Error Mining on Dependency Trees

Claire Gardent, Shashi Narayan

Error Mining on Dependency Trees

Claire Gardent
CNRS, LORIA, UMR 7503
Vandoeuvre-lès-Nancy, F-54500, France
claire.gardent@loria.fr

Shashi Narayan
Université de Lorraine, LORIA, UMR 7503
Villers-lès-Nancy, F-54600, France
shashi.narayan@loria.fr

Abstract

In recent years, error mining approaches were developed to help identify the most likely sources of parsing failures in parsing systems using handcrafted grammars and lexicons. However the techniques they use to enumerate and count n-grams builds on the sequential nature of a text corpus and do not easily extend to structured data. In this paper, we propose an algorithm for mining trees and apply it to detect the most likely sources of generation failure. We show that this tree mining algorithm permits identifying not only errors in the generation system (grammar, lexicon) but also mismatches between the structures contained in the input and the input structures expected by our generator as well as a few idiosyncracies/error in the input data.

1 Introduction

In recent years, error mining techniques have been developed to help identify the most likely sources of parsing failure (van Noord, 2004; Sagot and de la Clergerie, 2006; de Kok et al., 2009). First, the input data (text) is separated into two subcorpora, a corpus of sentences that could be parsed (PASS) and a corpus of sentences that failed to be parsed (FAIL). For each n-gram of words (and/or part of speech tag) occurring in the corpus to be parsed, a suspicion rate is then computed which, in essence, captures the likelihood that this n-gram causes parsing to fail.

These error mining techniques have been applied with good results on parsing output and shown to help improve the large scale symbolic grammars and lexicons used by the parser. However the techniques they use (e.g., suffix arrays) to enumerate and count n-grams builds on the sequential nature of a text corpus and cannot easily extend to structured data.

There are some NLP applications though where the processed data is structured data such as trees or graphs and which would benefit from error mining. For instance, when generating sentences from dependency trees, as was proposed recently in the Generation Challenge Surface Realisation Task (SR Task, (Belz et al., 2011)), it would be useful to be able to apply error mining on the input trees to find the most likely causes of generation failure.

In this paper, we address this issue and propose an approach that supports error mining on trees. We adapt an existing algorithm for tree mining which we then use to mine the Generation Challenge dependency trees and identify the most likely causes of generation failure. We show in particular, that this tree mining algorithm permits identifying not only errors in the grammar and the lexicon used by generation but also a few idiosyncracies/error in the input data as well as mismatches between the structures contained in the SR input and the input structures expected by our generator. The latter is an important point since, for symbolic approaches, a major hurdle to participation in the SR challenge is known to be precisely these mismatches i.e., the fact that the input provided by the SR task fails to match the input expected by the symbolic generation systems (Belz et al., 2011).

The paper is structured as follows. Section 2 presents the HybridTreeMiner algorithm, a complete and computationally efficient algorithm developed

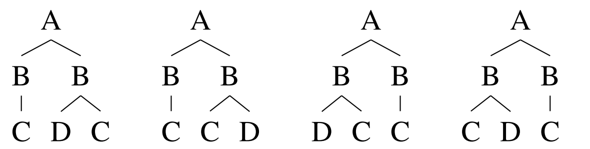

Figure 1: Four unordered labelled trees. The rightmost is in Breadth-First Canonical Form

by (Chi et al., 2004) for discovering frequently occurring subtrees in a database of labelled unordered trees. Section 3 shows how to adapt this algorithm to mine the SR dependency trees for subtrees with high suspicion rate. Section 4 presents an experiment we made using the resulting tree mining algorithm on SR dependency trees and summarises the results. Section 5 discusses related work. Section 6 concludes.

2 Mining Trees

Mining for frequent subtrees is an important problem that has many applications such as XML data mining, web usage analysis and RNA classification. The HybridTreeMiner (HTM) algorithm presented in (Chi et al., 2004) provides a complete and computationally efficient method for discovering frequently occurring subtrees in a database of labelled unordered trees and counting them. We now sketch the intuition underlying this algorithm[1]. In the next section, we will show how to modify this algorithm to mine for errors in dependency trees.

