource graph respectively. The order of parameters in these functions could not be changed, that is, the first parameter must be query item and the second must be resource item. This is because the similarity defined here is not purely tree similarity but the extent to which the resource satisfies the query. The function $W(T_Q)$ represents the weight value of the subtree in the query tree and will be defined later. It represents the importance of the subtree.

The similarity between the query and resource trees or subtrees could be recursively defined as follows, with $SIM_C$ and $SIM_R$ defined in [3]:

**Definition 3.** $SIM_T(T_Q, T_R) =$

$$\begin{cases} \dfrac{\sum_{i=1}^{Num(C_Q)} SIM_T(T_Q^i, T_R) W(T_Q^i)}{\sum_{i=1}^{Num(C_Q)} W(T_Q^i)} & (C_Q = dummy) \\ \dfrac{\sum_{i=1}^{Num(C_R)} SIM_T(T_Q, T_R^i)}{Num(C_R)} & (C_Q \neq dummy, C_R = dummy) \\ \dfrac{P\left( \underset{A(i)}{Max} \sum_{i=1}^{Num(C_Q)} SIM_T(T_Q^i, T_R^{A(i)})\, SIM_R(R_Q^i, R_R^{A(i)}) W(T_Q^i) \right) + SIM_C(C_Q, C_R)}{P\left( \sum_{i=1}^{Num(C_Q)} W(T_Q^i) \right) + 1} & (C_Q \neq dummy, C_R \neq dummy) \end{cases}$$

In the definition, $T$ with subscript represents a tree or subtree, and $C$ with the same subscript represents the root node of $T$. $T^i$ represents the subtree rooted on the $i$-th child node of the root of $T$, and $R^i$ represents the $i$-th relation of the root of $T$. $Num(C)$ represents the number of child nodes of the root node $C$. $A(i)$ is an injection from the subtree labels of the query tree to the subtree labels of the resource tree. If the subtree $T^i$ in the query tree can't find a corresponding subtree in the resource tree, then $A(i)$ is assigned a special value so that the $SIM$ function with $T^{A(i)}$ or $R^{A(i)}$ as the parameter will return 0. $P$ is a discount factor between 0 and 1. The function $W(T_Q)$ represents the weight value of the subtree in the query tree and will be defined later. It represents the importance of the subtree.

The first two formulas in the definition describe how to calculate the similarity while there's dummy node. If the dummy node is the root of the query tree, then the similarity is the weighted average of the similarities of its subtrees with the resource tree. If the dummy node is the root of the resource tree, as there is no weight definition for the resource tree, the result equals to the average of similarities of its subtrees with the query tree. The third formula calculates the similarity between two normal nodes. The main calculation is to find the best possible matching between the query subtrees and the resource subtrees by the Max function. The result similarity is the weighted average of the similarities of subtrees and that of the roots.

In the formula, $P$ is a constant between 0 and 1, which determines how fast the weight of nodes decreases with the depth of the tree. If $P = 1$, all nodes in the query are as important as each other. On the contrary, if $P = 0$, only the entry of the query is considered useful. In our approach, we use $P = 0.5$ by default.

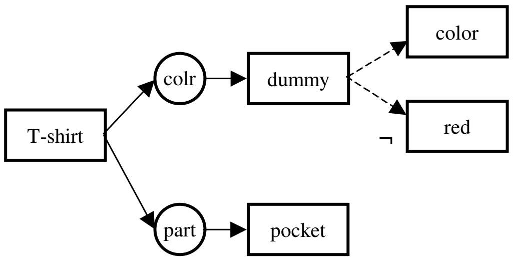
$W(T_Q)$ represents the importance to the user of the description that is represented by the subtree $T_Q$. If the user describes one of the attributes at length, then that attribute is likely of more importance. So we should add more weight to the subtrees that have more nodes than others. Therefore $W(T_Q)$ is defined as the following:

$$W\left(T_Q\right)=\begin{cases}\sum_{i=1}^{Num(C_Q)} W\left(T_Q^i\right) & (C_Q=\text{dummy})\\ P\left(\sum_{i=1}^{Num(C_Q)} W\left(T_Q^i\right)\right)+1 & (C_Q\neq\text{dummy})\end{cases}$$

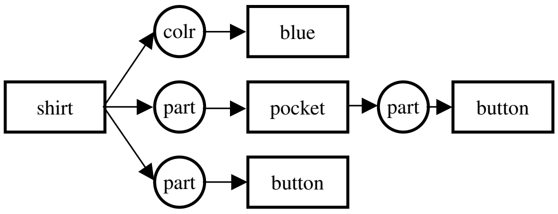
Users can also modify this function by pointing out the important attributes.

