The Inner Works of an Automatic Rule Refiner for Machine Translation

Ariadna Font Llitjós
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, 15217
aria@cs.cmu.edu

William A. Ridmann
Department of Computer Science
Carnegie Mellon University
Pittsburgh, 15217
war@andrew.cmu.edu

Abstract

Achieving high translation quality remains the biggest challenge Machine Translation (MT) systems currently face. Researchers have explored a variety of methods to include user feedback in the MT loop. However, most MT systems have failed to incorporate post-editing efforts beyond the addition of corrected translations to the parallel training data for Statistical and Example-Based system or to a translation memory database. In this paper, we describe the nuts and bolts of an Automatic Rule Refiner that, given online post-editing information, traces the errors back to lexical and grammar rules responsible for the errors and proposes concrete fixes to such rules. Initial results on a Diagnostic Test set show that this approach generalizes beyond input sentences corrected by bilingual speakers, and allows for the correct translation of unseen data.

1 Introduction

In the field of Machine Translation, the most popular trend of recent years has been adding more bilingual data to try to improve output quality. This strategy works reasonably well for Statistical and Example-Based MT systems. For Transfer-Based approaches to MT, however, having more bilingual data is rarely the solution to getting higher quality output.

Traditional solutions to improve Transfer-Based MT systems are costly and time-consuming, since they involve many computational linguist hours to develop new rules and refine old ones. Moreover, in any MT system, out-of-vocabulary words are constantly jeopardizing translation quality.

In this context, finding a way to automatically improve Transfer-Based Machine Translation systems without the need of computational linguistics experts constitutes a new promising research direction that deserves attention. This approach is particularly relevant for resource-poor scenarios.

In this paper, we describe an Automatic Rule Refiner that improves the quality of MT output. The refinement process targets bilingual speakers' corrections gathered through an online tool. These corrections allow the Rule Refiner (RR) to propose modifications that result in direct improvement of the grammar and the lexicon, yielding an improvement on overall translation quality of the MT system, even on unseen data. First, the RR parses and stores correction instances for specific translation pairs as provided by several bilingual speakers. Next, it proceeds to do blame assignment based on the transfer tree generated by the MT system. At this stage, the system retrieves the error-causing rules and lexical entries and it executes specific refining operations.

In a nutshell, the RR can add a lexical entry, modify a current lexical entry, bifurcate a rule and modify the copy, usually making it into a more specific rule, or refine a rule that is too general, by adding a missing agreement constraint.

2 Related Work

Nishida (1988) and colleagues described a Post-Editing Correction information Feedback system

(PECOF) in its early stages that also sought to improve a transfer-based MT system. The main differences with our approach are: 1) requiring computational linguists, whose work is not only to correct MT output but also to formulate correcting procedures corresponding to unseen error patterns, which are then executed by the PECOF system, and 2) using two MT systems in order to detect discrepancies between intermediate representations of the source language and the target language side, namely an *original* MT system (Japanese to English) and a *reverse* MT system (English to Japanese) that applied to the post-edited English translation.

Our transfer-based MT system is particular in the sense that its rules integrate information from the three components of a typical transfer system, namely parsing, transfer and generation. And thus, in comparison with the PECOF system, blame assignment is more directly inferable from corrections via the translation tree output by the system.

More recently, researchers have looked at other ways of including user feedback in the MT loop. Phaholphinyo and colleagues (2005) proposed adding post-editing rules to their English-Thai MT system via a post-editing tool. However, they use context sensitive pattern-matching rules, which make it impossible to fix errors involving missing words. Their system, unlike our approach, requires experienced linguists as well as a large corpus. They mention an experiment with 6,000 bilingual sentences but report no results due to data sparseness.[1]

3 Error Correction Extraction

The first part of the rule refinement process is the extraction of error correction information. Our approach relies on bilingual speaker post-editing information, collected via an online Translation Correction Tool (TCTool) as described in Font-Llitjós and Carbonell (2004).

Each translation pair corrected by a user via the TCTool generates a log file, which can be processed and parsed by the Rule Refiner to extract all the relevant correction and error information, and store it into a correction instance (CI).

CIs store all correction actions taken by a user, with related error information (Figure 1), into a vector of actions. Actions are processed by the Rule Refiner one at a time, following the algorithm described in Section 5. It is important to note that the order in which the user corrected errors has an impact on the order in which refinements apply and, consequently, on the resulting refined grammar.

