The Metric System: Transforming Prose to Verse

Richard Frankel
frankel@cs.stanford.edu

Kara Levy
karalevy@cs.stanford.edu

Kevin Montag
kmontag@stanford.edu

> Invoke thy aid to my adventrous Song,
> That with no middle flight intends to soar
> Above th' Aonian Mount, while it pursues
> Things unattempted yet in Prose or Rhime.
> – John Milton, *Paradise Lost*, I, 13-16

ABSTRACT

The Metric System attempts the following task: Given English text, generate semantically equivalent English text which conforms to a specific poetic meter. The system employs a genetic algorithm to iteratively transform a prose source text into similar-meaning lines of metrical verse based on inputs and analyses from a language model, a semantic similarity model, a metrical model, a pronunciation dictionary, word sense disambiguation, parsing, and part-of-speech tagging.

1. INTRODUCTION

Prose-to-verse "translation" is an exercise with a long history of practice by human authors. While, given the current chasm between the abilities of humans and machines in natural language understanding and in artistry more generally, there is likely no direct practical application of our particular system or one like it, we believe this problem is interesting and worthwhile from a research standpoint for its combination of various popular natural language processing (NLP) tools in novel, unusual ways.

Our underlying aim is to explore quantitative approaches to processing and evaluating poetic verse.

1.1. Motivation and Related Work

A number of NLP projects in the past decades have experimented with poetry generation. These can be categorized by how they go about generating poetry, both externally in terms of what semantic constraints are placed on a poem—i.e. what the generated poem is a supposed to be about, topically—and internally in terms of constraints on and methodology for how the actual words are chosen. External semantic constraints in such projects have historically been abstract and open-ended. An extreme example is *the Poet* system, which generated poems using probabilistic text generation and hand-set scripts based on the abstract end user stimuli of visual line drawings [4]. Other projects have attempted rule-based poetry generation loosely based on phrases selected by end users, such as the *Hitch Haiku* system [8]. The database-driven *Muse* system [7] and the *Poetry Mix-Up* project [2] both generated poems by recombining complete units of preexisting (i.e. human-composed) poetic phrases.

In contrast, for our project we were interested in poetry generation with more rigid constraints: externally, by basing poems on complete and precise ideas in the form of prose passages (as opposed to vague themes or feelings); and internally, by conforming to complex notions of poetic meter, that is, iambic pentameter specifically. We are not aware of any other projects attempting such rigid direct "translation" of complete thoughts and sentences into poetry, and while the abovementioned *Hitch Haiku* and other haiku projects such as the *Keats* program do conform loosely to metrical constraints [5], we are also not aware of any projects doing similar quantitative metrical work in more traditional English-language verse such as iambic pentameter. Similarly, while, there has been substantial work in the fields of English literature and linguistics attempting to quantify and parameterize complete rules for metrical verse [3], efforts to encode metrical rules for computational use are still new and relatively unexplored.

2. METHODOLOGY

2.1. Genetic algorithm

The overarching framework for our project is a genetic algorithm which iteratively transforms a source text—i.e. prose—into *semantically similar* (faithful), *English-language* (fluent), and *metrical* (poetic) lines of verse.

First, a brief description of genetic algorithms (readers familiar with the concept can safely skip this paragraph). "Genetic algorithms" (GA) is the name given to a certain class of biologically-inspired randomized heuristic search strategies. In simplest terms, GA focuses on evolving a population of candidate solutions to a problem. First, an initial population is created, either of random candidates or of blank/default candidates. Then, while no candidate in the population satisfies some set of goal criteria (and/or a maximum number of generations has not been exceeded), the population is replaced by its own offspring. Offspring are generated by both mutation and "crossover" (mating) of members of the population, which are selected for these operations with likelihood proportional to their fitness. A candidate's fitness is a heuristic evaluation of its merit as a solution to the problem the GA is attempting to solve. As a trivial example, imagine that we want to find a string of 32 consecutive 1s. First, we would initialize a population of random-length 32-bit vectors. A candidate's fitness might be the number of 1s it contains. Mutating a candidate could mean flipping each of its bits with low probability. And mating two candidates could mean combining the first 16 bits of one candidate with the second 16 bits of the other.

We use a genetic algorithm to modify passages into other, semantically equivalent but more metrical passages. The reasons for this are threefold: first, using GA is relatively tractable approach, as the entire space of semantically equivalent English passages is huge; second, our problem is appropriate for GA—it is possible to change a passage in small, incremental ways, and such incremental changes can have a correspondingly incremental effect on the "fitness" of a passage as a semantically equivalent poem; third, there is a certain aesthetic appeal to using GA, which is relatively organic and random in nature, in order to attempt generation of something as ethereal as poetry. It is also worth noting that there is precedence for using GA in other poetry generation projects [5].

2.1.1. Fitness

We define a passage's fitness as a weighted sum of its score according to three models: An *N*-gram *language model*, a *semantic similarity model*, and a *metrical model*. The *N*-gram language model serves to penalize passages which are no longer correct English. The semantic similarity model is used to constrain the "semantic drift" of a passage from the original text. And the metrical model evaluates the metricality of a passage.

2.1.2. Mutation

We mutate passages in several ways: *synonym substitution*, *hypernym substitution*, *intensifier insertion*, and *structural sentence rewrites*.

Synonym substitution: Each token in the passage is, with low probability, replaced by a synonym. When performing synonym substitution, we first attempt to choose the correct sense of the word (see §2.6 below), pruning senses of different parts of speech. We use WordNet to provide a list of synonyms for the chosen sense. Because synonymy is an equivalence relation, we avoid substitutions of no metrical benefit; that is, we further prune the list of potential synonyms by removing any that do not introduce new potential stress patterns according to the pronunciation dictionary (see §2.5 below). We then randomly select one of the remaining synonyms, if any exist, and replace the token's word with that synonym.

Hypernym substitution: In object-oriented programming terminology, a hypernym can be thought of as a superclass. For example the hypernyms of (one sense of) *dog* are *domestic animal* and *canine*. We perform hypernym substitution much in the same way as synonym substitution. However, there are two differences. First, subsequent hypernym substitutions become decreasingly likely for a given token. This is because hypernymy involves small, but incremental, semantic drift (consider the sentence: "The entity acted upon the entity in a manner"); of course, because synonym is an equivalence relation, no such restriction is necessary there. Second, the stipulation that substitutions must introduce new potential stress patterns does not apply to hypernym substitutions. This is because a hypernym substitution introduces new potential synonym substitutions; therefore a hypernym substitution may indirectly introduce new stress patterns.

Intensifier insertion: Whenever an adjective or adverb does not undergo substitution, we, with low probability, insert an intensifier in front of it. By "intensifier" we really refer to any modifier of magnitude; words like "slightly" and "somewhat" are included, as well as those like "extremely". Similar to hypernym substitution, in this case in order to prevent really somewhat completely terribly silly sentences, the probability of intensifier insertion becomes increasingly less likely for a token each time it is performed on that token.

Structural sentence rewrites: With low probability, each sentence in a passage is rewritten according to some hand-written rules that maintain meaning and correctness in English. Given a parse tree for a sentence, a random traversal of the parse tree is performed. At each sub-tree, a rewrite is attempted with 50% probability. If at least one rewrite rule for the "heads" of that sub-tree exists, we randomly select one of the possible rewrite rules, perform it, and return, performing no further rewrites. (The heads of a sub-tree are the list of its types and the types of its children. For example, the heads of `[S [NP [NN Jim]] [VP [V hates] [NNS cookies]] [. .]]` are `(S, NP, VP, .)`.) If the rewrite was not attempted, or failed, then the same process is performed on each of the tree's sub-trees, in random order; if one succeeds, the function returns immediately. Obviously, the base case of this recurrence is that leaves of the parse tree are not rewritten.

2.1.3. Crossover

When a passage is first read, it is tokenized, and each token is assigned a chunk identifier (chunk ID). While many things about a chunk may change, the chunk is guaranteed to be the same part of speech, remain in the same sentence, and maintain roughly the same semantic value. When two passages mate, their offspring inherits the chunk ordering of one parent, but the actual contents of each chunk are selected randomly from each of two parents [Appendix A].