Given a set of trees T, the HybridTreeMiner algorithm proceeds in two steps. First, the unordered labelled trees contained in T are converted to a canonical form called BFCF (Breadth-First Canonical Form). In that way, distinct instantiations of the same unordered trees have a unique representation. Second, the subtrees of the BFCF trees are enumerated in increasing size order using two tree operations called join and extension and their support (the number of trees in the database that contains each subtree) is recorded. In effect, the algorithm builds *an enumeration tree* whose nodes are the possible subtrees of T and such that, at depth d of this enumeration tree, all possible frequent subtrees consisting of d nodes are listed.

[1]For a more complete definition see (Chi et al., 2004).

The BFCF canonical form of an unordered tree is an ordered tree t such that t has the smallest breath-first canonical string (BFCS) encoding according to lexicographic order. The BFCS encoding of a tree is obtained by breadth-first traversal of the tree, recording the string labelling each node, "$" to separate siblings with distinct parents and "#" to represent the end of the tree[2]. For instance, the BFCS encodings of the four trees shown in Figure 1 are 'ABBC$DC#', 'A$BBCCD#', 'ABBDC$C#' and 'A$BBCDC#' respectively. Hence, the rightmost tree is the BFCF of all four trees.

The join and extension operations used to iteratively enumerate subtrees are depicted in Figure 2 and can be defined as follows.

- A *leg* is a leaf of maximal depth.
- Extension: Given a tree t of height h_t and a node n, extending t with n yields a tree t' (a child of t in the enumeration tree) with height $h_{t'}$ such that n is a child of one of t's legs and $h_{t'}$ is $h_t + 1$.
- Join: Given two trees t_1 and t_2 of same height h differing only in their rightmost leg and such that t_1 sorts lower than t_2, joining t_1 and t_2 yields a tree t' (a child of t_1 in the enumeration tree) of same height h by adding the rightmost leg of t_2 to t_1 at level $h - 1$.

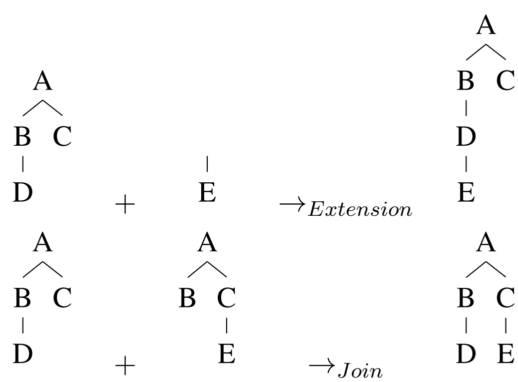

Figure 2: Join and Extension Operations

To support counting, the algorithm additionally records for each subtree a list (called *occurrence list*)

[2]Assuming "#" sorts greater than "$" and both sort greater than any other alphabets in node labels.

of all trees in which this subtree occurs and of its position in the tree (represented by the list of tree nodes mapped onto by the subtree). Thus for a given subtree t, the *support* of t is the number of elements in that list. Occurrence lists are also used to check that trees that are combined occur in the data. For the join operation, the subtrees being combined must occur in the same tree at the same position (the intersection of their occurrence lists must be non empty and the tree nodes must match except the last node). For the extension operation, the extension of a tree t is licensed for any given occurrence in the occurrence list only if the planned extension maps onto the tree identified by the occurrence.

3 Mining Dependency Trees

We develop an algorithm (called ErrorTreeMiner, ETM) which adapts the HybridTreeMiner algorithm to mine sources of generation errors in the Generation Challenge SR shallow input data. The main modification is that instead of simply counting trees, we want to compute their suspicion rate. Following (de Kok et al., 2009), we take the suspicion rate of a given subtree t to be the proportion of cases where t occurs in an input tree for which generation fails:

$$Sus(t) = \frac{count(t|\text{FAIL})}{count(t)}$$

where $count(t)$ is the number of occurrences of t in all input trees and $count(t|\text{FAIL})$ is the number of occurrences of t in input trees for which no output was produced.