3.1 Correction Instance Handling

It is crucial that the correction actions stored in our system correspond to the essence of what the bilingual speaker did to correct a specific translation pair while using the TCTool. This is actually a rather hard task. Even with just four correction actions (add, modify, delete and change word order), users can choose to correct the same mistake in more than one different way. For example, instead of modifying a word directly by editing it, deleting the incorrect word and adding a correct word would lead to the same final translation, but there would be no automatic way to relate the correction actions to the same error. In addition to intended corrections, users often change their mind and some times even make mistakes.

SL: John and Mary fell
TL: Juan y María cayeron
Alignments: ((1,1),(2,2),(3,3),(4,4))

Action 1: add (*se* in position 4)
Temp_CTL: Juan y María se cayeron
Alignments: ((1,1),(2,2),(3,3),(4,5))

Action 2: add alignment (*fell—se* (4,4))

CTL: Juan y María se cayeron
Alignments: ((1,1),(2,2),(3,3),(4,5),(4,4))

Figure 1. Correction Instance for Add Action. CIs store the source language sentence (SL), the target language sentence (TL) and the initial alignments (AL), as well as all the correction actions done by the user. It also provides the corrected translation (CTL) and final alignments.

Thus the goal of this component is to extract all the post-editing actions taken by non-expert users and process them while filtering out as much noise as possible at this early stage, so that the error information can be used effectively by the rest of the system.

[1] For a more detailed discussion of related work, see Font Llitjós and Carbonell (2006)

3.1.1 Spurious Correction Detection

There are several ways in which users change their mind, the first one being to correct a sentence that is already correct. If at some point during the correction session, the user decides to go back and mark the translation as being correct, the RR ignores any correction actions registered and assumes the translation is correct, effectively filtering out the noise introduced by the users' hesitation.

3.1.2 Spurious Loop Detection

In other cases, users carry out a correction action and then change their mind. Examples of this are when users decide to add a word, but then realize that it is not needed, or modify a word from form1 into form2, and then decide that it was already correct before, and so changes form2 back to form1.

The RR addresses all this issues with a Spurious Loop Detector. The Spurious Loop Detector operates by iterating over each action (A*i*) and searching for an action (A*i*') that will subsequently have had a reverse effect on the translation correction. Both A*i* and A*i*' are removed from the list of actions the user performed. Then each action lying in between A*i* and A*i*' is updated to reflect the removal of A*i* and A*i*'. Such updates can result in even more actions removed from the user action history.

More specifically, the following user actions can reverse each other:

- **Adding** and **Deleting** the same word (and vice-versa).
- **Editing** a word more than once (first action deleted if last edit on word reverts back to original word, first action change to last edit otherwise).
- **Changing Word Order** to previous order.
- **Adding** and **Deleting** the same SL-TL word alignment (and vice-versa)

Spurious Loop Detection runs in $O(|A|^2)$ time.

Given a Source Language (SL) and Target Language (TL) sentence pair, correctly detecting and discarding spurious loops allows for more reliable comparison of CIs that were parsed from log files generated by different users.

3.2 Collection of Correction Instances

Since users of the TCTool are not linguists or translation experts, the need to compare different correction instances and filter out noise becomes even more relevant.

On the other hand, all posterior blame assignment and refinement decisions made by the system fully depend on the correct extraction and processing of error correction information given by bilingual speakers.

In batch mode, the RR reads in multiple correction instances affecting multiple translation pairs, and stores them in a Collection. This allows the RR to compare all the CI affecting a SL-TL pair and, if they contain equivalent information[2], they are stored only once in the Collection with a weight proportional to the number of different CIs that were found to be equivalent. This weight directly indicates how much evidence there is in the data to support a correction action set as being more appropriate than another one with less weight for any given SL-TL pair. Namely, the relevance of a particular CI can be precisely estimated by its weight, which corresponds to the number of log files (and thus different users) that agree with it.

3.2.1 Error Complexity

In addition to taking into account the number of users who agreed on a specific set of correction actions, the RR also scores CIs according the complexity of their set of correction actions, or error complexity.

To estimate the error complexity of a given CI, both the number of errors addressed (approximated by counting different correction actions) as well as whether there is any dependency among the errors (the assumption being that when two different correction actions affect the same word they are targeting the same error, and thus are considered dependent), are factored in.

More specifically, CIs are sorted with polynomial sort, first by degree of dependency and then by coefficient, namely the amount of clusters with that degree.

[2] Equivalent CIs are CIs that in addition to having the same SL-TL and Corrected TL, once the spurious loops have been detected and removed, they also have the same set of correction actions affecting the same words.

