Automatic Analysis of Rhythmic Poetry with Applications to Generation and Translation

Erica Greene
Haverford College
370 Lancaster Ave.
Haverford, PA 19041
ericagreene@gmail.com

Tugba Bodrumlu
Dept. of Computer Science
Univ. of Southern California
Los Angeles, CA 90089
bodrumlu@cs.usc.edu

Kevin Knight
Information Sciences Institute
Univ. of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
knight@isi.edu

Abstract

We employ statistical methods to analyze, generate, and translate rhythmic poetry. We first apply unsupervised learning to reveal word-stress patterns in a corpus of raw poetry. We then use these word-stress patterns, in addition to rhyme and discourse models, to generate English love poetry. Finally, we translate Italian poetry into English, choosing target realizations that conform to desired rhythmic patterns.

1 Introduction

When it comes to generating creative language (poems, stories, jokes, etc), people have massive advantages over machines:

- people can construct grammatical, sensible utterances,
- people have a wide range of topics to talk about, and
- people experience joy and heart-break.

On the other hand, machines have some minor advantages:

- a machine can easily come up with a five-syllable word that starts with *p* and rhymes with *early*, and
- a machine can analyze very large online text repositories of human works and maintain these in memory.

In this paper we concentrate on statistical methods applied to the analysis, generation, and translation of poetry. By analysis, we mean extracting patterns from existing online poetry corpora. We use these patterns to generate new poems and translate existing poems. When translating, we render target text in a rhythmic scheme determined by the user.

Poetry generation has received research attention in the past (Manurung et al., 2000; Gervas, 2001; Diaz-Agudo et al., 2002; Manurung, 2003; Wong and Chun, 2008; Tosa et al., 2008; Jiang and Zhou, 2008; Netzer et al., 2009), including the use of statistical methods, although there is a long way to go. One difficulty has been the evaluation of machine-generated poetry—this continues to be a difficulty in the present paper. Less research effort has been spent on poetry analysis and poetry translation, which we tackle here.

2 Terms

Meter refers to the rhythmic beat of poetic text when read aloud. Iambic is a common meter that sounds like *da-DUM da-DUM da-DUM*, etc. Each *da-DUM* is called a foot. Anapest meter sounds like *da-da-DUM da-da-DUM da-da-DUM*, etc.

Trimeter refers to a line with three feet, pentameter to a line with five feet, etc. Examples include:

- a VE-ry NAS-ty CUT (iambic trimeter)
- shall I com-PARE thee TO a SUM-mer's DAY? (iambic pentameter)
- twas the NIGHT before CHRIST-mas and ALL through the HOUSE (anapest tetrameter)

Classical English sonnets are poems most often composed of 14 lines of iambic pentameter.

Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 524–533,
MIT, Massachusetts, USA, 9-11 October 2010.

3 Analysis

We focus on English rhythmic poetry. We define the following analysis task: *given poetic lines in a known meter (such as sonnets written in iambic pentameter), assign a syllable-stress pattern to each word in each line*. Making such decisions is part of the larger task of reading poetry aloud. Later in the paper, we will employ the concrete statistical tables from analysis to the problems of poetry generation and translation.

We create a test set consisting of 70 lines from Shakespeare's sonnets, which are written in iambic pentameter. Here is an input line annotated with gold output.

```
shall i compare thee to a summers day
|     |    /\    |   |  |   /\     |
S     S*  S  S*  S   S* S  S* S    S*
```

S refers to an unstressed syllable, and S* refers to a stressed syllable. One of the authors created gold-standard output by listening to Internet recordings of the 70 lines and marking words according to the speaker's stress. The task evaluation consists of *per-word accuracy* (how many words are assigned the correct stress pattern) and *per-line accuracy* (how many lines have all words analyzed perfectly).

