# "Poetic" Statistical Machine Translation: Rhyme and Meter

**Dmitriy Genzel** **Jakob Uszkoreit** **Franz Och**
Google, Inc.
1600 Amphitheatre Pkwy
Mountain View, CA 94043, USA
{dmitriy,uszkoreit,och}@google.com

## Abstract

As a prerequisite to translation of poetry, we implement the ability to produce translations with meter and rhyme for phrase-based MT, examine whether the hypothesis space of such a system is flexible enough to accomodate such constraints, and investigate the impact of such constraints on translation quality.

## 1 Introduction

Machine translation of poetry is probably one of the hardest possible tasks that can be considered in computational linguistics, MT, or even AI in general. It is a task that most humans are not truly capable of. Robert Frost is reported to have said that poetry is that which gets lost in translation. Not surprisingly, given the task's difficulty, we are not aware of any work in the field that attempts to solve this problem, or even discuss it, except to mention its difficulty, and professional translators like to cite it as an example of an area where MT will never replace a human translator. This may well be true in the near or even long term. However, there are aspects of the problem that we can already tackle, namely the problem of the poetic form.

Vladimir Nabokov, in his famous translation of *Eugene Onegin* (Nabokov, 1965), a poem with a very strict meter and rhyming scheme, heavily disparages those translators that attempt to preserve the form, claiming that since it is impossible to perfectly preserve both the form and the meaning, the form must be entirely sacrificed. On the other hand, Douglas Hofstadter, who spends 600 pages describing how to translate a 60 word poem in 80 different ways in *Le Ton beau de Marot* (1998), makes a strong case that a poem's form must be preserved in translation, if at all possible. Leaving the controversy to the professional translators, we investigate whether or not it is possible to produce translations that conform to certain metrical constraints common in poetry.

Statistical machine translation techniques, unlike their traditional rule-based counterparts, are in fact well-suited to the task. Because the number of potential translation hypotheses is very large, it is not unreasonable to expect that some of them should conform to an externally imposed standard. The goal of this paper is to investigate how these hypotheses can be efficiently identified, how often they are present, and what the quality penalty for imposing them is.

## 2 Related Work

There has been very little work related to the translation of poetry. There has been some work where MT techniques were used to produce poetry (Jiang and Zhou, 2008). In other computational poetry work, Ramakrishnan et al (2009) generate song lyrics from melody and various algorithms for poetry generation (Manurung et al., 2000; Díaz-Agudo et al., 2002). There are books (Hartman, 1996) and articles (Bootz, 1996) on the subject of computer poetry from a literary point of view. Finally, we must mention Donald Knuth's seminal work on complexity of songs (Knuth, 1984).



*Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 158–166,
MIT, Massachusetts, USA, 9-11 October 2010. 

## 3 Statistical MT and Poetry

We can treat any poetic form as a constraint on the potential outputs. A naive approach to ensure that an output of the MT system is, say, a *haiku*, is to create a *haiku* detector and to examine a (very large) n-best list of translations. This approach would not succeed very often, since the *haikus* that may be among the possible translations are a very small fraction of all translations, and the MT decoder is not actively looking for them, since it is not part of the cost it attempts to minimize. Instead, we would want to recast "Haikuness" as a feature function, such that a real *haiku* has 0 cost, and those outputs that are not, have large cost. This feature function must be local, rather than global, so as to guide the decoder search.

The concept of feature functions as used in statistical MT is described by Och and Ney (Och and Ney, 2002). For a phrase based system, a feature function is a function whose inputs are a partial hypothesis state $s_{in}$, and a phrase pair $p$, and whose outputs are the hypothesis state after $p$ is appended to the hypothesis: $s_{out}$, and the cost incurred, $c$. For hierarchical, tree-to-string and some other types of MT systems which combine two partial hypotheses and are not generating translations left-to-right, one instead has two partial hypotheses states $s_{left}$ and $s_{right}$ as inputs, and the outputs are the same. Our first goal is to describe how these functions can be efficiently implemented.