Since we work with subtrees of arbitrary length, we also need to check whether constructing a longer subtree is useful that is, whether its suspicion rate is equal or higher than the suspicion rate of any of the subtrees it contains. In that way, we avoid computing all subtrees (thus saving time and space). As noted in (de Kok et al., 2009), this also permits bypassing *suspicion sharing* that is the fact that, if n_2 is the cause of a generation failure, and if n_2 is contained in larger trees n_3 and n_4, then all three trees will have high suspicion rate making it difficult to identify the actual source of failure namely n_2. Because we use a milder condition however (we accept bigger trees whose suspicion rate is *equal* to the suspicion rate of any of their subtrees), some amount of

Algorithm 1 ErrorTreeMiner($D, minsup$)

```
Note: D consists of D_fail and D_pass
F_1 ← {Frequent 1-trees}
F_2 ← ∅
for i ← 1, ..., |F_1| do
  for j ← 1, ..., |F_1| do
    q ← f_i plus legf_j
    if Noord-Validation(q, minsup) then
      F_2 ← F_2 ∪ q
    end if
  end for
end for
F ← F_1 ∪ F_2
PUSH: sort(F_2) → L_Queue
Enum-Grow(L_Queue, F, minsup)
return F
```

Algorithm 2 Enum-Grow($L_{Queue}, F, minsup$)

```
while L_Queue ≠ empty do
  POP: pop(L_Queue) → C
  for i ← 1, ..., |C| do
    ◇The join operation
    J ← ∅
    for j ← i, ..., |C| do
      p ← join(c_i, c_j)
      if Noord-Validation(p, minsup) then
        J ← J ∪ p
      end if
    end for
    F ← F ∪ J
    PUSH: sort(J) → L_Queue
    ◇The extension operation
    E ← ∅
    for possible leg l_m of c_i do
      for possible new leg l_n(∈ F_1) do
        q ← extend c_i with l_n at position l_m
        if Noord-Validation(q, minsup) then
          E ← E ∪ q
        end if
      end for
    end for
    F ← F ∪ E
    PUSH: sort(E) → L_Queue
  end for
end while
```

Algorithm 3 Noord-Validation($t_n, minsup$)

Note: $t_n,$ tree with n nodes
if $Sup(t_n) \geq minsup$ **then**
  if $Sus(t_n) \geq Sus(t_{n-1}), \forall t_{n-1}$ in t_n **then**
    return true
  end if
end if
return false

suspicion sharing remains. As we shall see in Section 4.3.2, relaxing this check though allows us to extract frequent larger tree patterns and thereby get a more precise picture of the context in which highly suspicious items occur.

Finally, we only keep subtrees whose support is above a given threshold where the support $Sup(t)$ of a tree t is defined as the ratio between the number of times it occurs in an input for which generation fails and the total number of generation failures:

$$Sup(t) = \frac{count(t|\text{FAIL})}{count(FAIL)}$$

The modified algorithm we use for error mining is given in Algorithm 1, 2 and 3. It can be summarised as follows.

First, dependency trees are converted to Breadth-First Canonical Form whereby lexicographic order can apply to the word forms labelling tree nodes, to their part of speech, to their dependency relation or to any combination thereof[3].

Next, the algorithm iteratively enumerates the subtrees occurring in the input data in increasing size order and associating each subtree t with two occurrence lists namely, the list of input trees in which t occurs and for which generation was successful (PASS(t)); and the list of input trees in which t occurs and for which generation failed (FAIL(t)).

This process is initiated by building trees of size one (i.e., one-node tree) and extending them to trees of size two. It is then continued by extending the trees using the join and extension operations. As explained in Section 2 above, join and extension only apply provided the resulting trees occur in the data (this is checked by looking up occurrence lists). Each time an n-node tree t_n, is built, it is checked that (i) its support is above the set threshold and (ii) its suspicion rate is higher than or equal to the suspicion rate of all $(n-1)$-node subtrees of t_n.

In sum, the ETM algorithm differs from the HTM algorithm in two main ways. First, while HTM explores the enumeration tree depth-first, ETM proceeds breadth-first to ensure that the suspicion rate of (n-1)-node trees is always available when checking whether an n-node tree should be introduced. Second, while the HTM algorithm uses support to prune the search space (only trees with a minimum support bigger than the set threshold are stored), the ETM algorithm drastically prunes the search space by additionally checking that the suspicion rate of all subtrees contained in a new tree t is smaller or equal to the suspicion rate of t . As a result, while ETM looses the space advantage of HTM by a small margin[4], it benefits from a much stronger pruning of the search space than HTM through suspicion rate checking. In practice, the ETM algorithm allows us to process e.g., all NP chunks of size 4 and 6 present in the SR data (roughly 60 000 trees) in roughly 20 minutes on a PC.

4 Experiment and Results

Using the input data provided by the Generation Challenge SR Task, we applied the error mining algorithm described in the preceding Section to debug and extend a symbolic surface realiser developed for this task.

