Poetic Precision

Imperial College London

Nitin Nihalani

Department of Computing

Imperial College London

An interim report for

MEng Computing (AI)

Individual Project

Contents

Chapter 1

Introduction

"Dream in a pragmatic way." - Aldous Huxley

1.1 Motivation

Computational Semantics is a relatively young and fashionable topic in Computational Linguistics. It involves finding representations and algorithms that are able to cope with the *meaning* of linguistic utterances.

Pragmatics is an even younger discipline, concentrating on the context of those utterances. For example, the phrase *"I have a green light"* may mean that I have been granted permission for something, or that I literally have a lamp with a green tint. When we as humans read or listen to any linguistic output, we build a representation of the objects, people, actions, descriptions, relationships and anything else to provide context to the experience.

A similar approach is taken when writing or speaking - one has a purpose and a message that would like to be passed and the language used helps to build such context. For example, if I aim to tell you that I have a lot of tea, I could simply say so or say I am 'drowning' in tea. The latter helps you, the reader, realise that not only do I have a lot of tea but also that I cannot handle it and that I am talking about the drink rather than the leaves.

For a computer to truly converse in a manner indistinguishable to humans, as is the aim of the elusive Turing Test, it must be able to handle pragmatics along with syntax and semantics. This requires a deep understanding of words not as a linguistic unit, but as the objects, actions and descriptions they represent. They should then be used with a purpose when generating text, as human writers and speakers do.

Poetry is a linguistic art form designed to help convey a particular, well-defined message in a memorable and powerful way. Poets write poems that are succinct in length but dense in meaning by employing a number of techniques, such as rhyme and rhythm.

Different types of poetry each have their own sets of constraints, features and priorities that define their best usage. For example, narrative poems convey characters, relationships and actions while descriptive ones are used to give a comprehensive linguistic illustration. Furthermore, rhythm and rhyme can be used to provide melody making it pleasant to hear and applicable to songs, but alliteration can be used to create suspense and a sense of danger.

We aim to create a system that can analyse poetry and build a contextual representation of it, as well as generate poetry with an underlying purpose and message. This process will also act as a catalyst with which to develop computational semantics and pragmatics.

The proposed implementation comes in three phases:

1. First we will write the analysis module, which detects a wide range of poetic features in a single poem. It also aims to represent the context of the poem with Discourse Representation Structures (DRSs), outlined in section 2.4.4.

2. Then we run many poems of the same type through the analysis module and abstract the common features between the given poems into a template. This includes a general structure for the DRSs of that class of poem.

3. Finally we generate poems with a purpose as guided by the structure of the DRS. We will also utilise third-party libraries to build semantically and syntactically valid lines of poetry that also use poetic features. Poeticness and creativity is prioritised in the selection of words and phrases during the generation process. A dry run of this phase is given in the appendix.

1.2 Objectives

The overarching primary objective of this system is *pragmatic competence*. We aim to generate poetry that demonstrates some understanding of context with regards to descriptions, actions and relationships of people and objects, and with careful text and sentence planning for that context.

Thereafter, we wish to create a system that:

- identifies a broad list of features in a single poem.
- abstracts features of a given class of poems or texts that have been analysed.
- learns the features of a wide variety of different classes of poems.
- produces poems, given natural language 'seeds' of inspiration, that are:
  - novel,
  - syntactically valid,
  - semantically interpretable,
  - pragmatically consistent,
  - evident of a set of desired features.
- is able to digress slightly from learned features in its use of poetic techniques in search of creativity.

1.3 Contributions

This project makes contributions towards both Computational Creativity and Computational Linguistics.

- We demonstrate the ability for computer systems to assess written natural language works in terms of its structure, common words and phrases, rhetoric and poetic features such as rhyme, rhythm and alliteration.
- We will investigate the appropriateness of Discourse Representation Theory as a semantic representation of poetry from which we can derive pragmatics in terms of characters, objects and locations, descriptions of them, relationships between them and the actions that they executed.
- We demonstrate the ability to abstract common written features out of a large number of comparable texts.
- We take in a step in the direction of using the web as a source of material and conceptual inspiration for creative acts.

- We demonstrate Discourse Representation Theory as an effective tool for guiding the macro- and micro- planning stages of natural language generation.
- We demonstrate the effectiveness of third-party libraries for the surface realisation stage of Natural Language Generation.
- We provide a tool for poetry creation from natural language seeds of inspiration.
- We investigate the applicability of the new FACE and IDEA descriptive models for evaluation.

Chapter 2

Background

This chapter gives a brief overview of the features and classes of poetry that exist, along with why people are interested in writing them. We will discuss, critique and gather inspiration from the related work in the area of poetry generation that most relate to the approach taken in this paper. We then give brief overviews into the fields of Computational Linguistics and Computational Creativity, both of which are involved in the task of automatic poetry generation. These overviews are by no means complete or comprehensive, but should provide enough information for those not familiar with the areas to understand and appreciate this paper.

2.1 Poetry Theory

To fully comprehend the task that we are about to undertake, we need to have an understanding of poetry as an art form. Poetry is believed to predate even literacy and has been documented ever since. Over the ages, different styles of poetry have evolved. Here we discuss those styles and the underlying reason for why poetry is written.

