Using Transformations to Improve Semantic Matching

Peter Z. Yeh pzyeh@cs.utexas.edu
Bruce Porter porter@cs.utexas.edu
Ken Barker kbarker@cs.utexas.edu

Department of Computer Sciences
University of Texas, Austin
Austin, TX 78712 USA

ABSTRACT

Many AI tasks require determining whether two knowledge representations encode the same knowledge. Solving this *matching problem* is hard because representations may encode the same content but differ substantially in form. Previous approaches to this problem have used either syntactic measures, such as graph edit distance, or semantic knowledge to determine the "distance" between two representations. Although semantic approaches outperform syntactic ones, previous research has focused primarily on the use of taxonomic knowledge. We show that this is not enough because mismatches between representations go largely unaddressed. In this paper, we describe how transformations can augment existing semantic approaches to further improve matching. We also describe the application of our approach to the task of critiquing military Courses of Action and compare its performance to other leading algorithms.

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalisms and Methods**]: semantic networks

General Terms

Algorithms

Keywords

conceptual graphs, inexact matching, ontology, semantic matching, transformations

1. INTRODUCTION

A requirement common to many AI applications is determining whether (and how) two knowledge representations, encoded using the same ontology, encode the same knowledge. For example, rule-based classification systems match rule antecedents with working memory; information retrieval systems match queries with documents; and some knowledge-acquisition tools match new information with already encoded knowledge to expand upon and debug both of them.

The core problem, of course, is that multiple encodings of the same knowledge rarely match exactly, so a matcher must be flexible to avoid a high rate of false-negatives. However, a matcher that is too flexible can suffer from a high rate of false-positives. This problem has various causes, including:

- the ontology is expressive enough to allow the same information to be encoded in different ways
- the representations are built by different knowledge engineers (or computer programs), raising the likelihood they differ
- the representations are large, increasing the opportunity for differences

Our goal is to build a matcher that does well under these conditions.

Previous solutions to this problem have produced two types of matchers. Syntactic matchers use only the graphical form of the representations, judging their similarity by the amount of common structures shared [3, 4] or the number of edit operations required to transform one graph into the other [17, 23, 24, 26]. Approaches that focus on the amount of shared common structures do not handle mismatches. Approaches that use edit operations can handle mismatches but are sensitive to the cost assigned to the edit operations and tuning these parameters optimally is problematic.

In contrast, semantic matchers use knowledge, stored in an ontology, of the terms referenced in the representations. Previous semantic matchers use this knowledge to determine, for example, that two representations match because one consistently generalizes the

K-CAP'03, October 23–25, 2003, Sanibel Island, Florida, USA.

other [12, 14, 20, 30]. However, these matchers do not fully account for the various types of mismatches that can occur. Both syntactic and semantic approaches are examined in detail in section 5.

Our approach, described in section 3, is to augment existing semantic matchers with additional non-taxonomic knowledge in the form of transformation rules. We have methodically compiled a library of about 300 transformations for a general-purpose ontology. The result is a knowledge rich method for matching representations that handles a wide range of mismatches.

We applied our method to the task of critiquing military Courses of Action (COAs), one of the challenge problems posed in DARPA's Rapid Knowledge Formation (RKF) project. This application requires matching critiquing knowledge – engineered by military analysts – with Courses of Action – engineered by military commanders – to generate a report on their strengths and weaknesses along various dimensions. We thoroughly evaluated our method's ability to handle mismatches, and compared it with other match algorithms. In section 4, we show empirically that our matcher performs significantly better than either syntactic matchers or semantic matchers that rely solely on taxonomic knowledge.

Our underlying formalism for this project is a frame-based knowledge representation language called KM [6]. The knowledge encoded in this language can be equivalently encoded as conceptual graphs [25] with one exception: conceptual graphs support full negation, but we only support the negation of relations. For ease of exposition and generality, we will use conceptual graphs as the representation medium throughout the rest of this paper.

2. TYPES OF MISMATCH

As an initial study – which is neither formal nor complete – we analyzed the results from the COA critiquing task to get a sense of the types of mismatches that can occur. These are cases in which a military analyst expected that a piece of critiquing knowledge (a rule) would match a particular COA, but it did not. We found that most of the mismatches fall into one of a few categories.

Equivalent Alternatives. We found that often encodings of the same knowledge by different sources differ only in form. As an example, Figure 1 shows two independently encoded representations of a simple concept: one military unit engaging another unit to prevent it from moving. Structurally, the two representations differ substantially. The top graph represents an `Attack-By-Fire` that has the `objective` to *enable* `Block`ing the enemy's movement. The bottom graph says that the `Engagement-Military-Task` *prevent*s the enemy's movement. Despite this difference, we would like these representations to match. The match should be based on: (1) the assumption that actions (such as the `Attack-By-Fire`) achieve their stated objectives; and (2) the knowledge that to `Block` means to *prevent* movement.

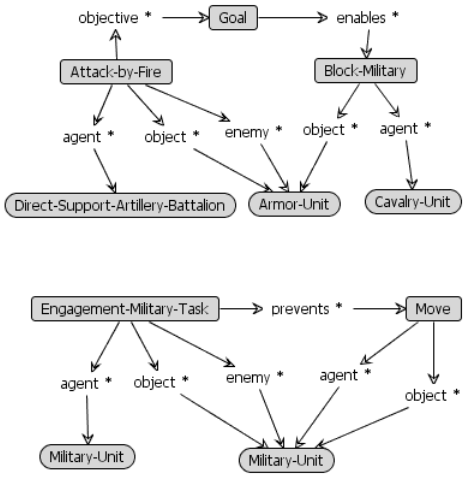

Figure 1: Two independent encodings of the same knowledge by a Subject Matter Expert participating in the RKF project. Attack-by-Fire is a subclass of Engagement-Military-Task.

