Natural Language Generation through Case-Based Text Modification

Josep Valls and Santiago Ontañón

Computer Science Department
Drexel University
Philadelphia, PA, USA 19104
josep@valls.name, santi@cs.drexel.edu

Abstract. Natural Language Generation (NLG) is one of the longstanding problems in Artificial Intelligence. In this paper, we focus on a subproblem in NLG, namely *surface realization through text modification*: given a source sentence and a desired change, produce a grammatically correct and semantically coherent sentence that implements the desired change. Text modification has many applications within text generation like interactive narrative systems, where stories tailored to specific users are generated by adapting or instantiating a pre-authored story. We present a case-based approach where cases correspond to pairs of sentences implementing specific modifications. We describe our retrieval, adaptation and revise procedures. The main contribution of this paper is an approach to perform case-adaptation in textual domains.

1 Introduction

Natural Language Generation (NLG) is one of the longstanding problems in Artificial Intelligence. In this paper, we focus on a specific subarea in NLG, namely *surface realization*. Surface realization is the final stage in the NLG pipeline [11], and is in charge of generating actual natural language text given a sentence specification, embodying all decisions related to the grammar and morphology of a particular language [6]. We present a case-based approach to the problem of *surface realization through text modification*: given a source sentence and a desired change (e.g. replace a phrase from a sentence by another phrase, change the subject, change the tense of a verb, etc.), produce a grammatically correct sentence that implements the desired change. In the remainder of this paper we will refer to this problem simply as "text modification".

The research reported in this paper was motivated by work in the Riu system [9], an interactive fiction system that generates stories in natural language by combining and modifying pre-authored story snippets. One of the major drawbacks in Riu is the quality of the text presented to the user, due to the difficulty of generating grammatically correct and semantically coherent natural language. The approach presented in this paper can greatly increase the quality of systems like Riu, but it can also be applied to any other system that generates text by instantiating and modifying predefined text or templates.

In order to address the problem of text modification, we present a CBR approach called CeBeTA. With a few notable exceptions such as CR2N [2] and some recent work in jCOLIBRI [10], the vast majority of work in CBR for natural language focuses on retrieval and spell checking. The main contribution of CeBeTA is that we propose not only retrieval, but also adaptation and revision methods for natural language domains. In particular, cases in CeBeTA contain pairs of sentences, one being a modification of the other. When a new problem arrives to CeBeTA requesting to modify a sentence, the system infers which are the modifications that need to be done to the sentence based on retrieved cases, and they are applied to obtain a collection of candidate solutions. CeBeTA then uses an automatic revision module that selects the solution most likely to be grammatically correct and semantically coherent.

The remainder of this paper is structured as follows. Section 2 formally presents the problem of surface realization through text modification. Section 3 presents the CeBeTA system to address this problem. After that, Section 4 describes our empirical evaluation, and Section 5 compares CeBeTA with existing related work. The paper closes with conclusions and future work.

2 Text Modification through CBR

In this article we present a CBR approach to the surface realization phase of the NLG pipeline. Surface realization has been traditionally dealt with the use of annotated templates. In this paper, we present an alternative solution, based on case-based text modification, which doesn't require annotated templates.

In this section we will look at the specific problem we are trying to address, our proposed solution and a system that implements a subset of this proposal.

2.1 Problem Statement

The problem we address in this paper is the natural language generation by modification of given sentences, specifically:

Given:

1. A correct sentence s (for example: "Alice loves pizza").
2. A desired modification, represented as a replacement $p_1 \rightarrow p_2$ (for example "Alice" $\rightarrow$ "The girls').

Find: A new grammatically correct sentence s' where p_1 has been replaced by p_2 (for example "The girls love pizza").

Finding tokens in a string of text and performing replacements is a trivial task for computers. For example, given the sentence "Alice loves Bob", a simple replacement instruction could be stated as: "Bob" $\rightarrow$ "pizza". The result of the desired replacement would be: "Alice loves pizza". However, simple token replacement doesn't always result in grammatically correct sentences. For example if we would like to execute the replacement instruction "Alice" $\rightarrow$ "I', then the

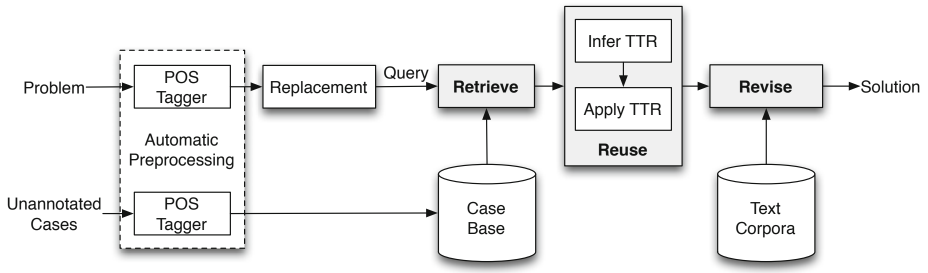

Fig. 1. Overview of the CeBeTA architecture for case-based text modification

output of the replacement would be: “I loves Bob”, which is incorrect due to the verb inflexion rules of the English language.