The feature function costs are multiplied by fixed weights and added together to obtain the total hypothesis cost. Normally feature functions include the logarithm of probability of target phrase given source, source given target and other phrase-local features which require no state to be kept, as well as features like language model, which require non-trivial state. The weights are usually learned automatically, however we will set them manually for our feature functions to be effectively infinite, since we want them to override all other sources of information.

We will now examine some different kinds of poetry and consider the properties of such feature functions, especially with regard to keeping necessary state. We are concerned with minimizing the amount of information to be kept, both due to memory requirements, and especially to ensure that compatible hypotheses can be efficiently recombined by the decoder.

### 3.1 Line-length constrained poetry

Some poetic genres, like the above-mentioned *haiku*, require that a poem contain a certain number of lines (3 for *haiku*), each containing a certain number of syllables (5,7,5 for *haiku*). These genres include *lanternes*, *fibs*, *tankas*, and many others. These genres impose two constraints. The first constraint is on total length. This requires that each hypothesis state contain the current translation length (in syllables). In addition, whenever a hypothesis is expanded, we must keep track of whether or not it would be possible to achieve the desired final length with such an expansion. For example, if in the initial state, we have a choice between two phrases, and picking the longer of the two would make it impossible to have a 17-syllable translation later on, we must impose a high cost on it, so as to avoid going down a garden path.

The second constraint is on placing line breaks: they must come at word boundaries. Therefore the 5th and 12th (and obviously 17th) syllable must end words. This also requires knowing the current hypothesis' syllable length, but unlike the first constraint, it can be scored entirely locally, without considering possible future expansions. For either constraint, however, the sentence has to be assembled strictly left-to-right, which makes it impossible to build partial hypotheses that do not start the sentence, which hierarchical and tree-to-string decoders require.

### 3.2 Rhythmic poetry

Some famous Western poetry, notably Shakespeare, is written in rhythmic poetry, also known as *blank verse*. This poetry imposes a constraint on the pattern of stressed and unstressed syllables. For example, if we use 0 to indicate no stress, and 1 to indicate stress, *blank verse* with iambic foot obeys the regular expression $(01)*$, while one with a dactylic foot looks like $(100)*$. This genre is the easiest to handle, because it does not require current position, but only its value modulo foot length (e.g. for an iamb, whether the offset is even or odd). It is even possible, as described in Section 4, to track this form in a decoder that is not left-to-right.

### 3.3 Rhythmic and rhyming poetry

The majority of English poetry that was written until recently has both rhythm and rhyme. Generally speaking, a poetic genre of this form can be described by two properties. The first is a rhyming scheme. A rhyming scheme is a string of letters, each corresponding to a line of a poem, such that the same letter is used for the lines that rhyme. E.g. *aa* is a scheme for a couplet, a 2-line poem whose lines rhyme. A *sonnet* might have a complicated scheme like *abbaabbacdecde*. The second property concerns meter. Usually lines that rhyme have the same meter (i.e. the exact sequence of stressed and unstressed syllables). For example, an *iambic pentameter* is an *iamb* repeated 5 times, i.e. *0101010101*. We can describe a genre completely by its rhyming scheme and a meter for each letter used in the rhyming scheme. We will refer to this object as *genre description*. E.g. $\{abab, a : 010101, b : 10101010\}$ is a quatrain with trimeter iambic and tetrameter trochaic lines. Note that the other two kinds of poetry can also be fit by this structure, if one permits another symbol (we use *) to stand for the syllables whose stress is not important, e.g. a *haiku*: $\{abc, a : *****, b : *******, c : *****\}$. For this type of genre, we need to obey the same two constraints as in the line-based poetry, but also to ensure that rhyming constraints hold. This requires us to include in a state, for any outstanding rhyme letter, the word that occurred at the end of that line. It is not sufficient to include just the syllable that must rhyme, because we wish to avoid self-rhymes (word rhyming with an identical word).

