A chart generation system for topical metrical poetry

Berty Chrismartin Lumban Tobing and Ruli Manurung
Faculty of Computer Science
Universitas Indonesia
Depok 16424, West Java, Indonesia
berty.chrismartin@ui.ac.id,maruli@cs.ui.ac.id

Abstract

Several poetry generation systems that are in some way inspired or motivated by existing articles such as newspaper stories have recently appeared. However, most if not all of them employ template-based generation, which limits both the expressiveness of the system and the ability to faithfully convey the message of the source article. In this paper we present our work on a poetry generation system that uses a dependency parser to extract the predicate argument structure of the input article, and tries to maintain this structure through deep syntactic text generation whilst complying with a given target form. The combinatorial nature of this task presents huge challenges, and we describe several improvements that have been applied in an attempt to produce poetry in a tractable fashion.

Introduction

Poetry generators are systems that are capable of automatically generating poetry given certain restrictions and contexts. Gervás (2002) presents an overall evaluation of various poetry generators. Various generation approaches are employed, e.g. evolutionary algorithms (Manurung, Ritchie, and Thompson 2012), case-based reasoning (Diaz-Agudo, Gervás, and González-Calero 2002), template-based generation (Colton, Goodwin, and Veale 2012), (Rashel and Manurung 2014), and constraint programming (Toivanen, Järvisalo, and Toivonen 2013).

In this paper, the task we are aiming to solve can be referred to as *meaningful poetry generation*, where the goal is to generate a text that exhibits poetic aspects such as rhyme, metre, alliteration, and other phonetic or orthographic patterns, but also broadly tries to convey a given meaning representation. This last requirement is what distinguishes this task from other forms of poetry generation, which primarily focus on generating texts that take the form of a poem.

The way in which an input meaning representation is provided, and the manner in which a poetry generation system attempts to preserve the fidelity of the input meaning representation, varies. Most systems can be said to be "loosely inspired" by their input meaning representations, as they use words and phrases from the input as fillers for textual templates. Although the resulting poems may include words derived from the input, they do not necessarily take into account aspects such as predicate-argument structure and head-modifier relationships that are crucial towards semantic interpretation. For example, it is possible that a poetry generator that extracts keywords from an article about the Gulf War writes a poem about Iraq invading the USA, and not the other way around. Some notable exceptions are PoeTryMe (Gonçalo Oliveira 2012) and McGonagall (Manurung, Ritchie, and Thompson 2012). PoeTryMe selects content in the form of a set of words and relations between them that are obtained from a semantic graph, and conveys this content using templates that are known to express such specific relations. McGonagall employs a fitness function that measures the semantic similarity between a candidate poem and a given target semantics in such a way that structural similarity is significantly preferred. One other interesting related work is Gervás (2015), which explores various modifications and extensions to an existing poetry generation system, WASP, to consider much tighter constraints on the content of generated poems.

This paper describes a system that uses a meaning representation that explicitly captures predicate-argument structure and tries to maintain this structure through deep syntactic text generation whilst complying with a given target form.

We first discuss chart generation, the basic mechanism the system employs to produce text, before discussing how we extract meaning representations from input news articles. Some results from experiments using this initial system are shown. We then present a complexity analysis of the algorithm and suggest four different improvements to make the system generate meaningful metrical poems in a much more tractable manner. Finally, some results are shown from experiments using the revised system.

Chart Generation

Moreso than an author of prose, an author of poetry may have to perform a lot of rewording, paraphrasing, and various other alterations to the text, in order that the end result can satisfy the various poetic constraints such as rhyme and metre. Moreover, in literary texts, creative language use often results in more exibility of lexical choice, word-order, and grammaticality, hence an even larger search space for the paraphrasing.

One efficient method for constructing all valid para-

phrases of a natural language utterance is chart generation (Kay 1996). Given an input meaning representation, a set of grammar rules, and a lexicon, it systematically generates all syntactically well-formed texts that convey the input meaning. It employs a dynamic programming technique to overcome the inefficiency caused by backtracking due to the pervasive non-determinism in natural language grammar rules.