Omissions. We found that different sources representing the same knowledge leave out different pieces of information from the encoding. Omissions can result from indirect references or missing co-references in the representation. A piece of information may also be omitted from a representation because it is deemed irrelevant. Figure 2 provides an example. The two conceptual structures, built independently, represent knowledge about a mechanized infantry brigade engaging a mechanized infantry battalion. The conceptual structure on the top does not include information about the subevent and its participant.

Granularity. We found that different sources encode the same knowledge at different levels of detail. For example, a subject matter expert encoding an attack by a battalion that is part of a brigade might also encode that the attack is by the brigade. A different subject matter expert might leave out this detail from his encoding.

Differing Viewpoints. We found that sometimes different sources encode the same knowledge from different conceptual views. For example, a river can be viewed as either a barrier preventing movement or a conduit enabling it, and representations of these viewpoints will include different features of the river.

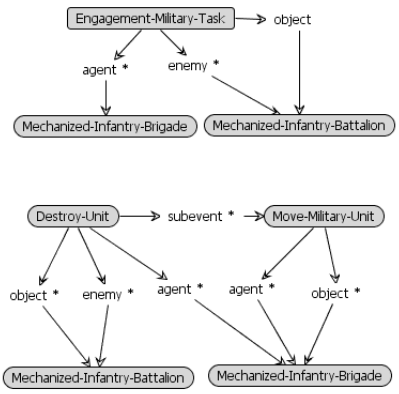

Figure 2: The concept "a brigade engaging a battalion" encoded by two different Subject Matter Experts participating in the RKF project. The top graph omits information present in the bottom graph. Also, *Destroy-Unit* **is a subclass of** *Engagement-Military-Task***.**

3. KNOWLEDGE-BASED MATCHING

Match algorithms that use knowledge about the terms in a representation are not new. Previous methods, described more fully in section 5, use taxonomic knowledge to determine, for example, that the terms in one representation consistently generalize the terms in the other. Our work extends these methods by using additional, non-taxonomic knowledge in the form of transformation rules. Each rule is of the form $lhs \Rightarrow rhs$ and is applied in a forward chaining manner. Furthermore, the antecedent and consequent of each rule are conceptual graphs encoded using the same ontology as the representations to which they are applied.

Previous research [9, 22, 27] has studied a related problem: reasoning with conceptual graphs using both deductive rules and projection. Although our method of applying rules is derived from this body of research, it differs in two significant ways.

First, we permit only those transformation rules whose antecedent and consequent are alternative encodings of the same information. We impose this restriction because a transformation's purpose is to resolve mismatches, and mismatches are alternative – but not necessarily equivalent – encodings of the same information.

Second, we modify the procedure for determining when a rule is applicable. With conventional forward chaining, a rule is applicable to a representation G if there is a projection from the rule's lhs to G [22, p. 251]. This procedure, however, does not consider that when performing a match there are two representations involved and the purpose for applying a rule is to better align them. Failure to consider this factor can generate a lot of unused work and increase the size of the search space. We account for these factors in the following way. When matching two graphs G_1 and G_2, we say that a rule, $R : lhs \Rightarrow rhs$, is applicable to G_1 w.r.t. G_2 if both of the following conditions are satisfied:

- There is a projection from lhs to G_1.
- There is a projection either from rhs to G_2 or from G_2 to rhs.

The application of R to G_1 with respect to G_2, i.e. $apply(R, G_1, G_2)$, adds the match instance of R's rhs to G_1, just as in conventional forward chaining.

Imposing these conditions to determine if a rule is applicable might cause a match to fail in one interesting case. Consider the rules $R_1 : X \Rightarrow Y$ and $R_2 : Y \Rightarrow Z$ and the graphs $G_1 : Z$ and $G_2 : X$, and note that R_1 and R_2 are not applicable under the conditions established above. As a result, G_1 and G_2 will not match because Z cannot be derived from G_2. We avoid this problem by computing the transitive closure of all the transformation rules to establish transitive relations not explicitly defined. This is feasible because we require that the consequent and antecedent of every rule be alternative encodings of the same information.

3.1 Types of Transformations

It's important to understand our ontology – at least, its design considerations – before delving into the transformations themselves. Our ontology is designed to enable Subject Matter Experts (SMEs), with little help from knowledge engineers, to build knowledge-bases about specialized topics without training in knowledge representation and logic. The approach is to give them a domain-neutral ontology containing a few hundred general concepts [1] that can be instantiated and assembled – using a set of 80 binary relations – to build new representations. Aspects of this work are described more fully in [2, 7, 8].

Because we kept our ontology small and gave each concept in our ontology well-defined semantics, we were able to methodically enumerate many transformations. These transformations were intentionally kept simple because each rule's antecedent and consequent are also susceptible to mismatches when matched against a representation. The resulting transformations are based on three types of general inference (see Table 1).