## 4 Stress pattern feature function

We will first discuss an easier special case, namely a feature function for blank verse, which we will refer to as *stress pattern* feature function. This feature function can be used for both phrase-based and hierarchical systems.

In addition to a statistical MT system (Och and Ney, 2004; Koehn et al., 2007), it is necessary to have the means to count the syllables in a word and to find out which ones are stressed. This can be done with a pronunciation module of a text-to-speech system, or a freely available pronunciation dictionary, such as CMUDict (Rudnicky, 2010). Out-of-vocabulary words can be treated as always imposing a high cost.

### 4.1 Stress pattern for a phrase-based system

In a phrase based system, the feature function state consists of the current hypothesis length modulo foot length. For a 2-syllable foot, it is either 0 or 1, for a 3-syllable foot, 0, 1, or 2. The proposed target phrase is converted into a stress pattern using the pronunciation module, and the desired stress pattern is left shifted by the current offset. The cost is the number of mismatches of the target phrase vs. the pattern. For example, if the desired pattern is *010*, current offset is 1, and the proposed new phrase has pattern *10011*, we shift the desired pattern by 1, obtaining *100* and extend it to length 5, obtaining *10010*, matching it against the proposal. There is one mismatch, at the fifth position, and we report a cost of 1. The new state is simply the old state plus phrase length, modulo foot length, 0 in this example.

### 4.2 Stress pattern for a hierarchical system

In a hierarchical system, we in general do not know how a partial hypothesis might be combined on the left. A hypothesis that is a perfect fit for pattern *010* would be horrible if it is placed at an offset that is not divisible by 3, and vice versa, an apparently bad hypothesis might be perfectly good if placed at such an offset. To solve this problem, we create states that track how well a partial hypothesis fits not only the desired pattern, but all patterns obtained by placing this pattern at any offset, and also the hypothesis length (modulo foot length, as usual). For instance, if we observe a pattern *10101*, we record the following state: {*length: 1, 01 cost: 5, 10 cost: 0*}. If we now combine this state with another, such as {*length: 0, 01 cost: 1, 10 cost: 0*}, we simply add the lengths, and combine the costs either of the same kind (if left state's length is even), or the opposite (if it is odd). In this instance we get {*length: 1, 01 cost: 5, 10 cost: 1*}. If both costs are greater than 0, we can subtract the minimum cost and immediately output it as cost: this is the unavoidable cost of this combination. For this example we get cost of 1, and a new state: {*length: 1, 01 cost: 4, 10 cost: 0*}. For the final state, we output the remaining cost for the pattern we desire. The approach is very similar for feet of length 3.

### 4.3 Stress pattern: Whatever fits

With a trivial modification we can output translations that can fit any one of the patterns, as long as we do not care which. The approach is identical for both hierarchical and phrase-based systems. We simply track all foot patterns (length 2 and length 3 are the only ones used in poetry) as in the above algorithm, taking care to combine the right pattern scores based on length offset. The length offset now has to be tracked modulo 2*3.

This feature function can now be used to translate arbitrary text into blank verse, picking whatever meter fits best. If no meters can fit completely, it will produce translations with the fewest violations (assuming the weight for this feature function is set high).

## 5 General poetic form feature function

In this section we discuss a framework for tracking any poetic genre, specified as a genre description object (Section 3.3 above). As in the case of the stress pattern function, we use a statistical MT system, which is now required to be phrase-based only. We also use a pronunciation dictionary, but in addition to tracking the number and stress of syllables, we must now be able to provide a function that classifies a pair of words as rhyming or non-rhyming. This is in itself a non-trivial task (Byrd and Chodorow, 1985), due to lack of a clear definition of what constitutes a rhyme. In fact rhyming is a continuum, from very strong rhymes to weak ones. We use a very weak definition which is limited to a single syllable: if the final syllables of both words have the same nucleus and coda[1], we say that the words rhyme. We accept this weak definition because we prefer to err on the side of over-generation and accept even really bad poetry.