This problem has been addressed in previous work with the use of annotated templates that contain *part-of-speech* (POS) information and some sort of dependency tree or typed dependency list [4] that reflects the relationships between components in phrases and phrase relationships within sentences (i.e. the system needs to know that “Alice” is the subject, singular, female, etc.).

In order to achieve this solution, the surface realizer still needs to understand and process all the annotations properly. Annotated templates have been in use since the early days of NLG and are still in use by modern systems. Hand annotation of templates is a significant bottleneck in NLG systems, for which some automated solutions have been proposed. However, automated annotation is unreliable, due to the ambiguous nature of natural language.

In this paper, we present a CBR approach that can be used to address the problem of surface realization through text modification. Furthermore we propose a model that is domain and language independent and that can be easily extended by feeding domain and language specific corpora along with a very simple grammar modification module.

This problem was motivated by the Riu system [9], a computational analogy-based story generation system, that, as part of its story generation process, produces text by modifying existing natural language sentences. Currently, Riu modifies existing sentences by performing simple replacements, often yielding incorrect sentences. The proposed solution can be applied to range of problems related to text generation by either enriching replacement-based systems like Riu or replacing annotated templates with sets of unannotated documents.

2.2 Conceptual Model

From a CBR point of view, NLG poses major problems in all the stages of the CBR cycle [1]. This section provides an overview of our approach, that we call *Surface Realization through Case-Based Text Modification* (CeBeTA), to deal with the problem of case-based text adaptation. Figure 1 shows all the elements of the CeBeTA architecture, which we describe below.

CeBeTA takes as its input a problem consisting of two parts: a sentence s, and a desired change $p_1 \rightarrow p_2$, where p_1 is a phrase appearing in s. Moreover, we require the problem specification to also include a second sentence s_2 containing p_2; this second sentence is used during the automatic preprocessing to obtain information about p_2 and is then discarded. An example problem is:

- s : "Alice wants to have a pet dog"
- $p_1 \rightarrow p_2$: "Alice" $\rightarrow$ "The girls"
- s_2 : "The girls sing many happy songs"

The previous problem amounts to asking the system to produce a valid sentence after replacing "Alice" by "The girls" from the sentence "Alice wants to have a pet dog". The expected solution of the system to this problem is "The girls want to have a pet dog". Moreover, the implemented version of our system allows for the problem to contain more than one desired replacement.

Cases are defined as pairs of grammatically correct sentences (s_s, s_t) where s_t is the result of performing some sort of replacement to s_s and later modifying the result to make sure it is grammatically correct. In the rest of this paper we will call s_s the *source* sentence, and s_t the *target* sentence.

As shown in Figure 1, before the CBR system can actually process either the problem or the cases, they are analyzed using a POS tagger. The tagging process determines the grammatical category of each word (determinant, adjective, etc.) in the context it is found. The second sentence, s_2, included in the problem is used only to obtain the POS tags for p_2, being subsequently discarded.

As shown in Figure 1, retrieval is the first process executed when a problem is received. During case retrieval, a case from the case base is selected by using a similarity measure between problem and cases, as described in Section 3.1. The similarity measure utilises the POS information to find cases which contain similar sentences and similar replacements.

As stated above, one of the main aims of our system is to be able to work from unannotated text, thus, cases in the case base are just pairs of sentences in natural language. Although the cases are automatically POS tagged, there are no additional annotations describing which modifications were performed to the source sentence to produce the target sentence. For that reason, reuse in our system works in two stages: in a first stage, the system tries to infer the transformations that led from the source sentence in the retrieved case to the target sentence in the case ("Infer TTR" in Figure 1). Then, in a second stage the system applies the inferred transformations to the sentence in the problem at hand, in order to produce candidate solutions ("Apply TTR" in Figure 1). These transformations consist of small grammatic operators, that we call *Text Transformation Routines* (TTR) (described in Section 3.2).

Finally, our model includes an automatic revision phase using a *Probabilistic Evaluator* (PE). The PE is a module implementing a language model capable of predicting the likelihood that a sentence is grammatically correct and semantically coherent. The PE ranks the solution candidates and selects the highest ranking candidate as the solution.