For example, CIs with one correction action can be codified as (1); CIs with two independent correction actions, as (2), and with two dependent actions, as (1,0); CIs with three independent correction actions, can be codified as (3), with two dependent actions and one independent action, as (1,1), and with three dependent correction actions, as (1,0,0), and so on.

Descendent order of these vectors provides a natural and intuitive way to sort correction instances, since it correctly prioritizing CIs with a larger number of independent errors over CIs with smaller number errors that are dependent among them: 001, 002, 003, 010, 011, 100, etc.[3]

3.2.2 Ranking of CI Collection

Since we want to prioritize correction instances with more user support and tackle simpler errors first, the RR uses the following ranking algorithm:

For each CIcollection:

1. For each SL-TL pair, find the CI with the highest weight (more evidence) → BestCI
2. For each BestCI, compute error complexity
3. Rank BestCI with lowest error complexity higher.

This algorithm picks the CI with more user support for each SL-TL pair (BestCI) and then computes their error complexity in order to rank simpler CIs higher. The resulting ranking is used by the Rule Refiner to determine in which order to process correction data stored in a Collection of Correction Instances.

This "Tetris" approach is based on the underlying assumption that once simpler errors are fixed, more complex errors will be simplified (thus moving up in the ranking) and become easier to fix automatically.

4 Rule Blame Assignment

After having correctly stored and processed error correction information, rule blame assignment is executed by the RR. This is a key step of the rule refinement process, and is what differentiates Rule-Based MT systems form most Statistical MT (SMT) or Example-Based MT (EBMT) systems. Namely, for systems that do not have explicit rules, an approach like the one proposed here cannot be applied directly.

[3] Currently, the implementation of error complexity does not take alignment correction actions into account.

Given the error and correction words and the transfer tree output by the transfer engine, the RR can identify the incorrect rules and/or lexical entries, as the case might be, that are responsible for the error.

4.1 Rule handling

In order for the blame assignment algorithm to be effective, the RR pre-processes the lexicon and the grammar and assigns unique Rule IDs to all the entries that do not already have an ID.

To ensure fast look up of rules, red-black trees are used to index all rules by their respective Rule IDs. Additionally, lexical entries are indexed by their SL and TL sides, including exact and partial matches. Red-black trees are a balanced-tree data structure that ensures amortized look-up times of $O(\log|R|)$. Logarithmic lookup time is vital as the lexicon could potentially have hundreds of thousands of rules.

When rules are bifurcated, a Refined Rule Hierarchy is created (with each child being a derived rule from its parent). Since refined rules are stored in a text file that needs to be parsed by the transfer engine, hierarchy information is stored as meta data encapsulated by comments that are unparsed by the transfer engine. Such a hierarchy allows reverting back to the grammar and lexicon previous to refinements that did not lead to an improvement of MT quality.

In general, all meta data specific to the Rule Refiner is stored as comments in the grammar and lexicon text files so as not to disturb transfer engine parsing.

4.2 Translation Trees

Similar to the modified transfer approach discussed in the early METAL system (Hutchins and Somers, 1992), the rules in our transfer-based MT system contain analysis, transfer and generation information (Lavie *et al.*, 2004). This representation greatly facilitates blame assignment.

The translation tree that is output by the MT system contains a precise trace of what translation rules were applied to what lexical entries in order to generate the target sentence that the user corrected. This is done via unique rule IDs displayed by the tree, which are used by the Blame assign-

ment process to retrieve the relevant rules that need to be refined.

```
(S,1 (NP,6    (NP,2 (N,2:1 'JUAN') )
              (CONJ,1:2 'Y')
              (NP,2 (N,3:3 'MARÍA') ) )
     (VP,1 (V,6:4 'CAYERON') ) ) )
```

Figure 2. Translation tree output by the MT system for the SL sentence *John and Mary fell*.

5 Rule Refinement Operations

The core component of the rule refinement process is the one that decides what rule refinement operations need to apply to address a specific error (correction). This is also the component that is most sensitive to the set of correction actions currently allowed by the TCTool, for the kinds of rule refinement operations that are applied crucially depend on what types of correction actions were chosen by users.

The two main pieces of information that determine the rule refinement operation that will be applied by the RR are the correction action (taken by the user) and the error information available at refinement time. Given the correction action type (add, edit, delete and change_word_order) and the error and correction words, the RR applies a different refinement algorithm. In general, the Rule Refiner addresses lexical refinements first and then moves on to refinements of the grammar rules, if necessary.