A data structure known as the chart stores all complete constituents once they are generated, so regardless of the number of paraphrases they may appear in, they will only be constructed once. The chart also stores incomplete constituents, which are predictions of larger constituents yet to be generated. A chart contains entries that are labelled with 'dotted rules' which describe both complete constituents, called inactive edges, and incomplete constituents, called active edges. An active and an inactive edge can combine to yield a new edge that represents a larger constituent.

An example of an inactive edge is $np \rightarrow det\ noun\ \bullet$, which represents a noun phrase (np) constituent that consists of a determiner followed by a noun, whereas an example of an active edge is $np \rightarrow det \bullet noun$ which represents a partially constructed noun phrase which is still lacking a noun. Note the position of the dot ($\bullet$) that delineates the portion of the constituent that has been constructed from that which is still lacking.

For chart generation, it is not enough for the dotted rules to simply state syntactic constituency. They must also state the semantics of each constituent, and how their arguments must unify when being combined. When two edges combine, their semantics must also be unioned to obtain the semantics of the new edge. Moreover, some semantic subsumption checking must be performed to prevent false sentences from being generated. For example, given the input semantics $loves(john, mary)$, an edge with the semantics $loves(john, X)$, where the variable X indicates an unbound argument, can be added to the chart, because its semantics still subsume the input. However, according to the input grammar, a chart generator may also construct an edge with the semantics $loves(X, john)$, whose semantics does not subsume the input. Therefore, it must be rejected.

The algorithm can be informally described as follows:

1. Add entries for all words whose semantics subsume the target semantics to the chart.
2. *Bottom-up prediction*: for each inactive edge in the chart, add new active edges to the chart for each grammar rule that have it as the first constituent on the right hand side.
3. *Scanning*: for each active edge in the chart, look for inactive edges whose category matches that of the first constituent needed, and add a new edge that combines the two.
4. *Completion*: for each inactive edge in the chart, look for an active edge that is looking for a constituent with a matching category.
5. The above processes are repeatedly applied to all new entries to the chart until no more new entries can be added.

Let us consider a simple example. Suppose the following target semantics are to be generated: {*dog(d), definite(d), see(s), cat(c), definite(c), arg1(s,d), arg2(s,c)*}.

Assume the grammar consists of the following three rules:
$s(x) \rightarrow np(y)vp(x, y)$
$np(x) \rightarrow det(x)noun(x)$
$vp(x, y) \rightarrow verb(x, y, z)np(z)$

and the lexicon consists of the following four entries:

Word	Category	Semantics
cat	noun(x)	x:{cat(x)}
saw	verb(x,y,z)	x:{see(x),arg1(x,y),arg2(x,z)}
dog	noun(x)	x:{dog(x)}
the	det(x)	x:{definite(x)}

Following the algorithm described above, edges are entered to form the chart seen in Table 1. The process is as follows:

- Initially, edges 1,2,4,6, and 7 enter the chart. They represent the lexical items that convey a portion of the target semantics.
- Edges 3,5, and 8 enter the chart as a result of the *prediction* operation. Based on the grammar, the algorithm hypothesises the existence of larger constituents.
- Edges 9 and 11 enter the chart as a result of combining the inactive and active edges 1+3 and 6+8 respectively.
- Edges 10 and 12 enter the chart as a result of the *prediction* operation on edges 9 and 11. Note that edge 12, although cannot form any part of a sentence that conveys the input semantics, still enters the chart, but will not combine with any other edge due to the semantic subsumption checking.
- Edge 13 and subsequently edge 14 enter the chart as a result of combining edges 5+11 and 10+13 respectively.

Metre compatibility

Manurung (1999) first introduced an extension to chart generation to take into account rhythmic constraints of poetry.

In most forms of poetry, metre is the arrangement of words such that rhythmic patterns emerge from their lexical stress, which is the relative prominence of stress received by syllables in a word. To simplify matters, we will assume that syllables may receive one of either two types of lexical stress: weak stress or strong stress. Thus, the rhythm of natural language strings can be represented as lists, which we call stress patterns, denoting the type of stress received by each syllable in an utterance, which can be either weak (denoted as '`w`') or strong (denoted as '`s`'). For example, the list `[w,s,w,s,w,s,w,s,w,s]` would be a stress pattern that represents a line of iambic pentameter.