2.1.1 The Purpose of Poetry

Merriam-Webster dictionary defines poetry as:
writing that formulates a concentrated imaginative awareness of experience in language chosen and arranged to create a specific emotional response through meaning, sound, and rhythm.

Let us break this down:

- *Formulates* implies that there is method to the process of writing a poem.
- *Concentrated* accentuates the fact that poems are generally short, as they are counted in stanzas in lines rather than paragraphs and pages.

- *Imaginative* confirms the fact that this is a creative act, that has a level of non-determinism and need not be entirely realistic.
- *Awareness of Experience* embodies the need for general background knowledge based on a particular set of experiences that surface when writing a poem.
- *Language Chosen and Arranged* reiterates that this is a methodical and systematic art - decisions are made carefully with precise intention.
- *Create a specific emotional response* gives the main purpose of the poem - to express a feeling or an idea and elicit an emotional response.
- *Through meaning, sound, and rhythm* describes that this purpose is not reached purely by words, but other language features.

To summarise the purpose of poetry from this, we would say that it is using one's imagination and experience to trigger empathy about a particular topic from the reader. This means that poetry is a vehicle through which poets can share a very personal message that they want the reader to experience and remember.

We must keep this in mind throughout the project, as it is important to realise that the features of poetry discussed in the next section might be seen as arbitrary rules on form rather than purposeful techniques used to make the language more concise and effective.

2.1.2 Features of Poetry

The definition above mentions the use of *meaning, sound and rhythm* in poetry. These add an extra layer of subtext to poems to help the author remain concise while still eliciting a full emotional response from the reader. We call these techniques *features* of poetry throughout this paper. There are many features of poetry to address, but we have scoped this project down to concentrate on the following popular ones.

2.1.2.1 Rhyme

Two words rhyme when they sound similar when spoken out loud. *Cat* and *fat* in figure 2.1 rhyme, as do *mice* and *nice*. Rhyming words need not spell the same way,

There once was a big brown ***cat***
That liked to eat a lot of ***mice***
He got all round and ***fat***
Because they tasted so ***nice***

Figure 2.1: A rhyming quatrain often used in teaching poetry

for example, *kite* and *height*.

Strict rhyme enforces the exact same sound while weak rhyme only requires that the vowel sounds are the same. Examples of weak rhyme are *turtle* with *purple* and *tragedy* with *strategy*.

A piece of text has a rhyme scheme if there is a pattern of rhyme between its lines. For example, the poem in Figure 2.1 has an *ABAB* rhyme scheme.

Rhyme can also occur within a line (internal rhyme) or between words in the middle of different lines.

Major Purposes

- Pleasant to hear, making the listener feel more comfortable and listen carefully.
- As a mnemonic device.
- Used at the end of lines of poetry and songs making the rhythmic structure more distinct.

In this project, we will detect and reproduce end-line rhyme and internal rhyme where applicable. It will also be prioritised when producing poetry for which a rhyme scheme is not mandatory but fits the purposes.

2.1.2.2 Rhythm

Rhythm is the pattern of emphasis of syllables that occurs in a line of poetry. There are three major ways of measuring rhythm, often used in tandem - syllabic, quantitative and accentual.

Syllabic rhythm enforces a certain number of syllables to be used in a particular line of poetry. Haikus are the most famous type of poem with this feature - they are three

The bartender said
to the neutron, 'For you, sir,
there will be no charge.'

Figure 2.2: A humourous Haiku

lines long with the first and last lines restricted to 5 syllables and the second to 7. An example is given in Figure 2.2.

Quantitative measures use the fact that some syllables *sound* longer than others when spoken out loud. Long sounding syllables are *stressed* while short ones are *unstressed*. Accentual measures are similar to Quantitative, but they work on the *tendency to emphasize a particular syllable* when spoken out loud, rather than its length. It is important to note that a word's meaning can change depending on stress. For example, '**ob**ject' is a noun whereas 'ob**ject**' is a verb.

Lines of pre-defined patterns of stressed and unstressed syllables are called *meters*. Lines with meter are made up of individual units called *feet*. The five major foot types in poetry are given in Table 2.1.

Foot Type	Pattern	Example
iamb	unstressed - stressed	des**cribe**
trochee	stressed - unstressed	**po**em
spondee	stressed - stressed	**popcorn**
anapest	unstressed - unstressed - stressed	meta**phor**
dactyl	stressed - unstressed - unstressed	**po**etry

Table 2.1: The major poetic foot types with their corresponding pattern and an illustrative example.

The metre is formed by repeating feet, typically with up to six feet:

- Monometer: 1 foot
- Dimeter: 2 feet
- Trimeter: 3 feet
- Tetrameter: 4 feet
- Pentameter: 5 feet
- Hexameter: 6 feet

All Shakepeare's sonnets are written in iambic pentameter, i.e. five repetitions of unstressed-stressed syllables. The first line of his Sonnet II as an example:
*When **for**ty **win**ters **shall** be**siege** thy **brow***

Major Purposes

- Introduces a melody based on the natural intonations of speech.
- Adds a level of predictability and structure that resonates with readers and listeners.
- Emphasizes the message by putting stress on the words that matter.

Rhythm is a fundamentally important feature of poetry, so this project aims to be able to detect and reproduce it consistently and to a high level of accuracy.

2.1.2.3 Sound Devices

This project considers four types of sound devices.