First let's introduce some notation to describe error and correction information. The RR represents TL sentences as vectors of words from 1 to *n* (sentence length), indexed from 1 to *m* (corpus length) $TL_m = (W_1,...W_i,...W_n)$ and the corrected sentences (CTL) as follows:

$$CTL_m = (W_1,...W_i',...W_{clue},...W_{n'})$$

where W_i represents the error, namely the word that needs to be modified, deleted or dragged into a different position by the user in order for the sentence to be correct; and W_i' represents the correction, namely the user modification of W_i or the word that needs to be added by the user in order for the sentence to be correct.

W_{clue}, or clue word, represents a word that provides a clue with respect to what triggered the correction, namely the cause of the error. For example, in the case of lack of agreement between a noun and the adjective that modifies it, as in **el coche roja* (the red car), W_{clue} should be instantiated to *coche*, namely the word that gives us the clue about what the gender agreement feature value of W_i should be, namely masculine (*rojo*). W_{clue} can also be a phrase or constituent like a plural subject (eg. *[*Juan y Maria*] *cayó*, where the plural is implied by the conjoined NP).

W_{clue} is not always present and it can be before or after W_i. They can be contiguous or separated by one or more words.

For more information about the theoretical framework of the Rule Refiner, refer to Font-Llitjós and Carbonell (2006).

The following subsections describe a simplified version of algorithm underlying the Rule Refiner illustrated by a few examples.

5.1 Add Word

When users add a word (by clicking on the [New Word] button on the TCTool interface and then writing the word in the newly created box), there is no error word *per se*, however the RR can reliably identify a correction word (W_i'). See Figure 3.

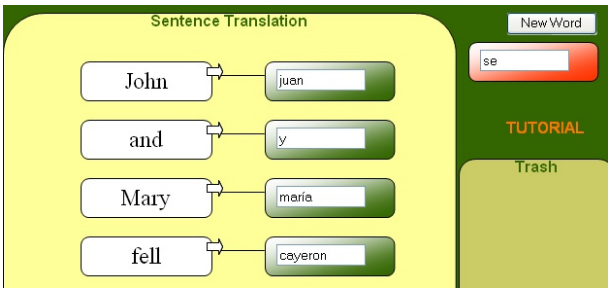

Figure 3. TCTool snapshot after having created a new word (*se*).

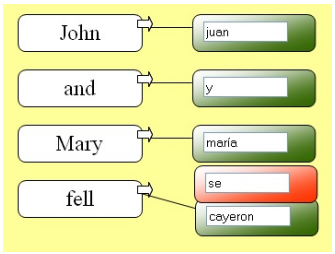

Figure 4. TCTool snapshot after having added the newly created word into the right position (Action 1).

Having instantiated W_i' with a word in the CTL vector (Figure 4), the next step is to check if the user added any alignments from the word in the SL sentence to this W_i', and if so, to retrieve them. Alignment information, however, can only be ex-

tracted after later correction actions are processed by the RR, and thus at this point a look ahead in the Action vector is required.

In the *John and Mary fell* example, when the user adds the word *se* between *María* and *cayeron* (*Juan y María* ***se*** *cayeron*), there is no alignment information available for W_i' (Figure 4), and so the algorithm looks ahead in the Action vector trying to find an alignment added to position *i*. In this case, it finds that *se* is aligned to *fell* by the user later on, and thus it extracts the corresponding alignment (4,4). Figure 5.

However, the SL word aligned to W_i' could also be aligned to other TL words, in this case *fell* also happens to be aligned to *cayeron*. And so next, the RR algorithm extracts all the alignments from SL word to other TL words (in this case (4,5)).

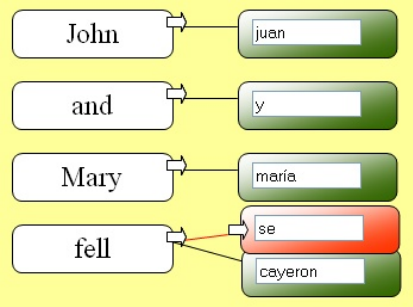

Figure 5. TCTool snapshot showing Action 2: Adding manual alignment.

Alignment information is required in order to retrieve the relevant lexical entries and determine the necessary refinements accordingly.

First, the entry for [SLW→ W_i'] is sought in the lexicon, if it's not there, [SLW→OtherTLWord] and [SL→ W_i'+OtherTLWord] are looked up.

In our example, [fell→se] and [fell→se cayeron] are not in the lexicon, however [fell→cayeron] is there (V,6).