These stress patterns can be used as the representation for specifying the metrical constraints that are provided as input for the chart generator. The starting point for constructing stress patterns is lexical stress, which can be obtained from pronunciation dictionaries such as the CMU Pronouncing Dictionary[1].

[1] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

No.	Phrase	Category	Semantics	Operator
1	dog	$noun(d)$	d:dog(d)	Lexical
2	the	$det(d)$	d:definite(d)	Lexical
3	the	$np(d) \rightarrow det(d) \bullet noun(d)$	d:definite(d)	Prediction (2)
4	saw	$verb(s, d, c)$	s:see(s), arg1(s,d), arg2(s,c)	Lexical
5	saw	$vp(s, d) \rightarrow verb(s, d, c) \bullet np(c)$	s:see(s), arg1(s,d), arg2(s,c)	Prediction (4)
6	cat	$noun(c)$	c:cat(c)	Lexical
7	the	$det(c)$	c:definite(c)	Lexical
8	the	$np(c) \rightarrow det(c) \bullet noun(c))$	c:definite(c)	Prediction (7)
9	the dog	$np(d) \rightarrow det(d)\ noun(d)\bullet$	d:definite(d),dog(d)	(1)+(3)
10	the dog	$s(\_) \rightarrow np(d) \bullet vp(\_, d)$	d:definite(d),dog(d)	Prediction (9)
11	the cat	$np(c) \rightarrow det(c)\ noun(c)\bullet$	c:definite(c),cat(c)	(6)+(8)
12	the cat	$s(\_) \rightarrow np(c) \bullet vp(\_, c)$	c:definite(c),cat(c)	Prediction (11)
13	saw the cat	$vp(s, d) \rightarrow verb(s, d, c)\ np(c)\bullet$	s:see(s), arg1(s,d), arg2(s,c), definite(c), cat(c)	(5)+(11)
14	the dog saw the cat	$s(s) \rightarrow np(d)\ vp(s, d)\bullet$	s:see(s), arg1(s,d), arg2(s,c), definite(c), cat(c), definite(d), dog(d)	(10)+(13)

Table 1: Sample entries during chart generation for "the dog saw the cat"

Stress patterns are not only used to represent input target forms, but also the metre of texts that are incrementally constructed through chart generation. When two edges are combined, their stress patterns are appended to obtain the stress pattern of the new edge that arises. Therefore, when attempting to add a new edge to the chart, the system can first check whether or not its stress pattern can appear as a contiguous subsequence of the target stress pattern. For example, the verb phrase "saw the cat" has a stress pattern `[s,w,s]`, and can thus be said to be compatible with an iambic pentameter metre because it can appear as a subsequence of `[w,s,w,s,w,s,w,s,w,s]`. However, the prepositional phrase "with the cat" has a stress pattern of `[w,w,s]`, and is thus not compatible, and hence should not be added to the chart.

By applying this metre check everytime an entry is attempted to be added to the chart, the search space can be significantly reduced, as it ensures that only texts that satisfy the metre constraints will be added to the chart.

Implementing topicality

As mentioned above, we aim to generate poems that explicitly convey a given meaning representation, preserving the fidelity of the message by taking into account predicate-argument structure, head-modifier relationships, and lexical semantics.

To that end, we implemented a preprocessing module that obtains meaning representations from a given text, which in our case is a newspaper article from an online website. An input URL is provided, and the main article content is extracted using a popular context extraction tool[2].

The article is split up into sentences, and each sentence is parsed using the Stanford Dependency parser[3] (Klein and Manning 2003). The set of dependency relations produced is taken to be the input meaning representation for the poem to be generated. Strictly speaking, a dependency parse cannot be said to be a genuine semantic representation, as it is still closely related to the constituent structure of the original sentence. Although the dependency relations do include semantic relations of an entity being the *agent*, *subject*, or *object* of another entity, a genuine semantic representation should abstract away from any syntactic decisions, whereas the dependency parse still contains relations such as *advmod* (adverb modifier) and *xcomp* (open clausal complement).