The first is **onomatopoeia** - words that imitate or suggest sounds of particular sources. For example, the *pow* of a punch or the *tick-tock* of a clock. This technique has mostly been used in comic books to help the reader experience the sound of the scene to go with the image.

The next three devices are repetitions of a pattern of similar sounds, like rhyme. **Consonance** is the repetition of similar consonant sounds (e.g. *pitter patter* repeats the 'p', 't', and 'r' sounds), while **assonance** is that of vowels (e.g. *doom and gloom* repeats the 'oo' sound). **Alliteration** is a special case where the repeated sound occurs at the beginning of consecutive words. *Zany zebras zigzagged through the zoo* has alliteration on the letter 'z'.

Major Purposes

- Poets use onomatopoeia to help describe actions or atmosphere richly. A famous example is the nursery rhyme 'Old MacDonald', which uses onomatopoeia of the sounds that animals make to describe the farm, figuratively placing the reader or listener in the farm itself.

- Alliteration, consonance and assonance are pleasant to listen to when spoken out loud.
- Can be used to add drama to an action.
- Sometimes used to suggest danger.

We aim to detect and reproduce all forms of sound devices explained here. However, there will be a limit to the onomatopoeic vocabulary and it will be unable to create brand new onomatopoeia.

2.1.2.4 Structure

The structure of the poem is the organisation of lines in a poem. The main unit is the stanza, which is a fixed amount of lines grouped by rhythmical pattern.

There are four major types of stanza:

- Couplet: 2 lines
- Tercet: 3 lines
- Quatrain: 4 lines
- Cinquain: 5 lines

Stanzas can also be called *verses*, which have the added property of a rhyme scheme. A *chorus* is a special type of verse that is repeated throughout a poem.

Features of the structure of a poem include:

- The number of stanzas.
- The number of lines per stanza.
- The number and positions of repeated lines.
- The number and positions of repeated stanzas.
- Enjambment; the continuation of a sentence over a line-break

The Haiku in Figure 2.2 has a single tercet structure with no repetitions. Songs are generally several stanzas long, with a chorus interleaving longer non-repeating verses.

Major Purposes

- Helps to guide the reader through the story.
- Forces the poet to be more succinct and purposeful.
- Manages the storyline - changes in stanza often suggest a change in perspective or message.
- Repetition helps drive home the main message.
- Ties several thoughts together into one continuous flow.

We concentrate on detecting and reproducing accurate structures of common poetry types in this project.

2.1.2.5 Symbolism and Imagery

Symbolism and imagery are general terms for creating an overall image in the reader's mind by describing a subject or object as something else with desired qualities.

Techniques include:

- **Metaphor**: an object is described as another object with a set of desirable characteristics. For example, saying someone is a lion immediately creates the image of bravery, intimidation and power.
- **Simile**: an object or action is specifically described using an adjective or adverb, but compared to another object that is a stereotypical example of that description. The phrases 'like a' and 'as a' are often used, e.g. *Runs like a cheetah*, *Slippery as an eel*.
- **Hyperbole**: unrealistic exaggeration, often used in tandem with metaphor e.g. *Cried a river of tears.*
- **Powerful Verb**: a more exciting way to describe an action using unusual verbs, e.g. *Wormed through the crowd.*

Major Purposes

- Explain complex concepts concisely.
- Induce empathy from the reader by relating it to something they understand.

This paper attempts to detect use of metaphor and simile as well as reproduce it. The pragmatics around the use of metaphor and simile will be considered, for example using the ocean to show power rather than the sun if the poem generally speaks of water. We do not consider Hyperbole and Powerful Verb as they are similar to Metaphor, but it is worth noting that they exist because including them should be an accessible extension for future work.

2.1.2.6 Pragmatics and Personification

Poetry is similar to storytelling in that it has characters around which the poem is written. Understanding who or what they are, their descriptions and their actions are all part of the underlying message that the poet wants to get across. This is the context of the poem and defining it is a pragmatics problem.

Personification is a technique used by poets to give inanimate objects life, expressing actions and descriptions as if it were human. This is a powerful technique that relates to imagery, helping poets make more abstract messages clearer. For example, *the moon smiled* gives the moon life by describing it as having performed a human-like action with full intention of doing so. Noting the use of personification can make the context of the poem clearer, as often inanimate objects are the subject of the poem.

Major Purposes
Context is the underlying message in its bare form. It is the story that the poet wishes to tell and guides the use of all other features.

In this paper, we aim to extract characters and differentiate them by their descriptions and actions. This is vital in understanding the poem and can help us generalise the uses of features when attempting to produce a coherent story as the backbone to the generated poem. Furthermore, it will help determine the type of poem (narrative, lyrical, descriptive etc.) and will help guide generation of poems of a particular type.

2.1.2.7 Theme

Theme is a very abstract term that is difficult to define. It can be thought of as the worldly context of the poem, often requiring better understanding and background knowledge of the poet. For example, Maya Angelou's *I Know Why The Caged Bird Sings* is about oppression of African Americans. It was her way of helping readers empathise with her situation, by personifying a caged bird and the similarities she shares with it. The theme is therefore *oppression*. However, without knowing Angelou's life situation, it would have been impossible to determine.

Theme is beyond the scope of this project, both in interpretation of poetry and generation. However, we will attempt themes that can be summarised into a single word, which we will refer to as *topics*. For example, we will try to emulate the styles of *Love* poems or *War* poems from given examples. This will help give more meaning to the features described here and help in using them in the right context during the generation phase.