This would seem simple enough, if we are armed with something like the CMU pronunciation dictionary: we look up syllable-stress patterns for each word token and lay these down on top of the sequence S S* S S* S S* S S* S S*. However, there are difficulties:

- The test data contains many words that are unknown to the CMU dictionary.
- Even when all words are known, many lines do not seem to contain 10 syllables. Some lines contain eleven words.
- Spoken recordings include stress reversals, such as poin-TING instead of POIN-ting.
- Archaic pronunciations abound, such as PROV-ed (two syllables) instead of PROVED (one syllable).
- In usage, syllables are often subtracted (PRIS-ner instead of PRIS-o-ner), added (SOV-e-reign instead of SOV-reign), or merged.
- Some one-syllable words are mostly stressed, and others mostly unstressed, but the dictionary provides no guidance. When we generate rhythmic text, it is important to use one-syllable words properly. For example, we would be happy for an iambic generator to output *big thoughts are not quite here*, but not *quite big thoughts are not here*.

$$\text{e} \rightarrow \boxed{\text{P}(m|e)} \rightarrow \text{m}$$

Figure 1: Finite-state transducer (FST) for mapping sequences of English words (e) onto sequences of S* and S symbols (m), representing stressed and unstressed syllables.

Therefore, we take a different tack and apply unsupervised learning to acquire word-stress patterns directly from raw poetry, without relying on a dictionary. This method easily ports to other languages, where dictionaries may not exist and where morphology is a severe complication. It may also be used for dead languages.

For raw data, we start with all Shakespeare sonnets (17,134 word tokens). Because our learning is unsupervised, we do not mind including our 70-line test set in this data (*open testing*).

Figures 1 and 2 show a finite-state transducer (FST) that converts sequences of English words to sequences of S* and S symbols. The FST's transitions initially map each English word onto all output sub-sequences of lengths 1 to 4 (i.e., S, S*, S-S, S-S*, S*-S, S*-S*, S-S-S, ...) plus the sequences S-S*-S-S*-S and S*-S-S*-S-S*. Initial probabilities are set to 1/32. The FST's main loop allows it to process a sequence of word tokens. If the same word appears twice in a sequence, then it may receive two different pronunciations, since the mapping is probabilistic. However, a token's syllable/stress pattern is chosen independently of other tokens in the sequence; we look at relaxing this assumption later.

We next use finite-state EM training[1] to train the machine on input/output sequences such as these:

```
from fairest creatures we desire increase
S S* S S* S S* S S* S S*

but thou contracted to thine own bright eyes
S S* S S* S S* S S* S S*
```

[1] All operations in this paper are carried out with the generic finite-state toolkit Carmel (Graehl, 1997). For example, the *train-cascade* command uses EM to learn probabilities in an arbitrary FST cascade from end-to-end input/output string pairs.