4.1 Input Data and Surface Realisation System

The shallow input data provided by the SR Task was obtained from the Penn Treebank using the LTH Constituent-to-Dependency Conversion Tool for Penn-style Treebanks (Pennconverter, (Johansson and Nugues, 2007)). It consists of a set of unordered labelled syntactic dependency trees whose nodes are labelled with word forms, part of speech categories, partial morphosyntactic information such as tense and number and, in some cases, a sense tag identifier. The edges are labelled with the syntactic labels provided by the Pennconverter. All words (including punctuation) of the original sen-

[3]For convenience, the dependency relation labelling the edges of dependency trees is brought down to the daughter node of the edge.

[4]ETM needs to store all (n-1)-node trees in queues before producing n-node trees.

tence are represented by a node in the tree and the alignment between nodes and word forms was provided by the organisers.

The surface realiser used is a system based on a Feature-Based Lexicalised Tree Adjoining Grammar (FB-LTAG) for English extended with a unification based compositional semantics. Both the grammars and the lexicon were developed in view of the Generation Challenge and the data provided by this challenge was used as a means to debug and extend the system. Unknown words are assigned a default TAG family/tree based on the part of speech they are associated with in the SR data. The surface realisation algorithm extends the algorithm proposed in (Gardent and Perez-Beltrachini, 2010) and adapts it to work on the SR dependency input rather than on flat semantic representations.

4.2 Experimental Setup

To facilitate interpretation, we first chunked the input data in NPs, PPs and Clauses and performed error mining on the resulting sets of data. The chunking was performed by retrieving from the Penn Treebank (PTB), for each phrase type, the yields of the constituents of that type and by using the alignment between words and dependency tree nodes provided by the organisers of the SR Task. For instance, given the sentence *"The most troublesome report may be the August merchandise trade deficit due out tomorrow"*, the NPs *"The most troublesome report"* and *"the August merchandise trade deficit due out tomorrow"* will be extracted from the PTB and the corresponding dependency structures from the SR Task data.

Using this chunked data, we then ran the generator on the corresponding SR Task dependency trees and stored separately, the input dependency trees for which generation succeeded and the input dependency trees for which generation failed. Using information provided by the generator, we then removed from the failed data, those cases where generation failed either because a word was missing in the lexicon or because a TAG tree/family was missing in the grammar but required by the lexicon and the input data. These cases can easily be detected using the generation system and thus do not need to be handled by error mining.

Finally, we performed error mining on the data using different minimal support thresholds, different display modes (sorted first by size and second by suspicion rate *vs* sorted by suspicion rate) and different labels (part of speech, words and part of speech, dependency, dependency and part of speech).

4.3 Results

One feature of our approach is that it permits mining the data for tree patterns of arbitrary size using different types of labelling information (POS tags, dependencies, word forms and any combination thereof). In what follows, we focus on the NP chunk data and illustrate by means of examples how these features can be exploited to extract complementary debugging information from the data.

4.3.1 Mining on single labels (word form, POS tag or dependency)

Mining on a single label permits (i) assessing the relative impact of each category in a given label category and (ii) identifying different sources of errors depending on the type of label considered (POS tag, dependency or word form).

Mining on POS tags Table 1 illustrates how mining on a single label (in this case, POS tags) gives a good overview of how the different categories in that label type impact generation: two POS tags (POS and CC) have a suspicion rate of 0.99 indicating that these categories always lead generation to fail. Other POS tag with much lower suspicion rate indicate that there are unresolved issues with, in decreasing order of suspicion rate, cardinal numbers (CD), proper names (NNP), nouns (NN), prepositions (IN) and determiners (DT).

The highest ranking category (POS[5]) points to a mismatch between the representation of genitive NPs (e.g., *John's father*) in the SR Task data and in the grammar. While our generator expects the representation of 'John's father' to be FATHER("S"(JOHN)), the structure provided by the SR Task is FATHER(JOHN("S")). Hence whenever a possessive appears in the input data, generation fails. This is in line with (Rajkumar et al., 2011)'s finding that the logical forms expected by their system for possessives differed from the shared task inputs.

[5] In the Penn Treebank, the POS tag is the category assigned to possessive *'s*.