Nevertheless, such a representation is still a useful abstraction from the original text, and arguably does convey the semantics of the original text. In fact, the Stanford CoreNLP tool[4] actually refers to the dependency parse as a "semantic graph". In particular, it represents predicate-argument and head-modifier relations very well. It is precisely such relations that the keyword and phrase extraction-based approaches of previous topical poetry generation systems fail to capture.

For example, given an input sentence "The fox jumps over the dog.", the dependency parse output is as follows:

```
{det(fox-2, The-1),
 nsubj(jumps-3, fox-2),
 root(ROOT-0, jumps-3),
 det(dog-6, the-5),
 prep_over(jumps-3, dog-6)}
```

Since the chart generator will populate the initial chart with lexical entries based on the input meaning representation, whereas the relations above actually define relations between words, we must first explicitly add clauses for each word by introducing a `lex` relation and introduce a new variable for each word. Subsequently, we replace the arguments in the dependency relations to unify with these new variables, yielding the following representation:

```
{lex(a, [the, det]),
 lex(b, [fox, noun]),
 lex(c, [jumps, verb]),
 lex(d, [over, prep]),
 lex(e, [the, det]),
```

[2]https://code.google.com/p/boilerpipe/

[3]http://nlp.stanford.edu/software/lex-parser.shtml

[4]http://nlp.stanford.edu/software/corenlp.shtml

Synset ID	Gloss
#102118333	alert carnivorous mammal with pointed muzzle and ears and a bushy tail; most are predators that do not hunt in packs
#110022759	a shifty deceptive person
#114764910	the grey or reddish-brown fur of a fox

Table 2: WordNet entries for the noun "fox"

```
lex(f, [dog, n]),
det(b,a),
nsubj(c,b),
det(f,e),
prep_over(c,f)}
```

Mapping words to concepts

Although the meaning representation from the dependency parse explicitly states predicate-argument and head-modifier relations, it does so over strings of text such as "fox" and "jumps". To properly treat this as a semantic input, and to maximize the paraphrasing power of the generation component, these strings must first be transformed into semantic concepts. To achieve this, these strings are mapped onto appropriate WordNet synsets (Fellbaum 1998).

This increases the paraphrasing power of the generator, as it enables the generator to select synonyms to convey the concept, which may be necessary to satisfy rhythmic constraints.

Unfortunately, words are ambiguous symbols that may have many meanings, and such a mapping process raises the issue of word sense disambiguation (Agirre and Edmonds 2007). For example, given the word "fox" as a noun, WordNet has three different senses, which can be seen in Table 2.

In the initial version of the system that we develop, we simply take all possible senses for the word given the appropriate part of speech tag as returned by the dependency parser. Thus, in the example of "fox" above, all three senses of the noun are considered, but the three senses of "fox" as a verb are not.

Lexical resources

To accommodate the input meaning representations obtained from the dependency parser, the grammar, lexicon, and semantic representations must first be suitably modified. The lexicon is constructed by consulting WordNet and the CMU pronouncing dictionary. Before mapping to WordNet synsets, lemmatization is first applied to the words found in the dependency parses. Since WordNet only contains open class words, entries for closed class words such as determiners, prepositions, etc. are added manually.

Initial Experiments

To summarize the previous two sections, given an input URL and a target form, the poetry generation system proceeds as follows:

Ask in french surface
Call her years, check her
Think were toy tennis
Skid in chase, land her
(a)

Game were this tuesday
Is her but basket
James were drill friday
He were this target
(b)

This baby tell me
That I can miss you?
That you should hold me
Will know I miss you
(c)

Table 3: Sample output of initial experiments

1. Given an input URL, download the page and extract the main news content.
2. Parse each sentence of the text using the dependency parser.
3. For each sentence, apply chart generation to produce a text that conveys target semantics in the form of the target stress pattern.
4. Assemble all possible poems from the successfully generated sentences.