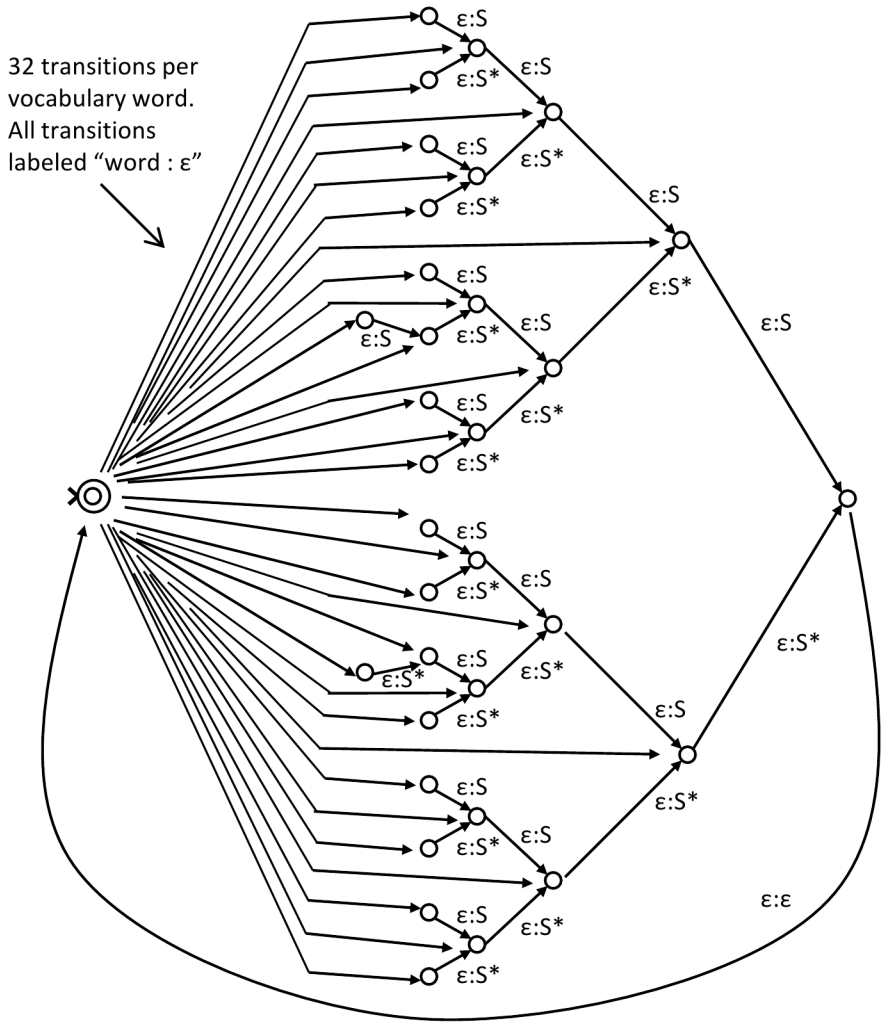

Figure 2: An efficient FST implementing P($m|e$). This machine maps sequences of English words onto sequences of S* and S symbols, representing stressed and unstressed syllables. Initially every vocabulary word has 32 transitions, each with probability 1/32. After EM training, far fewer transitions remain.

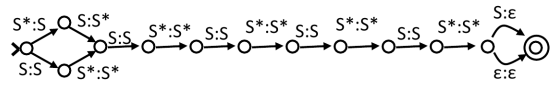

Figure 3: An FST that accepts any of four input meters and deterministically normalizes its input to strict iambic pentameter. We call this FST *norm*.

$$\text{e} \rightarrow \boxed{\text{P}(m|e)} \rightarrow \text{m} \rightarrow \boxed{norm} \rightarrow \text{m}$$

Figure 4: FST cascade that encodes a loose interpretation of iambic pentameter. The *norm* FST accepts any of four near-iambic-pentameter sequences and normalizes them into strict iambic pentameter.

Note that the output sequences are all the same, representing our belief that each line should be read as iambic pentameter.[2] After we train the FST, we can use Viterbi decoding to recover the highest-probability alignments, e.g.:

```
from fairest creatures we desire increase
 |     |       /| \    |    /\      /\
 S     S*     S S* S   S*  S  S*   S  S*

but thou contracted to thine own bright eyes
 |    |    /| \     |    |    |    |     |
 S    S*  S S* S    S*   S    S*   S     S*
```

Note that the first example contains an error—the words *fairest* and *creatures* should each be read with two syllables. There are many such errors. We next improve the system in two ways: more data and better modeling.

First, we augment the Shakespeare sonnets with data from the website *sonnets.org*, increasing the number of word tokens from 17,134 to 235,463. The *sonnets.org* data is noisier, because it contains some non-iambic-pentameter poetry, but overall we find that alignments improve, e.g.:

```
from fairest creatures we desire increase
 |     /\       /\     |    /\      /\
 S    S* S     S* S    S*  S  S*   S  S*
```

Second, we loosen our model. When we listen to recordings, we discover that not all lines are read S S* S S* S S* S S* S S*. Indeed, some lines in our data contain eleven words—these are unexplainable by the EM training system. We also observe that

[2]We can augment the data with lines of poetry written in meters other than iambic pentameter, so long as we supply the desired output pattern for each input line.

Training data	Training tokens	Test token accuracy	Test line accuracy
Shakespeare	17,134	82.3%	55.7%
sonnets.org	235,463	94.2%	81.4%

Figure 5: Analysis task accuracy.

poets often use the word *mother* (S* S) at the beginnings and ends of lines, where it theoretically should not appear.

Two well-known variations explain these facts. One is optional *inversion* of the first foot (S S* → S* S). Second is the optional addition of an eleventh unstressed syllable (the *feminine ending*). These variations yield four possible syllable-stress sequences:

```
S  S* S S* S S* S S* S S*
S* S  S S* S S* S S* S S*
S  S* S S* S S* S S* S S* S
S* S  S S* S S* S S* S S* S
```

We want to offer EM the freedom to analyze lines into any of these four variations. We therefore construct a second FST (Figure 3), $norm$, which maps all four sequences onto the canonical pattern S S* S S* S S* S S*. We then arrange both FSTs in a cascade (Figure 4), and we train the whole cascade on the same input/output sequences as before. Because $norm$ has no trainable parameters, we wind up training only the lexical mapping parameters. Viterbi decoding through the two-step cascade now reveals EM's proposed internal meter analysis as well as token mappings, e.g.:

```
to be or not to be that is the question
|  |  |   |  |  |    |  |   |     /\
S  S* S   S* S  S*   S  S*  S    S* S
|  |  |   |  |  |    |  |   |    |
S  S* S   S* S  S*   S  S*  S    S*
```