2.2. Language model

We use a standard N-gram *language model* to evaluate the English-language fluency of a passage, i.e. how likely it is that a given sentence is grammatically-correct English as opposed to gibberish. Ours is a trigram language model which employs backoff to bigram and unigram language models in the event of unknown N-grams, and uses absolute discounting to smooth the unigram model and handle unknown word tokens.

For the purposes of this system, we wanted to evaluate sentences (and passages) of all lengths equally, without penalizing long sentences over short ones. Therefore, the "score" of a sentence is defined as the N^{th} root of its probability according to the language model, where N is the number of tokens in the sentence. Similarly, the language model scores passages by returning the M^{th} root of the product of the scores of each of the M sentences in the passage.

2.3. Semantic similarity model

To keep the meaning of generated sentences reasonably close to the meaning of their sources, our fitness function incorporates a measure of *semantic similarity*. When a passage of text is initialized, we assign a unique chunk identifier to each word in the sentence, and preserve these identifiers across mutations (when intensifiers are inserted, they become part of the same chunk as word they are modifying). To compute the semantic similarity between two sentences, we then average the semantic similarities between matching chunks. To compute the semantic similarity between two chunks, we use a measure which utilizes the WordNet hierarchy as well as our predictions about the likelihoods for the various senses of words in the chunks. Our chunk similarity measure is most easily introduced in pseudocode [Appendix B].

In broader terms, we match each word in the original chunk with its best-matching word (based on a similarity measure) in the mutated chunk, and use the harmonic mean of these best-match similarities as the overall chunk similarity. The similarity between two words is computed by summing the WordNet similarities of all possible senses of both words, weighting each distance by the combined probabilities of both of those senses (these probabilities are obtained in our word sense disambiguation code, and depend on the language model and context of the words). Thus words whose senses are strongly predicted and are semantically similar will have high similarity, whereas words with ambiguous senses, or whose maximum-likelihood senses are semantically distant, will have low similarity.

Note that this gives the possibly surprising consequence that a word compared with itself might not have the maximum similarity measure. If a word has several senses which are pair-wise semantically distant, and if its sense in context is not well-determined, we may see a low similarity by our measure [Appendix C].

We might expect that this similarity be 1. In fact, our implementation explicitly assigns a similarity of 1 to identical words (this also allows us to handle words without an associated WordNet synset), but this issue appears for words which are very close synonyms in many of their senses. At the heart of this issue is the fact that we use a probability product to weight the WordNet similarity of two senses, rather than using a single probability and summing over a distribution; this means that the similarity score is penalized by words which are ambiguous in sense. We found empirically, however, that penalizing for ambiguity of word sense was a useful feature of the similarity model, though in some ways it creates a slight overlap with the language model.

NLTK implements a number of WordNet similarity measures, and we experimented with many different options—in general we found that it was appropriate to penalize more for steps downward in a hypernym hierarchy (e.g. *dog* to *cocker-spaniel*) than upward (e.g. *dog* to *animal*). The similarity measure which captures this intuition most closely is the Wu-Palmer similarity, described in the WordNet documentation as a score "denoting how similar two word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node)," and which ranges from 0 to 1. We also experimented with similarity measures based on information content [6], which involves weighting steps in the WordNet hierarchy by a factor related to the change in "overall content" of a synset and determined by training data. However, likely due to a shortage of training data, we found that this generally made the similarity measure behave less intuitively, and also required significant additional computation time for each similarity comparison.

Finally, it is worth noting that our chunk similarity measure is not symmetric—we compare all the words in one passage to any of the words in another, so adding words to the second passage will not affect the score. This was a deliberate choice; we wish to be able to add modifiers (such as intensifiers) to mutated passages without affecting their similarity with their source.

2.4. Metrical model

For the purposes of our genetic algorithm, we need to be able to examine a passage of verse and answer the question of how well that series of lines conforms to a target meter; beyond simple classification between prose from verse, we aim to quantify metrical quality. We do so by defining a *metrical distance* function between a line of source text and the meter we wish to emulate.

(A brief diversion for background in prosodic theory: Poetic meter is defined by a number of *foot* units, where each foot contains a set pattern of strong and weak metrical positions. Iambic pentameter, for example—the canonical meter of Shakespeare and Milton—is defined by five iambic feet, or iambs, where each iamb consists of a weak position followed by a strong position, yielding its cardiac rhythm of *da-DUM-da-DUM-da-DUM-da-DUM-da-DUM*. While phonological theory distinguishes between concepts of syllable weight, stress, strength, and pitch accent as distinct qualities of words and syllables each of which have their own formal rules for how they can map to strong and weak metrical positions, it happens to be the case in English that syllable stress corresponds to metrical position strength in the large majority of cases [3]. Moreover, the pedagogy of English poetics traditionally teaches "word accent" syllable stress as the primary determinant of metrical position stresses in a line of verse [1]—which is to say that human poets, even if their versification choices may also be informed by phonological features beyond their knowledge, themselves learn and understand metrical rules according to syllabic stress. We therefore use syllable stress as a proxy measure for metrical position strength with confidence that this is a consistently reliable heuristic, and will proceed from here on to use these concepts and terms interchangeably.)

Our metrical distance function is an adaptation of the classic Levenshtein dynamic programming algorithm for computing edit distance between two sequences, in our case sequences of syllabic stress. Using CMU Prodict notation (0 for no stress, 1 for primary stress, 2 for secondary stress—see §2.5 below), we measure the metrical distance between the stress sequence of a poetic line (e.g. '1001010101' for "Satan exalted sat, by merit rais'd") and an archetypical metrical pattern (e.g. '0101010101' for iambic pentameter).

Where the traditional Levenshtein algorithm measures edit distance by counting element insertions, deletions, and substitutions as equal-weight edits, we classify edits according to codified poetic categories of metrical deviation: *inversion* (swapping the stresses in a binary foot, for example substituting a trochee/'10' for an iamb/'01'); *extra-metrical syllables* (inserting additional unstressed syllables at the end of a foot); *resolution* (inserting additional unstressed syllables inside of a foot); and *catalexis* (omitting/deleting expected unstressed syllables). Each of these metrical deviations is further categorized according to where in the line it occurs: *initial* deviations occur at the beginning of a line; *cola* deviations occur on cola boundaries (a colon—the singular—is a grouping of feet within a metrical line—for example a line of iambic pentameter has two top-level cola, one 2-foot colon and one 3-foot colon, not necessarily in that order, constituting subgroups within the 5-foot line); and *internal* deviations occurring anywhere else in the line.

Each possible deviation is assigned an edit penalty. In addition, to cover edits not encapsulated by the above "allowed" deviations, we also assign substitution penalties for substituting different stresses for the expected metrical stresses, for deleting or inserting primary stresses, and for having metrical feet which cross clause boundaries (where clauses are loosely inferred as delineated by sentence boundaries and commas). Note that unlike traditional Levenshtein distance, metrical distance is not symmetric—we only consider editing from a target meter archetype to a source line, in other words switching the identities of these sequences affects the resulting distance.

In the absence of quantitative data about the real-world frequencies of these regular metrical irregularities, the penalties we assign to each deviation are at best crude approximations of their relative likelihoods in authentic verse; however, these penalties are based in the collected wisdom, albeit imprecise, of metrical analysts. For example, we know anecdotally that inversion is most likely at the beginning of a line and is more likely at the beginning of a cola than inside of one, and that inversion is in general more common than catalexis, and the relative magnitudes of our edit penalties attempt to capture such knowledge.

Since the Levenshtein algorithm on which we elaborate guarantees optimal (minimum possible) distance between two sequences, we calculate the optimal metrical distance from a target meter to a line of text by enumerating all possible stress sequences of the line and both possible cola boundaries for each enumeration and taking the minimum distance between all of those. We then perform pseudo-normalization on this distance, scaling it by a substantially bad stress sequence of the same length and clamping it to the (0,1] range, and use an inverted version of this value as the final metrical fitness of the source line. We define the overall fitness of a passage of text to be the harmonic mean of this metrical fitness function for each of its constituent full lines.