At this point, the RR BIFURCATES the lexical entry [fell→cayeron] creating a copy of it (V,11), and REFINES it by replacing the TL side with W_i' + OtherTLWord (aligned to SL word): [fell→se cayeron]. The resulting refined entry is displayed below:

{V,11}
V::V |: [fell] -> [**"se cayeron"**]
(;**(P:{V,6})**
(X1::Y1)
((x0 form) = fall)
((x0 tense) = past)
((y0 agr pers) = 3)
((y0 agr num) = pl)

The new lexical entry is added to the Lexicon and the Refined Lexicon is loaded to the transfer engine in order to assess the effect of the rule refinement.

The lattice output by the transfer engine when translating the SL sentence is checked against the CTL sentence as corrected by the user.

If the RR finds that CTL is being generated by the MT system, it stops, otherwise, it proceeds to grammar refinements. For this example, the algorithm described above successfully refined the lexicon and the lattice output by the refined MT system, and so the Rule Refiner moves on to the next best CI in the Collection ranking.

If the word added (W_i') is not aligned to any word in the SL sentence, then there is nothing to be done at the lexical level and the algorithm skips to grammar refinements.

The first step is blame assignment by looking at the translation tree. For example, given the translation pair *you saw the woman − viste la mujer* and the user correction of adding the word "a" in front of *mujer*, the RR detects that "a" is not aligned to any words in the SL sentence, and it proceeds to look at the translation tree to extract the appropriate rule that needs to be refined.

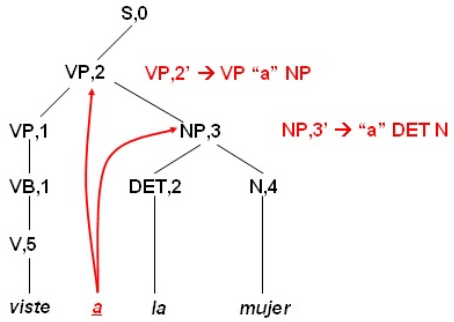

Figure 6: Translation Tree showing user insertion ("a") with two potentially relevant rules highlighted (VP,2 and NP,3).

In this case, since "a" is inserted between "viste" (V) and "la" (DET), there are two candidate rules for refinement, namely VP,2 and NP,3 (Figure 6).

Adding an "a" in the right position to any of these two rules (["a" DET N] and [V "a" NP]) would have the desired effect for this example. However, only the second option generalizes well to other sentences. If the first one were chosen, all instances of NP[DET N] would be generated in Spanish with an "a" preceding them, even when the NP is a subject or an oblique, this would result in an unnecessary ambiguity increase.

In general, to handle these cases in batch mode (when there is no option for further user interaction), the RR needs to refine the most specific rule, namely the candidate rule that encodes the most amount of context (Figure 7). This ensures that the refinement applies to syntactic environments most similar to the original corrected sentence. In this case, this means the refinement applies to object NPs only and not to all NPs.

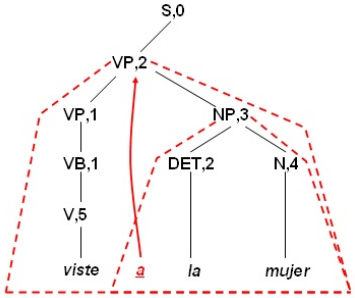

Figure 7: Depicting context captured by each candidate rule. VP,2 encapsulates more context than NP,3, and thus is more specific.

This is still not the ideal level of generalization, since one would want to only add an "a" in front of animated object NPs in Spanish. The RR could further refine the bifurcated rule to have a value constraint that restricts its application to NPs with *mujer* as a head. However, in the absence of semantic features in the lexicon (such as animacy), not adding any further refinements is the best strategy to strive high accuracy and control unnecessary ambiguity.

5.2 Edit Word

When users modify a word (W_i) into a related form or sense (W_i'), there are two possible scenarios. The one most favorable to generalization, is that the lexicon already discriminates between these two forms, usually by giving them a different value for the same feature attribute (example: [red-roja] and [red-rojo]). The one with less immediate impact is that the two senses are identically defined in the lexicon, namely they have the same POS and the same feature attributes and values (ex: [women-mujer] and [guitar-guitarra] are both singular feminine nouns in Spanish).

If the lexicon already discriminates between the two lexical entries, the RR extracts the grammar rule for the immediate common parent of W_i and W_{clue} (as identified by the user) and adds an agreement constraint with the triggering feature[4] between the constituents corresponding to W_i and W_{clue}.