Figure 5 shows accuracy results on the 70-line test corpus mentioned at the beginning of this section. Over 94% of word tokens are assigned a syllable-stress pattern that matches the pattern transcribed from audio. Over 81% of whole lines are also scanned correctly. The upper limit for whole-line scanning under our constraints is 88.6%, because 11.4% of gold outputs do not match any of the four patterns we allow.

We further obtain a probabilistic table of word mappings that we can use for generation and trans-

```
P(S* S S* | altitude) = 1.00

P(S* S | creatures)   = 1.00

P(S* S  | pointed)    = 0.95
P(S  S* | pointed)    = 0.05

P(S* S    | prisoner) = 0.74
P(S* S S* | prisoner) = 0.26

P(S* S  | mother)     = 0.95
P(S*    | mother)     = 0.03
P(S  S* | mother)     = 0.02
```

Figure 6: Sample learned mappings between words and syllable-stress patterns.

word	P(S* | word)	P(S | word)
a	0.04	0.96
the	0.06	0.94
their	0.09	0.91
mens	0.10	0.90
thy	0.10	0.90
be	0.48	0.52
me	0.49	0.51
quick	0.50	0.50
split	0.50	0.50
just	0.51	0.49
food	0.90	0.10
near	0.90	0.10
raised	0.91	0.09
dog	0.93	0.07
thought	0.95	0.05

Figure 7: Sample mappings for one-syllable words.

lation tasks. Figure 6 shows a portion of this table. Note that P(S S* | mother) has a very small probability of 0.02. We would incorrectly learn a much higher value if we did not loosen the iambic pentameter model, as many *mother* tokens occur line-initial and line-final.

Figure 7 shows which one-syllable words are more often stressed (or unstressed) in iambic pentameter poetry. Function words and possessives tend to be unstressed, while content words tend to be stressed, though many words are used both ways. This useful information is not available in typical pronunciation dictionaries.

Alignment errors still occur, especially in noisy

$$\boxed{\text{P}(m)} \rightarrow \text{m} \rightarrow \boxed{\text{P}(e|m)} \rightarrow \text{e} \rightarrow \boxed{\text{P}(e)} \rightarrow \text{e}$$

Figure 8: Finite-state cascade for poetry generation.

portions of the data that are not actually written in iambic pentameter, but also in clean portions, e.g.:

```
the perfect ceremony of loves rite
 |     /\      /|\    |   |    /\
 S    S* S   S* S S*  S   S*  S  S*
```

The word *ceremony* only occurs this once in the data, so it is willing to accept any stress pattern. While *rite* is correctly analyzed elsewhere as a one-syllable word, *loves* prefers S*, and this overwhelms the one-syllable preference for *rite*. We can blame our tokenizer for this, as it conflates *loves* and *love's*, despite the fact that these words have different stress probabilities.

4 Generation

Figure 8 shows our concept of generation as a cascade of weighted FSTs.

P(m) is a user-supplied model of desired meters—normally it deterministically generates a single string of S* and S symbols. (The user also supplies a rhyme scheme—see below).

P($e|m$) is the reverse of Section 3's P($m|e$), being a model of word selection. Its generative story is: (1) probabilistically select n tokens (n = 1 to 5) from the input, (2) probabilistically select a word w that realizes that n-token sequence, and (3) recurse until the input is consumed. Instead of asking how a given word is likely to be pronounced (e.g., S or S*), we now ask how a given stress-pattern (e.g., S or S*) is likely to be realized. This model is trained with the same method described in Section 3 and is augmented with the CMU pronunciation dictionary.

Finally, P(e) is a word-trigram model built from a 10,000-line corpus of 105 English love poems.