3 Technical Approach

This section describes our implementation of the CeBeTA architecture presented in the previous section, detailing how retrieval, reuse and revision work.

3.1 Retrieval

As described above, each case $c = (s_s, s_t)$ in the CeBeTA system consists of a pair of natural language sentences in plain text. During the retrieval process, CeBeTA aims at finding one case that contains a pair of sentences similar to the problem at hand. Moreover, as described above, both cases and problems are automatically analyzed using a POS tagger before being used by the system, so, in the rest of this paper, we will assume that sentences and replacements have been already POS tagged.

To accomplish that task the system performs two steps:

1. Replacement: given the problem at hand, composed of a sentence s_1, and a given replacement $p_1 \rightarrow p_2$, the system generates a new sentence s_r, by directly replacing p_1 by p_2 in s_1, without performing any further changes. The result of preprocessing is the pair (s_1, s_r), which we call the *query*.
2. Nearest neighbor retrieval: after preprocessing, the query now consists of a pair of sentences, like the cases in the case base. Therefore, it is now possible to use a distance metric between pairs of sentences to retrieve the case that is most similar to the query.

The distance metric used in the CeBeTA system between a query (s_1, s_r) and a case (s_s, s_t) is defined as:

$$d((s_1, s_r), (s_s, s_t)) = SED(s_1, s_s) + SED(s_r, s_t)$$

where $SED(s, s')$ is defined as a custom *Sentence Edit Distance* (SED)[12] function. In other words, the distance between the query and a given case is the sum of the distances between the source sentence of the query to the source sentence of the case, and the target sentence of the query to the target sentence of the case. Other forms of aggregating the two distances could be employed by using other t-norms, the choice of the optimal aggregation operator is part of our future work. However, in our experiments, we observed that a simple addition was enough to obtain acceptable results.

The SED distance uses the POS-tagged words as tokens, and, as other edit distance algorithms, computes the shortest possible way in which one sentence could be edited in order to be identical to the other. Each of these edit operations has a different cost. In order to determine the cost of each operation, an undirected weighted graph was built with the different linguistic components of a phrase. The arcs between a particle a and a particle b are estimated according to the impact of replacing a by b, or vice-versa. A subset of the graph used for the experiments reported in this paper is shown in Figure 2.

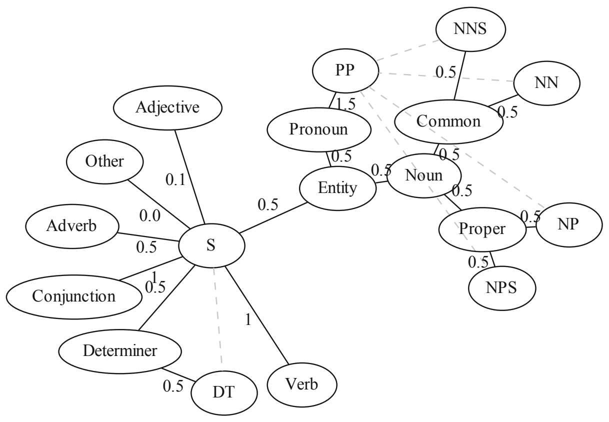

Fig. 2. Subset of the cost graph used in the sentence edit distance used by CeBeTA

The full graph features 54 nodes and 53 arcs plus 5 path overrides (shown in Figure 2 as dashed lines). We estimated the cost of inserting and deleting a word by adding the cost from the POS-class of the word to the root element of the graph (S). For example, the cost of deleting a singular proper noun (NP) would be $0.5 + 0.5 + 0.5 + 0.5$. And the cost of inserting a plural common noun (NNS) would be $0.5 + 0.5 + 0.5 + 0.5$.

For the edit operation, the current system computes an additional cost in order to account for lemma or morphological differences. During POS tagging, words are also *stemmed* (i.e. the stem or lemma of each word is computed).

The edit operation between two words has an additional cost of 0.1 when the lemma does not match. For example, the cost of replacing "dogs" by "dog" is $0.5+0.5+0.0$ but "dogs" by "bird" is then $0.5+0.5+0.1$. For pronoun and noun-pronoun changes, we add 0.05 when the gender does not match and 0.05 when the number does not match; the cost of replacing "girl" by "her" is $1.0+0.1+0.0+0.0$ but "girls" by "him" is then $1.0 + 0.1 + 0.05 + 0.05$. Lemma information could be used to further enhance the distance metric using semantic and conceptual information. Part of our future work would be to enhance the graph, fine-tune arc costs and account for lemma compatibility, for example, using the Regressive Imagery Dictionary (RID) or the Wordnet lexical database.