SL: I see the red car
TL: veo el auto roja
Alignments: ((2,1),(3,2),(4,4),(5,3))

Action 1: edit (W_i=roja → W_i'=rojo; W_{clue}=auto)

CTL: veo el auto rojo
Alignments: ((2,1),(3,2),(4,4),(5,3))

Figure 8. Correction Instance for edit action.

For the CI represented in Figure 8 (*I see the red car)*, the user edits *roja* into *rojo* (by clicking on the word and changing 'a' into 'o'), and the system finds that the difference (delta set) between the lexical entry for roja and rojo is [agr gen].[5]

At this point, the RR moves to the Grammar Refinement.

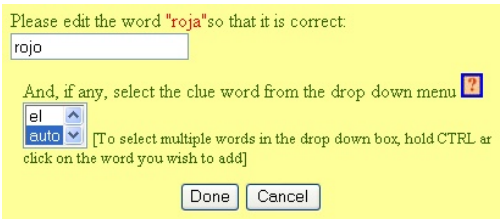

Figure 9. Edit Word window eliciting for Clue word Information.

Since the user identified *auto* as being the clue word as shown in Figure 9, the RR algorithm can now instantiate what variables do W_i and W_{clue} correspond to in the relevant rule (NP,8), namely *y3* and *y2*.

Next, the Rule Refiner adds an [agr gen] constraint to the rule copy (NP,9) between y2 and y3:

```
{NP,9}                          ;; y1   y2   y3
NP::NP : [DET ADJ N] -> [DET N ADJ]
(;(P:{NP,8})
  (X1::Y1)  (X2::Y3)  (X3::Y2)
  ((x0 det) = x1)
  ((x0 mod) = x2)
  (x0 = x3)
  (y0 = x0)
  (y1 == (y0 det))
  (y3 == (y0 mod))
  (y2 = y0)
  ((y2 agr gen) = (y3 agr gen)))
```

[4] The triggering feature is the attribute name for which the two lexical entries have a different value.

[5] *roja* is the feminine form of red in Spanish and *rojo* is the masculine form (*auto rojo* vs *casa roja*).

However, if the lexicon does not already discriminate between the two lexical entries (W_i and W_i'), the RR postulates a new feature attribute and adds a binary value constraint to each lexical entry, in order to allow the grammar to distinguish between the two senses of the same SL word.

For example, given the sentence *Mary plays guitar* and its translation as produced by our MT system, **María juega guitarra*, the user will edit *juega* into *toca* and since this new sense is not listed in the lexicon, the RR will BIFURCATE the original lexical entry [play→juega] and REFINE it by replacing the TL side (as in 5.1. Add above). Since in this case, [play→toca] is otherwise an exact copy of [play→juega] (with the same POS and features), the system postulates a new feature (*feat_0*) to distinguish between the two and adds the following constraints to the lexical entries: [play→toca ((*feat_0*) = +)] and [play→juega ((*feat_0*) = −)].

If W_{clue} were instantiated with *guitarra*, in order to effectively add an agreement constraint between "toca" and "guitarra", in addition to adding an agreement constraint to the rule that subsumes both words (VP,2), a percolate method recursively adds the appropriate agreement constraints to all intermediate rules (in this case, VP,1 and NP,3):

(S,1 (NP,2 (N,3:1 'MARÍA'))
(**VP,2** (**VP**,1 (V,5:2 '**JUEGA**'))
(**NP**,3 (DET,2:3'LA')
(N,5:4 '**GUITARRA**')))))

5.3 Delete Word

If a user deletes a word (by dragging it to the trash), it could be that the user really meant to delete this word or that she just wanted to modify the incorrect word and thus deleted it and then added a new word with the correct word in it.

In order to detect this, the RR algorithm checks if there were any alignments from it to one or more SL words, and if so, it looks ahead to see if there was any other word in the TL sentence that was aligned to that SL word at a later point in the session. If there is a TL word aligned to the SL word, then the RR algorithm checks if it's already in the lexicon, and if it isn't, it adds it.

If W_i' is in the lexicon, the RR algorithm adds a new lexical entry for the SL word aligned to it with an empty TL side ([SL word → ""]), which results into the MT system not translating the SL word.

5.4 Word Order Change

In order to change the order of the TL words, users can drag and drop words into a different position in the TL sentence. (Alignments stay the same, unless manually changed by users).