We select the first line of our poem from the FST cascade's 100,000-best list, or by hand. To generate each subsequent line, we modify the cascade and run it again. The first modification is to incorporate a discourse model. From our poetry corpus, we estimate a word's unigram probability given the words on the previous line, via IBM Model 1 (Brown et al., 1993). We modify P(e) by interpolating in these probabilities. Second, we check if any previous line

```
The women of the night
Again and all the way
Like a mouse in the white
Not the heart of the day.
  - - -
Of the bed to trust me
Around her twists the string
But i will not tell thee
Fire changes everything.
  - - -
A son of the right hand confines
His uncle could have broken in
Towards the high bank and the pines
Upon the eyes and i have been
  - - -
Into one of her hundred year old
Or the house in a house in a cold
The first time she met him
Like a mouse in the dim
For me to the moon and when i told
  - - -
Into one of them some years before
His own man or the house in a more
The moon and when the day
Into one of the way
With the breath from the first time
  she swore
```

Figure 9: Sample poems generated with a weighted FST cascade.

$w_1, w_2, ...w_n$ needs to be rhymed with, according to the user-supplied scheme. If so, we build an additional FST that accepts only strings whose final word rhymes with w_n. This is a reasonable approach, though it will not, for example, rhyme *...tar me* with *...army*. We say two non-identical words rhyme if their phoneme strings share a common suffix that includes the last stressed vowel.

Figure 9 shows several poems that we automatically generate with this scheme.

5 Translation

Automatically generated poetry can sound good when read aloud, but it often has a "nonsense" feel to it. According to (Gervas, 2010), creative-language researchers interested in realization and surface language statistics ("how to say") have tended to gravitate to poetry generation, while researchers interested in characters, goals, and story-line ("what to say") have tended to gravitate to prose story generation.

Translation provides one way to tie things together. The source language provides the input ("what to say"), and the target language can be shaped to desired specifications ("how to say"). For example, we may want to translate Italian sonnets into fluent English iambic pentameter. This is certainly a difficult task for people, and one which is generally assumed to be impossible for computers.

$$\text{i} \rightarrow \boxed{\text{P}(e|i)} \rightarrow \text{e} \rightarrow \boxed{\text{P}(m|e)} \rightarrow \text{m} \rightarrow \boxed{\text{P}(m)} \rightarrow \text{m}$$

Figure 10: Finite-state cascade for poetry translation.

Here we investigate translating Dante's *Divine Comedy* (DC) from Italian into English by machine. The poem begins:

```
nel mezzo del cammin di nostra vita
mi ritrovai per una selva oscura
che la via diritta era smarrita.
```

DC is a long sequence of such three-line stanzas (*tercets*). The meter in Italian is hendecasyllabic, which has ten syllables and ensures three beats. Dante's Italian rhyme scheme is: **ABA**, **BCB**, **CDC**, etc, meaning that lines 2, 4, and 6 rhyme with each other; lines 5, 7, and 9 rhyme with each other, and so forth. There is also internal rhyme (e.g., *diritta/smarrita*).

Because DC has been translated many times into English, we have examples of good outputs. Some translations target iambic pentameter, but even the most respected translations give up on rhyme, since English is much harder to rhyme than Italian. Longfellow's translation begins:

```
midway upon the journey of our life
i found myself within a forest dark
for the straightforward pathway had
  been lost.
```

We arrange the translation problem as a cascade of WFSTs, as shown in Figure 10. We call our Italian input i. In lieu of the first WFST, we use the statistical phrase-based machine translation (PBMT) system Moses (Koehn et al., 2007), which generates a target-language lattice with paths scored by P($e|i$). We send this lattice through the same P($m|e$) device we trained in Section 3. Finally, we filter the resulting syllable sequences with a strict, single-path, deterministic iambic pentameter acceptor, P(m).[3] Our

[3]It is also possible to use a looser iambic P(m) model, as described in Section 3.

Parallel Italian/English Data

Collection	Word count (English)
DC-train	400,670
Il Fiore	25,995
Detto Damare	2,483
Egloghe	3,120
Misc.	557
Europarl	*32,780,960*

English Language Model Data

Collection	Word count (English)
DC-train	400,670
poemhunter.com poetry.eserver.org poetrymountain.com	686,714
poetryarchive.org	58,739
everypoet.com	574,322
sonnets.org	166,465
Europarl	*32,780,960*

Tune and Blind Test Data (4 reference)

Collection	Word count (Italian)
DC-tune	7,674
DC-test	2,861

Figure 11: Data for Italian/English statistical translation.

finite-state toolkit's top-k paths represent the translations with the highest product of scores P($e|i$) · P($m|e$) · P(m).

