SEGUE: A Hybrid Case-Based Surface Natural Language Generator

Shimei Pan and James Shaw

IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
`shimei@us.ibm.com, shawjc@us.ibm.com`

Abstract. This paper presents Segue, a hybrid surface natural language generator that employs case-based paradigm but performs rule-based adaptations. It uses an annotated corpus as its knowledge source and employs grammatical rules to construct new sentences. By using *adaptation-guided retrieval* to select cases that can be adapted easily to the desired output, Segue simplifies the process and avoids generating ungrammatical sentences. The evaluation results show the system generates grammatically correct sentences (91%), but disfluency is still an issue.

1 Introduction

This paper presents an overview of Segue, a hybrid system that combines both case-based reasoning (CBR) and rule-based approaches for natural language generation (NLG). CBR is a machine learning paradigm that reuses prior solutions to solve new problems [1, 2]. It is closely related to instance-based learning where the representation of the instances is simpler (e.g. feature vectors).

There are several approaches to build a surface generator. Template-based systems are easy to develop, but the text they produce lacks variety. Rule-based systems, on the other hand, can produce more varied text, but developers with linguistic sophistication are required to develop and maintain the generators. Recently, statistics-based generation systems have achieved some success [3–5]. The main deterrent to the use of statistics-based generators is that they require a large training corpus. Without enough training instances, their performance degrades. A hybrid case-based generator can work with a relatively small annotated corpus. Because it uses rule-based adaptation to ensure the adapted sentences are grammatically correct, the hybrid generator is more accurate than statistical ones. More significantly, because CBR has learning capability, case-based generators can perform tasks more efficiently as they accumulate solutions.

This paper offers three significant contributions. First, we propose a case-based natural language generator with high accuracy that only requires a small corpus, thus easier to incorporate into applications than other corpus-based approaches. Second, We use rule-based adaptation operators to ensure that the generated sentences are grammatically correct, a property that other corpus-based approaches cannot guarantee. Third, to our knowledge, our work is the first to incorporate *adaptation-guided retrieval* in a case-based NLG system. Case retrieval in SEGUE is not only based on a similarity measure, but also on how easily the retrieved cases can be adapted to the desired outputs.

In the remainder of the paper, Section 2 provides an overview of SEGUE's system architecture. Section 3 describes the representation of cases in the case repository. The details of the case-based generation process are described in Section 4. An evaluation experiment of the generator is described in Section 5. Related work is discussed in Section 6. Section 7 summarizes our results and discusses future work.

2 System Overview

SEGUE is a domain independent surface generator currently embedded in a multimodal dialogue system called RealHunter[6]. Users interact with the system to identify potential real estate they would like to purchase in a particular region. Below is an example of a user interacting with RealHunter:

User: Show houses in the north.

RealHunter: I found 145 houses in northern Westchester county. I cannot display all the houses at once. Please give me more constraints.

User: Show houses along the Hudson River.

RealHunter: I found 23 houses based on your criteria. I cannot display all the houses at once. Steve. You can specify more constraints using asking price.

User: Just those with brick sidings.

RealHunter: I found 2 houses in northern Westchester County with brick sidings next to the Hudson River Shore. The houses are a Colonial and a Cape cod. House features, such as asking price and city name, are displayed on the screen.

In the first and second responses, RealHunter's strategic component determined that there are too many houses to display on the screen, so it asks for more constraints to reduce the number of retrieved houses. In the third response, the first sentence is formulated by using revision operators described in Section 4.2. In the second sentence, a conjunction operator is used.

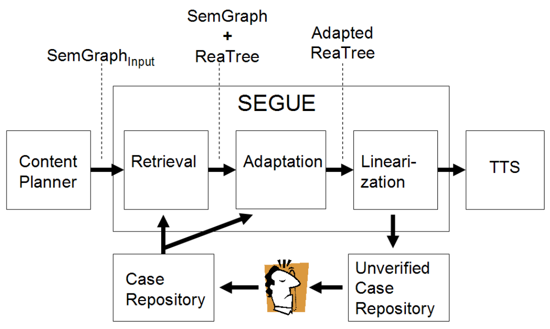
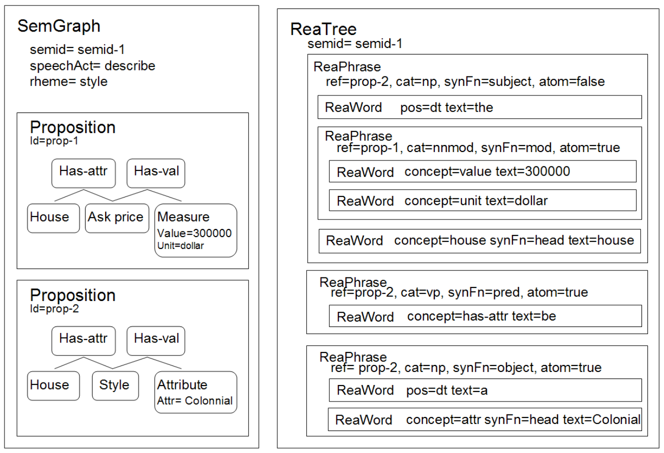