The Rule Refiner detects which word(s) were moved to a different position and extracts what were their initial (*i*) and final (*i*') positions. The Rule Refiner can only reliably execute refinement operations if, given a word that has moved (W_i), both the initial and final positions fall inside the scope of a rule in the grammar. If a word undergoes a long-distance move and thus is placed at the beginning or the end of the sentence far from its original position, automatic refinements become less reliable.

If the initial and final positions are subsumed by a rule in the grammar, then the RR algorithm can extract the rule that immediately subsumes the constituents in both positions and BIFURCATE it in order to change the constituents on the right hand side of the rule copy.

For example, if the grammar already contains a general NP rule that reverses the order of the adjectives and nouns in Spanish, but is lacking a specific rule for pre-nominal adjectives, given relevant correction feedback, the RR can extract the general NP rule and flip the order of N and ADJ on the RHS (TL side) of the rule. This also requires updating the alignment information as well as affected indices in the value and agreement constraints.

The next step is to further constrain the newly created rule so that it only applies in the right context. Again this can be done in a general way if the lexicon already distinguishes between the lexical entries that are affected by this change and the general cases. A constraint with the appropriate feature attribute is added to the specific rule and a blocking constraint is added to the general rule.

If there is no current feature attribute to distinguish between the special case and the general case, the RR postulates a new binary feature and add a value constraint to the appropriate lexical entries as well as to the specific and general grammar rules.

To see a concrete example of this (*Gaudi was a great artist* → **Gaudí era un artista grande* → *Gaudí era un gran artista*), see Font Llitjós and Carbonell (2006).

6 Generalization Power

The main difference between this approach and mere post-editing is that the resulting refinements affect not only the translation instances corrected by the user, but also other similar sentences where the same error would manifest. After the types of refinements described in Section 5 have been applied to the grammar and lexicon, sentences like *I gave the girl a book* and *Juan is a great person* will now correctly be translated as *Doy un libro a la niña* (instead of **Doy un libro la niña*) and *Juan es una gran persona* (instead of **Juan es una persona grande*), to name just two examples.

7 Evaluation of Automatic Refinements

The ultimate goal of the Automatic Rule Refiner is to improve MT output accuracy and quality. Initial experiments with an English-Spanish MT system, designed to measure the effects of the refinement process, clearly show that refinements generalize well beyond the specific sentences corrected by users.

A Diagnostic Test (DT) set was developed with 55 sentences expected to exhibit the same kinds of errors that users corrected, but that are significantly different from the ones corrected. For this initial experiment the seven different log files shown in Figure 10 were processed by the Rule Refiner.

1. I see the red car – veo el auto *roja* – veo el auto rojo
TCTool: Edit → RR: Add gender agreement constraint

2. you saw the woman – viste la mujer – viste a la mujer
TCTool: Add → RR: Add "a" to the appropriate Gr rule

3. I see the red unicorn – veo el *unicorn* rojo – unicornio
TCTool: Edit → RR: Add OOV word to the lexicon

4. Mary plays the guitar – María *juega* la guitarra – toca
TCTool: Edit → RR: Add new sense of the word "play"

5. John and Mary fell – Juan y María cayeron – se cayeron.
TCTool: Add → RR: Add reflexive form to the lexicon

6. Gaudi was a great artist – Gaudí era un *artista gran* – gran artista. TCTool: Move *gran* in front of *artista*
RR: Add NP rule to cover pre-nominal ADJ

7: I would like to go – me gustaría *que* ir – me gustaría ir. TCTool: Delete *que* → RR: allow no translation for "to" ([to→ " "]).

Figure 10: Log Files processed by the Rule Refiner.

The DT set was translated both with the initial grammar and lexicon (19 grammar rules and 250 lexical entries) and with the final refined grammar and lexicon, result of all the refinements triggered by the seven log files (21 grammar rules and 254 lexical entries).

The MT system went from not producing a correct translation to producing one to two correct translations for most SL sentences in the DT set, while the number of total translations per sentence went from an average of 4 to an average of 6 translations, as shown in Table 1. Note that the average increase in the number of correct translations (recall) is larger than the average increase in the total number of translations (ambiguity). In this context, ambiguity corresponds to the denominator of precision, and so when ambiguity increases, precision decreases.

Recall	**Before**	**After**
Avg. Num of *Correct* Tr.	0.38	1.20
Avg. Increase in %		214%
Ambiguity		
Avg. Num of *Total* Tr.	4.09	6.11
Avg. Increase in %		49%

Table 1: The first row measures recall achieved *before* any refinements and *after* 7 refinements were applied, the average number of correct translations; the second row shows the average increase of the number of correct translations. The last two rows illustrate ambiguity: the average number of total translations, and the average increase of the number of total translations.