How to calculate the similarity when negations are concerned? Actually, if exactly one of the $T_Q$ and $T_R$ has a negation symbol on the root, the result similarity will be one minus the original similarity of the two trees.

Here is an example showing how the similarity is calculated.



(a) The query tree: a T-shirt which has a pocket and is not red



(b) The resource tree: a blue shirt with a buttoned pocket

**Fig. 5.** An example of mathcing

Before calculating the similarity, we should first calculate the default weight of each subtree in the query tree. According to the formula above, the weights of "color", "blue" and "pocket" are 1, the weight of subtree rooted on "dummy" is 2, and therefore the weight of the whole tree is 2.5.

To calculate the similarity between "T-shirt" and "shirt", the algorithm will first try to match their child nodes. In theory, six similarities (two children in the query match three children in the resource) should be calculated. Because similarity between different relations is zero, in fact only similarities between "dummy" and "blue", "pocket" and "pocket" and "pocket" and "button" are calculated.

The similarity between "dummy" and "blue" is the weighted average of the similarity between "color" and "blue" and the similarity between "not red" and "blue". "Blue" is a kind of "color" and "blue" is in the resource, so the sub-query "color" is satisfied and the similarity is 1. Assume the similarity between "red" and "blue" is 0.2, so the similarity between "not red" and "blue" is 0.8. We now get the similarity between "dummy" and "blue" to be 0.9. On the other side, the similarity between "pocket" and "pocket" equals to 1. As the matching is based on the query, the "button" linked to "pocket" in the resource is ignored. The similarity between "pocket" and "button" is small. So the best matching is "blue" to "dummy" and "pocket" to "pocket".

Finally, although "shirt" is the super type of "T-shirt", it is in the resource. So we assume the similarity between "T-shirt" and "shirt" is 0.7. By the formula we get the final result 0.84.

## 5.3 Matching Algorithm and Its Evaluation

As described above, our matching method has two steps. The first is to convert the query and resource graphs into trees. The algorithm is similar to depth-first graph traversal. The second is to calculate the similarity of two trees. As the similarity is recursively defined, we use recursion to carry out this step. In matching every pair of nodes, the min cost max flow algorithm is used to get the best match of their child nodes. The evaluation of the time complexity of our algorithm is given below.

The first step is to convert a graph into a tree. In practice, the number of nodes of the generated tree is just a little larger than that of the graph. So the time complexity is linear in the size of the graph in most cases.

The second step is to match two trees. Assume the height of both trees is no larger than $h$, and there are at most $n$ nodes in one level of a tree. While the min cost max flow algorithm is used for matching, the time complexity is $O\left((a+b)^3 \min(a,b)\right)$, where $a$ and $b$ denote the node number in the two groups for matching. In each level of a tree, we can group the nodes by their parents. Suppose in query tree the number of nodes in each group is $a_1, a_2, \ldots, a_q$ respectively, and in resource tree the number of nodes in each group is $b_1, b_2, \ldots, b_r$ respectively. Then we can calculate the time complexity of the matching in each level of the trees as the following:

$$\begin{aligned}
T &= \sum_{i=1}^{q} \sum_{j=1}^{r} \Big( (a_i + b_j)^3 \min(a_i, b_j) \Big) \\
&= \sum_{i=1}^{q} \sum_{j=1}^{r} \left( \left(a_i^3 + 3a_i^2 b_j + 3a_i b_j^2 + b_j^3\right) \min(a_i, b_j) \right) \\
&\le \sum_{i=1}^{q} \sum_{j=1}^{r} \left( \left(a_i^3 + 3a_i^2 b_j\right) b_j + \left(3a_i b_j^2 + b_j^3\right) a_i \right) \\
&= \sum_{i=1}^{q} \sum_{j=1}^{r} \left( a_i^3 b_j + 6a_i^2 b_j^2 + a_i b_j^3 \right)
\end{aligned}$$

$$< \left( \sum_{i=1}^{q} a_i + \sum_{j=1}^{r} b_j \right)^4$$
$$\leq (2n)^4$$

So, the actual time complexity of the whole algorithm is $O\left(n^4 h\right)$. In practice, most graphs have no more than a few tens of nodes, so we think this time complexity is acceptable.