In general, the P($e|i$) and P($m|e$) models fight each other in ranking candidate outputs. In experiments, we find that the P($e|i$) preference is sometimes so strong that the P($m|e$) model is pushed into using a low-probability word-to-stress mapping. This creates output lines that do not scan easily. We solve this problem by assigning a higher weight to the P($m|e$) model.[4]

Figure 11 shows the data we used to train the PBMT system. The vast majority of parallel Italian/English poetry is DC itself, for which we have four English translations. We break DC up into DC-train, DC-tune, and DC-test. We augment our target language model with English poetry collected from many sources. We also add Europarl data, which is out of domain, but which reduces the unknown word-token rate in DC-test from 9% to 6%, and the unknown word-type rate from 22% to 13%.

[4]We set this weight manually to 3.0, i.e., we raise all probabilities in the P($m|e$) model to the power of 3.0. Setting the weight too high results in lines that scan very well, but whose translation quality is low.

Original:

nel mezzo del cammin di nostra vita
mi ritrovai per una selva oscura
che la via diritta era smarrita.

Phrase-based translation (PBMT):

midway in the journey of our life
i found myself within a forest dark
for the straight way was lost.

PBMT + meter model:

midway upon the journey of our life
i found myself within a forest dark
for the straightforward pathway had been lost.

Figure 12: Automatic translation of lines from Dante's *Divine Comedy*. In this test-on-train scenario, the machine reproduces lines from human translations it has seen.

We first experiment in a test-on-train scenario, where we translate parts of DC that are in our training set. This is a normal scenario in human poetry translation, where people have access to previous translations.

Figure 12 shows how we translate the first lines of DC, first using only PBMT, then using the full system. When we use the full system, we not only get an output string, but also the system's intended scan, e.g.:

```
midway upon the journey of our life
 /\     /\   |    /\    |   |   |
S  S*  S  S* S   S* S   S*  S   S*
```

The machine's translation here is the same as Longfellow's, which is in the training data. In other cases, we observe the machine combining existing translations, e.g.:

```
i:  bedi la bestia per cu io mi volsi
I5: behold the beast that made me turn aside

H1: BEHOLD THE BEAST for which i have turned back
H2: you see the beast THAT MADE ME TURN ASIDE
H3: see the beast that forced me to turn back
H4: look at the beast that drove me to turn back
```

I5 refs to the machine's iambic pentameter transla-

tion, while H1-4 refer to human translations. The machine also creates new translations:

```
i:  diro‘ de laltre cose chi vho scorte
I5: i shall explain the other things i saw

H1: speak will i of the other things i saw there
H2: ill also tell THE OTHER THINGS I SAW
H3: i will recount the other things i saw
H4: i here will tell the other things i saw
```

We can further change the target meter to anything we desire. To obtain iambic tetrameter (4-beat) translations, we delete the last two transitions of the P(m) model. We then get:

```
I4: in our life the journey way
    i found myself deep on dark wood
    that lost straightforward pathway had.

    ah how to say the what is hard
    this forest savage rough and stern
    the very thought renews the fear.
```

Translations and scans are uneven, but we have significant flexibility. We can even request translations that avoid the English letter A, by adding a filter to the end of the FST cascade, obtaining:

```
I5: in midst upon the journey of our life
    i found myself within the wood obscure
    <fail>
```

To steer clear of the adjective *dark* in the second line, the system switches from *forest* to *wood*, so obtain a proper scan. The third line fails because all paths through the translation lattice contain an A somewhere.

Translating blind-test data proves to be more difficult. We hold out Canto XXXI of DC's Paradiso section for testing. Figure 13 shows a portion of the translation results. The MT system handles unknown Italian words by passing them through to the output. The P($m|e$) meter model cannot process those words, accounting for the I5 failure rate.

Here, we get a first look at statistical MT translating poetry into rhythmic structures—as with all MT, there are successes and problems, and certainly more to do.

Original: in forma dunque di candida rosa mi si mostrava la milizia santa che nel suo sangue cristo fece sposa ma laltra che volando vede e canta la gloria di colui che la nnamora e la bonta‘ che la fece cotanta
Human translation: in fashion then as of a snow white rose displayed itself to me the saintly host whom christ in his own blood had made his bride but the other host that flying sees and sings the glory of him who doth enamour it and the goodness that created it so noble
Phrase-based translation (PBMT): in the form so rose candida i now was shown the militia holy that in his blood christ did bride but the other that flying sees and sings the glory of him that the nnamora and the goodness that the made cotanta
PBMT + meter model: <fail> i now was shown the holy soldiery that in his blood he married jesus christ but flying sees and sings the other which <fail> <fail>

Figure 13: Automatic translation of blind-test data from Dante's *Divine Comedy*.