3.2 Reuse

There are two main ideas behind the reuse process in the CeBeTA system:

- Cases do not contain the adaptation routines used to produce the solutions. CeBeTA infers the adaptation path that lead from the source to the target sentence in a case, to then apply it to the problem at hand (this is motivated by the fact that we want our system to reason from unannotated sentences).
- In order to deal with text modification, we use what we call *Text Transformation Routines* (TTR), which serve as our adaptation operators.

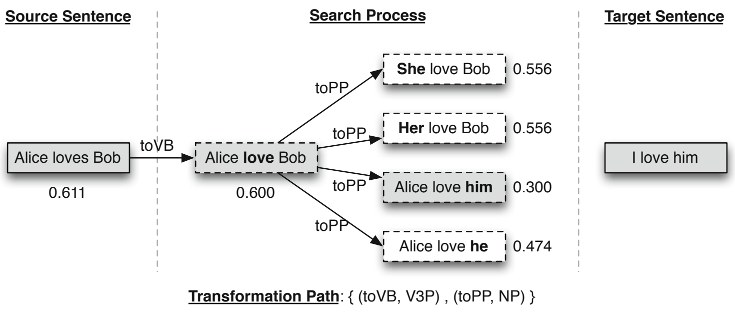

Fig. 3. Illustration of the search process required to obtain the set of TTRs required to reach the target sentence in a case, starting from the source sentence. toVB is the *verb-to-first-person* TTR, toPP is the *noun-to-pronoun* TTR, V3P means "Verb singular 3rd person", and "NP" means proper noun.

A TTR is an operator that given a word, e.g. "loves", and its given POS class, e.g. "verb, 3rd person singular, present tense", generates variations of it, e.g. "loved" or "loving". We used TTRs that enable word elimination, gender transformations; noun, pronoun and determiner transformations, etc. The specific TTRs we used are: *remove-word*, *pluralize*, *singularize*, *make-male*, *make-female*, *insert-determinant*, *verb-to-past-tense*, *verb-to-past-participle-tense*, *verb-to-first-person*, *verb-to-future-tense*, *verb-to-present-tense*, *verb-to-third-person*, *verb-to-gerund*, *verb-to-infinitive*, and *noun-to-pronoun*, with their intuitive effects.

In the rest of this paper we will represent a TTR as a pair $(type, POS)$, where $type$ is the type of TTR (*remove-word*, *noun-to-pronoun*, etc.), and POS is a POS-class (adjective, preposition, verb-3rd-person-singular, etc.), to which the TTR will be applied. Given a POS tagged sentence, we can generate variations of this sentence by applying TTRs to the different words of the sentence.

Given a collection of TTRs, the reuse phase in CeBeTA proceeds in two steps (as shown in Figure 1). In a first step (*Infer TTR*), the system infers which is the set of TTRs that needs to be applied to the source sentence in the retrieved case to produce the target sentence, we call this the *transformation path*. In a second step (*Apply TTR*), candidate solutions to the problem at hand are generated by applying the set of TTRs inferred in the first step. Some TTRs can produce more than one variation, and thus the outcome of the reuse phase is not just a single candidate solution, but a list of candidate solutions to the problem at hand, that will be evaluated by the revise process explained in the next section.

During the *Infer TTR* step, the CeBeTA system uses a hill climbing search method in order to find the sequence of TTRs that can better transform the source sentence in a case into the target sentence, as follows:

1. Set the current sentence to the source sentence in the retrieved case, the current transformation path is empty.
2. Select an applicable TTR to the current sentence, and generate all the possible variations of it using such TTR.

3. Measure the distance between the newly generated variations and the target sentence in the retrieved case.
4. If any of the newly generated variations is more similar to the target than the current sentence, we add the TTR to the transformation path, set that variation as the current sentence, and go back to 2.
5. If there is no applicable TTR that can generate a variation that is more similar to the target sentence than the current sentence, finish, and return the current transformation path.

An example of the execution of this process is shown in Figure 3. The source sentence is "Alice loves Bob", and the target sentence is "I love him". The distance between source and target is 0.611. "Alice loves Bob" is set as the current sentence, and the *verb-to-first-person* TTR is applied to obtain the sentence "Alice love Bob", which is more similar to the target (distance 0.600) than the current sentence, thus, the pair (toVB, V3P) is added to the current transformation path, indicating that the TTR type *verb-to-first-person* was applied to a singular 3rd person verb. Then "Alice love Bob" is selected to be the next current sentence. This process continues, until the system reaches "Alice love him", which is the closest it can get to the target using the available TTRs.