2.1.3 Classification of Poetry

We define a type of poetry as a particular form of poem with a set of unique features, including those described in the previous section. Some types are very popular and have had their styles, features and purposes documented and taught. Out of these grew categories of different types that tend to be used for similar purposes.

This project attempts to derive these categories and some popular types of poetry by analysing many comparable poems.

2.1.3.1 Categories

There are many types of poem all with different form. However, there are only three main categories of purpose for a poem:

1. *Lyrical* poems have an identifiable speaker whose thoughts and emotions are being expressed in the poem. This means that poems of this category have very few characters, a song-like structure and tend to be in a reflective tone, generally

using a lot of symbolism. Maya Angelou's *I Know Why The Caged Bird Sings* is an example of this, along with many songs.

2. *Descriptive* poems describe the surroundings of the speaker. This is identifiable by the use of adjectives and complex imagery. Many objects may appear in this type of poem to be able to give an in-depth description of the environment and atmosphere. There will be very few action verbs used.

3. *Narrative* poems concentrate on telling a story. It therefore has a coherent plot line, several characters with explicit relationships between them, action and climax. Ballads and Epics are types of narrative poems.

Some popular poem types do not fall under any one bracket as they can be used in any of the above categories. Examples include Haikus and Limericks. This project aims to be able to place any poem accurately into one of these categories.

2.1.3.2 Popular Types

As well as determining the category of poems, we aim to be able to detect and reproduce some popular types of poetry. For this project, we will concentrate on:

- *Haiku:* single tercet structure with 5-7-5 syllabic rhythm.
- *Limerick:* single cinquain structure with AABBA rhyme scheme. Lines 1, 2 and 5 have 7-10 syllables, while lines 3 and 4 have 5-7 syllables. The first line tends to begin with "There was a..." and ending with a person or location. Limericks are usually used for humour as the last line is generally a punchline.
- *Sonnet:* 14 lines, each in iambic pentameter with an ABAB CDCD EFEF GG rhyme scheme, i.e. three quatrains followed by a rhyming couplet.
- *Elegy:* usually used to mourn the dead, its lines alternates between dactylic hexameter and pentameter in rhythm. It has no particular rhyme scheme, although does still use rhyme.
- *Ode:* Description of a particular person or thing, using plenty of similes, metaphors and hyperbole.

- *Ballad:* Tells a story and has a number of quatrains, each with an AABB rhyme scheme. Lines alternate between iambic tetrameter and iambic trimeter.
- *Cinquain:* as the name suggests, this has 5 lines. They are not rhymed, but have a 2-4-6-8-2 syllabic pattern.
- *Riddle:* Riddles describe things without telling what it is, using anaphora to refer to it. Ususally told in a number of rhyming couplets.
- *Free Verse:* No particular features attached to this type.

Some of these poems are harder to read and generate than others, particularly in terms of pragmatics. However, this selection covers the main features of poetry that we would like to address so it a good way to evaluate this project.

2.2 Lessons from Related Work

This section looks at six important previous attempts at automatic poetry generation. They each have some aspect of investigation or experimentation that have influenced this project. However, each of these attempts has its limitations that we look to overcome in this project.

2.2.1 Actively Gather Inspiration

Colton et al. published a paper in the International Conference of Computational Creativity 2012[13], whose main objective was to describe the first poetry generation system that satisfied the FACE Descriptive model[14]. It is a *Form Aware*[29] implementation that constructs templates of poems based on constraints of poetic features.

The most interesting point of this paper was its admission that inspiration cannot come from the technology and must come from the user. By taking this into account, it now takes inspiration from news articles as seen in Figure 2.3. However, since its objective was focused on passing a particular evaluation model, the poems created by this system are relatively simple and the processes rudimentary - using randomness rather than semantic applicability and coherent pragmatics.

It was generally a bad news day. I read an article in the Guardian entitled: "Police investigate alleged race hate crime in Rochdale". Apparently, "Stringer-Prince, 17, has undergone surgery following the attack on Saturday in which his skull, eye sockets and cheekbone were fractured" and "This was a completely unprovoked and relentless attack that has left both victims shocked by their ordeal". I decided to focus on mood and lyricism, with an emphasis on syllables and matching line lengths, with very occasional rhyming. I like how words like attack and snake sound together. I wrote this poem.

Relentless attack
a glacier-relentless attack
the wild unprovoked attack of a snake
the wild relentless attack of a snake
a relentless attack, like a glacier
the high-level function of eye sockets
a relentless attack, like a machine
the low-level role of eye sockets
a relentless attack, like the tick of a machine
the high-level role of eye sockets
a relentless attack, like a bloodhound

Figure 2.3: The Guardian article used for inspiration(left) and the resulting poem(right).

2.2.2 Constrain to Improve Creativity

Recently, Toivanen et al. attempted a solution that used off-the-shelf constraint solvers[37] to produce poetry. Their solution, illustrated in figure 2.4, also received inspiration from another source. This was then combined with other sources to build the set of candidate words, form requirements and content requirements. These were passed into a constraint solver with a manually encoded static constraint library powered by Answer Set Programming.