Transitivity: Causal relations like *causes, enables,* and *by-means-of* are transitive. However, there are subtle

[1]The complete library of general concepts and relations can be browsed and downloaded at http://www.cs.utexas.edu/users/mfkb/RKF/public

Table 1: Some of the transformations in our library – those involving causal relations.

relation	Transitive	Part Ascension	Transfers Through
causes	X	-	subevent, resulting-state
caused-by	X	subevent-of	resulting-from
defeats	-	-	-
defeated-by	-	subevent-of	caused-by
enables	X	-	causes, resulting-state, subevent
enabled-by	X	subevent-of	caused-by, resulting-from
inhibits	-	subevent-of	resulting-state
inhibited-by	-	subevent-of	caused-by, resulting-from
by-means-of	X	-	-
means-by-which	X	-	-
prevents	-	subevent-of	-
prevented-by	-	subevent-of	caused-by, resulting-from
resulting-state	-	-	causes
resulting-from	-	-	-

exceptions, such as *prevents* and *inhibits*. Partonomic relations (i.e. *has-part, subevent, element,* and *has-region*), and their inverses, are transitive. Also, some spatial and temporal relations (e.g. *is-inside, is-near,* and *before*) are transitive.

Part Ascension: Some information about a part applies to the whole to which it belongs. For example, the *caused-by* relation ascends through *subevent-of*. Therefore, because "Terrorism is *caused-by* Drug-Purchases which is a *subevent-of* the Drug-Trade", "Terrorism itself is *caused-by* the Drug-Trade". Other examples of part ascension include the *object* of an action, which ascends through *is-part-of*. However, exceptions include the *agent* of an action, which does not ascend through parts.

Transfers Through: Transitive and part ascendent transformations conform to a more general notion called "transfers through". A relation r transfers through another relation r' if

$$X - r \rightarrow Y - r' \rightarrow Z \Rightarrow X - r \rightarrow Z$$

As shown in Table 1, *enables* transfers through *causes* and *inhibits* transfers through *resulting-state*. This notion is similar to the one used in Cyc [15].

Thus, one of our contributions is identifying, in a systematic manner, about 300 transformations and the general class of inference to which they belong. Because these transformations are based on a domain-neutral ontology and are expressed at the knowledge-level, they should be generally useful to others. For the complete list of transformations see [29].

3.2 Match Algorithm

Before we describe our matching algorithm, we define some of the terms used:

- A **triple** is a 3-tuple of the form $(head, rel, tail)$ where *head* and *tail* are concepts or instances (i.e. nodes in a conceptual graph) and *rel* is an edge in the graph. Every two nodes connected by an edge in a conceptual graph can be converted to a triple. Thus, a conceptual graph can be mechanically converted into a set of triples.

- $t_1 = (head_1, rel_1, tail_1)$ and $t_2 = (head_2, rel_2, tail_2)$ **align** if $head_1 \geq head_2$, $uneg(rel_1) \geq uneg(rel_2)$, and $tail_1 \geq tail_2$. $uneg(rel)$ unnegates *rel* if it's negated, otherwise it just returns *rel*.

- Given $l = \{(t_{11}, t_{21}),...,(t_{1n}, t_{2n})\}$, a list of aligned triples. The **bindings** for l, i.e. $binding(l)$, is $\{head_{11}/head_{21}, tail_{11}/tail_{21}, ..., head_{1n}/head_{2n}, tail_{1n}/tail_{2n}\}$.

Figure 3 shows our algorithm for finding a match between two representations. Below, we describe the steps for finding this match and illustrate them with a running example. This example will match the two graphs, G_1 and G_2, shown in Figure 4. The taxonomy and the transformations used in this example are shown in Figure 5. For reference, we label each triple in G_1 with a unique number and each triple in G_2 with a unique letter. We use subscripts to differentiate terms that appear multiple times (e.g. Military-Unit).

Step 1: Our algorithm compares each triple in G_1 with each triple in G_2 to find all possible alignments. In our example, triple 1 aligns with triple A. Triple 1, however, does not align with triple B because the relations differ. This process is illustrated in Figure 6. The result of this initial match (we'll call it M) is also shown in Figure 6. We note that each element of M is a list called l_i. For example, $\{(1, A)\}$ is called l_1, $\{(2, B)\}$ is called l_2, etc..

GIVEN: Two graphs $G_1 = \{t_{11}, ..., t_{1n}\}$ and $G_2 = \{t_{21}, ..., t_{2m}\}$ where t_{1i} and t_{2j} are triples in G_1 and G_2 respectively, and a set of transformations R where $R = \{R_1, ..., R_n\}$.
FIND: A common subgraph of G_1 and G_2 called SG .
1. M = NIL and l = NIL
   FOR each triple t_{1i} in G_1
      FOR each triple t_{2j} in G_2
         IF t_{1i} aligns t_{2j}
         THEN add (t_{1i}, t_{2j}) to l.
      Add l to M and reset l to NIL.
2. Use M to construct a common subgraph of G_1 and G_2 called SG. $SG = \{(t_{11}, t_{21}),...,(t_{1n}, t_{2n})\}$ where (t_{1i}, t_{2i}) are the aligned triples between G_1 and G_2 respectively.
3. IF SG is inconsistent
   THEN stop and return NIL.
4. FOR each rule R_i in R,
      IF R_i is applicable to G_1 w.r.t. G_2,
      THEN $apply(R_i, G_1, G_2)$.
      ELSE IF R_i is applicable to G_2 w.r.t. G_1,
      THEN $apply(R_i, G_2, G_1)$
5. FOR each unaligned triple t_{1i} in G_1
      FOR each unaligned triple t_{2j} in G_2
         IF t_{1i} aligns t_{2j} and $binding(\{(t_{1i}, t_{2j})\})$ is consistent with $binding(SG)$,
         THEN add (t_{1i}, t_{2j}) to SG and break.
   UNTIL SG reaches quiescence go to step 4.
6. RETURN SG.