POS	Sus	Sup	Fail	Pass
POS	0.99	0.38	3237	1
CC	0.99	0.21	1774	9
CD	0.39	0.16	1419	2148
NNP	0.35	0.32	2749	5014
NN	0.30	0.81	6798	15663
IN	0.30	0.16	1355	3128
DT	0.09	0.12	1079	10254

Table 1: Error Mining on POS tags with frequency cutoff 0.1 and displaying only trees of size 1 sorted by decreasing suspicion rate (Sus)

The second highest ranked category is CC for coordinations. In this case, error mining unveils a bug in the grammar trees associated with conjunction which made all sentences containing a conjunction fail. Because the grammar is compiled out of a strongly factorised description, errors in this description can propagate to a large number of trees in the grammar. It turned out that an error occurred in a class inherited by all conjunction trees thereby blocking the generation of any sentence requiring the use of a conjunction.

Next but with a much lower suspicion rate come cardinal numbers (CD), proper names (NNP), nouns (NN), prepositions (IN) and determiners (DT). We will see below how the richer information provided by mining for larger tree patterns with mixed labelling information permits identifying the contexts in which these POS tags lead to generation failure.

Mining on Word Forms Because we remove from the failure set all cases of errors due to a missing word form in the lexicon, a high suspicion rate for a word form usually indicates a missing or incorrect lexical entry: the word is present in the lexicon but associated with either the wrong POS tag and/or the wrong TAG tree/family. To capture such cases, we therefore mine not on word forms alone but on pairs of word forms and POS tag. In this way, we found for instance, that cardinal numbers induced many generation failures whenever they were categorised as determiners but not as nouns in our lexicon. As we will see below, larger tree patterns help identify the specific contexts inducing such failures.

One interesting case stood out which pointed to idiosyncracies in the input data: The word form *$* (Sus=1) was assigned the POS tag $ in the input data, a POS tag which is unknown to our system and not documented in the SR Task guidelines. The SR guidelines specify that the Penn Treebank tagset is used modulo the modifications which are explicitly listed. However for the $ symbol, the Penn treebank used SYM as a POS tag and the SR Task $, but the modification is not listed. Similarly, while in the Penn treebank, punctuations are assigned the SYM POS tag, in the SR data "," is used for the comma, "(" for an opening bracket and so on.

Mining on Dependencies When mining on dependencies, suspects can point to syntactic constructions (rather than words or word categories) that are not easily spotted when mining on words or parts of speech. Thus, while problems with coordination could easily be spotted through a high suspicion rate for the CC POS tag, some constructions are linked neither to a specific POS tag nor to a specific word. This is the case, for instance, for apposition which a suspicion rate of 0.19 (286F/1148P) identified as problematic. Similarly, a high suspicion rate (0.54, 183F/155P) on the TMP dependency indicates that temporal modifiers are not correctly handled either because of missing or erroneous information in the grammar or because of a mismatch between the input data and the fomat expected by the surface realiser.

Interestingly, the underspecified dependency relation DEP which is typically used in cases for which no obvious syntactic dependency comes to mind shows a suspicion rate of 0.61 (595F/371P).

4.3.2 Mining on trees of arbitrary size and complex labelling patterns

While error mining with tree patterns of size one permits ranking and qualifying the various sources of errors, larger patterns often provide more detailed contextual information about these errors. For instance, Table 1 shows that the CD POS tag has a suspicion rate of 0.39 (1419F/2148P). The larger tree patterns identified below permits a more specific characterization of the context in which this POS tag co-occurs with generation failure:

TP1	CD(IN,RBR)	*more than 10*
TP2	IN(CD)	*of 1991*
TP3	NNP(CD)	*November 1*
TP4	CD(NNP(CD))	*Nov. 1, 1997*

Two patterns clearly emerge: a pattern where cardinal numbers are parts of a date (tree patterns TP2-TP4) and a more specific pattern (TP1) involving the comparative construction (e.g., *more than 10*). All these patterns in fact point to a missing category for cardinals in the lexicon: they are only associated with determiner TAG trees, not nouns, and therefore fail to combine with prepositions (e.g., *of 1991, than 10*) and with proper names (e.g., *November 1*).