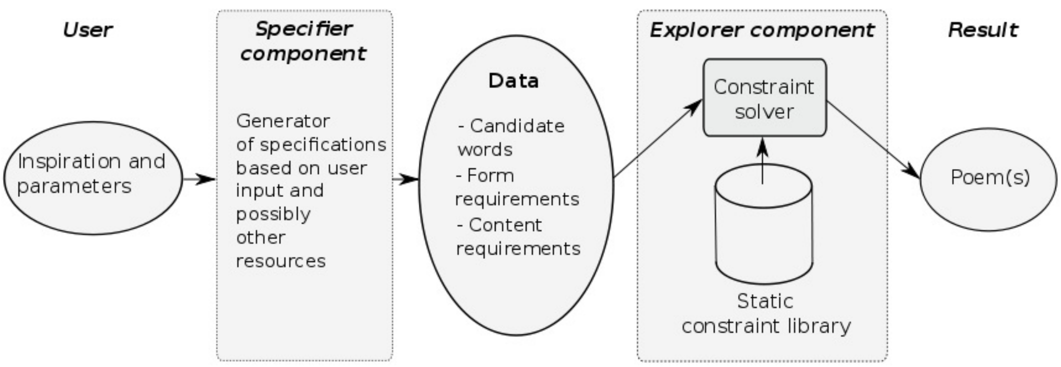

Figure 2.4: Complete poetry composition workflow.

N SG VB, N SG VB, N SG VB! PR PS ADJ N PL ADJ PRE PR PS N SG: – C ADV, ADV ADV DT N SG PR VB! DT N SG PRE DT N PL PRE N SG!	*Music swells, accent practises, theatre hears!* *Her delighted epiphanies bent in her universe:* *– And then, singing directly a universe she disappears!* *An anthem in the judgements after verse!*

Figure 2.5: The POS template used for constraint input(left) and the resulting poem(right).

The idea that constraints do not hinder but rather help the creative process is an attractive one for Computational Creativity research. Constraining words and other requirements for each particular word position is a natural technique for constraint programming, but extremely restrictive. First, the size of each line must be defined by number of words *and then* by rhythm and other poetic features. Secondly, once the candidate words are chosen there is no scope for further filtering. Finally, the structure of the poem in terms of its parts-of-speech (section 2.4.2) tags must be defined beforehand and is taken from previous poems of the same type, as seen in Figure 2.5. Even though this is an efficient method that has produced impressive results, it is too restricted to produce truly creative work.

2.2.3 Learn from Experience

Ray Kurtzweil Cybernetic Poet (RKCP), created by Kurtzweil himself[27], addresses the issue of having a predefined template. He uses a stochastic approach that takes advantage of n-grams to build lines from words. The system was trained on a selection of poems and created a template and n-gram corpus from those poems. RKCP would then be able to create similar types of poems. Algorithms were employed to ensure that poems were not exact copies of other poems and to maintain a coherent theme.

Scattered sandals
a call back to myself,
so hollow I would echo.

Figure 2.6: A haiku written by Ray Kurzweil's Cybernetic Poet after reading poems by Kimberly McLauchlin and Ray Kurzweil

This method is more flexible and has granular word selection. However, the vocabulary would still be limited and the form of the poem is not well defined due to being probabilistic. We can see that in Figure 2.6, the attempted Haiku has a syllabic rhythm is 4-6-7 as opposed to 5-7-5. A specific purpose or storyline is not definable and the use of imagery is only probabilistic. A lot also depends on the poems given as examples, limiting the pragmatic and semantic capabilities.

2.2.4 Choose Words Carefully

MCGONAGALL[29] takes a semantic representation of a sentence, called *semantic expressions*, as input into an NLG system. For example, the semantic expression of *"John loves Mary"* would be {*john(j), mary(m), love(l, j, m)*}

These are used as starting points for initialisation of his evolutionary system that uses stochastic methods to determine the best values to be carried forward to further iterations.

They play. An expense is a waist.
A lion, he dwells in a dish.
He dwells in a skin.
A sensitive child,
he dwells in a child with a fish.

Figure 2.7: Resulting MCGONNAGAL poem when seeded with a couple of lines of Hilaire Belloc.

Of particular note is the structure of a lexical entry into the system. It is enriched with much semantic information, as in Figure 2.8, that backs up the fitness score and helps MCGONAGALL form syntactically and semantically correct sentences. We will use much of his ideas in this area. However, pragmatism is still lacking because of the restrictions on evolution. It does not take particular types of poetry into account and there is little scope for creativity due to the strictness of grammar generated.

2.2.5 Derive Insight from Worldly Knowledge

Tony Veale's daring approach to knowledge-based poetry generation[38] concentrates on symbolism and imagery - arguably the hardest tasks in automatic poetry gener-

Field	**Value**
Key	`lion_n`
Orthographic spelling	*lion*
Phonetic spelling	`[L,AY1,AH0,N]`
Semantic expression	lion(X,Y)
Semantic signature	$\boxed{X,Y}$
Anchored trees	`I_N_N, I_C_NP, A_R_C_NP`
Feature structure	$\begin{bmatrix} \text{CAT} & n \\ \text{AGR} & \begin{bmatrix} \text{NUM} & sg \\ \text{PERS} & 3 \\ \text{3RDSG} & + \end{bmatrix} \\ \text{PRON} & - \\ \text{PROP} & - \\ \text{SUB} & \begin{bmatrix} \text{ANIM} & + \end{bmatrix} \end{bmatrix}$

Figure 2.8: Semantically enriched lexical entry for *lion* in MCGONNAGAL

ation. He uses the theory of Mutual Knowledge through norms and stereotypes to build a structure that uses various words to describe objects and derive stereotypical characteristics. Out of this grew a very useful tool: Metaphor Magnet[39].