Note that given the novelty and limited scope of our project, we designed our metrical distance function and weight penalties specifically for iambic pentameter; however, with more work this same framework could be generalized to encompass any arbitrary meter.

The result is a moderately reasonable measure of the metricality of a passage. Run on an arbitrary collection sample fragments, we obtain fitness scores such as:

SOURCE	METRICAL FITNESS (IAMBIC PENTAMETER)
Shakespeare’s sonnets*	`0.852`
Milton, *Paradise Lost*, Book I*	`0.715`
Dickens, *A Christmas Carol*, Stave I	`0.610`
The New York Times, January 1, 2004	`0.574`

(* = actually written in iambic pentameter)

2.5. Prodict module

We use the CMU Pronouncing Dictionary (“Prodict”) to get the set of possible stress patterns for any given word, a free resource which has precedent being used for metrical haiku generation [5]. Our Prodict module augments the base CMU resource to decent include handling for unknown words not in the original dictionary.

The Prodict module reads the CMU Pronouncing Dictionary and creates two main dictionaries: one from words to sets of stress patterns, and one from stress patterns to sets of words [Appendix D].

This is a relatively trivial matter of text processing. However, handling unknown words requires a little more sophistication. We try the following three (3) methods for unknowns, returning the result of the first one which works [Appendix E for examples]:

(1) Hyphenated compounds: If the word contains one or more hyphens, we return the cross-product of the patterns of each component stem.

(2) *Consonant reduction:* We try three increasing levels of Soundex-based consonant reduction until a collision with a known word is found. We define a three-level word reduction function as follows:

Consonant reduction level 1:
First, reduce the following diphthongs, removing spurious H's:

```
* 0 <= TH, SH, CH
* G <= GH
* F <= PH
* W <= WH
```

Next, reduce consonants to equivalence classes as follows:

```
* B <= B, F, P, V
* C <= C, G, J, K, Q, S, X, Z
* D <= D, T
* L <= L
* M <= M, N
* R <= R
* W <= W
* H <= H
* 0 <= 0
```

Consonant reduction level 2:
Do level 1 reduction, and then collapse consecutive consonants into a single consonant.

Consonant reduction level 3:
At the first step, performed in level 1, put all consonants into the same equivalence class. In addition, perform the same collapse as in level 2.

The reasoning here is that actual consonants tend not to influence syllable stress—rather, it is vowels in each syllable that determine stress patterns. When we read in the CMU dictionary, we store a dictionary from reductions to original words for each of the three levels. Given an unknown, we keep performing reductions until we get a collision with a known word. We then return the patterns of whichever colliding word has the lowest Levenshtein distance to the unknown.

There are two reasons for using this method rather than just taking the Levenshtein-closest known word. First, it is more time-efficient. Second, it treats differences between consonants as less important than differences between vowels (and differences between similar consonants as less important than those between dissimilar consonants), which reflects how stress patterns are actually determined.

(3) Rule-base system: As a last resort, we use a rule-based system. First, we attempt to determine the number of syllables in the word. Then based on the number of syllables we:

1: Return `set(['0', '1'])`

2: If WordNet knows a sense of the word which is a verb, or if the word ends with `AxE` (for all consonants `x`), include `'01'`. If WordNet knows a sense of the word which is not a verb, or if WordNet doesn't know the word, add `'10'`.

3+: At this point, by the succinctness of Python, it is simpler for both the author and the reader to refer to the actual code. See the function `get_patterns_by_rule` in the Prodict module.

Refer to the Appendix E for overall results. We judge these results to be generally quite good. The only clear failure is on the compound word *psycholinguists*. We considered attempting to detect and handle compound words as we handle hyphenated words, but were concerned that false-positives and cases where the cross-product of the stems is incorrect would cause this to do more harm than good (e.g. *classmate*, which would return `'11'`).

2.6. Word sense disambiguation

We use a variant of the classic Lesk algorithm for word sense disambiguation, relying on WordNet for definitions and example sentences. Our version takes as input a word W, a part of speech P, a context C, and a language model LM; it returns a WordNet sense of W matching P, if one exists.

First, we define the IW (interesting words) of a sense S to be the pruned union of the words in the WordNet definition of S and its example sentences. The pruning in question removes W itself, removes common stop words, and removes words with a unigram probability above some threshold (according to LM).

When disambiguating a word in a context, we take the combination of all the interesting words of all the senses of each word in C and call it the greater context GC. One might balk at our flagrant use of all senses of each word in C; however, consider how unlikely it is that the incorrect senses of words in C will have many matches (see below) with an incorrect sense of W. The benefits of the increased context to Lesk far outweigh any rare occasions in which this leads to an incorrect disambiguation.

Then, for each sense S of W in WordNet matching P, we find the matches between GC and the interesting words of S. We use fuzzy matching, in which two words match if one contains the other or if their Levenshtein distance is less than some threshold (default of 3). We then return the sense maximizing the sum of the inverses of the unigram probabilities of its matches (with unknowns treated as one in a million) according to LM [Appendix F].

2.7. Parsing and chunking

As described earlier, one form of mutation in our genetic algorithm is the reordering of sentence sections (for example, “I walked around him” can become “Around him I walked”). Rules for these mutations are specified as a reordering of children for particular parse tree nodes, so it is necessary to obtain a reasonable semantic parse of the sentences we are processing. We explored a number of possibilities to this end, and ultimately settled on using the open-source, pre-trained Stanford Parser <http://nlp.stanford.edu/software/lex-parser.shtml>. This proved worthwhile despite technical obstacles (specifically, integration between the Stanford parser, which is implemented in Java, and our system, which is implemented in Python); however it is also informative to describe the options we pursued before arriving at the pre-trained parser.

Our initial approach was to build a weighted grammar from tagged training data, and use this to create a probabilistic parser. NLTK provides a number of probabilistic parsers which can be built from grammars, and a large collection of training data from which to build these grammars. To parse sentences, we built a grammar containing only non-terminal productions (with each production weighted by its frequency in the training data), and then added the appropriate terminal productions on a per-sentence basis based on part-of-speech tags assigned to tokens. It was sufficient to give the terminal productions a small fixed weight, since each such production was the only one in the grammar which yielded its terminal. We found, however, that the grammars we obtained were highly specific to the training data, even when a large number of productions were added; this was largely due to the use of rather obscure variants on parts of speech and parse groups in the training data. For example, our trained grammar contained the production `VP -> VB PP-MNR PP-CLR`, but not `VP -> VB PP PP`. As such, many simple sentences from outside the training set could not be parsed by the parsers built on our grammar. A solution to this issue would have been to hand-tune our grammar and the part of speech tags it interpreted for tokens, restricting them to a more limited set; however, this would have been time-consuming and tedious, and was unnecessary given the availability of other options.

We also experimented with chunking sentences—that is, parsing them “loosely” and shallowly by dividing them into chunks of a small number of semantic types. To do this, we used NLTK data to train a trigram tagger which labeled part-of-speech tags (in context) with a maximum-likelihood chunk type. Note that this means we performed chunking based purely on the part of speech tags we had determined for a sentence, ignoring the text of the sentence itself. The potential chunk types were verb phrases; noun phrases; prepositional phrases (each of which encompassed two labels, one representing the beginning of a chunk and one representing the trailing portion of a chunk, to make training data more effective); other words (a catch-all chunk type for extraneous words); and unknowns (words for which the tagger could not make an effective decision). This performed better than parsers generated from our grammar, and was significantly more robust, since by construction it could output a list of chunks for any sentence provided.