Figure 3: Outline of our algorithm.

Step 2: Our algorithm uses M to construct a common subgraph of G_1 and G_2 called SG. Our algorithm begins by selecting a member, l_i, of M to serve as the seed of the construction process (recall that $M = \{l_1, ..., l_m\}$ and $l_i = \{(t_{1i}, t_{2i}), ..., (t_{1i}, t_{2k})\}$). This seed is selected based on a heuristic scoring function:

$$h(l_i) = \frac{\sum_{j=i}^{k} num(head_{1i}/head_{2j}) + num(tail_{1i}/tail_{2j})}{k} \quad (1)$$

where $head_{1i}/head_{2j}$ and $tail_{1i}/tail_{2j}$ are the bindings between t_{1i} and t_{2j}, and $num(b)$ is the number of times the binding b occurs in $binding(M)$. This function is a heuristic that favors those l_i in M with high interconnectivity. Bindings that occur frequently indicate high interconnectivity. We want to select these l_i as the seeds because they have more potential for allowing larger common subgraphs to be constructed. Therefore, the algorithm selects the l_i in M with the highest score, as determined by the function h.

Returning to our example, the bindings for $l_1 = \{(1, A)\}$ is {Support-Attack$_1$/Support-Attack$_2$, Military-Unit$_1$/ Military-Unit$_3$}. The number of times Support-Attack$_1$/ Support-Attack$_2$ occurs in $binding(M)$ (see Figure 6) is

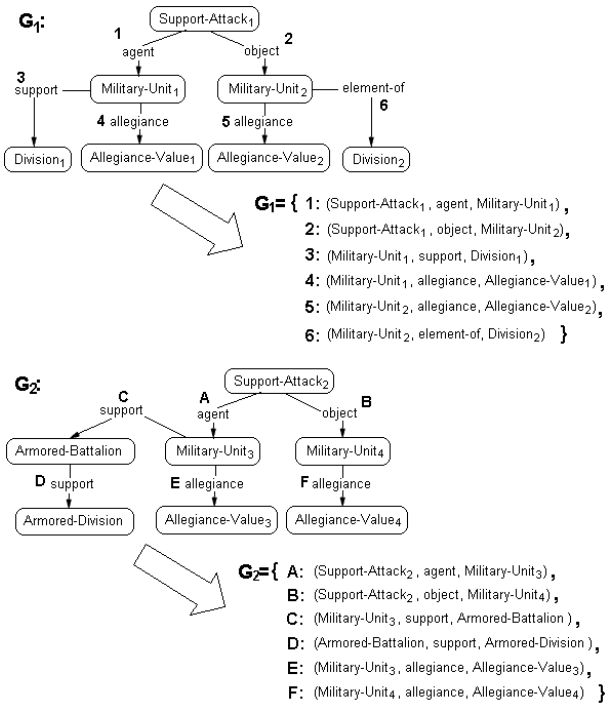

Figure 4: The graphs being matched.

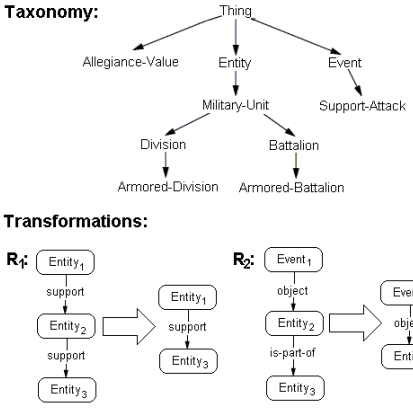

Figure 5: The taxonomy and transformations used in our example.

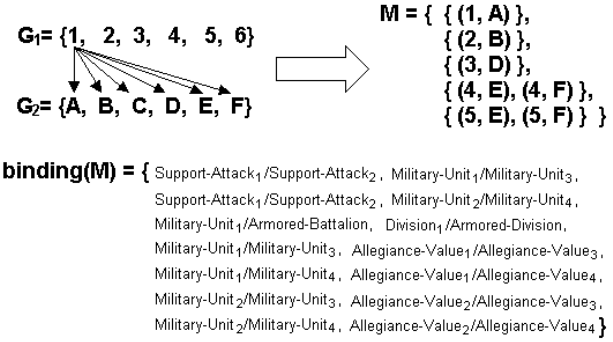

Figure 6: Illustration of Step 1.

2. The number of times Military-Unit$_1$/Military-Unit$_3$ occurs in $binding(M)$ is 2 also. Therefore, $h(l_1) = 4$. Applying the h to each l_i of M, we find that l_1 and l_2 have the highest scores (each has a score of 4). To break this tie, we randomly select l_1 as the seed.

After a l_i is selected as the seed, it is removed from M. Our algorithm will then construct a SG for each pair p_j in l_i (i.e. the seed). Each SG is constructed in the following way. First, SG is set to $\{p_j\}$. SG is then extended with those pairs of aligned triples in M whose bindings intersect the bindings of the pairs in SG. Pairs in M that extend SG are removed from M along with the l_j they belong to. This process is repeated until SG can no longer be extended. After a SG has been constructed for each pair in the seed, our algorithm selects the SG that is the largest. The result is an approximation of maximal common subgraph of G_1 and G_2.