Fig. 1. Architecture of SEGUE

Fig. 1 shows the architecture of SEGUE. Similar to the standard NLG architecture, the dialogue system has a content planner that creates semantic representations. SEGUE takes the specified semantic information and transforms it into sentences. Its knowledge resource is a case repository. In the repository, each training sentence is associated with a semantic representation of the sentence (SemGraph), and a realization tree (ReaTree), its corresponding syntactic lexical representation. Further details of the SemGraphs and ReaTrees can be found in [7]. Text generation in SEGUE has three phases: *retrieval*, *adaptation*, and *learning*. In the retrieval phase, the SemGraph$_I$[1] is compared with SemGraphs in the case repository to identify a ranked list of similar SemGraphs for later adaptation. In the adaptation phase, SEGUE applies one or more adaptation operators to the ReaTrees corresponding to the SemGraphs identified in the retrieval phase. After the adaptation, the adapted ReaTree is sent to a linearization module which handles subject-verb agreements and other boundary friction issues [8]. In the learning phase, the SemGraph$_I$, its corresponding adapted ReaTree, and the generated sentences are first stored in a temporary case repository. These new cases are manually verified before being incorporated into the main case repository for future reuse. Currently, SEGUE sends the generated sentences to a commercial TTS system to transform them into speech. SEGUE is implemented in Java. On a 1.8-GHz P4-M machine with 512 MB memory, it takes about 0.2 seconds to generate sentences with an average length of 23 words.

3 Case Repository

The corpus used to establish SEGUE's case repository is described in [7]. A SemGraph in the case repository contains a list of propositions. A proposition represents roughly a clause-sized information, as shown in Fig. 2. Other information in a SemGraph includes speech act and theme/rheme information. The SemGraphs in the case repository support the computation of similarity measures in the retrieval phase.



SemGraph
semid= semid-1
speechAct= describe
rheme= style
Proposition
Id=prop-1
Has-attr
Has-val
House
Ask price
Measure
Value=300000
Unit=dollar
Proposition
Id=prop-2
Has-attr
Has-val
House
Style
Attribute
Attr= Colonnial
ReaTree
semid= semid-1
ReaPhrase
ref=prop-2, cat=np, synFn=subject, atom=false
ReaWord pos=dt text=the
ReaPhrase
ref=prop-1, cat=nnmod, synFn=mod, atom=true
ReaWord concept=value text=300000
ReaWord concept=unit text=dollar
ReaWord concept=house synFn=head text=house
ReaPhrase
ref=prop-2, cat=vp, synFn=pred, atom=true
ReaWord concept=has-attr text=be
ReaPhrase
ref= prop-2, cat=np, synFn=object, atom=true
ReaWord pos=dt text=a
ReaWord concept=attr synFn=head text=Colonial

Fig. 2. The SemGraph and ReaTree for "The 300,000 dollar house is a Colonial."

[1] In the rest of the paper, we use SemGraph$_I$ for "input semantic graph", and SemGraph$_C$ for "semantic graphs in the case repository."

The ReaTrees in the case repository are mainly used during the adaptation phase. A ReaTree contains a SemGraph ID and recursive phrase structures. The SemGraph ID establishes the correspondence between a SemGraph and a ReaTree. Because there can be several paraphrases for a particular SemGraph, multiple ReaTrees might have the same SemGraph ID. A ReaPhrase corresponds to a syntactic phrase that in turn contains recursive ReaPhrases and ReaWords, as shown in Fig. 2. A ReaPhrase typically has 4 attributes, *reference*, *syntactic category*, *syntactic function*, and *atomicity*. The *reference* attribute points to a specific proposition in the corresponding SemGraph. The *syntactic category* attribute specifies whether a ReaPhrase is realized as a noun phrase, an adjective phrase, or etc. The *syntactic function* attribute specifies that either the phrase is a subject, predicate, object, complement, or modifier. The *atomicity* attribute indicates that either the phrase realizes only one proposition or it is a composition of multiple propositions. A ReaWord typically has the following attributes: *concept*, *syntactic function*, and *text*. The *concept* attribute points to a particular semantic relation or concept in the proposition. The *syntactic function* attribute specifies whether the ReaWord is the head of a ReaPhrase. The *text* attribute represents the surface string used to realize the *concept*.