This model was, however, highly dependent on the success of our part of speech tagging, and (as we would expect for an *N*-gram model) errors in the part of speech tag on a token would propagate to its neighbors. For example, we received a number of sentences similar to:

```
(NP: The expanded tour) (O: still)
(VP: begins) (PP: with) (NP: the
rarefied reception rooms) (PP: on)
(NP: the eighth floor) (O: ,) (O:
but) (O: now) (VP: adds stops) (PP:
on) (NP: the first floor) (O: ,)
(O: including) (NP: the main lobby)
(O: ,) (O: dominated) (O: by) (NP:
the) (VP: flags) (PP: of) (NP: the
180 nations) (PP: with) (NP: which)
(U: the) (U: United) (U: States)
(U: has) (NP: diplomatic
relations.)
```

Here *flags* was tagged as a present-tense verb; note that this causes *flags* as well as its preceding *the* to be improperly chunked. This gives us the local chunk sequence `(NP VP PP NP)`, as in “Barbara looked past the house,” which we might think to reorganize as “past the house looked Barbara.” However, this reorganization applied to the sentence above would give the undesirable fragment “of the 180 nations flags the." Additionally, *United* was incorrectly tagged as a past-tense verb in our example, causing words in its vicinity to be tagged as belonging to chunks of unknown type and making them impossible to sensibly reorganize (though something like "with which the United States diplomatic relations has" would be semantically acceptable in a poetic context).

In a deeper sense, the interdependence of phrasal chunks within a sentence cannot be adequately captured with this shallow chunking method. For example, the sentence "The man with the X-ray glasses looked through walls" would be chunked as `(NP: The man) (PP: with) (NP: the X-ray glasses) (VP: looked) (PP: through) (NP: walls)`. Here again we see the local chunk sequence `(NP VP PP NP)`, corresponding to the fragment "the X-ray glasses looked through walls." The chunk reordering rule used above would replace this with "through walls looked the X-ray glasses" (a valid reordering if this fragment were an entire sentence), resulting in the nonsensical mutated sentence "The man with through walls looked the X-ray glasses." A full parse tree, on the other hand, would identify "The man with the X-ray glasses" as a self-contained noun phrase which should not be split during reordering.

Since chunking was insufficient for our purposes, and since training a parser from scratch proved too sensitive to training data, we ultimately opted to use the pre-trained Stanford Parser to parse sentences. We use the JPype module to create a usable bridge between our Python modules and the parser's Java classes—from an efficiency perspective, this approach allows us to run our code with the efficiency of CPython except when directly accessing the Java classes, as opposed to other pure-Java approaches (such as using Jython) which would result in significant performance hits.

2.8. Part-of-speech tagging

To generate part of speech tags for use in word sense disambiguation, we use a Brill tagger trained on a collection of NLTK corpora (specifically, the Treebank, CoNLL-2000, and Brown corpora). We used the built-in NLTK Brill implementation, specified a number of fairly standard initial rules for the trainer (inspired by a tutorial cited in our code—for example, words ending in *-ment* are tagged as nouns), and used a trigram tagger as our backoff when training. The Brill tagger gives a noticeable performance improvement over simple *N*-gram taggers.

Since we tested on sentences with words not in the lexicon of our tag training data, the Brill tagger is sometimes unable to assign a part of speech to words. To solve this, we implemented a tag model which extends our trigram language model, using part of speech tags themselves for inputs (similar to the chunking model discussed below) as well as outputs. We used this as a fallback to tag words which could not be tagged by our Brill tagger due to incompleteness of its lexicon.

3. RESULTS AND DISCUSSION

When we decided to tackle this problem, we knew that anything but the most modest results would be beyond the scope of the project, if not the very state of the art. We were unsurprised to find that we did not generate anything which would qualify as real poetry. However, we did experience some local success within passages. In particular, we frequently found valid synonym substitutions that improved the metricality of a passage.

We first thought of attempting to transform the works of Hemingway, thinking of his famously simple prose, but no corpus of his collected works was freely available. We then considered that Dickens might be a better subject, for his prolific output and his residence in the public domain, but were deterred by his fanciful names, often immoderate sentence length, and large amounts of dialogue. In the end, we decided to sample text from the Simple English Wikipedia.

In all results, our language model was trained on the New York Times corpus, selected primarily for its volume, broad range of subject matter, and relatively straightforward prose.

3.1. Qualitative examination of results

We explore several examples in this section. But before proceeding, we note one further caveat:

poetry is, by definition, non-quantifiable, and it is most certainly more than the sum of metricality, fluency, and semantic faithfulness. One consequence of this is that there is no obvious way to weigh these three models against one another, even if each is normalized. Therefore, we chose weights empirically, and with the mindset that our primary goal was to improve the (iambic pentameter) metricality of a passage, while using the language and semantic models to prune absurd or unfaithful transformations.

First, we examine an example from the Simple English Wikipedia "Algorithm" article.

Original passage

...
algorithm. When we write algorithms,
we like our algorithm to take the
least amount of time so that we can solve
our problem as quickly as possible.

Transformed passage

...
algorithmic rule. When we write algorithms,
we want our rule to take the very least
amount of time so that we can see our
job as quickly as possible.

There are a few things worth noticing here. First, the substitution of "algorithmic rule" for "algorithm" in the first line slightly improves that line's metricality. However, the next two lines are far more interesting. Note that each of these two lines is in perfect iambic pentameter. The substitution of "want" for "like" would appear to have been guided by the language model, and indeed it does sound like more natural English. Fortunately, "want" and "like" have the same possible stress patterns; had this substitution disrupted the metricality of the line, the passage's fitness would have been lowered and it may not have survived. In addition, the insertion of the "very" intensifier has no local effect on the metricality of the line; were it not there, the inferred line would include "amount," maintaining perfect iambic pentameter. However, were "amount" on the second line of the excerpt above, the third line would read "of time so that we can see our job as." If the insertion of "very" occurred before the substitution of "job" for "problem," then this would have forced the third line above to read "of time so that we can see our problem." Even with the generous interpretation of "our" as having a 10 stress pattern, this potential third line would still contain an extrametrical syllable at its end. So the real value of the "very" insertion is that it improved the metricality of the subsequent line. (Recall that the final line is ignored by the metrical model, as it is incomplete.) Again, it is worth noting that this insertion feels like very natural English, the credit for which belongs to our language model.

Next, we consider another passage from the Simple English Wikipedia, this time the "Evolution" article.

Original passage

...
this with evolution. All insects are
the descendants of a group of animals
who lived a long time ago. They still keep
...

Transformed passage

...
explain this with evolution. All invertebrates
are the descendants of a group of brutes
who lived a long time ago. They all the same
...

It is obvious that the changes to this passage have made the middle of the three excerpted lines more metrical. In fact, the middle line is nearly perfect iambic pentameter; it suffers only from an initial inversion, a common indulgence of the Bard himself (and one which is therefore penalized only lightly by our metrical model). Further, the word "animals," which has a '100' stress pattern, is less conducive to iambic pentameter than the single-syllable "brutes"; therefore the metrical flexibility of "brutes" likely outweighs the semantic cost when substituting it for "animal."

The reader may wonder, at this point, why the final transformed passages are so similar to their original counterparts. Indeed, we expected slightly more fantastical output ourselves. The answer to the question of why our output was often so conservative is that the combination of the language model and the semantic model generally kept some of the more outrageous transformations in check. However, this was not always the case. For the sake of completeness, we now present two slightly more bodacious passages. The first is an excerpt from a transformation of part of the Simple English Wikipedia "Mutations" article; for the sake of consistency, the second is the result of 200 unconstrained mutations of the Simple English Wikipedia "Genetics" article.

Original passage

...
the organism's ability to
live and reproduce . This is an important
part of the theory of evolution .
The amount of heritable variation
carried by a population can be
huge , and as a consequence natural populations
have the capacity to change and adapt
to conditions in their environment .

Transformed passage

...
noesis to live and procreate . This is
an important part of the theory of
phylogenesis . The amount of heritable
variation carried by an integer
can be Brobdingnagian , and as an
upshot natural populations have the
capacity to change and accommodate
to conditions in their billet .