To test this system, three input articles were provided: two news articles from the sports section of the New York Times: "Maria Sharapova Is Finding Her Stride On Clay at Roland Garros"[5] and "James and the Heat Coolly Even the N.B.A. Finals"[6], and the lyrics to a contemporary R&B song, "Officially Missing You"[7]. From each of these input articles poems were generated using target stress patterns of 4 lines long, each consisting of 5 syllables, with a rhyming pattern of AB-AB. For the two news articles a stress pattern of `[s,w,s,s,w]` was specified, but for the song lyrics the generator was only constrained by the number of syllables. Table 3 shows some randomly selected sample output for each input article. Note that they are all perfect in terms of rhyme and metre, and they all roughly convey some aspects of semantics of their respective input articles.

Improving runtime complexity

Despite the fact that chart generation utilizes dynamic programming to make the process efficient, and that metre compatibility checking can substantially reduce the search space, the system as described is still very inefficient, and takes sev-

[5] http://www.nytimes.com/2014/06/07/sports/tennis/maria-sharapova-is-finding-her-stride-on-clay-at-roland-garros.html

[6] http://www.nytimes.com/2014/06/09/sports/basketball/lebron-james-and-miami-heat-coolly-even-the-series.html

[7] http://en.wikipedia.org/wiki/Officially_Missing_You

eral hours on a modern desktop PC to compute the sample output presented in the previous section.

A brief algorithm analysis will now be presented, together with some insights on how to speed up the process.

Assume that we are trying to generate a poem consisting of A lines based on an input article containing Z sentences. Assume also an input target semantics of N clauses, where for each word appearing in the semantics there are L possible WordNet synsets, with each synset having K synonyms. The lexicon that needs to be considered contains a total of $N \times L \times K$ entries. Finally, assume a grammar that contains M rules.

The chart generation process starts by considering all words from the lexicon that can possibly convey a section of the input semantics, and the bottom-up operator checks the M rules whether they predict the appearance of a word with the appropriate syntactic category. By taking into consideration repeated application of the scanning and completion operators discussed previously, until no more entries can be added to the chart, the theoretical worst case complexity is estimated to be:

$$O((((L \times K)^A \times P(N, A) \times M \times A) \times Z)^P) \quad (1)$$

where $P(N, A)$ is the permutation function, $\frac{N!}{(N-A)!}$.

Idea 1: Summarizing input text

In our initial experiment described above, the entire input news article is parsed and processed. As an example, the New York Times article about Maria Sharapova consisted of 1198 words. One idea to reduce complexity would be to try to summarize the article beforehand, and extract the semantic representation of the summary as input for the poetry generator instead. Aside from issues of complexity, attempting to convey the meaning of an entire news article in a short poem without really considering issues of discourse processing and coherence is slightly naive. Document summarization systems are precisely designed to analyse a text at the discourse level and to determine the most salient portions. Thus, aside from reducing complexity, this approach may also leverage the ability of such summarization systems to select a subset of content from the input news article that is more relevant to be conveyed.

Assuming that the Z sentences of the news article is summarized into P sentences, where P is the number of lines in the target form to be generated and is $< Z$, the complexity becomes:

$$O((((L \times K)^A \times P(N, A) \times M \times A) \times P)^P) \quad (2)$$

In our experiments, we use the popular document summarization tool MEAD[8] (Radev et al. 2003).

Idea 2: Sense disambiguation

In our initial version, the system simply considers all possible senses of a word when mapping to WordNet concepts. Given that this is done for all words in the input text, this creates a combinatorial explosion, many of which are likely to be incoherent combinations of senses.

To select the most appropriate word sense, the context of the target word, in this case the sentence in the news article to which it belongs, is compared against the context of the various available senses, i.e. the gloss and/or example sentences from WordNet. The modified Lesk algorithm is a well-known instance of this approach (Banerjee and Pedersen 2002). We employ a vector space model approach, where the two contexts are represented as vectors in a high-dimensional space and the sense that yields the highest cosine similarity is selected as the appropriate sense. In recent years, so-called word embeddings that have been trained using neural networks on very large corpora have yielded very good results. We use pre-trained vectors that have been made available as part of the GloVe[9] (Global Vectors for Word Representation) toolkit.