### 5.1 Tracking the target length

The hardest constraint to track efficiently is the range of lengths of the resulting sentence. Phrase-based decoders use a limited-width beam as they build up possible translations. Once a hypothesis drops out of the beam, it cannot be recovered, since no backtracking is done. Therefore we cannot afford to explore a part of the hypothesis space which has no possible solutions for our constraints, we must be able to prune a hypothesis as soon as it leads us to such a subspace, otherwise we will end up on an unrecoverable garden path. To avoid this problem, we need to have a set of possible sentence lengths available at any point in the search, and to impose a high cost if the desired length is not in that set.

Computing this set exactly involves a standard dynamic programming sweep over the phrase lattice, including only uncovered source spans. If the maximum source phrase size is $k$, source sentence length is $n$ and maximum target/source length ratio for a phrase is $l$ (and therefore target sentence is limited to at most $ln$ words), this sweep requires going over $O(n^2)$ source ranges, each of which can be produced in $k$ ways, and tracking $ln$ potential lengths in each, resulting in $O(n^3kl)$ algorithm. This is unacceptably slow to be done for each hypothesis (even noting that hypotheses with the same set of already covered source position can share this computation).

We will therefore solve this task approximately. First, we can note that in most cases the set of possible target lengths is a range. This is due to phrase extraction constraints, which normally ensure that the lengths of target phrases form a complete range. This means that it is sufficient to track only a minimum and maximum value for each range, reducing time to $O(n^2k)$. Second, we can note that whenever a source range is interrupted by a covered phrase and split into two ranges, the minimal and maximal sentence length is simply the sum of the corresponding lengths over the two uncovered subranges, plus the current hypothesis length. Therefore, if we precompute the minimum and maximum lengths over all ranges, using the same dynamic programming algorithm in advance, it is only necessary to iterate over the uncovered ranges (at most $O(n)$, and $O(1)$ in practice, due to reordering constraints) at runtime and sum their minimum and maximum values. As a result, we only need to do $O(n^2k)$ work upfront, and on average $O(1)$ extra work for each hypothesis.

### 5.2 State space

A state for the feature function must contain the following elements:

- Current sentence length (in syllables)

[1] In phonology, nucleus and coda together are in fact called *rhyme* or *rime*

- Set of uncovered ranges (as needed for the computation above)
- Zero or more letters from the rhyming scheme with the associated word that has an outstanding rhyme

### 5.3 The combination algorithm

To combine the hypothesis state $s_{in}$ with a phrase pair $p$, do the following

1. Initialize cost as 0, $s_{out}$ as $s_{in}$
2. Update $s_{out}$: increment sentence length by target phrase length (in syllables), update coverage range
3. Compute minimum and maximum achievable sentence length; if desired length not in range, increment cost by a penalty
4. For each word in the target phrase
   (a) If the word's syllable pattern does not match against desired pattern, add number of mismatches to cost
   (b) If at the end of a line:
      i. If the line would end mid-word, increment cost by a penalty
      ii. Let $x$ be this line's rhyme scheme letter
      iii. If $x$ is present in the state $s_{out}$, check if the word associated with $x$ rhymes with the current word, if not, increment cost by a penalty
      iv. Remove $x$ with associated word from the state $s_{out}$
      v. If letter $x$ occurs further in the rhyming scheme, add $x$ with the current word to the state $s_{out}$

### 5.4 Tracking multiple patterns

The above algorithm will allow to efficiently search the hypothesis space for a single *genre description* object. In practice, however, there may be several desirable patterns, any one of which would be acceptable. A naive approach, to use multiple feature functions, one with each pattern, does not work, since the decoder is using a (log-)linear model, in which costs are additive. As a result, a pattern that matches one pattern, but not another, will still have high cost, perhaps as high as a pattern that partially matches both. We need to combine feature functions not linearly, but with a *min* operator. This is easily achieved by creating a combined state that encodes the union of each individual function's states (which can share most of the information), and in addition each feature function's current total cost. As long as at least one function has zero cost (i.e. can potentially match), no cost is reported to the decoder. As soon as all costs become positive, the minimum over all costs is reported to the decoder as unavoidable cost, and should be subtracted from each feature function cost, bringing the minimum stored in the output state back to 0.