While this may seem a liberal interpretation of the source, with the exception of one or two questionable substitutions (likely the result of word sense disambiguation failures), the original meaning of the passage is generally preserved. In order to appreciate fully the magnitude of the space of possible transformations, the reader should compare the following:

Original passage

Since human beings are not bred experimentally ,
human genetics must be studied by
other means . One way is by twin studies .
Identical twins are natural clones . They
carry the same genes , they may be used to
investigate how much heredity
contributes to individual people .
Studies with twins have been quite interesting .
If we make a list of characteristic
traits , we find that they vary in how much
they owe to heredity .

Passage mutated 200 times, without regard to fitness

Since considerably human parts take in
not modify slightly by experimentation ,
slightly human natural sciences
must constitute by barely other administrative units
gleam . Mean by duplicate totalities
one keeping . Indistinguishable Geminis
change integrity connatural court games .
They go away the same series , they
may give tongue to utilise to poll very
extraordinarily hardly at all
how utterly somewhat utterly adequately
much target contributes to rather deeply
single substances . Make be quite interesting
immunological disorders with
Geminis . If we bring up a enactment
of characteristic removals , we
appear that they vary in how much they
to retention owe .

It is worth noting that while our mutations do aim to preserve meaning, they prioritize increasing the space of possible transformations. The fact that even our more radical output remains comprehensible is a testament to the utility of our language and semantic models.

On a lighter note, we did occasionally generate snippets of notable aesthetic value. For the pleasure of the reader, we include two below:

Original (from the Simple English Wikipedia "Fuzzy Logic" article)

The human brain switches
to fuzzy logic that says get me closer ,
get me closer , and so on .

Transformed

The human encephalon substitutions
to fuzzy logic that enunciate
turn me immensely closer , get me unaired ,
and so on .

-

Original (from the Simple English Wikipedia article on murder)

is not a crime to defend yourself , even
if it kills the attacker .

Transformed

is not an offence to keep yourself , even
if it efface the wrongdoer .

3.2. Data

In spite of the non-quantifiable nature of poetry, it is worthwhile to examine how we fared with respect to our three models. That is, given our weights, how the maximum scores of the population changed over a 200-generation run. (In these data, we report language model “scores” rather than probabilities; see §2.2 for an explanation. Furthermore, we multiply these scores by 75 in order that they might be displayed on the same order of magnitude as those of the metrical and semantic models.)

In Appendix G you will find graphs for the scores according to each model of three different Simple English Wikipedia articles. The results of each are shown with two different weighting schemes, one emphasizing the metrical model score (“MM emphasis”: 10.0 metrical, 100.0 language, 0.5 semantic) and one emphasizing the language model score (“LM emphasis”: 5.0 metrical, 300.0 language, 0.5 semantic). The scores are displayed un-weighted. All GA simulations were run for 200 generations, with a population of size 25; the scores of the best candidate in the population at each generation are what the graphs display.

These graphs make clear a few things. First, the obvious: As a passage’s semantic similarity to its original can only decrease as it changes, improvements in language and metrical score nearly always come at the cost of semantic score (though there occasional tradeoffs between the two). We can also see that the language score improves more steadily, and for longer. Indeed, in many cases it would appear that the language score would have continued to improve had the simulation lasted longer.

There is another phenomenon which appears in the MM-emphasis data for both the “Evolution” and “Fuzzy Logic” articles, which is that once metrical scores cease to improve (at around 0.84, which is a good score—see §2.4), the language score starts increasing, presumably by making fluency-increasing but metrically-equivalent substitutions. That metricality-preserving substitutions are consistently available provides, perhaps, insight into one small part of the structure of natural language.

4. FUTURE IMPROVEMENTS

Our efforts notwithstanding, there is substantial room for improvement in prose-to-verse translation. We suggest a few ways in which our system could be improved, given sufficient time and resources:

4.1. Language model training data

The New York Times corpus is both voluminous and convenient. However, it is entirely prose, and there are certain grammatical structures never seen in prose but perfectly acceptable in poetry. While our parsing/rewrite mutation system provided the desired increase in the magnitude of our passage transformation space, many rewrites were effectively rejected by the language model. We are confident that the results of our system would be improved by training both the language model and the part-of-speech tagger on poetry rather than prose. However, we are not aware of any poetic corpus of sufficient size which is annotated with parts of speech.

4.2. Metrical model

This metrical model would also benefit from further sophistication. Most generally, metrical word stress is context-sensitive in ways not currently captured by our metrical model.

Because analysis is performed only locally we assign too-gentle penalties on lines with multiple metrical deviations since, for example, the third deviation in a line is penalized just the same as if it were the first deviation.

Additionally, polysyllabic words ought to receive stress priority, while whether or not monosyllabic words are stressed or unstressed depends both on their grammatical function and their “rhetorical accent” [1]. Incorporating grammatical function as a feature would require parse tree integration; incorporating rhetorical function would require a degree of natural language understanding beyond anything our project has yet attempted.

Finally, we currently assign no penalty at all to *enjambment*, the breaking of clauses across lines, since this is a fairly common occurrence in actual (human) poetry; however, in actual poetry enjambment is used selectively to enhance and highlight semantic meaning, whereas our system's ignorance of enjambment results in our generated poetry often exhibiting meaningless, jarring enjambment.

4.3. Prodict module

Because the Prodict module currently determines the possible stress patterns for a given word form based exclusively on its orthography, it misses opportunities to use context for disambiguating between different possible stress patterns. For example, the noun *present* has an '10' stress pattern and the verb *present* has an '01' stress pattern, meaning that part-of-speech knowledge would allow us to pick a single correct pattern, but our current Prodict module naively returns `set(['10', '01'])` when queried. Given a dictionary resource which coupled pronunciation information along with different word senses for the same word form (which the CMU dictionary does not currently do), we would be able to improve the Prodict module by integrating it with part-of-speech tagging and word-sense disambiguation.

4.4. Semantic network limitations

We very much appreciate the gargantuan effort put into building and maintaining WordNet. However, as anyone who has used it knows, there is a great deal of room for improvement. One need look no further than WordNet's synonyms for dog (the animal)—"domestic dog" and "Canis familiaris", but neither "pooch" nor "doggie" nor "mutt" nor "hound"—to see the problem. Some groups, such as Memidex <http://www.memidex.com>, have made substantial improvements to WordNet proper, but none of these are publicly available. As synonym substitutions are one of our system's safest and most common mutations (based on anecdotal experience as well as the phenomenon described in §3.2), having access to data such as Memidex's would be highly beneficial to our system's performance.

4.5. Word sense disambiguation

Word sense disambiguation is currently an unsolved problem in the field of natural language processing, and, given the subtleties of natural language, some might even argue that it is an inherently unsolvable (or, at least, AI-complete) problem. However, incremental improvement is a realistic goal, and as better WSD systems are devised, incorporating them into our system would surely improve its results.

5. CLOSING REMARKS

Prose-to-verse translation is a relatively unusual NLP task, with potentially unsolvable evaluation challenges, significant influence from the fine arts, and little more to offer than whimsy. However, this makes it no less challenging, and a complete approach involves several nontrivial NLP subproblems. This has made it an entertaining and illustrative project.

6. ACKNOWLEDGEMENTS

We would like to pay special thanks to Stephanie Shih (Ph.D. candidate in Linguistics, Stanford University), Ryan Heuser (Ph.D. candidate in English, Stanford University), Arto Anttila (Associate Professor in Linguistics, Stanford University), and Brooke Lewis (Lecturer in Language, New York University) for their consultations about this project.

6.1. Collaborator contributions

High-level design, planning, testing, problem-solving, and analysis were performed jointly by all collaborators. Rich was responsible for the implementation and management of the overarching genetic algorithm framework, the Prodict module, and word sense disambiguation; likewise Kara for the metrical model and background research into related work; and likewise Kevin for semantic distance, part of speech tagging, parsing, and chunking.

7. REFERENCES

[1] M. H. Abrams. *A Glossary of Literary Terms*, pages 194-199. Ninth edition. Wadsworth, 2009.