For proper names (NNP), dates also show up because months are tagged as proper names (TP3,TP4) as well as addresses TP5:

TP5 NNP(",",",") *Brooklyn, n.y.,*

For prepositions (IN), we find, in addition to the TP1-TP2, the following two main patterns:

TP6 DT(IN) *those with, some of*

TP7 RB(IN) *just under, little more*

Pattern TP6 points to a missing entry for words such as *those* and *some* which are categorised in the lexicon as determiners but not as nouns. TP7 points to a mismatch between the SR data and the format expected by the generator: while the latter expects the structure IN(RB), the input format provided by the SR Task is RB(IN).

4.4 Improving Generation Using the Results of Error Mining

Table 2 shows how implementing some of the corrections suggested by error mining impacts the number of NP chunks (size 4) that can be generated. In this experiment, the total number of input (NP) dependency trees is 24995. Before error mining, generation failed on 33% of these input. Correcting the erroneous class inherited by all conjuction trees mentioned in Section 4.3.1 brings generation failure down to 26%. Converting the input data to the correct input format to resolve the mismatch induced by possessive *'s* (cf. Section 4.3.1) reduce generation failure to 21%[6] and combining both corrections results in a failure rate of 13%. In other words, error mining permits quickly identifying two issues which, once corrected, reduces generation failure by 20 points.

When mining on clause size chunks, other mismatches were identified such as in particular, mismatches introduced by subjects and auxiliaries: while our generator expects both the subject and the auxiliary to be children of the verb, the SR data represent the subject and the verb as children of the auxiliary.

[6] For NP of size 4, 3264 structures with possessive *'s* were rewritten.

NP 4	Before	After
SR Data	8361	6511
Rewritten SR Data	5255	3401

Table 2: Diminishing the number of errors using information from error mining. The table compares the number of failures on NP chunks of size 4 before (first row) and after (second row) rewritting the SR data to the format expected by our generator and before (second column) and after (third column) correcting the grammar and lexicon errors discussed in Section 4.3.1

5 Related Work

We now relate our proposal (i) to previous proposals on error mining and (ii) to the use of error mining in natural language generation.

Previous work on error mining. (van Noord, 2004) initiated error mining on parsing results with a very simple approach computing the parsability rate of each n-gram in a very large corpus. The parsability rate of an n-gram $w_i \dots w_n$ is the ratio $R(w_i \dots w_n) = \frac{C(w_i \dots w_n|OK)}{C(w_i \dots w_n)}$ with $C(w_i \dots w_n)$ the number of sentences in which the n-gram $w_i \dots w_n$ occurs and $C(w_i \dots w_n \mid OK)$ the number of sentences containing $w_i \dots w_n$ which could be parsed. The corpus is stored in a suffix array and the sorted suffixes are used to compute the frequency of each n-grams in the total corpus and in the corpus of parsed sentences. The approach was later extended and refined in (Sagot and de la Clergerie, 2006) and (de Kok et al., 2009) whereby (Sagot and de la Clergerie, 2006) defines a suspicion rate for n-grams which takes into account the number of occurrences of a given word form and iteratively defines the suspicion rate of each word form in a sentence based on the suspicion rate of this word form in the corpus; (de Kok et al., 2009) combined the iterative error mining proposed by (Sagot and de la Clergerie, 2006) with expansion of forms to n-grams of words and POS tags of arbitrary length.

Our approach differs from these previous approaches in several ways. First, error mining is performed on trees. Second, it can be parameterised to use any combination of POS tag, dependency and/or word form information. Third, it is applied to generation input rather than parsing output. Typically, the input to surface realisation is a structured representation (i.e., a flat semantic representation, a first order logic formula or a dependency tree) rather than a string. Mining these structured representations thus permits identifying causes of undergeneration in surface realisation systems.

Error Mining for Generation Not much work has been done on mining the results of surface realisers. Nonetheless, (Gardent and Kow, 2007) describes an error mining approach which works on the output of surface realisation (the generated sentences), manually separates correct from incorrect output and looks for derivation items which systematically occur in incorrect output but not in correct ones. In contrast, our approach works on the input to surface realisation, automatically separates correct from incorrect items using surface realisation and targets the most likely sources of errors rather than the absolute ones.