Figure 11 shows a few examples from the DT set which were not being correctly translated by the original MT system and are now being correctly translated by the refined MT system.

(1) I meet some didactic professors at the conference – conocí a algunos profesores didácticos en el congreso
(2) I saw the children – vi a los niños
(3) The unicorn slept – el unicornio durmió
(4) The boy plays the viola – el niño toca la viola
(5) The little boys fell – los niños pequeños se cayeron
(6) Irina is a great friend – Irina es una gran amiga.
(7) They want to contribute – Ellas quieren contribuir

Figure 11: Translation examples from DT set after refinements in Figure 10. The number preceding each example corresponds to the log file number responsible for the correction.

These results represent a lower-bound on recall for this test set, since the MT system used for these

experiments did not include a morphology module. If the system had a morphology module, refinements would generalize to all forms of a specific word, and thus the impact on unseen data would be much larger.

8 Conclusions and Future Work

The main goal of the Automatic Rule Refiner is to extend the lexicon and the grammar to account for exceptions not originally encoded in the translation rules. A secondary goal is to make overly general rules more specific to reduce grammar ambiguity.

Beyond increasing lexical coverage, the errors most efficiently corrected by the Rule Refiner are syntactic in nature. The reason for this is that the generalization power of the RR is greatest when refinements involve existing feature constraints. Since our lexicon currently contains purely morpho-syntactic features (such as *gender*, *number* and *person*), grammar refinements affecting those features will generalize well on unseen data.

Semantic errors, however, usually require the system to postulate a new binary feature to distinguish between the different senses of the word. Since the RR cannot populate other lexical entries with newly hypothesized features automatically, in the absence of a generalization mechanism, this process represents just a first step towards semantic correction. An extension of this work could be to query an external ontology, and derive semantic distinctions from it.

Initial results show that our Automatic Rule Refinement achieves the main goal to add generation capability to the translation model, and thus improve MT output recall, without proportional increases in ambiguity.

The space of solutions for an Automatic Rule Refiner is large and there is a clear tradeoff between adding constraints to control ambiguity and loosing refinement generality.

Adding generation capabilities to the model by always bifurcating, for example, increases ambiguity exponentially. In a modular translation model, where rules plug into each other, blind bifurcation increases the complexity of the grammar unnecessarily and can pose a serious problem.

Adding more feature constraints will decrease ambiguity, but unless features already exist in the lexicon, refinements will not generalize over unseen words and syntactic structures.

In the ideal oracle case, only the right constraints at the right level of generality, are added to the model. In the practical learning case, the Automatic RR learns refinements that increase the generation capabilities, but do not increase ambiguity exponentially (much like a partial constraint oracle).

When applying refinements automatically, it cannot always be determined whether a refinement should be as general as possible or, on the contrary, it should only be made as specific as possible. Therefore, choosing between prioritizing generalization versus ambiguity reduction becomes a practical matter and can be decided according to specific needs.

9 Acknowledgements

We would like to thank Jaime Carbonell and Alon Lavie for fruitful discussion about the theoretical aspects of this work. This research was funded in part by NSF grant number IIS-0121-631.

References

Font Llitjós, Ariadna and Jaime Carbonell. 2006. *Automating Post-Editing To Improve MT Systems*. AMTA, APE Workshop. Boston, USA.

Font Llitjós, Ariadna and Jaime Carbonell. 2004. *The Translation Correction Tool: English-Spanish user studies*. LREC, Lisbon, Portugal.

Hutchins, W. J., and Harold L. Somers. 1992. *An Introduction to Machine Translation*. London: Academic Press.

Alon Lavie, Stephan Vogel, Erik Peterson, Katharina Probst, Ariadna Font-Llitjós, Rachel Reynolds, Jaime Carbonell, and Richard Cohen. 2004. *Experiments with a Hindi-to-English Transfer-based MT System under a Miserly Data Scenario*. Transactions on Asian Language Information Processing (TALIP), Special Issue on Rapid Deployment Hindi-English Translation Systems.

Nishida, F.; S. Takamatsu, T. Tani and T. Doi. 1988. *Feedback of Correcting Information in Postediting to a Machine Translation System*. COLING.

Phaholphinyo, Sitthaa; Teerapong Modhiran, Nattapol Kritsuthikul and Thepchai Supnithi. 2005. *A Practical of Memory-based Approach for Improving Accuracy of MT*. MT Summit X. Phuket Island, Thailand.