Depending on the complexity of the desired output sentences, the number of instances needed varies widely for different applications. Furthermore, this number is greatly influenced by the type of information annotated in the case repository. Brown [9] demonstrated that by encoding more linguistic information into a machine translation system, he reduced the required number of instances by a factor of six or more while retaining comparable translation quality. Currently there are 210 instances in the repository for the real-estate application.

4 Algorithm

The generation process in SEGUE has three phases: *retrieval*, *adaptation*, and *learning*.

4.1 Retrieval Phase

The goal of the retrieval phase is to identify a ranked list of instances in the case repository that can later be adapted with minimal effort to the new SemGraph. There are many ways to compute the similarity measure between two SemGraphs, such as using the string-edit distance [10] or computing the overlap of concepts in the semantic representation. In SEGUE, we compute the similarity measure based on the distance between all pairs of propositions in SemGraph$_C$ and SemGraph$_I$. From knowing which propositions are the same and different, the following adaptation operators can be created:

- *null()*: both SemGraph$_C$ and SemGraph$_I$ have the exact same propositions.
- *substitution(prop$_C$,prop$_I$)*: the propositions have the same relation but with different values. $Prop_C$ is a proposition in SemGraph$_C$ and $prop_I$ is a proposition in SemGraph$_I$.
- *deletion(prop$_C$)*: SemGraph$_C$ has the proposition $prop_C$, but SemGraph$_I$ does not.

– *insertion($prop_I$)*: SemGraph$_I$ has the proposition $prop_I$, but SemGraph$_C$ does not.

At this point in the retrieval process, even though which operation will be applied is known, whether it can be applied successfully is unknown. To retrieve cases with a high chance to be adapted successfully, we employ a technique known as *adaptation-guided retrieval* in CBR, where adaptation knowledge is integrated into the retrieval process. The rational behind this is that the most similar case may not be the most useful and easiest to adapt [11, 12]. As a result, we used the adaptation cost as the basis for case retrieval in SEGUE. The adaptation cost is estimated based on how likely an adaptation operator is to be successful. The cost for a *null* operator is zero. When applying a *substitution* operator, the overall sentence structure and the grammaticality of the final sentence are likely preserved. As a result, the cost of a *substitution* operator is relatively small. In contrast, the cost of applying *deletion* operators is higher because syntactic constituents are removed from the ReaTree$_C$ and might render the final sentences ungrammatical. The *insertion* operators have the highest cost because multiple sentences are involved and there are more opportunities for them to create ungrammatical sentences. By assigning lower cost to operators more likely to produce grammatically correct sentences, the retrieval in SEGUE is adaptation-guided.

Using only the adaptation operator cost described above, SEGUE sometimes still retrieves cases that are very similar at the semantic level, but are difficult to adapt at the syntactic level. This problem is particularly pronounced when the proposition realizing the main structure of the retrieved ReaTree is to be deleted. The following example illustrates the situation:

SemGraph$_I$: The {1995} house is a {Colonial}.

SemGraph$_C$: The {1995} {Colonial} house is {in Ardsley}.

In this example, SemGraph$_I$ is sent to SEGUE, and SemGraph$_C$ is retrieved for adaptation. Since the house attribute, *located-in-city*, does not exist in SemGraph$_I$, the phrases that express this information will be deleted from the ReaTree$_C$. In this particular example, applying the *deletion* operator to ReaTree$_C$ will remove both the verb "is" and the complement "in Ardsley" and render the generated sentence incomplete: "*The 1995 Colonial house." To prevent deletion of main verbs, we assign high costs to SemGraph candidates that do not share the main syntactic structure. This is another case of adaptation-guided retrieval. Later we will show it simplifies the *deletion* operation during adaptation.

4.2 Adaptation Phase

After the retrieval phase, a short list of similar SemGraphs have been retrieved and the list of adaptation operators to be applied for each SemGraph is computed. Among all the adaptation operators, we apply *deletion* before *insertion.* Because the operators manipulate the intermediate ReaTree, reducing the size of ReaTree first also reduces the complexity.