In our example, we select l_1 as the seed. Since l_1 contains only one pair, we only need to construct one SG. We begin by setting SG to $\{(1, A)\}$. We then extend SG with those pairs in M whose bindings intersect the bindings of $(1, A)$. For example, we extend SG with $(2, B)$. The bindings for $(2, B)$ is {Support-Attack$_1$/Support-Attack$_2$, Military-Unit$_2$/Military-Unit$_4$} and Support-Attack$_1$/Support-Attack$_2$ intersects with the bindings of $(1, A)$. We cannot, however, extend SG with $(3, D)$ because its bindings do not intersect with those of $(1, A)$. Because we extended SG with the pair $(2, B)$, we must remove it and the list it belongs to (i.e. we remove $l_2 = \{(2, B)\}$) from M. This process is repeated until SG cannot be extended. Figure 7 illustrates this process.

Steps 3-5: Our algorithm checks if SG is consistent. SG is inconsistent if it contains an aligned pair of triples (t_{1i}, t_{2i}) where the relation of t_{1i} is negated and the relation of t_{2i} is not negated (or vice versa). If SG contains such a pair, then our algorithm stops and returns NIL. Otherwise, our algorithm applies transformations to improve the match (i.e. steps 4 and 5). Steps 4 and 5 are repeated until SG reaches quiescence. In step 4, our algorithm applies transformations to resolve mismatches between G_1 and G_2. In step 5, our algorithm will try to align additional triples between G_1 and G_2. Step 5 is like step 1 except it focuses on the unaligned triples.

Returning to our example, SG is consistent, so we apply transformations to improve the match. In step 4, R_1 is applicable to G_2 w.r.t. G_1 because the lhs of R_1 projects onto G_2, and the rhs of R_1 projects onto G_1. Therefore, we add the match instance of R_1's rhs to G_2 (i.e. (Military-Unit$_3$, supports, Armored-Division)). For future reference, let's label this new triple G. R_2, however, is not applicable because only the rhs of R_2 can project onto G_2.

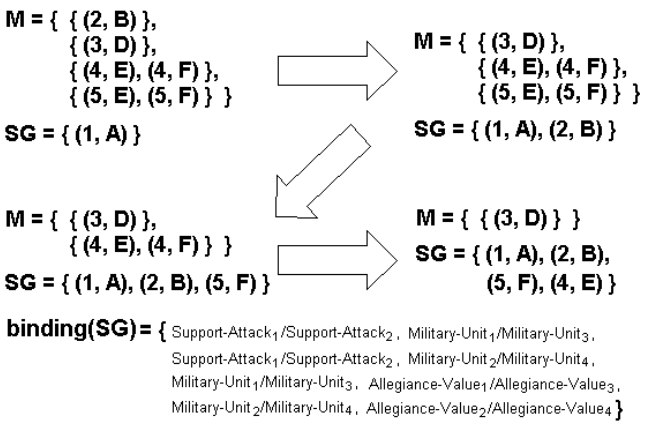

Figure 7: Illustration of Step 2.

After transformations have been applied, we try to align the remaining triples in G_1 (i.e. {3,6}) with the remaining triples in G_2 (i.e. {C,D,G}). We start with triple 3 and try to align it with triples C, D, and G. Triple 3 does not align with triple C. Triple 3 aligns with triple D, but the bindings for $(3, D)$ (i.e. {Military-Unit$_1$/Armored-Battalion, Military-Division$_1$/Armored-Division}) are inconsistent with SG's bindings: Military-Unit$_1$ is already bound to Military-Unit$_3$ in $binding(SG)$. Triple 3 aligns with triple G, and the bindings for $(3, G)$ are consistent with SG's bindings. Therefore, we add $(3, G)$ to SG. We use the same procedure to align triple 6. Through this process, only $(3, G)$ is added to SG. Furthermore, SG reaches quiescence after one iteration, so our algorithm stops applying transformations to improve the match.

Step 6: After transformations have been applied, SG is returned along with a numeric score reflecting the fitness of the match between G_1 and G_2. This score is computed based on the number of matched triples over the size of the graphs being matched (see [29] for the details).

4. EVALUATION

We present the results from two experiments conducted in the context of the COA critiquing task. The first experiment evaluates our matcher's ability to cope with discrepancies between representations as compared with two other match algorithms. The second experiment evaluates the efficiency of our matcher compared with a variant that uses conventional forward chaining.

4.1 Performance Task

We evaluate our matcher in the context of the COA critiquing task – one of the challenge problems in DARPA's RKF project. COAs are large, detailed battle plans intended to meet a specific military objective. Because of their complexity, military analysts have difficulty evaluating them quickly and accurately. Thus, the task of

the COA critiquer is to analyze a COA along several dimensions to assess its strengths and weaknesses. Our matcher was one of the critiquing methods used in a knowledged-base tool called SHAKEN [1].

Our approach to this problem has two parts. First, military analysts build a knowledge-base of "critiquing patterns" (see Figure 8). Each pattern, encoded as a conceptual graph, describes a situation that might arise in a COA, for example "blue aviation units attack red artillery units before the main attack" or "blue holds an artillery unit in reserve during the main attack". Because COAs are evaluated on eleven dimensions – such as `risk, use-of-terrain` and `simplicity` – each pattern has an associated list of $< dimension, score >$ tuples. This library of patterns is compiled in the lab, without concern for any particular battle plan, based on military doctrine and experience.