## 6 Related Work

Semantic search has been studied for years to improve both recall and precision of information retrieval. Most projects emphasize on selecting from semantic data source the information which perfectly fits the query. In contrast, we use a graph matching approach to carry out imprecise semantic search. Some projects, such as OntoSeek [1], also employ graph matching methods. Our difference from these ones is discussed in the third paragraph. Another feature of our approach is the use of inference to introduce the background knowledge. While many projects have not employed inference, some ones just use limited inference to support query broadening or relaxation, such as *DAML Semantic Search Service* [7]. OntoBroker [8], with SiLRI [9] as its inference engine, employs additional rules expressed in F-logic to provide information about relationships between predicates, therefore it has the capacity to get information by inference which is not stated explicitly in the source structural data. Our approach also employs similar additional rules. Where it differs from OntoBroker is that we query by means of semantic matching. As a result, our approach has to adopt data-driven inference (or forward inference) so that after inference our matching is meaningful.

Sound and complete conceptual graph inference has long been well-established [6, 12]. But we only use a sound and non-complete inference in order to simplify the implementation and make the approach more efficient. The forward chaining inference discussed in [10] is similar to ours. It can be viewed as a simplified version of our inference, for it only deals with simple conceptual graphs which are non-nested with no negation. Therefore its inference is complete. But the expressive power of simple conceptual graphs is too limited, so while reserving the simplicity of the inference, we sacrifice the completeness, which is not important to our method.

Some previoufs work has studied the issue of semantic matching. OntoSeek [1] fetches corresponding nodes first and then checks the arc linkage between them, so as to avoid NP-completeness of such a computation known as Maximum Subgraph Matching. We think such simplification separate matching on nodes from the organization of the graph. So we try to retain graph structure in our similarity definition but confine comparison range to mitigate the computation. Cupid [11] matches on bottom-up traversal and biases matches of schema leaves to perform schema mapping. We share the idea of bottom-up traversal with Cupid, but we prefer matches of the entry. Moreover, like most semantic matching researches, neither of the above projects has the ability to represent and match negation, which is an important feature of our approach.

## 7 Conclusion and Future Work

In this paper we introduced a semantic search approach by graph matching. Our knowledge representation is based on conceptual graphs with some restrictions, while it is still expressive for semantic search. Before matching we enrich the resource graphs with linguistic and world knowledge by a means based on a formal deductive inference. The semantic matching method calculates the similarity between the query graph and the resource graph in polynomial time. At last the candidate resource graphs are ranked and the best corresponding answers are output. A prototype of our approach is currently under development.

There is still a lot of possible work that remains to be done. Various methods to convert resource and query to our representation could be developed. The rule base used in inference could be automatically built by means of DM methods. Confidence of rules could be introduced to the inference approach. Matching of different levels of the tree structure is also an interesting idea.

## References

1. N. Guarino, C. Masolo, and G. Vetere: OntoSeek: Content-Based Access to the Web. IEEE Intelligent Systems, 14(3), pp.70–80.
2. P. A. Martin: General documentation of WebKB-2. Available at http://kvo.itee.uq.edu.au/webkb/WebKB2/doc/generalDoc.html
3. Jiwei Zhong, Haiping Zhu, Jianming Li and Yong Yu: Conceptual Graph Matching for Semantic Search. In Proc. of the 10th Intl. Conf. on Conceptual Structures, (ICCS2002), LNAI 2393, Springer-Verlag, 2002.
4. Lei Zhang and Yong Yu: Learning to Generate CGs from Domain Specific Sentences. In Proc. of the 9th Intl. Conf. on Conceptual Structures, (ICCS2001), LNAI 2120, Springer-Verlag, 2001.
5. George A. Miller: WordNet: An On-line Lexical Database. In the Intl. Journal of Lexicography, Vol.3, No.4, 1990.
6. J. F. Sowa: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley. 1984.
7. DAML Semantic Search Service, http://reliant.teknowledge.com/DAML/
8. OntoBroker, http://ontobroker.aifb.uni-karlsruhe.de/index_ob.html
9. S. Decker, D. Brickley, J. Saarela, and J. Angele: A Query and Inference Service for RDF. In: QL'98 - The Query Languages Workshop. 1998.
10. E. Salvat and M.L. Mugnier: Sound and complete forward and backward chainings of graph rules. In Proc. of the 4th Intl. Conf. on Conceptual Structures, ICCS'96, LNAI 1115, Springer-Verlag, 1996.
11. J. Madhavan, P.A. Bernstein, and E. Rahm: Generic Schema Matching with Cupid. In Proc. of the 27th Intl. Conf. on Very Large Databases (VLDB 2001).
12. J. F. Sowa (ed.): Conceptual Graph Standard. Available at http://www.jfsowa.com/cg/cgstand.htm