Converting Prose into Rhyming Poetry

Luke Allen
Stanford University
source@cs.stanford.edu

Abstract

This paper explores several rule-based strategies to modify sentences in order to change length and word order while retaining the original sentence meaning. These strategies are then applied to create an application which converts English text into rhyming lines of poetry.

1 Introduction

Automated poetry generation is an interesting and mostly unexplored proving ground for paraphrase-generation algorithms. This paper briefly explores three types of rule-based sentence paraphrasing methods: deletion of non-essential text, rearrangement of phrases, and replacement of a word by its synonym. These methods are combined into an application which can rearrange an arbitrary input text into pairs of rhyming lines, while usually preserving a fair amount of the original meaning.

2 Related Work

Most work on sentence paraphrasing has focused on machine-translation methods, which use parallel corpora to identify phrases which have a similar meaning, as discussed in (Wubben et al., 2010). Machine translation is popular because it is the most general approach.

However, a smaller but still useful set of paraphrases can be generated by simple rule-based approaches using operations on parse trees. For example, Daelemans et al. (2004) discuss a system which can shorten sentences by deleting non-essential modifiers.

Automated poetry generation itself has received little research focus to date. A literature review identified only one other attempt at actual generation of poems, by Genzel et al. (2010), who modified the cost function of a machine translation system to control the syllable count of its translations, and then searched its possible translations of foreign-language poems to find English rhymes.

3 Algorithm

Each paraphrase method will be discussed as it applies to the poetry-generation algorithm:

3.1 Prerequisite: Testing Rhymes

The Carnegie Mellon Pronunciation Dictionary conveniently provides word pronunciations in the form of an array of phonemes. To determine if two words rhyme, we can simply compare their phoneme arrays, moving from the right. If the first consonants and the first vowel match, the words are considered to rhyme. Further matches produce a higher score.

Words' phoneme arrays are also used to determine syllable counts of a line of poetry. For words not in the CMU dictionary, a library by Sucher (2012) is used to guess the pronunciation of the last syllable and also to guess the word's syllable count.

3.2 Synonym Substitution

First, the algorithm uses the Stanford Parser to parse the original prose. This provides part-of-speech tags as well as other information which will

Figure 1: A pair of rhyming words defines two poetry lines. Paraphrasing rules can then be used to match their length.

be discussed later. WordNet and VerbNet are used, via the Python Natural Language Toolkit, to build a list of possible synonyms for each word. WordNet hypernyms are also included, because these usually preserve meaning. (Hyponyms were also tried, but experimentally these tended to be much worse at preserving meaning, even if we allow for very generous "poetic license.")

Each synonym in a word's synonym list is given a meaning-preservation score: The original word gets a perfect score. Synonyms from the most common (i.e. first) WordNet synonym set for the word are assumed to be most similar and are ranked next. Hypernyms are ranked lower than synonyms. (A full word sense disambiguation (WSD) algorithm has not been added, but good WSD would certainly improve accuracy here.)

3.3 Finding possible rhymes

Once a synonym list has been built for each word, each word is checked against all nearby words, to see if any of their synonyms rhyme. The result of this check is a list of possible rhymes in the local section of text. These possible rhymes are ranked by (rhyme quality score) * (meaning-preservation scores of the two rhyming synonyms). In a local area of 3 sentences, typically between 10 and 200 such rhyme pairs are identified.

At this point, we could simply take the best rhyme pair and insert a newline after each of the two rhyming words. (After replacing the words with synonyms, if necessary for the rhyme.) That would be a simple way to produce lines which rhyme. (See Figure 1, above)

However, the lines will probably be different lengths, in syllables. In order for the text to sound like a poem, the lines must be the same number of syllables, or very close. So, we need to paraphrase the involved sentences to match the line lengths. The following paraphrase methods will be attempted for each possible pair of rhyming words.

3.4 Phrase rearrangement

Note that prepositional phrases which act as modifiers can generally be moved freely within their larger phrase while preserving meaning. E.g. "I noticed that he walked under the bridge" → "I noticed that under the bridge he walked." The basic parse tree output by the Stanford Parser identifies prepositional phrases as a subtree under a larger phrase. Thus, during parsing, we can make a list of prepositional phrases and the destination they can move to. Currently, the only destination used is the beginning of the larger phrase. Other locations are possible, however. Also, other phrase types such as adverbs and temporal and locative pronouns (e.g. "now," "somewhere") could also be moved, but the algorithm currently does not attempt this.