Notice that the target sentence in a case is the result of first applying some replacement to the source sentence and then fixing the grammatical errors. The TTRs used in our system are designed to cover only the latter, and thus, it is not expected that the transformation path can perfectly produce the target.

Moreover, since the search process required in the *Infer TTR* step might explore a very large space, we don't use the same distance measure used for retrieval, but a simpler distance measure based on the Ratcliff-Obershelp algorithm [3]; enough for the purposes of finding a good transformation path.

During the *Apply TTR* step, the transformation path is applied to the problem in order to obtain candidate solutions. This process is performed as follows:

1. Given the query (s_1, s_r), where s_r was obtained by replacing p_1 by p_2 in s_1, and the inferred transformation path: $\{(type_1, POS_1), ..., (type_n, POS_n)\}$.
2. Apply the TTRs in the transformation path $\{(type_1, POS_1), ..., (type_n, POS_n)\}$ one at a time to obtain all the possible variants of s_r, but protecting all the words in s_r that belong to p_2 from any change.

The result is a collection of sentence variants that are candidate solutions to the problem at hand. Notice that we protect any changes to occur to the words in p_2 since we assume that the user wants p_2 to be in the solution as-is.

For example, imagine that the problem is s_1 = "Mary likes cake", $p_1 \rightarrow p_2$ = "Mary" $\rightarrow$ "I", and the inferred transformation path is: {(toVB, V3P), (toPP, NP)}. In this situation s_r would be "I likes cake". Then, all the inferred TTRs would be applied one by one:

- Applying (toVB, V3P) we obtain: "I like cake"
- The second TTR, (toPP, NP), cannot be applied since there is no proper noun in the sentence, so, we obtain no more variants.

The output of the reuse process in this example would be all the variants generated by applying all the TTRS, which in this example is just two sentences: "I likes cake", and "I like cake", which would be handed to the revise process to select which one is the best. Moreover, notice that we have used very simple transformations in the examples in this section, but in principle problems may imply very complex transformations of the sentence. For example, consider the sentence: s_1 = "Once upon a time there was a knight in Scotland, his name was Robert", and the replacement "Robert" $\rightarrow$ "Alice and Bob". The correct result would be "Once upon a time there were two knights in Scotland, their names were Alice and Bob". Notice that the resulting sentence has suffered many transformations, like "one knight" $\rightarrow$ "two knights", which are semantical rather than grammatical, but may be covered by a very simple transformation path like {(remove-word, determinant), (pluralize, NN), (insert-determinant, NN)}.

3.3 Revise

The revise process is performed automatically by a *Probabilistic Evaluator* (PE) module. In a nutshell, the PE is a standalone module implementing a language model capable of predicting the likelihood that a sentence is grammatically correct and semantically coherent. Given the candidate sentences returned by the reuse process, the PE ranks them and the one that is more likely to be correct (according to the PE evaluation) is selected as the final answer. The PE has been designed as a self contained, pluggable module so that multiple evaluators can be switched, combined or reused in other systems. Two PE modules have been developed as a proof-of-concept for the system presented in this article, each implementing a language model using a different database constructed from readily available data.

Instead of implementing a full language model capable of computing probabilities for the sentences, our PE takes for input a sentence and splits it in smaller fragments of fixed size n by using a sliding window. Then, it checks the probabilities for each fragment against a text corpus and aggregates the results into a single value. We used the arithmetic mean for aggregation after some experimentation. Although a full language model would certainly improve the results of the system, it was out of the scope of this project. In our tests with simple language models, we observed they would favor shorter sentences as the conditional probabilities diminished with longer sentences.

We used two PEs from language models trained from different corpora. One uses a frequencies database (Google n-grams) and the other uses a probability database (from the CMU Sphinx system, using the Gigaword corpus). The larger the n-grams considered by the evaluators, the better the performance, however, increasing n increases the memory required to handle the database of the evaluator exponentially. The results from each evaluator differ slightly, since the data used varies in nature. Below we briefly describe the two databases used in our experiments.

– **Google n-gram**: This evaluator uses a database based on the Google Books n-gram English dataset. The original dataset was extracted from a corpus

Table 1. Example results that the two probabilistic evaluators used in our experiments obtain for three different sentences using a 3-gram model

Sentence	*Google n-gram PE*	*Sphinx PE*
I have a pet dog.	6.695	-2.759
I have a pet wolf.	-19.743	-3.564
I have a pet hamburger.	-44.743	-4.174

of scanned books. We gathered several datasets containing n-grams up to size 4, limited to books between 1980 and 2008, discarding year information, words with non-English characters, etc. Our Google n-gram database stores, the probability of each n-gram relative to that of the first word of the n-gram (computed in standard deviations form the mean of the first word).