It is also possible to prune the set of functions that are still viable, based on their cost, to avoid keeping track of patterns that cannot possibly match. Using this approach we can translate arbitrary text, provide a large number of poetic patterns and expect to get some sort of poem at the end. Given a wide variety of poetic genres, it is not unreasonable to expect that for most inputs, some pattern will apply. Of course, for translating actual poetry, we would likely have a specific form in mind, and a positive outcome would be less likely.

## 6 Results

We train a baseline phrase-based French-English system using WMT-09 corpora (Callison-Burch et al., 2009) for training and evaluation. We use a proprietary pronunciation module to provide phonetic representation of English words.

### 6.1 Stress Pattern Feature Function

We have no objective means of "poetic" quality evaluation. We are instead interested in two metrics: percentage of sentences that can be translated while obeying a stress pattern constraint, and the impact of this constraint on BLEU score (Papineni et al., 2002). Obviously, WMT test set is not itself in any way poetic, so we use it merely to see if arbitrary text can be forced into this constraint.

The BLEU score impact on WMT has been fairly consistent during our experimentation: the BLEU score is roughly halved. In particular, for the above system the baseline score is **35.33**, and stress

Table 1: Stress pattern distribution

| Name | Pattern | % of matches |
|---|---|---|
| Iamb | 01 | 9.6% |
| Trochee | 10 | 7.2% |
| Anapest | 001 | 27.1% |
| Amphibrach | 010 | 32.1% |
| Dactyl | 100 | 23.8% |

pattern-constrained system only obtains **18.93**.

The proportion of sentences successfully matched is **85%**, and if we permit a single stress error, it is **93%**, which suggests that the constraint can be satisfied in the great majority of cases. The distribution of stress patterns among the perfect matches is given in Table 1.

Some of the more interesting example translations with stress pattern enforcement enabled are given in table 2.

### 6.2 Poetic Form Feature Function

For poetic form feature function, we perform the same evaluation as above, to estimate the impact of forcing prose into an arbitrary poetic form, but to get more relevant results we also translate a poetic work with a specific genre requirement.

Our poetic form feature function is given a list of some 210 genre descriptions which vary from Haikus to Shakespearean sonnets. Matching any one of them satisfies the constraint. We translate WMT blind set and obtain a BLEU score of **17.28** with the baseline of **35.33** as above. The proportion of sentences that satisfied one of the poetic constraints is **87%**. The distribution of matched genres is given in Table 3. Some of the more interesting example translations are given in table 2.

For a proper poetic evaluation, we use a French translation of Oscar Wilde's *Ballad of Reading Gaol* by Jean Guiloineau as input, and the original Wilde's text as reference. The poem consists of 109 stanzas of 6 lines each, with a genre description of {*abcbdb, a/c/d: 01010101, b: 010101*}. The French version obeys the same constraint. We treat each stanza as a sentence to be translated. The baseline BLEU score is **10.27**. This baseline score is quite low, as can be expected for matching a literal MT translation against a professional poetical translation. We evaluate our system with a poetic constraint given above.

Table 3: Genre distribution for WMT corpus.
(Descriptions of these genres can be found in Wikipedia, http://en.wikipedia.org)

| Genre | Number | Percentage |
|---|---|---|
| No poem | 809 | 13.1% |
| Blank verse | 5107 | 82.7% |
| Couplet | 81 | 1.3% |
| Haiku | 42 | 0.7% |
| Cinquain | 33 | 0.5% |
| Dodoitsu | 24 | 0.4% |
| Quinzaine | 23 | 0.4% |
| Choka | 18 | 0.3% |
| Fib | 15 | 0.2% |
| Tanka | 14 | 0.2% |
| Lanterne | 4 | 0.1% |
| Triplet | 1 | 0.02% |
| Quatrain | 1 | 0.02% |
| Total | 6172 | 100% |

The resulting score is **7.28**. Out of 109 stanzas, we found 12 translations that satisfy the genre constraint (If we allow any poetic form, 108 out of 109 stanzas match some form). Two sample stanzas that satisfied the constraints are given in Table 4.