So, after parsing the original text, this method yields a list of phrase movement options (in the form of phrase, destination pairs). The algorithm identifies which of these movements would affect the rhyming lines being investigated. Usually the number of relevant movement options is small (under 10), so it is feasible to try all 2^n possible combinations of movements. (This is easily done in code by using the bits of an integer counter to select the options to apply.) If the number of movement options is too large, the algorithm defaults to just trying each one individually.

Each possible rearranged section of text is passed to the final line-length-trimming stage below.

3.5 Deletion of non-essential words

It is often possible to delete adverbs, adjectives, prepositional phrases, and sometimes whole sentences without corrupting meaning. These deletable phrases are identified in the basic parse tree from the Stanford Parser. Each deletable phrase is given a meaning-preservation score based on its part of speech (adverbs are scored as most deletable, then adjectives, then sentences, then

prepositional phrases. Also, adverbs/adjectives which represent negation are not deletable.)

Because some deletable phrases are inside other deletable phrases, the algorithm stores them as a "forest" of trees. The root node of each tree is an instruction to delete an entire sentence, and its children are instructions to delete phrases within the sentence. (Note that because this structure came from a tree, sibling phrases are always independent of each other, with no overlap in the words they delete, and children are entirely contained in the parent phrase. See Figure 2, below.)

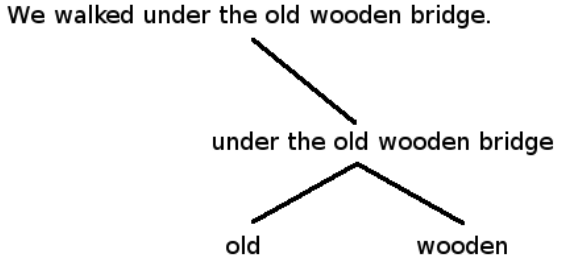


Figure 2: An example tree from the forest of deletion options. The deletable sentence contains one deletable prepositional phrase, which contains two independently deletable adjectives. The tree structure captures those relationships.

To match the length of two rhyming lines, the algorithm must search all combinations of relevant deletions. At first glance this appears very computationally complex, but it can actually be performed in polynomial time. (See Appendix B.)

If the lengths of two rhyming lines can be made to match, they are saved, along with a total rhyme-and-paraphrase score calculated as:

(rhyme quality) * (synonym quality) * (meaning preservation of rearrangements) * (meaning preservation of deletions)

The best-scoring rhyme pair is printed. Then, any word rearrangements are applied to the original text (to maintain consistency with the lines printed so far), and the whole algorithm is repeated on an excerpt which begins with the next word in the original text. (The algorithm simply chooses each rhyme pair greedily with no attempt at optimizing future rhymes.)

4 Results

When tested on excerpts from books, the algorithm almost always finds rhymes successfully. This is to be expected because of the very large combinatorial space provided by the paraphrase methods above.

The rate of successful meaning-preservation for each paraphrase method tends to vary with the writing style of the original text, which makes numerical success rates unreliable. So results are discussed somewhat subjectively here:

Of the tested paraphrase methods, synonym substitution was most likely to corrupt sentence meaning because of the difficulty of word sense disambiguation. Synonym substitution is unavoidable for automated rhyming, so better WSD would provide the single largest improvement to the algorithm.

The deletion of non-essential modifiers is a powerful tool for controlling line length, and it usually preserves meaning. However, the part-of-speech labels in the basic parse tree are not always sufficient to identify whether a phrase is non-essential. The more advanced Stanford dependency parse would likely allow better deletion rules. (E.g. it could identify that the prepositional phrase is essential in the sentence "He was under the bridge" but not essential in "He walked under the bridge.") This was not investigated further due to time constraints, but would likely yield a large improvement in meaning-preservation.

Phrase rearrangement was by far the best at preserving meaning. Unconventional phrase arrangements are often used by human poets, and are usually easy to understand. Therefore, investigating more rearrangement rules would likely be very fruitful future work in paraphrase generation.