[2] Owen Noel Newton Fernando, Adrian David Cheok, Nimesha Ranasinghe, Kening Zhu, Chamari Priyange Edirisinghe, Yan Yan Cao. Poetry mix-up: a poetry generating system for cultural communication. In *Advances in Computer Entertainment Technology 2009*, pages 396-399.

[3] Kristin Hanson and Paul Kiparsky. A Parametric Theory of Poetic Meter. In *Language*, Vol. 72, No. 2, pages 287-335, 1996.

[4] E. Mendelowitz. Drafting poems: inverted potentialities. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 1047-1048. ACM New York, NY, USA, 2006.

[5] Matt Thompson. Computational / Poetry thesis blog post / Stanza 三: Haiku. Blog. http://mattt.me/2009/03/computational-poetry-thesis-blog-post-stanza-3-haiku/. Posted March 18, 2009.

[6] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proc. of 14th International Joint Conference on Artificial Intelligence*, pages 448-453, Montreal, Canada, 1995.

[7] N. Tosa. Interactive poem. In *SIGGRAPH '98: ACM SIGGRAPH 98 Conference abstracts and applications*, page 300, New York, NY, USA, 1998. ACM.

[8] N. Tosa. Hitch-haiku. In *SIGGRAPH '07: ACM SIGGRAPH 2007 art gallery*, page 250, New York, NY, USA, 2007. ACM.

APPENDIX

A. Genetic algorithm: Crossover

Two hypothetical passages (shown in `[(word, chunk ID), ...]` form) and a potential offspring:

Passage 2:

```
[(With, 4), (a, 5), (grin, 6), (Dave, 1), (hailed, 2), (me, 3), (., 7), (I,
8), (smiled, 9), (back, 10), (., 11)]
```

Passage 1:

```
[(Dave, 1), (greeted, 2), (me, 3), (with, 4), (a, 5), (big, 6), (smile, 6),
(., 7), (I, 8), (beamed, 9), (back, 10), (., 11)]
```

Offspring:

```
[(With, 4), (a, 5), (big, 6), (smile, 6), (Dave, 1), (hailed, 2), (me, 3),
(., 7), (I, 8), (beamed, 9), (back, 10), (., 11)]
```

B. Semantic similarity model

Pseudocode:

```
semantic_similarities = empty list
for original_word in original_chunk:
 best_similarity = 0
 for mutated_word in mutated_chunk:
 if mutated_word == original_word:
 # For words with no WordNet senses, e.g. 'this'
 best_similarity = 1
 else:
 for original_sense in original_word.senses:
 for mutated_sense in mutated_word.senses:
  current_similarity =
  wordnet_similarity(original_sense, mutated_sense) *
  original_sense.probability *
  mutated_sense.probability
  if current_similarity > best_similarity
  best_similarity = current_similarity
 semantic_similarities.append(best_similarity)

return harmonic_mean(semantic_similarities)
```

C. Semantic similarity model

As a toy example, suppose a word has two senses, s_1 and s_2, with $WordNetSimilarity(s_1, s_1) = WordNetSimilarity(s_2, s_2) = 1$, $WordNetSimilarity(s_1, s_2) = 0$, and $P(s_1) = P(s_2) = 0.5$. Then our similarity measure gives: $similarity(s_1, s_2) = \big((1 \cdot 0.5 \cdot 0.5) + (0 \cdot 0.5 \cdot 0.5) + (0 \cdot 0.5 \cdot 0.5) + (1 \cdot 0.5 \cdot 0.5)\big) = 0.5$.

D. Prodict module

```
>>> prodict.words['INTRIGUE']
set(['10', '01'])
>>> prodict.words['INDIFFERENT']
set(['010', '0100'])
>>> prodict.words['HAPPINESS']
set(['100'])
>>> prodict.patterns['2001']
set(['MADEMOISELLE', 'BIODIVERSE', 'CABRIOLET', 'OVERSUPPLY', 'LEGERDEMAIN',
'ENTREPRENEUR', 'HULLABALOO', 'LAVIOLETTE', 'SUPERIMPOSED', 'OVEREXCITES',
"AEROPERU'S", 'MISUNDERSTAND', 'MISUNDERSTOOD', 'INTERRELATE',
'RESTAURATEURS', 'INTERCONNECT', 'OVERPROTECT', 'AUTOMOBILE', 'SOCIETE',
'OVEREXPOSE', 'RESTAURATEUR', 'OVERSUPPLIED', 'OVEREXTENDS', 'ENTREPRENEURS',
'AQUAMARINE', 'NASIONAL', 'MUJAHEDEEN', 'NOVOSIBIRSK', 'TELEPHONIQUES',
'OVEREXTEND', "AZERBAIJAN'S", 'UROKINASE', 'AZERBAIJAN', 'IDIOPATH',
"ENTREPRENEUR'S", 'AEROPERU', 'COUNTERATTACKED', 'SENEGALESE',
'MISUNDERSTANDS', 'MUJAHIDEEN', 'CATAMARAN', 'GUADALCANAL', 'KALAMAZOO',
'OVEREXPOSED', 'OVEREXCITE', 'NEVERTHELESS', 'OVERSUBSCRIBED',
'OVERSUBSCRIBE', 'AUTOMOBILES', 'BALAKUMAR', 'GOBBLEDYGOOK', 'RECITATIVES',
'SUPERIMPOSE', 'BELLEFEUILLE'])
```

E. Prodict module

i. Hyphenated compounds:

```
>>> prodict.words['SPINE']
set(['1'])
>>> prodict.words['TINGLING']
set(['10', '100'])
>>> prodict.get_pattern('SPINE-TINGLING')
set(['110', '1100'])

>>> prodict.words['FIANCE']
set(['012', '221'])
>>> prodict.words['TO']
set(['1', '0'])
>>> prodict.words['BE']
set(['1', '0'])
>>> prodict.get_pattern('FIANCE-TO-BE')
set(['22100', '22101', '22111', '22110', '01211', '01210', '01200', '01201'])
```

ii. Consonant reduction

Level 1:

```
>>> prodict.reduce_consonants('NEVERTHELESS', 1)
'MEBER0ELECC'
```

Level 2:

```
>>> prodict.reduce_consonants('NEVERTHELESS', 2)
'MEBER0ELEC'
```

Level 3:

```
>>> prodict.reduce_consonants('NEVERTHELESS', 3)
'CECECECEC'
```

Here is how this system would handle the unknown *Spitkovsky*, as performed manually in the interpreter:

```
>>> r1 = prodict.reduce_consonants('SPITKOVSKY', 1)
>>> r1 in prodict.reduced[1]
False
>>> r2 = prodict.reduce_consonants('SPITKOVSKY', 2)
>>> r2 in prodict.reduced[2]
False
>>> r3 = prodict.reduce_consonants('SPITKOVSKY', 3)
>>> r3 in prodict.reduced[3]
True
>>> guesses = prodict.reduced[3][r3]
>>> guesses
set(['TRILOGY', 'TIMOTHY', 'BIMONTHLY', 'PILLORY', 'HICKORY', 'SIROKY',
'BIGOTRY', 'SIMONY', 'GRIGORY', 'LIPOVSKY', 'WIKOWSKY', 'VISCLOSKY',
'KILLORY', 'GILLOGLY', 'WILLOWY', 'VICTORY', 'PITOFSKY', 'MITOFSKY',
'SIKORSKY', 'HISTORY'])
>>> best_guess = min([(lev('SPITKOVSKY', guess), guess) for guess in
guesses])[1]
>>> best_guess
'PITOFSKY'
>>> prodict.words[best_guess]
set(['010'])
```

iii. Unknown handling

Here are the results of some unknown “words” (many of which are realistic, but not real, words):