By applying word sense disambiguation, $L = 1$, thus the complexity becomes:

$$O(((K^A \times P(N, A) \times M \times A) \times P)^P) \quad (3)$$

Idea 3: Positional indexing

Chart generation is actually a dynamic programming approach to text generation that is motivated by chart parsing, which analyses a sentence and produces all parse trees based on a given grammar. In chart parsing, bottom-up processing starts with adding entries for each word appearing in the text to be parsed. However, since the order of the words is already known, entries in the chart are indexed based on the position they appear in the sentence. This index speeds up the process, since only edges that are incident to each other can possibly combine to yield new edges that represent larger constituent structures.

However, in chart generation such positional indexing is typically not used, as one does not know beforehand where words will appear in the sentence, and the overriding aspect that governs which edges can combine is that of semantic subsumption.

When considering metre compatibility during an attempt to add an edge to the chart, the system currently checks whether it can appear as a contiguous substring in the target form, but does not specify where precisely this substring is located. As a result, this substring matching process must be repeated every time, for every edge. When considering the interaction of this aspect with that of rhyme, it is possible that the chart generator spends a lot of time building partial structures that appear to be valid constructions early on, but eventually cannot fit the metre.

To overcome this, we augment the chart data structure by also recording the start and end position of each edge in terms of the syllable count within the poem. When adding lexical items to the chart at the beginning of the generation process, multiple instances are recorded for each word at each possible position within the poem. However, the metre compatibility check need only be computed once during the beginning, and when the generation subsequently proceeds,

[8] http://www.summarization.com/mead/

[9] http://nlp.stanford.edu/projects/glove/

the system need only ensure that pairs of incident edges are being combined, without having to perform any additional metre substring matching.

The complexity is thus further reduced to become:

$$O(((K^A \times P(N,A) \times M) \times P)^P) \quad (4)$$

Note, however, that due to the additional bookkeeping overhead and redundancy of having multiple entries for words based on the position they appear in, the memory complexity increases.

Idea 4: Greedy collation

In our initial version above, for each input sentence, chart generation is applied to produce a text that conveys the target semantics in the form of the target stress pattern for one line. Following this process, all possible combinations of these lines are assemble to yield all possible poems. This is a major source of inefficiency. The final modification that is carried out in an attempt to improve the efficiency of the generation algorithm is to replace this exhaustive combinatorial process with a greedy algorithm that selects subsequent lines so as to maximize an objective function that considers the aspects of rhyme, metre, and semantics.

Firstly, all possible candidates for the first line are tried in turn. For each subsequent line l, a candidate is selected that maximizes the following objective function:

$$f(l) = \alpha_1 \times rhyme(l) + \alpha_2 \times syll(l) + \alpha_3 \times sem(l)$$

where:

- $\alpha_1, \alpha_2, and \alpha_3$ are weight factors in the interval [0,1] and $\alpha_1 + \alpha_2 + \alpha_3 = 1$.
- $rhyme(l)$ is a function that returns a value of 1 if l ends with a correct rhyme, 0 otherwise.
- $syll(l)$ is a function that returns a normalized syllable count, e.g. the ratio of the number of syllables found in l to the number of syllables in the target form for that line.
- $sem(l)$ is a function that returns a normalized semantic content count, e.g. the ratio of the number of semantic clauses conveyed by l to the maximum number semantic clauses obtained for that sentence during generation.

The complexity is thus further reduced to become:

$$O((K^A \times P(N,A) \times M) \times P) \quad (5)$$

Subsequent experiment

To test the various modifications that were designed and implemented, the system was run with the exact same input as during the initial experiment, and results can be seen in Table 4. As can be seen, the overall quality of the results suffers as a result of some of the modifications, and possibly most notably the use of a greedy algorithm to assemble the resulting poem. For instance, from the point of view of the rhyme and metre the solutions are sub-optimal.

On the other hand, whereas previously the generator would run for many hours to complete, the empirical running time measurements from the modified system show that the modified system typically takes approximately 20-30 seconds to generate poems given the same size of input.

Is she
Court were full even
She were take couple
She were a woman
(a)

James were a system
Are an air system
Was a james
Aver get way
(b)

Tell me are you
Wish you now call me
Fix over with you
Guess was it was I
(c)

Table 4: Sample output of subsequent experiments

Discussion & summary

In this paper we have presented work in progress on the development of a poetry generation system that uses a dependency parser to extract the predicate argument structure of the input article, and tries to maintain this structure through deep syntactic text generation whilst complying with a given target form. The combinatorial nature of this task presents huge challenges, and several improvements have been suggested and applied in an attempt to produce poetry in a tractable fashion. Whilst this does drastically improve the complexity of the algorithm, changing the running time from several hours to a matter of seconds, the quality of the output seems to visibly suffer.