– **CMU Sphinx's Gigaword ARPA corpus**: This evaluator uses a database computed for a language model based on the Linguistic Data Consortium Gigaword text corpus consisting of 1.200 million words from various sources. This dataset stores the conditional probability of each n-gram given the first $n-1$ words (stored as its natural logarithm). The database contains n-grams up to size $n = 3$ but we use only one of the three available datasets.

Table 1 shows three examples of the numbers obtained with each of the two evaluators for comparison purposes. For example, for the sentence "I have a pet dog" the *Google n-gram* evaluator returns a score of 6.695, which is the average between 0.03,1.23, 4.52, and 21.0, that this evaluator gives respectively to the following n-grams: "I have a", "have a pet", "a pet dog", and "pet dog.". It is worth noting that the PE do not just check for grammatical correctness, but also for semantical consistency, as can be seen in 1, where both evaluators score a sentence containing the phrase "pet hamburger" very low, since hamburgers are typically not pets, and thus the two words appear together very rarely in the corpora used for training the language models.

4 Empirical Evaluation

In order to evaluate CeBeTA, we designed 3 experiments, reported in the following three subsections respectively. The first experiment aims at evaluating the distance metric used for retrieval. A second experiment explores the performance of the PE used for revision. The third experiment tests the design of the system as a whole.

4.1 Case Retrieval

In order to evaluate the similarity metric between sentences, we devised the following experiment. We constructed 8 groups of sentences, where each group contained between 7 and 15 sentences. The sentences in each group were constructed by taking a base sentence and modifying it in different ways. The sentences in a

Table 2. Example group of sentences used in our experiments to test the distance measure used for case retrieval. We also show the distance between the first sentence and the rest as computed by the measure used for retrieval (*Custom SED*).

Sentence	*Custom SED*
Alice loves Bob.	0.0
Alice loves Mary.	0.1
Alice loves him.	1.0
Alice loves me.	1.1
Alice hates pasta.	2.2
Alice ran away.	6.1
Alice wants to be a painter.	7.7
Alice used to have a pet dog.	9.2
Alice is going to be a painter.	9.7
Alice has always wanted to be a painter.	10.7

group were manually sorted by us according to intuitive similarity with the base sentence: the first sentence in a group is the base sentence, the second is the most similar to the base, and so on, until the last sentence, which is the most different from the base sentence. For example Table 2 shows an example of one such groups of sentences.

Different groups were authored to check for certain features. Some groups of sentences were crafted to test for basic noun and pronoun compatibility, adjective insertions and deletions, morphology changes and semantic modifications.

In order to evaluate the distance metric used for retrieval, we compared the ordering that results by sorting the sentences in each group by their similarity with the base sentence, with the order that was provided for the sentences in each group (that we consider to be the ground truth). We measured the mean square error of the position of each sentence in the order generated by the similarity measure. For example is a sentence was in position 3 in the ground truth and in position 5 in the order generated by the similarity measure, the squared error is $(3 - 5)^2 = 4$. The resulting mean square error was 5.13, which means that sentences are in average 2.26 positions off where they should be. Moreover, we observed that most sentences were actually exactly in the same position or at most one off, except for a few out-layers that caused an increase in the error. 36.5% of the sentences were exactly in the same position as in the ground truth, and 67.66% where exactly in the same position, or at most one position off. This shows strong evidence that the similarity measure used for retrieval strongly resembles intuitive sentence similarity.

4.2 Revision

In order to test the PE performance for solution revision, we followed a methodology similar to the previous experiment. We crafted several groups of sentences, designed to test evaluators against different grammatical and syntactic features.

Table 3. Example group of sentences used in our experiments to test performance of the PE for solution revision. We also show the ranked score given by both evaluators using 3-grams.

Sentence	*Google*	*Sphinx*
I have a pet dog.	6.703986723	-2.7593
I have a pet wolf.	-19.73622328	-3.5635
I have a pet hamburger.	-44.73622328	-4.174458333

Some sentences in each group are incorrect in different ways, some of them are grammatically incorrect (e.g. "They plays the piano."), some others are semantically incorrect (e.g. "I have a pet hamburger."), and some are correct ("I run fast."). The sentences were ordered in the group by us in increasing order of incorrectness. Specifically, we constructed 8 groups containing 3 sentences each.

Different groups have been written to check for grammatical errors and semantic incoherence. As in the previous experiment, each group contains several sentences with some text modification. Each sentence is ran trough the evaluator and they are ranked according to the output of the evaluator. We then compared the ranking obtained by the evaluator against the ground truth provided by us using the same metric as in the previous section.