Samples of inputs and outputs are in Appendix A below. Besides paraphrase quality, the largest weakness relative to human poems is in the rhythm of the rhymes: human poets almost always locate rhymes at natural pauses, such as the end of a sentence or phrase. The algorithm assigns a higher score to rhymes at the end of a phrase, but the existing paraphrase rules are not powerful enough to reliably find rhymes at the end of a phrase. More paraphrase options would improve

this.

5 Future work

As discussed above, the quality of the rules-based paraphrases would be improved by better WSD, better classification of deletable phrases, and more paraphrasing rules. Another unexplored paraphrase option is to pad sentence length by inserting filler words with low meaning content. This could be efficiently added to the existing word-deletion step, and would add flexibility for cases where the original text has few deletable modifiers.

Even with its currently limited number of paraphrase rules, the algorithm demonstrates the flexibility of rule-based paraphrase generation. It also highlights the potential for poem generation as a proving ground for paraphrase generation in the presence of constraints.

References

Daelemans, Hothker and Sang, 2004. Automatic Sentence Simplification for Subtitling in Dutch and English. University of Antwerp.

Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430.

Genzel, Uszkoreit, and Och, 2010. "Poetic" Statistical Machine Translation: Rhyme and Meter. Google Inc. Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 158–166, MIT, Massachusetts, USA

Princeton University "About WordNet." WordNet. Princeton University. 2010. <http://wordnet.princeton.edu>

Sucher, 2012. Nantucket: An Accidental Limerick Detector. http://www.daniellesucher.com/2012/04/nantucket-an-accidental-limerick-detector/

Wubben, van den Bosch, Krahmer, 2010. Paraphrase Generation as Monolingual Translation: Data and Evaluation. Tilburg University, The Netherlands

Appendix A. Sample Outputs

Below are sample inputs and the algorithm's output. The average execution time for these samples was 1.5 seconds per line on a modern quad-core PC. (No attempt was made to choose examples where the algorithm does well. Also, because it is a greedy algorithm, there are usually a few words left at the end with no possible rhyme.)

(From *Superiority* by Arthur C. Clarke):

The ultimate cause of our failure was a simple one: despite all statements to the contrary, it was not due to lack of bravery on the part of our men, or to any fault of the Fleet's. We were defeated by one thing only-by the inferior science of our enemies. I repeat-by the inferior science of our enemies.

When the war opened we had no doubt of our ultimate victory. The combined fleets of our allies greatly exceeded in number and armament those which the enemy could muster against us, and in almost all branches of military science we were their superiors. We were sure that we could maintain this superiority. Our belief proved, alas, to be only too well founded.

Of our failure the ultimate cause
was a simple one: of the Fleet 's, it was
not to lack, or to any fault. When the war capsize
we had no doubt. The combined fleets of our allies
exceeded those which the enemy could summon
against us, and of scientific discipline
in all branches we were their leader.
Our psychological feature
proved alas to be only too well founded. *<--no rhymes

[This example had some deletions of essential words but otherwise has good paraphrase quality.]

(From *The Adventures of Sherlock Holmes* by Arthur Conan Doyle):

Sherlock Holmes was transformed when he was hot upon such a scent as this. Men who had only known the quiet thinker and logician of Baker Street would have failed to recognise him. His face flushed and darkened. His brows were drawn into two hard black lines, while his eyes shone out from beneath them with a steely glitter. His face was bent downward, his shoulders bowed, his lips compressed, and the veins stood out like whipcord in his long, sinewy neck. His nostrils seemed to dilate with a purely animal lust for the chase, and his mind was so absolutely concentrated upon the matter before him that a question or remark fell unheeded upon his ears, or, at the most, only provoked a quick, impatient snarl in reply. Swiftly and silently he made his way along the track which ran through the meadows, and so by way of the woods to the Boscombe Pool. It

was damp, marshy ground, as is all that district, and there were marks of many feet, both upon the path and amid the short grass which bounded it on either side. Sometimes Holmes would hurry on, sometimes stop dead, and once he made quite a little detour into the meadow. Lestrade and I walked behind him, the detective indifferent and contemptuous, while I watched my friend with the interest which sprang from the conviction that every one of his actions was directed towards a definite end.