We have explored both statistical and rule-based approaches for adaptation. Currently, the rule-based approach is used because it requires fewer instances.

Substitution Operator The *substitution* operator takes two parameters: the proposition $prop_I$ from SemGraph$_I$ and its corresponding proposition $prop_C$ from SemGraph$_C$. Using the *ref* attribute in the ReaTree$_C$, a list of ReaPhrases matching the proposition $prop_C$ is retrieved. Then the operator goes through each ReaWord inside the retrieved ReaPhrases, verifying if the concept in each ReaWord is the same as those in the proposition $prop_I$. If they differ, a substitution appropriate for the particular concept is applied. After all ReaWords in the retrieved ReaPhrases are substituted, these ReaPhrases would convey the information expressed in the proposition, $prop_I$.

Deletion Operator In the retrieval phase, adaptation-guided retrieval is used to ensure that the proposition corresponding to the main structure of the retrieved ReaTree will not be deleted during the adaptation phase. This effort allows the *deletion* operators to be simple and ensures the soundness of the main sentence structure.

Deletion in SEGUE is basically a reverse-aggregation process. In the aggregation process, there are usually two ways to add new information into an existing sentence: *paratactic* or *hypotactic* transformations [13, 14]. Paratactic operators involve syntactically equivalent constituents, usually conjunctions. In contrast, hypotactic operators involve subordinate or modifying constructions, such as adjective or prepositional phrases. In SEGUE, there are two types of deletion: hypotactic and paratactic. For deleting hypotactic constructions, the *deletion* operator first recursively inspects all the ReaPhrases in the ReaTree$_C$ to retrieve those that express the propositions to be deleted. This is done through checking the *ref* attribute inside each ReaPhrase. If the *atomicity* attribute in the retrieved ReaPhrase is true, meaning it only expressed the proposition indicated in the *ref* attribute, then the ReaPhrase is removed from the ReaTree; otherwise, the system recursively inspects the subtree until an atomic ReaPhrase conveying the proposition is found. To delete constituents in paratactic constructions, a special procedure is used to ensure that the conjunctor "and" or "or" will be deleted or shifted to a grammatical correct position in the resulting phrase.

Insertion Operator *Insertion* operator is the most complex operator of all the adaptation operators. For propositions in the SemGraph$_I$ that do not exist in SemGraph$_C$, SEGUE incorporates ReaPhrases from various instances in the repository. In SEGUE, insertion is performed through two aggregation operations: *paratactic* and *hypotactic* operations.

Paratactic Operations. Without many training instances, conjunction is difficult to handle using pure case-based approaches. In SEGUE, paratactic operators use rules to avoid adding many instances. Currently, SEGUE supports two common types of paratactic operations: *quantification* ("**3** houses are Colonials. **1** is a Tudor."), and *simple conjunction* ("the names of the school districts are Lakeland School District **and** Panas School District.").

Both quantification and simple conjunction are performed mainly based on the semantic representations. When propositions from the SemGraph$_I$ are merged into the final ReaTree, the concepts in the new proposition are compared with the concepts in the final ReaTree. Depending on the results of the comparisons, the proposition is either merged into the final ReaTree using quantification or simple conjunction operator, or stored away, waiting to be realized as a part of a separate sentence. Currently, quantification and conjunctions in subject and object positions are implemented.

Hypotactic Operations. Hypotactic operators are the most used *insertion* operators in SEGUE. Applying hypotactic operators is a two-step process. In the first step, all the ReaTrees in the case repository are inspected to extract ReaPhrases that express the missing propositions. During this extraction process, the head modified by the extracted ReaPhrase is remembered, together with whether the ReaPhrase is used as a premodifier or post-modifier. In the second step, the extracted ReaPhrase is attached to the head constituent in the final ReaTree. Because SEGUE prefers ReaPhrases that can be realized in the shortest character length, a substitution operation (same as in Section 4.2) is performed on each of the extracted ReaPhrases to measure its length before selecting the shortest candidate. This heuristic results in the generation of concise sentences.

The hypotactic transformation operations in traditional rule-based generators also have these two similar steps [14]. In the first step, the rule-based system transforms the missing propositions into modifying constructions through a complex lexical process, which, in some cases, is domain dependent. In SEGUE, this transformation step is replaced by the extraction process to identify modifying constructions from the instances in the repository, which is much simpler and less error prone than the rule-based process. The second step in rule-based systems, attaching the modifying constructions to the final syntactic tree, is similar in SEGUE.