Second, to prepare for a battle, military commanders design a Course of Action to achieve stated objectives. It, too, is a conceptual graph. Our system evaluates the COA by applying the library of critiquing patterns to it to generate a report that assesses the COA's strengths and weaknesses.

The knowledge-base of critiquing patterns was built by two military analysts using the SHAKEN system. The COAs were built by the same personnel using NWU's NuSketch COA-authoring system [11]. The two SMEs produced a total of 44 patterns and three COAs. Because the patterns and COAs were authored separately, and by using different knowledge-authoring tools, there were many opportunities for mismatches.

4.2 Experiment: Coping with Mismatches

To evaluate our matcher's ability to cope with mismatches, we compared our matcher with two established algorithms:[2]

- Maximal Common Substructure (MCS): Uses a graph distance metric based on the maximal common substructure of two graphs [4][3].
- Semantic Search Lite (SSL): Uses taxonomic knowledge to match two graphs [14, 30]. Importantly, SSL is equivalent to our matcher with transformations ablated.

We used the following experimental methodology for this evaluation. We used the knowledge-base of critiquing patterns and COAs built by the two SMEs as the data set. For each match algorithm, we used that matcher to find the COAs that a pattern matched. We say that a pattern matches a COA if the match score meets or exceeds a prespecified threshold. This score is computed based on the fraction of information in the pattern that an algorithm matched with a COA.

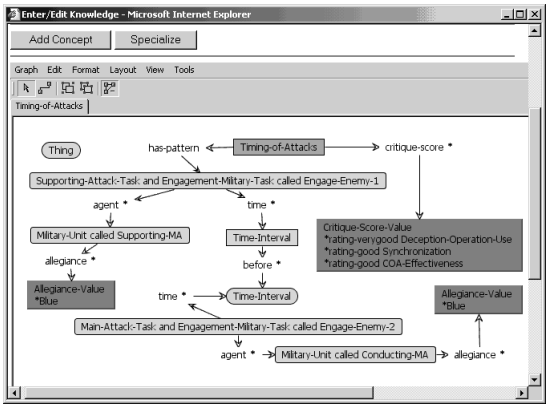

Figure 8: SHAKEN's interface for authoring and editing patterns. This screen capture shows a pattern authored by a SME regarding timing of attacks.

Because of the size and complexity of the COAs (each averaging a few thousand edges), a pattern often matches a COA multiple times, in different ways, yielding dozens, sometimes hundreds, of matches. To evaluate these matches, and hence the algorithms' performance, we note that applying patterns to COAs can be viewed as a form of information retrieval where information of interest is retrieved from a COA based on a pattern that is acting as a query. Therefore, we use metrics of precision and recall.

To calculate these metrics, we compared the results for each match algorithm against those of a human oracle, who determined that there were a total of 927 correct matches between the patterns and the COAs. To test for significant differences in the results, we used Pearson's $\mathcal{X}^2$ test. Table 2 shows the number of correct answers over the total number of answers given by each algorithm for match thresholds between 0.5 and 0.95.

Figure 9 shows the precision of the three algorithms. Our algorithm and SSL outperformed MCS at all levels of the match threshold ($p < 0.01$ for all points). MCS had terrible precision at all match thresholds. This was due to the complexity and high interconnectivity of the COAs (i.e. there were many substructures that were syntactically identical to the patterns). With MCS, a pattern would match all these substructures, although most were false-positives.

[2]Graph edit distance was not compared. Its performance relies heavily on the cost assigned to each edit operation. We did not have enough time to derive the optimal costs for a meaningful comparison. Taxonomic knowledge could have been used to derive the edit costs, but the resulting system would have been equivalent to SSL.

[3]Our implementation of MCS differs slightly from Bunke's. We do not require the labels of the nodes and edges to match.

Table 2: The number of correct answers over the total number of answers given by each algorithm for match thresholds between 0.5 and 0.95.

	KB Match	SSL	MCS
0.5	805/1839	508/1395	474/20304
0.6	805/1810	508/1239	474/17299
0.7	805/1219	459/686	425/8563
0.8	805/1087	459/667	425/8437
0.9	805/950	103/103	107/3915
0.95	750/750	35/35	34/3394

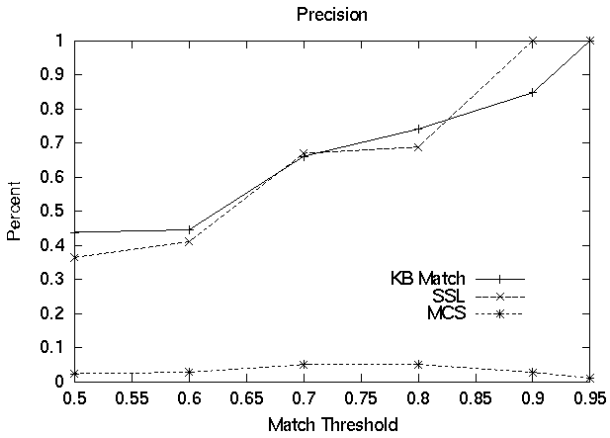

Figure 9: Precision is computed by dividing the number of correct answers given by the total number of answers given.