His face flushed and shrivel.
His brows were scribble
into two hard black lines, while his eyes
shone out. His face was bent, his shoulders rise,
his lips compressed, and in his long, neck the veins straddle
out like whipcord. His nostrils seemed with a animal
lust for the chase to dilate, and before
him his mind was so that a question or
remark fell disfigure
or, at the most, bewilder
a quick, snarl in reply. Swiftly
and silently he decree
his way along the track which, and through
the meadows by way of the woods brew
to the Boscombe Pool. It was, marshy ground, as is
all that district, and amid the short grass which whiz
it on either side there were marks of many pes,
both upon the path and. Sometimes Holmes would traipse
on, stop, and into the parcel
once he made quite a little
detour. Lestrade and I shuffle,
the detective and disdainful,
while with the interest which sprang that every single
was directed towards a definite terminal
I watched my friend. *<--no rhymes

[This example shows some bad synonym substitutions. And, the synonym-substitution algorithm currently only inserts the root word (lemma) of a synonym, so it doesn't match verb form or noun plurality. Also, the deletion algorithm currently doesn't remove conjunctions when necessary.]

(From *Atlas Shrugged* by Ayn Rand):

"To trade by means of money is the code of the men of good will. Money rests on the axiom that every man is the owner of his mind and his effort. Money allows no power to prescribe the value of your effort except the voluntary choice of the man who is willing to trade you his effort in return. Money permits you to obtain for your goods and your labor that which they are worth to the men who buy them, but no more. Money permits no deals except those to mutual benefit by the unforced judgment of the traders. Money demands of you the recognition that men must work for their own benefit, not for their own injury, for their gain, not their loss–the recognition that they are not beasts of burden, born to carry the weight of your misery–that you must offer them values, not wounds–that the common bond among men is not the exchange of suffering, but the exchange of goods. Money demands that you sell, not your weakness to men's stupidity, but your talent to their reason; it demands that you buy, not the shoddiest they offer, but the best that your money can find. And when men live by trade–with reason, not force, as their final arbiter–it is the best product that wins, the best performance, the man of best judgment and highest ability–and the degree of a man's productiveness is the degree of his reward. This is the code of existence whose tool and symbol is money. Is this what you consider evil?"

Money allows no power to prescribe the value
of your effort except the choice. Money permits you
to obtain for your goods and your
labor that which they are, but no more.
Money permits no muckle
except those to mutual
benefit by the view
of the traders. Of you
money demands the recognition that for
their injury, born of your misery to store
the weight-- ***** <--no rhymes
that you must offer them values, not wounds-- that the bail
is not the exchange, but the exchange adult male
must work, not. And when by trade--, not force, as their
final arbiter-- men live it is the best ware
that of best judgment and highest
ability wins, the practiced
performance, the humanity--
-- and the degree is the degree
This is the code whose tool and abstraction
is money. Is this what you imagine
evil? ***** <--no rhymes

[This example showed bad paraphrasing performance. This is because the original text had very few unnecessary modifiers and very carefully-chosen words, so both deletion and synonym-substitution tended to destroy meaning. This highlights the need for better WSD and for an option to add filler words.]

Appendix B. Word Deletion Algorithm

This section describes details of the algorithm which deletes words to match the length of two rhyming lines, introduced in section 3.5 above.

Our goal is to list the best ways to remove between 0 and MaxSyl syllables from a line, where MaxSyl is the difference between the original line length and a user-specified minimum line length. (We are obviously not interested in removing more syllables than that.) Once we have a list of the way to remove each number of syllables, we will know the achievable line lengths for line A, and after running the same algorithm on line B, we can identify possible matching lengths.

The problem of finding the best way to remove a given number of syllables from a line of text is actually an example of the known "0-1 Knapsack Problem with integer weights." It can be solved in polynomial time, using the insight that we are only interested in a finite number of deleted syllables, and we are not interested in multiple ways to delete the same number of syllables (only the best-scoring way).