To ensure a variety of syntactic constructions in the generated sentence, we apply paratactic operators first because they have more restrictive preconditions. By applying them first, less restrictive hypotactic operators won't prevent paratactic operators from being applied. There are other ways to add new information into a sentence, such as through nominalization [13]. SEGUE currently generates these constructions by adding them into the case repository.

4.3 Learning Phase

If the operator applied in the adaptation phase is not trivial, the new SemGraph and adapted ReaTree are added to the case repository. For the purpose of text generation, an adaptation is considered non-trivial if either *deletion* or *insertion* operator is applied. If only *substitution* operator is applied, new case is not added to the case repository. Because not all the "learned" cases are fluent and grammatically correct, the "learned" cases are stored in a temporary case repository. Periodically, we manually run tests on those learned cases before integrating them into the repository.

Comparing to knowledge-intensive approaches, the CBR approach can learn from past experiences and become faster and more accurate over time. In rule-based generation systems [13, 14], however, a complex sentence is always built from scratch in a bottom-up manner – starting from a nucleus proposition and each satellite proposition is attached sequentially afterward. When a sentence is complex, such revision process is inefficient. In Segue, once a solution is proposed and verified, it can be incorporated into the case repository. Next time the system encounters similar requests, it shortcuts the process and reuses the old solution. This speeds up the process tremendously and decreases the likelihood of making mistakes during the assembling of complex sentences from pieces.

5 Evaluation

This evaluation focuses on the generation of assertive sentences related to house attributes. Segue also generates sentences expressing other speech acts, such as expressive ("I am sorry."), questions, and commands. In this real-estate application, a typical house has about 20 main attributes (e.g., asking price, property tax, and city location). During evaluation, we created a repository of 20 SemGraphs, with each SemGraph expressing a different attribute of a house. Using these SemGraphs as candidates, we exhaustively generated all combinations with maximum 5 propositions[2]. 100 of the 21,699 synthesized SemGraphs are randomly selected and sent to Segue to be transformed into sentences. Although the current setup led to empirically rare inputs, it gives an objective account of the generator's performance. If the semantic inputs are collected from natural interactions with users, Segue is likely to perform even better.

Two human evaluators graded the sentences using 3 ratings: **good** means all the sentences generated from a SemGraph are grammatical[3]; **minor grammatical error** indicates that some of the sentences have minor grammatical errors, such as subject-verb agreement; **major grammatical/pragmatic error** indicates that the sentences have major grammatical or pragmatic problems. Whenever there is a disagreement in the ratings, the evaluators reached an agreement after discussion. 91% of the generated cases were classified as **good**. There were no (0%) **minor grammatical errors**, and all the **major grammatical/pragmatic errors** (9%) involve multiple sentences and were caused by the lack of a referring expression module. For example, in the sentence "I found 3 Colonial houses with 0.2 acre of land. *The house* has an asphalt roof.", the subject in the second sentence should be a plural noun. For a baseline, we consider a substitution-only generator provided with the same input, using the same corpus instances as Segue. Under a best case assumption, each of the 210 corpus instances encodes a unique combination of propositions. Since only those input SemGraphs containing the identical set of propositions would be handled correctly by the baseline generator, this would result in a probability of less than 1% (210/21699) of the SemGraphs being generated correctly.

[2] Since generating all combinations in this setting is an exponential process, to make this possible, we restricted each sentence to have a maximum of 5 propositions. The number 5 is chosen because the number of propositions in a SemGraph from the content planner is almost always less than 5.

[3] Based on a sentence boundary determination algorithm, Segue may generate more than one sentence from a SemGraph$_I$.

In our evaluation, we did find instances where the generated sentences are grammatically correct, but contained disfluencies. In addition to sentences that are too long, disfluencies are often caused by the lack of referring expression module and inadequate adaptation operators. An example of inadequate adaptation operator occurred in the sentence "?I found 2 2-bedroom houses" where the two numbers appeared consecutively and sounds quite strangely without proper prosody. A better sentence to be generated is "I found 2 houses with 2 bedrooms" Other disfluencies include awkward modifier ordering, such as "?a Colonial 1995 house" instead of "a 1995 Colonial house." We believe after adding a referring expression module and incorporating more corpus information to address disfluency issues, many of the major grammatical/pragmatic errors and disfluencies will be resolved.