With respect to precision, our algorithm and SSL were comparable. Our system performed better than SSL at the 0.5 ($p < 0.01$) and 0.8 ($p < 0.02$) threshold. SSL, however, performed better than our system at the 0.9 threshold ($p < 0.01$). The 0.9 threshold is an interesting case. At this high threshold, SSL was able to match only those patterns that aligned exactly (or almost exactly) with the COAs. Thus, there were no false-positives. Using transformations, our algorithm was able to match features that SSL was unable to match. This resulted in more patterns being matched, but there was still room for false-positives at the 0.9 threshold. This difference disappears at the 0.95 threshold level.

Figure 10 shows the recall of the three algorithms. SSL and MCS had low recall rates, which dropped significantly as the match threshold was raised. Our algorithm, however, performed significantly better than both SSL ($p < 0.01$ for all points) and MCS ($p < 0.01$ for all points). Because our algorithm and SSL differ only in the use of transformations, this feature alone must account for the observed difference in recall.

4.3 Experiment: Efficiency

This experiment is designed to evaluate the effectiveness of the conditions we defined for when a rule is

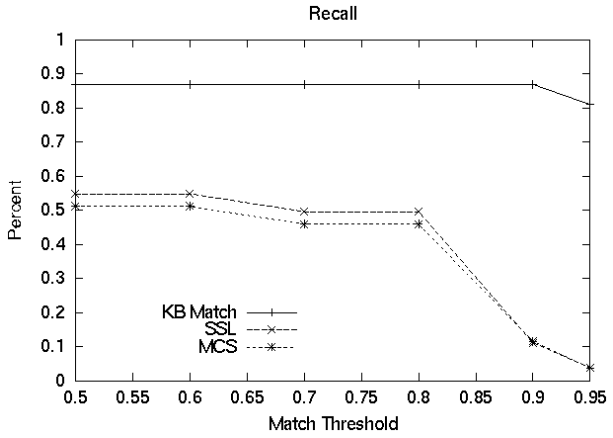

Figure 10: Recall is computed by dividing the number of correct answers given by the total number of correct answers given by the "oracle".

applicable. Specifically, we measure the efficiency gains (i.e. the reduction in the number of unnecessary rules applied) obtained by using our conditions for applying rules as compared with conventional forward chaining.

We used the same data set as that used in section 4.2, and the experimental methodology is as follows. We took our algorithm (see Figure 3) and generated a variant that uses conventional forward chaining to determine if a rule is applicable. This resulted in two frameworks: our specialized forward chaining (SFC) framework and the conventional forward chaining (FC) framework. For each framework, we applied all the patterns to the applicable COAs and recorded the number of transformation rules applied when determining whether a particular pattern matched a particular COA. Everything else (i.e. data set, transformation rules, etc..) was kept the same, and the answers returned by each framework was identical. Therefore, the only difference between the two frameworks is in how a rule is applied.

Table 3 shows the results for SFC and FC. The optimal number of transformations required to match a pattern to a COA is given as a baseline. As expected, there was a major difference in the average number of rules applied between SFC and FC. SFC was more efficient, and this difference was significant at the 0.01 level for the 2-tail t test.

The observed efficiency gains can be attributed to the additional condition we imposed for when a rule applies. By requiring both the *lhs* and *rhs* of a rule must match the representations being aligned (each to a different representation), the match process focused only on rules that can potentially align the mismatches between two representations. As a result, irrelevant transformations were not considered and not applied. This, in turn, helped to control the size of the search space.

Table 3: This table shows the average number of transformations applied under each framework. N is the sample size.

	Optimal	SFC	FC
Mean	17.38	343.76	2619.93
Std. Dev.	25.07	840.34	1619.33
Max	118	3410	4470
Min	0	0	0
N	71	71	71

5. RELATED WORK

Previous approaches to matching conceptual structures range from purely syntactic to semantically based. The Structure Mapping Engine[10, 13], for example, finds shared substructures between two graphs by matching the syntactic names of the relations. A heuristic called the systematicity principle is used to focus on shared substructures with high interconnectivity, for they are the most relevant. Similar metrics, such as the maximal common subgraph [4] and the minimal common supergraph [3], also find shared structures between two graphs. In the case of the maximal common subgraph, a subgraph that is isomorphic to both graphs is first obtained. A similarity distance between the two graphs is then calculated based on this subgraph. The minimal common supergraph operates in the inverse manner. These approaches do not account for mismatches between representations, and in our studies, MCS proved to be sensitive when the graphs being matched have high interconnectivity because many of the subgraphs appeared syntactically identical.

Graph edit/transformation approaches [17, 23, 24, 26] can cope with discrepancies and errors between two conceptual structures. A set of edit operations is defined, and a cost is usually assigned to each operation. The similarity between two graphs is the shortest sequence of edits that transforms one graph into the other. The performance of these approaches is sensitive to the cost of the edit operations and tuning these parameters optimally is problematic.

Conceptual retrieval or content-based search use semantics to improve performance. These approaches employ an ontology to aid in the matching of a query to a resource. In [28], a query is a simple conceptual structure consisting of the search terms and the relationships between the terms. The content being queried (or resource) is organized as a taxonomy. Structural subsumption is used as the matching mechanism. Works by [12, 14, 20, 30] also use taxonomic knowledge to aid in matching. The query and resource are represented as conceptual graphs and allow for more expressiveness. A query matches a resource if there is a projection from the query to the resource or the query shares a common subgraph with the resource. Although these approaches use semantics, its use is limited to taxonomic knowledge. As a result, mismatches between representations go largely unaddressed.