Deep natural language generation that is constrained by a target semantics at one end and a target form on the other end is a very difficult task. Whereas other poetry generation systems try to achieve this through the means of evolutionary computation and template-based generation, our work can be seen to be related to the work reported in (Toivanen, Järvisalo, and Toivonen 2013), as the task can be cast as a constraint satisfaction problem. Unfortunately, imposing syntactic constraints on a constraint satisfaction problem, where the syntactic constraints are defined as context-free grammar rules is a very computationally expensive problem. Our approach is to utilize chart generation, a well-known dynamic programming technique where the grammar rules are a fundamental component of the algorithm. Another strategy worth considering for future work is *context-free grammar filtering* (Kadioglu and Sellmann 2008), a time and space efficient arc-consistency algorithm that allows the formal specification of constraints as a context-free grammar within a constraint satisfaction problem framework.

References

Agirre, E., and Edmonds, P., eds. 2007. *Word Sense Disambiguation: Algorithms and Applications*. Springer.

Banerjee, S., and Pedersen, T. 2002. An adapted lesk al-

gorithm for word sense disambiguation using wordnet. In Gelbukh, A., ed., *Computational Linguistics and Intelligent Text Processing*, volume 2276 of *Lecture Notes in Computer Science*, 136–145. Springer Berlin Heidelberg.

Colton, S.; Goodwin, J.; and Veale, T. 2012. Full-face poetry generation. In *Proceedings of the Third International Conference on Computational Creativity*, 95–102.

Diaz-Agudo, B.; Gervás, P.; and González-Calero, P. 2002. Poetry generation in COLIBRI. In *Proceedings of the 6th European Conference on Case Based Reasoning (ECCBR 2002)*.

Fellbaum, C., ed. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

Gervás, P. 2002. Exploring quantitative evaluations of the creativity of automatic poets. In *Proceedings of the 2nd. Workshop on Creative Systems, Approaches to Creativity in Artificial Intelligence and Cognitive Science, 15th European Conference on Artificial Intelligence (ECAI 2002)*.

Gervás, P. 2015. Tightening the constraints on form and content for an existing computer poet. In *AISB 2015 Symposium on Computational Creativity*. University of Kent, Canterbury, United Kingdom: Society for the Study of Artificial Intelligence and Simulation of Behaviour.

Gonçalo Oliveira, H. 2012. PoeTryMe: a versatile platform for poetry generation. In *Proceedings of the ECAI 2012 Workshop on Computational Creativity, Concept Invention, and General Intelligence*, C3GI 2012.

Kadioglu, S., and Sellmann, M. 2008. Efficient context-free grammar constraints. In *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, 310–316.

Kay, M. 1996. Chart generation. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, 200–204. Santa Cruz, USA: ACL.

Klein, D., and Manning, C. D. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, 423–430. Association for Computational Linguistics.

Manurung, R.; Ritchie, G.; and Thompson, H. 2012. Using genetic algorithms to create meaningful poetic text. *Journal of Experimental & Theoretical Artificial Intelligence* 24(1):43–64.

Manurung, H. M. 1999. A chart generator for rhythm patterned text. In *Proceedings of the First International Workshop on Literature in Cognition and Computer*.

Radev, D.; Otterbacher, J.; Qi, H.; and Tam, D. 2003. MEAD ReDUCs: Michigan at DUC 2003. In *DUC03*. Edmonton, Alberta, Canada: Association for Computational Linguistics.

Rashel, F., and Manurung, R. 2014. Pemuisi: A constraint satisfaction-based generator of topical indonesian poetry. In *Proceedings of the Fifth International Conference on Computational Creativity*.

Toivanen, J. M.; Järvisalo, M.; and Toivonen, H. 2013. Harnessing constraint programming for poetry composition. In *Proceedings of the Fourth International Conference on Computational Creativity*, 160–167.