His methods have obvious limitations in that they do not consider rhyme, rhythm or any other poetic feature other than symbolism. However, we will take advantage of the tools that have been born out of his project to give this system more symbolic choices of words and phrases as Figure 2.9 shows that they are quite impressive.

My marriage is an emotional prison
Barred visitors do marriages allow
The most unitary collective scarcely organizes so much
Intimidate me with the official regulation of your prison
Let your sexual degradation charm me
Did ever an offender go to a more oppressive prison?
You confine me as securely as any locked prison cell
Does any prison punish more harshly than this marriage?
You punish me with your harsh security
The most isolated prisons inflict the most difficult hardships
O Marriage, you disgust me with your undesirable security

Figure 2.9: 'The legalized regime of this marriage', a poetic view of marriage as a prison

2.2.6 Dare to be Different

WASP is one of the first attempts at an automatic poetry generator. It is a rule based system that takes a set of words, a set of verse patterns and returns a set of verses[22]. It uses heuristics to guide the construction to fit structure, but no semantic limitations are enforced.

This has obvious limitations but Gervas, the creator of this system, does make a good point that poetry's creativeness is somewhat down to daringness of transgression. We keep this in mind to allow some level of randomness and mutation from expected norms in this project.

2.3 Brief Overview of Computational Creativity

Simon Colton and Geraint Wiggins, pioneers of Computational Creativity, define research in this area as:
The philosophy, science and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviours that unbiased observers would deem to be creative.[15]

In the context of Automatic Poetry Generation, we are creating a system that is *responsible* for generating aesthetically pleasing, meaningful and novel poems. The poems still need to be sufficiently similar to existing works created by humans such that it *exhibits behaviour* to which *unbiased observers* can relate and recognise.

This definition has evolved from one where behaviour was *deemed creative if exhibited by humans*[41]. However, recent developments in the area have lead to the requirement of more quantitative measures for evaluation than Turing-style tests, such as the FACE and IDEA descriptive models[14].

This area of research has come under scrutiny for philosophical reasons, but has had support from Alan Turing and other pioneers of Artificial Intelligence. It has since been accepted as a valid area of research, with the annual International Conference on Computational Creativity heading into its fifth round.

Successes of Computational Creativity:

- Simon Colton's *Painting Fool*[12] produced paintings that managed to trick art lovers into believing that it was the work of a talented human artist. An example is given in Figure 2.10.
- *JAPE*[7], created by Ritchie and Binsted in 1994 was given a general, non-humorous lexicon and generated puns as answers to questions. For example:
  Q:What do you call a strange market?
  A: A bizarre bazaar
- *Iamus* by Gustavo Diaz-Jerez[18], which composed music entirely on its own that was then recorded by London Symphony Orchestra
- *The Policeman's Beard is Half Constructed*[10] is recognised for being the first book, which included some poetry, to have been written entirely by a computer program, RACTER.

2.4 Brief Overview of Computational Linguistics

Computational Linguistics is a wide area of research, covering Speech Recognition, Natural Language Processing and Generation and with overlaps in several other areas such as Machine Learning and Knowledge Representation. In fact, Daniel Jurafsky and James H. Martin needed almost a thousand pages to cover the foundations of this area[23].

Figure 2.10: Chair #17 at the Performing Sciences Exhibition, La Maison Rouge, Paris, Sept 2011

Automatic Poetry Generation uses many lessons from Computational Linguistics. Here we will briefly discuss the major ones in general and in the context of machine poetry analysis and generation. For an in depth general study of Computational Linguistics, we refer the interested reader to Jurafsky and Martin's book.

2.4.1 Words

Words are the fundamental building blocks of language. They have been studied for creation of spell-checkers, text-to-speech synthesis and automatic speech recognition. Two major subsets where poetry is concerned is the study of pronunciation and morphology.

The CMU Pronunciation Dictionary[40] has taken the first steps towards computationally modelling the phonetics of words. It uses the ARPAbet phoneme set (see Table A.2 in the Appendix) is highly important for poetry generation as it helps machines reason about rhyme and sound devices by simply comparing phonemes. It has over 133,000 words mapped to its corresponding pronunciation.

To illustrate how this works, let us take two words that are spelled differently but pronounced the same - *kite* and *height*. The Jaro-Winkler distance, a normalised score of similarity between strings, for the tail of these words (in search of rhyme) gives 51.11%, indicating that it is barely probable that they rhyme if we only looked at spelling. Their corresponding phoneme sets are *'K AY1 T'* and *'HH AY1 T* respectively. Now it is trivial to compare them and see that the tails are exactly the same and therefore rhyme.

Notice the '1' appended to the 'AY' phonemes. Vowel phonemes come with a digit appended to them that defines the emphasis placed on this syllable:

- 0: unstressed
- 1: stressed
- 2: light/secondary stress

Morphology of words is the study of putting words together with morphemes, the smallest unit of grammar. To use Jarufsky and Martin's example, the word *fox* consists of a single morpheme that is itself, but *cats* has two morphemes, *cat* and *-s*. This is of vital importance in our project as we need to understand the difference between different forms of the same word and how they relate to context. Furthermore, when generating text we wish to produce coherent grammar with consistent tense and perspective.