6 Future Work

We plan to release all our of data in useful, processed form. Below we list directions for future research. In general, we see many interesting paths to pursue.

Analysis. Proper use of one-syllable words remains tricky. Lines coming out of generation

and translation do not always scan naturally when read aloud by a person. We trace such errors to the fact that our lexical probabilities are context-independent. For example, we have:

```
P(S  | off) = 0.39
P(S* | off) = 0.61
```

When we look at Viterbi alignments from the analysis task, we see that when *off* is preceded by the word *far*, the probabilities reverse dramatically:

```
P(S  | off, after far) = 0.95
P(S* | off, after far) = 0.05
```

Similarly, the probability of stressing *at* is 40% in general, but this increases to 91% when the next word is *the*. Developing a model with context-dependent probabilities may be useful not only for improving generation and translation, but also for improving poetry analysis itself, as measured by analysis task accuracy.

Other potential improvements include the use of prior knowledge, for example, taking word length and spelling into account, and exploiting incomplete pronunciation dictionary information.

Generation. Evaluation is a big open problem for automatic poetry generation—even evaluating human poetry is difficult. Previous suggestions for automatic generation include acceptance for publication in some established venue, or passing the Turing test, i.e., confounding judges attempts to distinguish machine poetry from human poetry. The Turing test is currently difficult to pass with medium-sized Western poetry.

Translation. The advantage of translation over generation is that the source text provides a coherent sequence of propositions and images, allowing the machine to focus on "how to say" instead of "what to say." However, translation output lattices offer limited material to work with, and as we dig deeper into those lattices, we encounter increasingly disfluent ways to string together renderings of the source substrings.

An appealing future direction is to combine translation and generation. Rather than translating the source text, a program may instead use the source text for inspiration. Such a hybrid translation/generation program would not be bound to translate every word, but rather it could more freely combine lexical material from its translation tables with other grammatical and lexical resources. Interestingly, human translators sometimes work this way when they translate poetry—many excellent works have been produced by people with very little knowledge of the source language.

Paraphrasing. Recently, e→f translation tables have been composed with f→e tables, to make e→e tables that can paraphrase English into English (Bannard and Callison-Burch, 2005). This makes it possible to consider statistical translation of English prose into English poetry.

Acknowledgments

This work was partially supported by NSF grant IIS-0904684.

References

C. Bannard and C. Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proc. ACL*.

P. Brown, V. Della Pietra, S. Della Pietra, and R. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2).

B. Diaz-Agudo, P. Gervas, and P. A. Gonzalez-Calero. 2002. Poetry generation in COLIBRI. In *Proc. EC-CBR*.

P. Gervas. 2001. An expert system for the composition of formal Spanish poetry. *Journal of Knowledge-Based Systems*, 14:200–1.

P. Gervas. 2010. Engineering linguistic creativity: Bird flight and jet planes. Invited talk, CALC-10.

J. Graehl. 1997. Carmel finite-state toolkit. http://www.isi.edu/licensed-sw/carmel.

L. Jiang and M. Zhou. 2008. Generating Chinese couplets using a statistical MT approach. In *Proc. COLING*.

P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2007. Moses: open source toolkit for statistical machine translation. In *Proc. ACL*.

H. Manurung, G. Ritchie, and H. Thompson. 2000. Towards a computational model of poetry generation. In *Proc. AISB'00 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*.

H. Manurung. 2003. *An evolutionary algorithm approach to poetry generation*. Ph.D. thesis, University of Edinburgh.

Y. Netzer, D. Gabay, Y. Goldberg, and M. Elhadad. 2009. Gaiku : Generating Haiku with word associations

norms. In *Proc. NAACL Workshop on Computational Approaches to Linguistic Creativity*.

N. Tosa, H. Obara, and M. Minoh. 2008. Hitch Haiku: An interactive supporting system for composing Haiku poem. In *Proc. International Conference on Entertainment Computing*.

M. T. Wong and A. H. W. Chun. 2008. Automatic Haiku generation using VSM. In *Proc. ACACOS*.