6 Related Work

In recent years, researchers have successfully applied machine learning techniques to the generation task [3–5, 15, 16]. These works can be described as "overgeneration-and-ranking"[17]. In these works, a large number of candidates are first generated either using corpus information or rule-based transformations. Then, an evaluation function is used to rank these candidates to pick out the best one to realize the desired input. Except for IGEN [16], they required large corpora for training. If they use grammatical rules, these rules are used during the overgeneration stage and not in the ranking stage [3, 4]. In contrast, our work can be described as "retrieve-and-adapt". SEGUE retrieves grammatically correct sentences and adapts the retrieved sentences using grammatical rules. It can guarantee the generated sentences will be grammatically correct – a property that other machine learning approaches do not have.

Researchers have applied CBR techniques to example-based machine translation (EBMT) [8, 9, 18, 19]. Typical EBMT input representation is a sequence of words which is very different from the semantic representation in SEGUE. Furthermore in SEGUE, the examples consist of a semantic representation and a syntactic tree, but in EBMT systems, the examples consist solely of strings or syntactic trees of different languages. Due to this marked difference in the input and representation, the adaptation operations in SEGUE and EBMT systems are drastically different.

7 Conclusion

We have presented a hybrid surface natural language generator that uses a case-based paradigm, but performs rule-based adaptations. By combining the benefits of both approaches, we were able to implement a natural language generator that uses a small training corpus but is fast, accurate, and extensible. For developing a dialogue system in a new domain, the requirements of deploying our approach are more manageable than statistical approaches. By requiring only a small annotated corpus and using accurate rule-based adaptation operators, SEGUE can be customized for new application with less effort. To our knowledge, incorporating adaptation-guided retrieval into a generation system is the first of its kind.

In the future, we want to build tools to facilitate annotation and demonstrate such a hybrid approach can significantly decrease the amount of effort needed to build NLG systems for different applications.

8 Acknowledgments

We would like to thank the anonymous reviewers for providing many useful suggestions. This material is based upon work supported by the Adventurous Research Program in IBM.

References

1. Kolodner, J.: Case-Based Reasoning. Morgan Kaufmann (1993)
2. Aamodt, A., Plaza, E.: Case-based reasoning; foundational issues, methodological variations, and system approaches. AI Communications **7** (1994)
3. Langkilde, I., Knight, K.: Generation that exploits corpus-based statistical knowledge. In: Proc. of the COLING and the ACL., Montreal, Canada (1998)
4. Bangalore, S., Rambow, O.: Exploiting a probabilistic hierarchical model for generation. In: Proc. of the COLING. (2000)
5. Ratnaparkhi, A.: Trainable methods for surface natural language generation. In: Proc. of the NAACL, Seattle, WA (2000)
6. Chai, J., Pan, S., Zhou, M., Houck, K.: Context-based multimodal understanding in conversational systems. In: Proc. of International Conference on Multimodal Interfaces. (2002)
7. Pan, S., Weng, W.: Designing a speech corpus for instance-based spoken language generation. In: Proc. of INLG, New York (2002)
8. Somers, H.: EBMT seen as case-based reasoning. In: MT Summit VIII Workshop on Example-Based Machine Translation, Santiago de Compostela, Spain (2001)
9. Brown, R.D.: Adding linguistic knowledge to a lexical example-based translation system. In: Proc. of the International Conference on Theoretical and Methodological Issues in Machine Translation, Chester, UK (1999)
10. Ristad, E.S., Yianilos, P.N.: Learning string edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence **20** (1998)
11. Smyth, B., Keane, M.T.: Experiments on adaptation-guided retrieval in case-based design. In: Proc. of the ICCBR. (1995)
12. Leake, D.B.: Adaptive similarity assessment for case-based explanation. International Journal of Expert Systems **8** (1995)
13. Robin, J., McKeown, K.R.: Corpus analysis for revision-based generation of complex sentences. In: Proc. of AAAI, Washington, DC (1993)
14. Shaw, J.: Clause aggregation using linguistic knowledge. In: Proc. of the IWNLG. (1998)
15. Walker, M., Rambow, O., Rogati, M.: SPot: A trainable sentence planner. In: Proc. of the NAACL. (2001)
16. Varges, S., Mellish, C.: Instance-based natural language generation. In: Proc. of the NAACL, Pittsburgh, PA (2001)
17. Oberlander, J., Brew, C.: Stochastic text generation. Philosophical Transactions of the Royal Society **358** (2000)
18. Collins, B., Cuningham, P.: Adaptation-guided retrieval in EBMT: A case-based approach to machine translation. In: Proc. of EWCBR. (1996)
19. Somers, H.: Example-based machine translation. Machine Translation **14** (1999)