The CLiPS Pattern library has a number of tools for morphology of words. It provides a method of changing a word into its first, second or third person version, pluralisation and finding superlatives.[17]

2.4.2 Syntax

Syntax is the glue that binds words together. It gives us an understanding of the grammatical relationship between words and guides the building of phrases and sentences.

Core to this area of research is *part-of-speech (POS)* analysis, which provides a model for grouping words together correctly, taking into account how words depend on each other. The big success story in this area is The Penn Treebank Tagset, an enormous corpus of annotated POS information [30]. The full tagset is given in Table A.3 in the APPENDIX. This accelerated progress of research in the area, as the paper had expected.

From these POS tags, we are able to create *grammars*, the structural rules of phrases and sentences, and *parsers* for those grammars to be able to extract grammatical structure from unstructured text.

For example, the phrase *John loves Mary* would be represented as in Figure 2.11 if parsed with a grammar based on The Penn Treebank tagset.

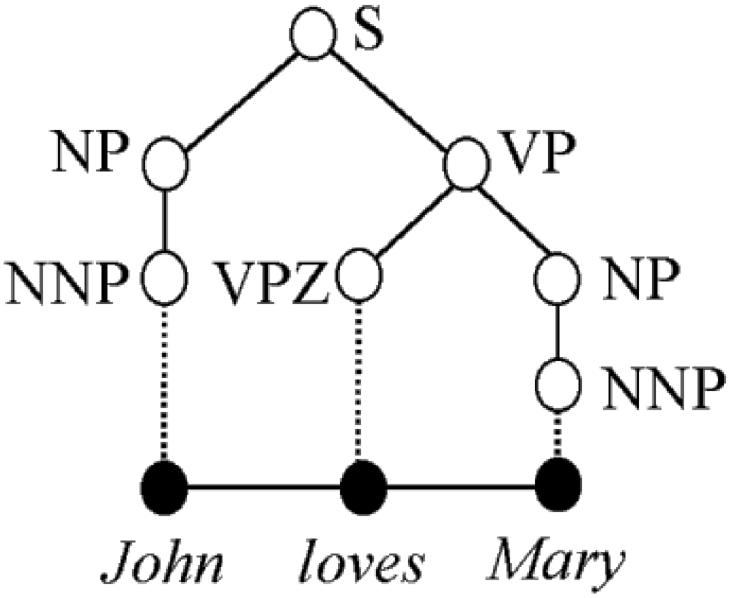

Figure 2.11: Parse tree of *'John loves Mary'*

Python's Natural Language Toolkit (NLTK)[8] is a suite of text processing libraries, corpora and lexical resources that is heavily used in this project, particularly for syntactical purposes. It will allow us to use The Penn Treebank tags as well as produce our own grammar and parser that can be used to parse most poetry. This is a challenge because we cannot expect poetry to follow grammatical rules as strictly as discourse. However, the pay-off will be that we can model the context of the poem, leading to better analysis of semantics and pragmatics of poetry.

2.4.3 Semantics and Pragmatics

Chomsky used the famous example *'Colourless green ideas sleep furiously'* to show that a valid grammatical syntax can be completely nonsensical[11]. This illustrates the importance of the study of semantics, the meaning of words and phrases, as well as pragmatics, the way context affects semantics. Suppose the context around Chomsky's example was a person with old (black-and-white, colourless) ideas of making money (green) that he wants to bring back (was put to sleep, but not peacefully), then this line would make some sense in a poetic way.

This is yet another major challenge for poetry generation. Poems are concise but have many layers of meaning that are subtle even to human readers and that only the best of poets can fully control. The different categories of poems each have their own pragmatics and there is always a fundamental message and purpose supporting these layers meaning. Veale argues that 'poetic licence' is not a licence but a contract that allows a speaker to take liberties in language in exchange for real insight[38]. We must be careful not to abuse this by slewing unrelated words together and expecting the reader to do the work of finding meaning and context.

Various methods of understanding semantics in natural language have been proposed. A very popular one is First Order Logic. In particular, Hans Kamp proposes Discourse Representation Theory (DRT) as a way of modelling language in such a way. We use this theory in this project and will go into more depth in the next section.

Johan Bos and his team have begun the Groningen Meaning Bank (GMB) project[5], a large semantically annotated corpus in lieu of The Penn Treebank, in the attempt to bring the same success and acceleration to this sub-field of research. They use DRT as the backbone to an assembly of third-party tools to annotate semantics, as can be seen in Figure 2.12.