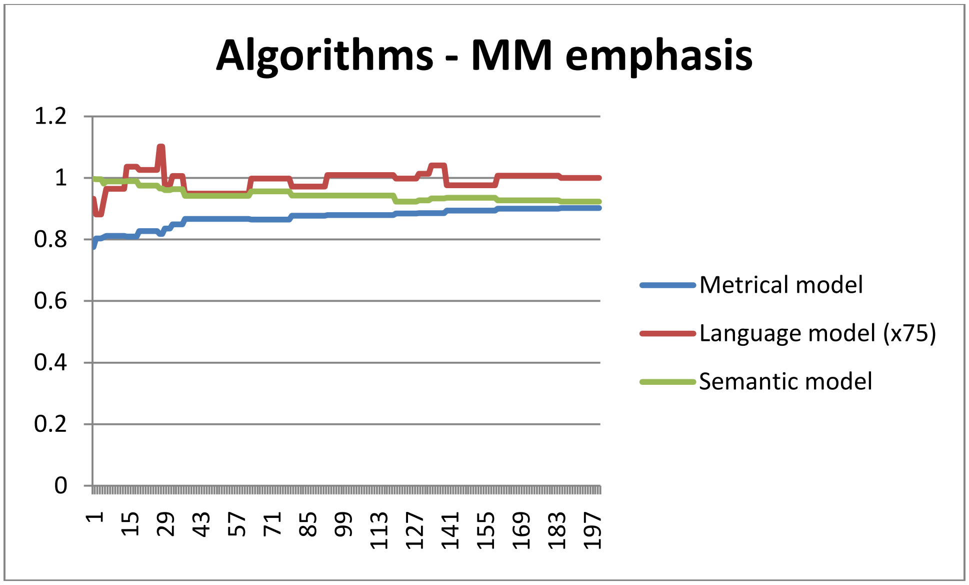
```
TEST WORD: PENNY-ARCADE
 CONSONANT-REDUCTION-BASED GUESS: None
 RULE-BASED GUESS: ['100']
 OVERALL GUESS: set(['1001'])

TEST WORD: PENNY-ARCADE-DINOSAUR-TELEPHONE
 CONSONANT-REDUCTION-BASED GUESS: None
 RULE-BASED GUESS: ['20000000100', '10000002100', '10000000100',
'20000002100']
 OVERALL GUESS: set(['1001102102'])

TEST WORD: SPITKOVSKY
 CONSONANT-REDUCTION-BASED GUESS: set(['010'])
 RULE-BASED GUESS: ['100']
 OVERALL GUESS: set(['010'])
```

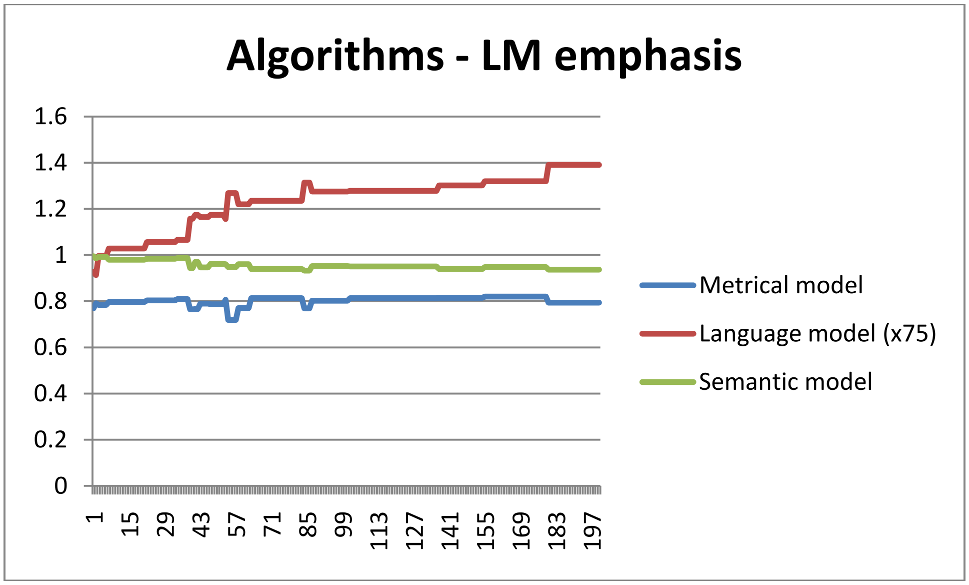
```
TEST WORD: ENTIMATE
 CONSONANT-REDUCTION-BASED GUESS: set(['102', '100'])
 RULE-BASED GUESS: ['100']
 OVERALL GUESS: set(['102', '100'])

TEST WORD: DALAGA
 CONSONANT-REDUCTION-BASED GUESS: set(['010'])
 RULE-BASED GUESS: ['100']
 OVERALL GUESS: set(['010'])

TEST WORD: RINDING
 CONSONANT-REDUCTION-BASED GUESS: set(['10'])
 RULE-BASED GUESS: ['10']
 OVERALL GUESS: set(['10'])

TEST WORD: MIRTHY
 CONSONANT-REDUCTION-BASED GUESS: set(['10'])
 RULE-BASED GUESS: ['10']
 OVERALL GUESS: set(['10'])

TEST WORD: EVOPSYCHOLOGY
 CONSONANT-REDUCTION-BASED GUESS: None
 RULE-BASED GUESS: ['200100', '100100', '102100', '202100']
 OVERALL GUESS: set(['200100', '100100', '102100', '202100'])

TEST WORD: PSYCHOLINGUISTICS
 CONSONANT-REDUCTION-BASED GUESS: None
 RULE-BASED GUESS: ['22010', '12010']
 OVERALL GUESS: set(['22010', '12010'])

TEST WORD: PSYCHOLINGUISTS
 CONSONANT-REDUCTION-BASED GUESS: None
 RULE-BASED GUESS: ['0100']
 OVERALL GUESS: set(['0100'])

TEST WORD: WITTICISM
 CONSONANT-REDUCTION-BASED GUESS: set(['1020'])
 RULE-BASED GUESS: ['0100']
 OVERALL GUESS: set(['1020'])

TEST WORD: HYPERDETERMINISTIC
 CONSONANT-REDUCTION-BASED GUESS: None
 RULE-BASED GUESS: ['2002010', '1022010', '2022010', '1002010']
 OVERALL GUESS: set(['2002010', '1022010', '2022010', '1002010'])

TEST WORD: ANTEDILUVIAN
 CONSONANT-REDUCTION-BASED GUESS: None
 RULE-BASED GUESS: ['200100', '100100', '102100', '202100']
 OVERALL GUESS: set(['200100', '100100', '102100', '202100'])

TEST WORD: TATTER'D
 CONSONANT-REDUCTION-BASED GUESS: set(['10'])
 RULE-BASED GUESS: ['10']
 OVERALL GUESS: set(['10'])
```