We compared the performance of the two evaluators described in Section 3.3 in the 8 groups of sentences. The Google PE using the 3-grams dataset gave the correct ordering for 4 out of the 8 test sets with an averaged squared error between the results and the ground truth of 0.333, while the PE using the CMU Sphinx database was able to correctly yield the expected order for 5 out of 8 test sets, with an average squared error between the results and the groudn truth of 0.250. Table 3 shows one example set and the results given by both evaluators.

The performance of both evaluators decreased as the length of the sentence increased, and, as expected, we observed that increasing the size of the n-gram dataset in use improves the evaluator performance. We ran the experiment using the 4-gram database for the Google PE and the numeric results matched the results from the CMU Sphinx. Going beyond 4-grams is hard in practice, due to memory requirements. For example, the *Google n-gram PE* requires 9GB of storage space for 3-grams and 14GB for 4-grams.

Moreover, the PE may use an external readily available database but using a dataset trained with domain-specific corpora, if available, could greatly improve the performance in certain scenarios.

4.3 CeBeTA

We designed a holistic experiment to confirm the validity of our system. For this experiment we collected a data set of 126 pairs of sentences, where each pair contains one source sentence, and a target sentence resulting from replacing a phrase in the source sentence by another phrase, and then fixing for grammatical and semantical errors. All the experiments reported here are the result of a 6-fold cross validation run.

In order to analyze the results, we used 3 different metrics:

1. Similarity between predicted solution and ground truth: we used the same simple distance metric based on the Ratcliff-Obershelp algorithm [3] we used for the reuse process to measure the distance between the solution provided by CeBeTA and the actual ground truth.
2. Percentage of times solution was amongst the candidates: the reuse process of CeBeTA returns a collection of candidate solutions. This metric measures whether the ground truth was one of those candidate solutions.
3. Probabilistic Evaluator: the third metric uses PE to evaluate the likelihood of the solution sentence being grammatically correct and semantically coherent.

During our experiments, we realized that one of the TTRs (*noun-to-pronoun*) was particularly problematic, since it would produce correct sentences towards which the PE was being biased, but which were further from the expected ground truth. Thus, we evaluated our system with and without that TTR. Our results are summarized in Table 4, as compared to a baseline approach that would just perform the requested text replacement without doing any further operations in the sentence.

Considering metric 1, we can see that both the baseline approach and the CeBeTA system produced sentences that were extremely similar to the ground truth. This is expected, since the task the system needs to perform is to replace a part of a sentence by another, and thus, most of the sentence is typically to be left unaltered. However, small differences, like verb morphology or a noun-pronoun coordination in a sentence are important, and thus, the improvement that we observe from the baseline system (obtaining sentences with average distance to ground truth of 0.0456) to the CeBeTA system (obtaining sentences with average distance to ground truth of up to 0.0367) is very significant. For example, the distance between a syntactically incorrect sentence as in our baseline like "Alice love Bob." to the ground truth ("Alice loves Bob.") is already very low, at 0.04. The numerical relevance of the improvement of a single letter is scaled down considerably as the sentences grow longer. Moreover, after carefully analyzing the results using this metric, we found out that sometimes the system was generating perfectly valid solutions that were being penalized because they were not actually matching the provided ground truth.

Considering metric 2, we see that a blind replacement only obtains the correct result a 30.952% of times, and that the CeBeTA system generates the correct result among its candidate solutions a much higher percentage of times (73.81%), even though the system only manages to yield the exact result as expected by the ground truth 55.96% of the time when using the Sphinx PE and 53.57% when using the Google PE.

Finally, metric 3 shows that sentences generated by the baseline approach are much less likely to be correct. The Sphinx PE gives the blind replacements an average score of -3.916 and the Google n-grams gives them a score of -32.031. Those scores go up to -3.522 and -22.337 respectively for the sentences produced by the CeBeTA system.

Table 4. Comparison of the performance of CeBeTA (both with and without the *noun-to-pronoun* (*toPP*) TTR activated) against a baseline system that would blindly perform the text replacement

	Metric 1	Metric 2	Metric 3
Baseline	0.0456	30.952%	-3.916 / -32.031
CeBeTA, Sphinx	0.0398	73.810%	-3.522
CeBeTA, Google	0.0397	73.810%	-22.337
CeBeTA (without *toPP*), Sphinx	0.0388	73.810%	-3.554
CeBeTA (without *toPP*), Google	0.0367	73.810%	-23.965

5 Related Work

Several areas of work are related to the approach presented in this paper, in this section we describe related approaches to assess similarity between sentences and CBR approaches using natural language.