## 7 Discussion and Future Work

In this work we demonstrate how modern-day statistical MT system can be constrained to search for translations obeying particular length, meter, and rhyming constraints, whether a single constraint, or any one of a set. We further demonstrate that the hypothesis space is often rich enough that these constraints can be satisfied. The impact on translation quality, however, is quite profound, as is to be expected. It seems that at the present state of machine translation, one does indeed have to choose between getting either the form or the meaning right. In the present form, however, we can already find good translations, as a sort of *found poetry* (Drury, 2006), by translating a large quantity of text, whether poetic or not.

This is the first attempt to deal with poetry translation, and the great majority of work to achieve reasonable quality in form and meaning still remains to be done. One major problem with the current feature function is that while it can avoid the areas of the search space where length constraints cannot be

Table 2: Example translations. Stressed syllables are italicized

| | |
|---|---|
| **Reference** | A police spokesman said three people had been arrested and the material was being examined. |
| **Baseline** | A policeman said that three people were arrested and that the material is currently being analyzed. |
| **Stress Pattern (001)** | A po*lice* said that *three* were ar*rest*ed and *that* the e*quip*ment is *cur*rently *be*ing e*xa*mined. |
| **Poetic**: *Couplet in amphibrachic tetrameter* | An *of*ficer *sta*ted that *three* were ar*rest*ed<br>and *that* the e*quip*ment is *cur*rently *test*ed. |

| | |
|---|---|
| **Reference** | A trio of retired generals launched a mutiny in the Lords, protesting against cuts in military spending: being armed-forces minister is, they claimed, a part-time job. |
| **Baseline** | A trio of retired generals have launched a revolt among the Lords, protesting against cuts in military spending: they have proclaimed only Minister of Defence is for them, a part-time employment. |
| **Stress Pattern (010)** | A *tri*o of *ge*neral re*ti*rement *launched* a re*bel*lion a*mong* Lords, pro*test*ing the *spen*ding cuts *troops*: they claimed *Mi*nister *on*ly de*fen*se is *for* them, a *part*-time job. |
| **Poetic**: *Blank Verse in amphibrachic trimeter* | A *tri*o of *ge*nerals re*ti*red<br>have *launched* an up*ri*sing a*mong* Lords,<br>pro*test*ing the *spend*ing cuts *mem*bers:<br>they *mi*nister *on*ly pro*claimed* the<br>de*fense* is for *them*, a part-*time* job. |

| | |
|---|---|
| **Reference** | We must continue to condemn human rights abuses in Burma. |
| **Baseline** | We must continue to denounce the violations of human rights abuses in Burma. |
| **Stress Pattern (100)** | *We* must con*ti*nue to *speak* out a*gainst* rights a*bu*ses com*mi*tted in *Bur*ma. |
| **Poetic**: *Haiku: 5-7-5 syllables* | We must continue<br>denounce violations of<br>human rights Burma. |

Table 4: Sample translations from Oscar Wilde's *Ballad of Reading Gaol*.