More generally, our approach is the first to our knowledge, which mines a surface realiser for undergeneration. Indeed, apart from (Gardent and Kow, 2007), most previous work on surface realisation evaluation has focused on evaluating the performance and the coverage of surface realisers. Approaches based on reversible grammars (Carroll et al., 1999) have used the semantic formulae output by parsing to evaluate the coverage and performance of their realiser; similarly, (Gardent et al., 2010) developed a tool called GenSem which traverses the grammar to produce flat semantic representations and thereby provide a benchmark for performance and coverage evaluation. In both cases however, because it is produced using the grammar exploited by the surface realiser, the input produced can only be used to test for overgeneration (and performance) . (Callaway, 2003) avoids this shortcoming by converting the Penn Treebank to the format expected by his realiser. However, this involves manually identifying the mismatches between two formats much like symbolic systems did in the Generation Challenge SR Task. The error mining approach we propose helps identifying such mismatches automatically.

6 Conclusion

Previous work on error mining has focused on applications (parsing) where the input data is sequential working mainly on words and part of speech tags. In this paper, we proposed a novel approach to error mining which permits mining trees. We applied it to the input data provided by the Generation Challenge SR Task. And we showed that this supports the identification of gaps and errors in the grammar and in the lexicon; and of mismatches between the input data format and the format expected by our realiser.

We applied our error mining approach to the input of a surface realiser to identify the most likely sources of *under-generation*. We plan to also explore how it can be used to detect the most likely sources of *over-generation* based on the output of this surface realiser on the SR Task data. Using the Penn Treebank sentences associated with each SR Task dependency tree, we will create the two tree sets necessary to support error mining by dividing the set of trees output by the surface realiser into a set of trees (FAIL) associated with overgeneration (the generated sentences do not match the original sentences) and a set of trees (SUCCESS) associated with success (the generated sentence matches the original sentences). Exactly which tree should populate the SUCCESS and FAIL set is an open question. The various evaluation metrics used by the SR Task (BLEU, NIST, METEOR and TER) could be used to determine a threshold under which an output is considered incorrect (and thus classificed as FAIL). Alternatively, a strict matching might be required. Similarly, since the surface realiser is non deterministic, the number of output trees to be kept will need to be experimented with.

Acknowledgments

We would like to thank Clément Jacq for useful discussions on the hybrid tree miner algorithm. The research presented in this paper was partially supported by the European Fund for Regional Development within the framework of the INTERREG IV A Allegro Project.

References

Anja Belz, Michael White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. 2011. The first surface realisation shared task: Overview and evaluation results. In *Proc. of the 13th European Workshop on Natural Language Generation*.

Charles B. Callaway. 2003. Evaluating coverage for large symbolic NLG grammars. In *18th IJCAI*, pages 811–817, Aug.

John Carroll, Ann Copestake, Dan Flickinger, and Viktor Paznański. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of EWNLG '99*.

Yun Chi, Yirong Yang, and Richard R. Muntz. 2004. Hybridtreeminer: An efficient algorithm for mining frequent rooted trees and free trees using canonical form. In *SSDBM'04: Proceedings of the 16th International Conference on and Statistical Database Management*, pages 11–20, Santorini Island, Greece. IEEE Computer Society.

Daniël de Kok, Jianqiang Ma, and Gertjan van Noord. 2009. A generalized method for iterative error mining in parsing results. In *ACL2009 Workshop Grammar Engineering Across Frameworks (GEAF)*, Singapore.

Claire Gardent and Eric Kow. 2007. Spotting overgeneration suspect. In *11th European Workshop on Natural Language Generation (ENLG)*.

Claire Gardent and Laura Perez-Beltrachini. 2010. Rtg based surface realisation for tag. In *COLING'10*, Beijing, China.

Claire Gardent, Benjamin Gottesman, and Laura Perez-Beltrachini. 2010. Comparing the performance of two TAG-based Surface Realisers using controlled Grammar Traversal. In *Proceedings of Coling 2010 (Poster session)*, pages 338–346, Beijing, China, August. Coling 2010 Organizing Committee.

Richert Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *Proceedings of NODALIDA*, pages 105–112, Tartu, Estonia.

Rajakrishnan Rajkumar, Dominic Espinosa, and Michael White. 2011. The osu system for surface realization at generation challenges 2011. Technical report, The Ohio State University. Proc. of the 13th European Workshop on Natural Language Generation.

Benoit Sagot and Eric de la Clergerie. 2006. Error mining in parsing results. In ACL, editor, *Proceedings of the ACL 2006*, pages 329–336, Morristown, NJ, USA.

Gertjan van Noord. 2004. Error mining for wide-coverage grammar engineering. In ACL, editor, *Proceedings of the ACL 2004*, pages 446–454, Morristown, NJ, USA.