Similarity assessment for natural language has been studied in the field of information retrieval. With a few exceptions, such as the work of Metzler et al. [8], similarity metrics for text focus on comparing large documents, and thus use representations, such as bag-of-words [7] or bag-of-concepts [13] that neglect the subtle syntactic and semantic details that are relevant for the work presented in this paper.

CBR approaches dealing with natural language (sometimes called *Textual CBR*, or TCBR) have been widely explored in the literature, but the vast majority of this work focuses on case retrieval. There has been an increased interested in the TCBR community in the past few years to seek and go beyond retrieval, although these are still the exception. Two representative efforts in this line are the CR2N system [2] that models reuse as selecting the appropriate portions of text from a retrieved document that could be reused, and some recent work in jCOLIBRI [10], that uses a mixed initiative approach where the user collaborates with the system in order to perform text reuse. In contrast, the adaptation approach presented in this paper is fully autonomous and automatically generates new sentences (using TTRs) to be candidate solutions, that are automatically evaluated using a probabilistic evaluator.

6 Conclusions

This paper has presented a case-based reasoning approach to natural language generation, and in particular to surface realization through text modification. We have presented the CeBeTA system, that combines a sentence similarity metric with a reuse approach based on text-transformation routines, in order to generate solutions to the text modification problem. Additionally, we have seen that by using an automatically trained statistical evaluator, it is possible to distinguish between correct and incorrect solutions.

The main contributions of this paper is a new approach to text adaptation based on text transformation routines (TTRs), and the idea of automatically inferring the *transformation path* from the retrieved case, as a way to capture the adaptations that need to be performed to the problem at hand in order to generate a solution. This idea allowed us to use unannotated cases in CeBeTA.

Our empirical evaluation showed promising results, although there is a lot of room for improvement. For example, a careful examination of the obtained results revealed that CeBeTA presents issues with entity handling, since the transformation path does not contain information concerning which specific entities in the sentence need to be transformed (it just contains their POS class). Existing work in the information extraction field on entity extraction could be used for this purpose [5]. Additionally, we want to evaluate the sensitivity of the system to the number of cases in the case-base, and integrate CeBeTA with the Riu system, which was the main motivation to develop CeBeTA.

References

[1] Aamodt, A., Plaza, E.: Case-based reasoning; foundational issues, methodological variations, and system approaches. AI Communications 7(1), 39–59 (1994)

[2] Adeyanju, I., Wiratunga, N., Lothian, R., Sripada, S., Lamontagne, L.: Case Retrieval Reuse Net (CR2N): An Architecture for Reuse of Textual Solutions. In: McGinty, L., Wilson, D.C. (eds.) ICCBR 2009. LNCS, vol. 5650, pp. 14–28. Springer, Heidelberg (2009)

[3] Apostolico, A.: String editing and longest common subsequences. In: Handbook of Formal Languages, pp. 361–398. Springer (1996)

[4] Catherine De Marneffe, M., Manning, C.D.: Stanford typed dependencies manual (2008)

[5] Etzioni, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D.S., Yates, A.: Unsupervised named-entity extraction from the web: An experimental study. Artificial Intelligence 165, 91–134 (2005)

[6] Gervás, P., Hervás, R., Recio-García, J.A.: The role of natural language generation during adaptation in textual cbr. In: Workshop on Textual Case-Based Reasoning: Beyond Retrieval, in 7th International Conference on Case-Based Reasoning (ICCBR 2007), Northern Ireland, pp. 227–235 (August 2007)

[7] Lewis, D.D.: Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 4–15. Springer, Heidelberg (1998)

[8] Metzler, D., Dumais, S., Meek, C.: Similarity measures for short segments of text. In: European Conference on Information Retrieval (2007)

[9] Ontañón, S., Zhu, J.: Story and Text Generation through Computational Analogy in the Riu System. In: AIIDE, pp. 51–56. The AAAI Press (2010)

[10] Recio-García, J.A., Díaz-Agudo, B., González-Calero, P.A.: Textual cbr in jcolibri: From retrieval to reuse. In: Wilson, D.C., Khemani, D. (eds.) Proceedings of the ICCBR 2007 Workshop on Textual Case-Based Reasoning: Beyond Retrieval, pp. 217–226 (August 2007)

[11] Reiter, E., Dale, R.: Building Natural Language Generation Systems (2000)

[12] Ristad, E.S., Yianilos, P.N.: Learning string-edit distance. IEEE Trans. Pattern Anal. Mach. Intell. 20, 522–532 (1998)

[13] Sahlgren, M., Cöster, R.: Using bag-of-concepts to improve the performance of support vector machines in text categorization. In: Proceedings of the 20th international conference on Computational Linguistics, COLING 2004. Association for Computational Linguistics, Stroudsburg (2004)