| **Wilde's original** | **Our translation** |
|---|---|
| He did not wring his hands, as do | Without hands twisted like these men, |
| Those witless men who dare | Poor men without hope, dare |
| To try to rear the changeling Hope | To nourish hope in our vault |
| In the cave of black Despair: | Of desperation there |
| He only looked upon the sun, | And looked toward the sun, drink cool |
| And drank the morning air. | Until the evening air. |
| With slouch and swing around the ring | We are in our circle we |
| We trod the Fool's Parade! | Dragged like the Fools' Parade! |
| We did not care: we knew we were | It mattered little, since we were |
| The Devil's Own Brigade: | The Devil's sad Brigade: |
| And shaven head and feet of lead | A shaved head and the feet of lead |
| Make a merry masquerade. | Regardless gay charade! |

satisfied, it cannot avoid the areas where rhyming constraints are impossible to satisfy. As a result, we need to allow a very wide hypothesis beam (5000 per each source phrase coverage), to ensure that enough hypotheses are considered, so that there are some that lead to correct solutions later. We do not currently have a way to ensure that this happens, although we can attempt to constrain the words that end lines to have possible rhymes, or employ other heuristics. A more radical solution is to create an entirely different decoding algorithm which places words not left-to-right, or in a hierarchical fashion, but first placing words that must rhyme, and building hypotheses around them, like human translators of poetry do.

As a result, the system at present is too slow, and we cannot make it available online as a demo, although we may be able to do so in the future.

The current approach relies on having enough lexical variety in the phrase table to satisfy constraints. Since our goal is not to be literal, but to obtain a satisfactory compromise between form and meaning, it would clearly be beneficial to augment target phrases with synonyms and paraphrases, or to allow for words to be dropped or added.

## 8 Acknowledgements

We would like to thank all the members of the MT team at Google, especially Richard Zens and Moshe Dubiner, for their help. We are thankful to the anonymous reviewers for their comments, especially to the one that to our amazement did the entire review in verse[2].

## References

P. Bootz. 1996. Poetic machinations. *Visible Language*, 30(2):118–37.

Roy J. Byrd and Martin S. Chodorow. 1985. Using an on-line dictionary to find rhyming words and pronunciations for unknown words. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 277–283, Chicago, Illinois, USA, July. Association for Computational Linguistics.

Chris Callison-Burch, Philipp Koehn, Christof Monz, and Josh Schroeder. 2009. Findings of the 2009 Workshop on Statistical Machine Translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 1–28, Athens, Greece, March. Association for Computational Linguistics.

B. Díaz-Agudo, P. Gervás, and P. González-Calero. 2002. Poetry generation in colibriza. In *Advances in Case-Based Reasoning*, pages 157–159. Springer.

John Drury. 2006. *The poetry dictionary*. Writer's Digest Books.

C.O. Hartman. 1996. *Virtual muse: experiments in computer poetry*. Wesleyan University Press.

Douglas R. Hofstadter. 1998. *Le Ton Beau De Marot: In Praise of the Music of Language*. Perseus Books Group.

[2]With the reviewer's permission, we feel that the extra work done by the reviewer deserves to be seen by more than a few people, and make it available for you to view at: `http://research.google.com/archive/papers/review_in_verse.html`

Long Jiang and Ming Zhou. 2008. Generating Chinese couplets using a statistical MT approach. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 377–384, Manchester, UK, August. Coling 2008 Organizing Committee.

D.E. Knuth. 1984. The complexity of songs. *Communications of the ACM*, 27(4):344–346.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June. Association for Computational Linguistics.

H.M. Manurung, G. Ritchie, and H. Thompson. 2000. Towards a computational model of poetry generation. In *Proceedings of AISB Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*, pages 79–86. Citeseer.

Vladimir Nabokov. 1965. *Eugene Onegin: A Novel in Verse by Alexandr Pushkin, Translated from the Russian*. Bollingen Foundation.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 295–302, Philadelphia, Pennsylvania, USA, July. Association for Computational Linguistics.

Franz Josef Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July. Association for Computational Linguistics.

Ananth Ramakrishnan A., Sankar Kuppan, and Sobha Lalitha Devi. 2009. Automatic generation of Tamil lyrics for melodies. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 40–46, Boulder, Colorado, June. Association for Computational Linguistics.

Alex Rudnicky. 2010. The Carnegie Mellon pronouncing dictionary, version 0.7a. Online.