Related research by [18, 19] addresses how representations can be normalized to improve the structural similarity of semantically similar objects for classification and indexing. Knowledge is normalized at acquisition time, so all representations in the knowledge-base conform to the same standards. Some of the proposed normalization procedures include determining, a priori, which relations are "privileged" in order to handle inverses, propagating transitive relations, and grammar validation check. We believe that normalizing a representation at acquisition time is unnecessary for matching and may even adversely affect performance. Furthermore, forcing the user to decide, a priori, which relations are "privileged" and what rules to propagate is arbitrary.

Another line of related research is ontology merging and translation [5, 16, 21]. Works in this area either merge multiple ontologies into one or translate a representation from one ontology into another. Some of the difficulties in ontology merging and translation also effect us.

6. SUMMARY

In this paper, we presented a matcher designed to handle mismatches between representations. Mismatches are addressed by using transformation rules whose antecedent and consequent are alternative encodings of the same information. These transformations along with the algorithm for applying them were described in detail, and our method was applied to the task of critiquing military Courses of Actions. Our method's performance on this task was compared with other match algorithms. The results showed that our matcher performed significantly better because of its use of transformations.

7. ACKNOWLEDGMENTS

Support for this research was provided by a contract from SRI international as part of DARPA's Rapid Knowledge Formation project. The authors would also like to thank Vinay Chaudhri, Peter Clark, James Fan, Sunil Mishra, Ken Murray, and Dan Tecuci for their help on this project.

8. REFERENCES

[1] K. Barker, J. Blythe, G. Borchardt, V. Chaudhri, P. Clark, P. Cohen, J. Fitzgerald, K. Forbus, Y. Gil, B. Katz, J. Kim, G. King, S. Mishra, C. Morrison, K. Murray, C. Otstott, B. Porter, R. Schrag, T. Uribe, J. Usher, and P. Yeh. A knowledge acquisition tool for course of action analysis. In *IAAI*, 2003.

[2] K. Barker, B. Porter, and P. Clark. A library of generic concepts for composing knowledge bases. In *K-Cap'01*, 2001.

[3] H. Bunke, X. Jiang, and A. Kandel. On the minimum common supergraph of two graphs. *Computing 65*, 1, 2000.

[4] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19, 1998.

[5] H. Chalupsky. Ontomorph: a translation system for symbolic knowledge. In *KR*, 2000.

[6] P. Clark and B. Porter. KM: The knowledge machine. *www.cs.utexas.edu/users/mfkb/km*.

[7] P. Clark and B. W. Porter. Building concept representations from reusable components. In *AAAI/IAAI*, 1997.

[8] P. Clark, J. Thompson, and B. Porter. Knowledge patterns. In *KR*, 2000.

[9] D. Corbett and R. Woodbury. Unification over constraints in conceptual graphs. In *ICCS*, 1999.

[10] B. Falkenhainer, K. D. Forbus, and D. Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41(1), 1989.

[11] K. Forbus and J. Usher. Sketching for knowledge capture: A progress report. In *Intelligent User Interfaces*, 2002.

[12] D. Genest and M. Chein. An experiment in document retrieval using conceptual graphs. In *ICCS*, 1997.

[13] D. Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7, 1983.

[14] N. Guarino, C. Masolo, and G. Vetere. Ontoseek: Content-based access to the web. *IEEE Intelligent Systems*, 14(3), 1999.

[15] D. B. Lenat and R. Guha. *Building Large Knowledge-Based Systems*. Addison-Wesley Publishing Company, 1990.

[16] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *KR*, 2000.

[17] B. T. Messmer and H. Bunke. A network based approach to exact and inexact graph matching. Technical Report IAM 93-021, Institut fur Informatik, Universitat Bern, 1993.

[18] G. W. Mineau. Normalizing conceptual graphs. In P. Eklund, T. Nagle, J. Nagle, L. Gerhotz, and E. Horwood, editors, *Current Directions in Conceptual Structure Research*, 1992.

[19] G. W. Mineau. Facilitating the creation of a multiple index on graph-described documents by transforming their descriptions. In *CIKM*, 1993.

[20] S. Myaeng. Conceptual graphs as a framework for text retrieval. In P. Eklund, T. Nagle, J. Nagle, L. Gerhotz, and E. Horwood, editors, *Current Directions in Conceptual Structure Research*, 1992.

[21] N. F. Noy and M. A. Musen. An algorithm for merging and aligning ontologies: automation and tool support. In *AAAI, Workshop on Ontology Management*, 1999.

[22] E. Salvat and M.-L. Mugnier. Sound and complete forward and backward chainingd of graph rules. In *ICCS*, 1996.

[23] A. Sanfeliu and K. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. on SMC*, 13, 1983.

[24] L. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Trans. on PAMI*, 3, 1981.

[25] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley Publishing Company, 1984.

[26] W. Tsai and K. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Trans. on SMC*, 9, 1979.

[27] M. Willems. Projection and unification for conceptual graphs. In *ICCS*, 1995.

[28] W. Woods. Conceptual indexing: A better way to organize knowledge. Technical Report TR-97-61, Sun Microsystems Laboratories, 1997.

[29] P. Yeh, B. Porter, and K. Barker. Transformation rules for knowledge-based pattern matching. Technical Report UT-AI-TR-03-299, University of Texas at Austin, 2003.

[30] J. Zhong, H. Zhu, J. Li, and Y. Yu. Conceptual graph matching for semantic search. In *ICCS*, 2002.