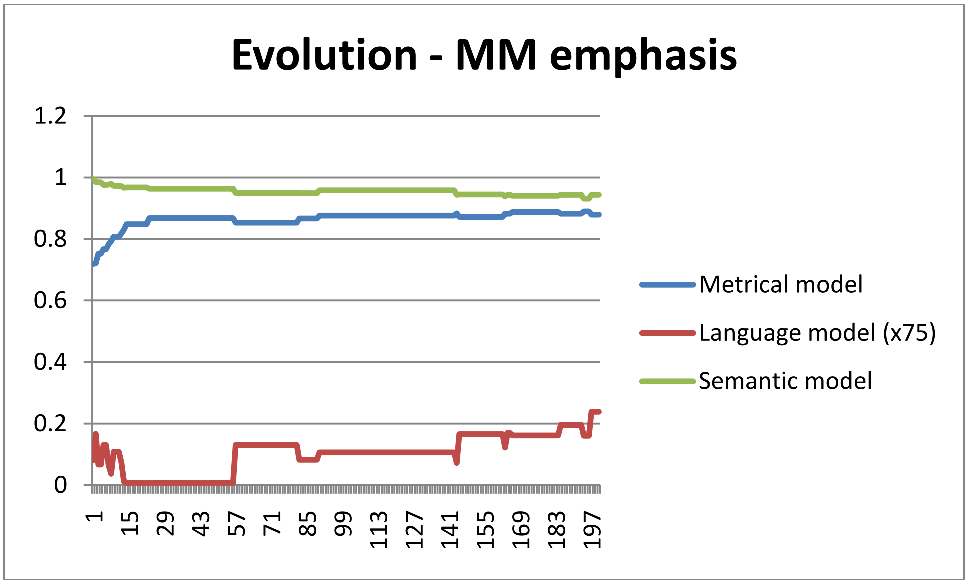
F. Word sense disambiguation

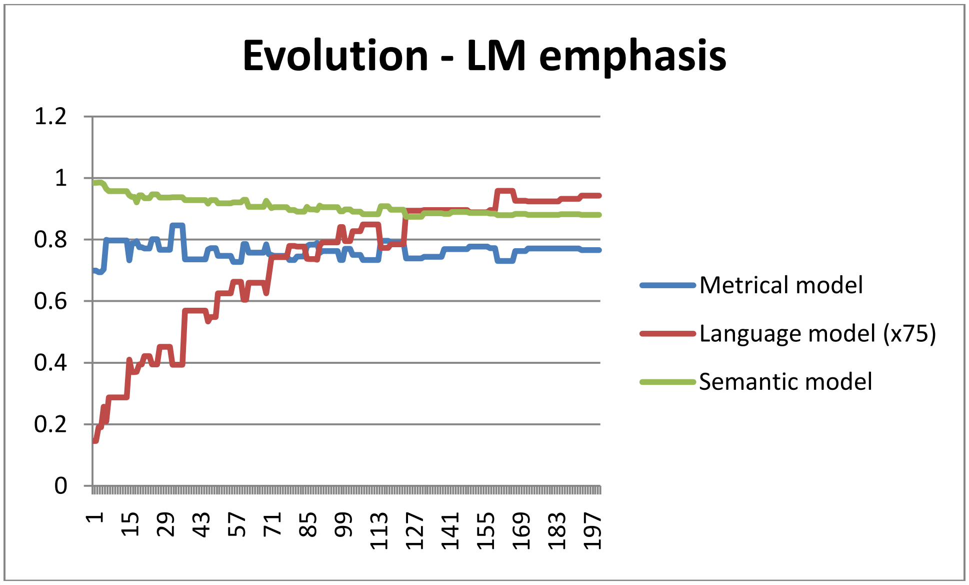
```
disambiguating dog in sentence ['She', 'is', 'an', 'unattractive', 'woman',
',', 'what', 'a', 'dog', '!']
a dull unattractive unpleasant girl or woman

disambiguating dog in sentence ['The', 'dog', 'is', 'a', 'canine', 'that',
'often', 'barks', '.']
a member of the genus Canis (probably descended from the common wolf) that
has been domesticated by man since prehistoric times; occurs in many breeds

disambiguating bass in sentence ['The', 'bass', 'line', 'of', 'the', 'song',
'is', 'too', 'weak', '.']
the lowest part of the musical range

disambiguating bass in sentence ['I', 'went', 'fishing', 'for', 'some',
'sea', 'bass', '.']
the lean flesh of a saltwater fish of the family Serranidae

disambiguating right in sentence ['The', 'government', 'should', 'grant',
'certain', 'civil', 'rights', '.']
an abstract idea of that which is due to a person or governmental body by law
or tradition or nature; it is something that nobody can take away"

disambiguating right in sentence ['If', 'you', "don't", 'know', 'which',
'direction', 'to', 'go', ',', 'just', 'turn', 'right', '.']
location near or direction toward the right side; i.e. the side to the south
when a person or object faces east
```

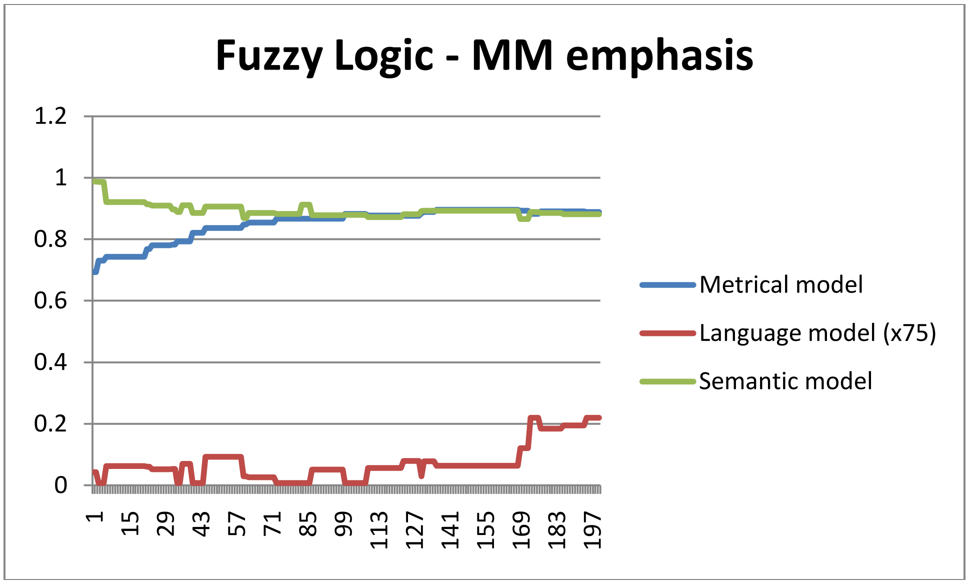
G. Data

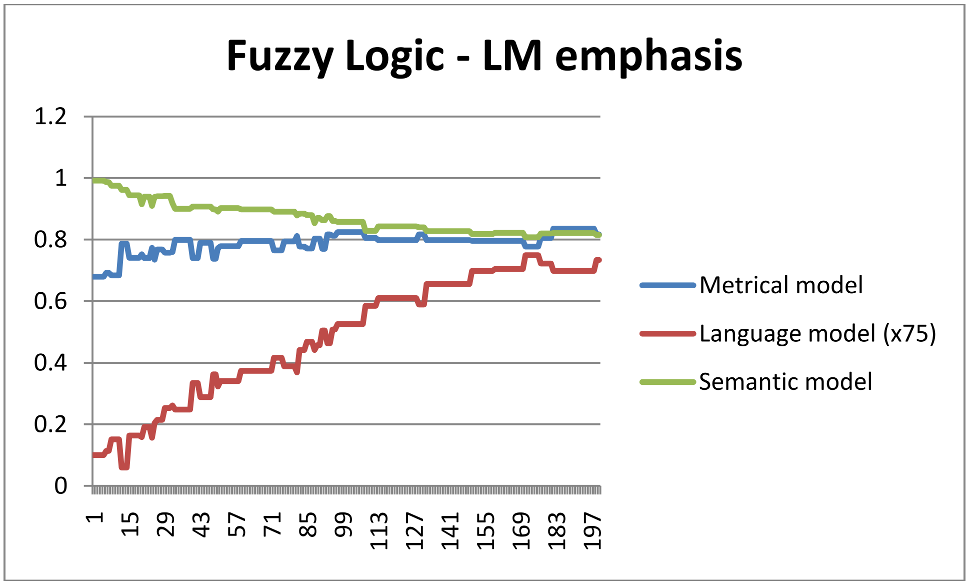

Algorithms - MM emphasis
1.2
1
0.8
0.6
0.4
0.2
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
Metrical model
Language model (x75)
Semantic model


Algorithms - LM emphasis
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
Metrical model
Language model (x75)
Semantic model


Evolution - MM emphasis
1.2
1
0.8
0.6
0.4
0.2
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
Metrical model
Language model (x75)
Semantic model


Evolution - LM emphasis
1.2
1
0.8
0.6
0.4
0.2
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
Metrical model
Language model (x75)
Semantic model


Fuzzy Logic - MM emphasis
1.2
1
0.8
0.6
0.4
0.2
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
Metrical model
Language model (x75)
Semantic model


Fuzzy Logic - LM emphasis
1.2
1
0.8
0.6
0.4
0.2
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
Metrical model
Language model (x75)
Semantic model