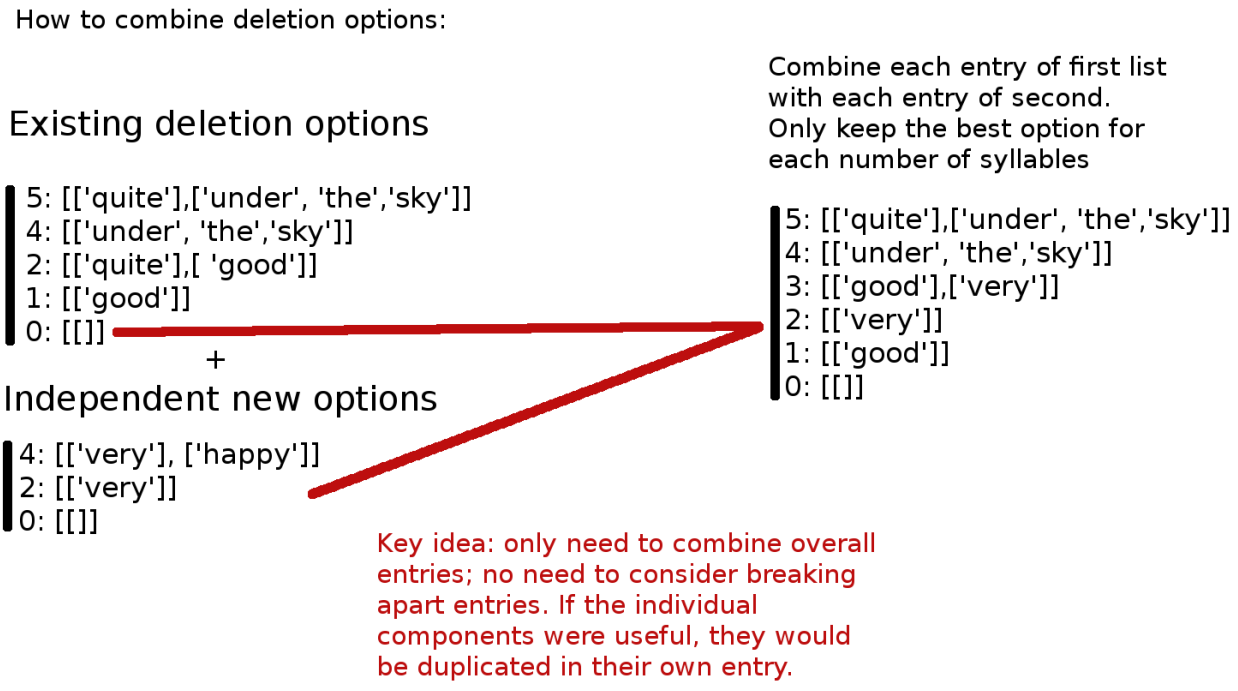
Recall Figure 2 in section 3.5. A given line to be shortened contains a "forest" of such sentence-level deletion trees (or portions of the sentence-level tree, in cases where the sentence is not entirely contained in the line). We can put an artificial node on top to turn that "forest" into a new tree which is entirely inside the line in question. Then we use a recursive algorithm to find the possible combinations of deletion options for that tree.

The heart of the algorithm is to build, at each node, a small Python dictionary which holds {key: value} entries of the form:

{numSyllables to delete: List of nodes to delete to best accomplish that}

The dictionaries for the bottom "leaf" nodes include only options to delete 0 syllables and to delete themselves. Each higher node builds a dictionary by combining its independent children. This is done by combining each entry from child 1's dictionary with each entry from child 2 (including the entry which removes 0 words). We will fill the dictionary for the higher-level node with the best resulting option for deleting each number of syllables. (See figure below.) Combinations which are not the best-scoring for their syllable count, or which yield more than MaxSyl deleted syllables, are discarded. Last, the high-level node adds the option to delete itself.

Note that the children's dictionary entries are only combined as **whole** units. It is never necessary to combinatorially break apart a child's



Appendix B Figure 1: Example of combining two children's dictionaries

entry; if the components of an entry were useful in other combinations, those would already be represented by other entries in that child's dictionary, because the child had tried all combinations while making its list.

After running this recursive algorithm, the very top node will contain the list of all possible numbers of syllables which can be deleted, up to MaxSyl. The complexity is roughly

$$O(MaxSyl^2 * \text{number of deletable nodes})$$

which scales roughly with (line length)^3. This is fast enough that we can easily run it on each of the possibly thousands of sentence rearrangements that the other algorithms generated.