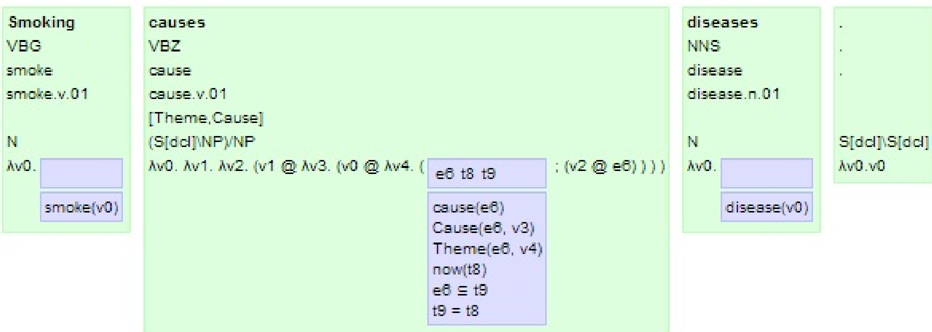

Figure 2.12: Semantic structure of the sentence *Smoking causes diseases.*

However, the GMB project is still in early stages and has only annotated open license news articles up until now, which is not a suitable corpus for poetry generation. However, we will support the coordination of third-party tools and will consider using the following to boost the semantic and pragmatic skill of our poetry generator, particularly with regards to the problem of producing vivid imagery and pragmatic parsing.

- *WordNet*: a lexical database that provides hierarchical, conceptual-semantic and lexical relations of 155,287 English words.[32]
- *VerbNet*: a lexical database that groups verbs by semantic and syntactic linking behaviour.[36]
- *FrameNet*: a lexicon with framing semantics to define and constrain the building of phrases around individual words.[3]
- *ACE*: a classification of names, places and other proper nouns for Named Entity Recognition.[19]
- *PropBank*: an annotated corpus a million words defining and providing argument role labels for verbs[25]
- *NomBank*: similar to PropBank, but for nouns instead of verbs.[31]
- *Wordnik*: a multi-faceted dictionary and semantic language database collected from a variety of sources[1]
- *Metaphor Magnet*: a web application that maps commonplace metaphors in everyday texts[39]
- *Oxford Collocations Dictionary*: a source of word pairings and phrases that occur with greater than chance probability[16]
- *University of South Florida Free Association Norms*, a database of various types of word associations[33]
- *LinguaTools DISCO*: a tool to derive semantic similarity between words based on statistical analysis of large text collections[26]
- *ConceptNet 5*: a semantic general knowledge network[28]
- *Written Sound*: a dictionary of onomatopoeia to their meanings and associated objects[2]

Not all of these tools will be used in this project due to the limited scope. The selected few will be justified in the implementation section with regards to meeting the objectives of this paper. Others will be suggested for future extensions to this project.

2.4.4 Discourse Representation Theory

Discourse Representation Theory (DRT)[24] is a framework for investigating semantics of natural language. DRT is becoming the accepted theory of meaning representation[5][9] and is fundamental to the task of semantic modelling of natural language.

Abstract mental representations of DRT are Discourse Representation Structures (DRSs). It is designed to be able to combine meaning across sentences and cope with anaphora (e.g. pronouns in place of nouns).

Using Kamp's example, if we take the sentence *A farmer owns a donkey* and convert it into a DRS, we get the following notation:
{[x,y: farmer(x), donkey(y), own(x,y)]}

If we then say *He beats it.*, it will produce:
{[x,y,z,w: farmer(x), donkey(y), own(x,y), PRO(z), PRO(w), beat(z,w)]}.

We can then use anaphora resolution on this DRS to produce:
{[x,y: farmer(x), donkey(y), own(x,y), beat(x,y)]}

We can see that this is similar to the notation used by Manurung in MCGONNAGAL, described in section 2.2.4.

This method has evolved over the years to take tense and aspect into account, providing temporal reasoning in natural language sentences. Accuracy of anaphora and presupposition (e.g. saying 'animal' instead of 'cat') resolution has improved with the use of the third party tools mentioned earlier in combination with the ideas of Blackburn and Bos[9].

Extending this example, we may wish to model the sentence *Every farmer who owns a donkey beats it.*. DRT provides an elegant solution for this using first order logic style 'for all':
{[x][y][farmer(x), donkey(y), own(x,y) ->beat(x,y)]}

This allows us to provide background knowledge to the system and make inferences on it. As a result of its usefulness in many applications, NLTK has included DRS manipulation and anaphora resolution into its core 'sem' package.

In poetry, like any text, we can use this to analyse the semantics and pragmatics of the storyline. Furthermore, we hypothesis that it can be used to model each poetry type since they each have their own forms and purposes, and provide coherency to the story when generating text.

2.4.5 Anaphora Resolution and Presupposition Projection

Need some stuff here. Probably remove the DRT section as well.

2.4.6 Natural Language Generation

Natural Language Generation is the term for putting some non-linguistic form of content into understandable text in a human language. Reiter and Dale give the framework[35] illustrated in figure 2.13 for the process of generating natural language.

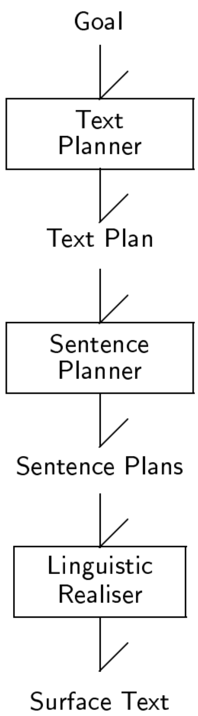

Figure 2.13: Reiter and Dale Natural Language Generation process

A similar model was propsed by Bateman and Zock[6], which includes four stages:

1. *Macro Planning:* Overall content of the text is structured.
2. *Micro Planning:* Specific words and expressions are decided.
3. *Surface Realisation:* Grammatical constructs and order are selected.
4. *Physical Presentation:* Final articulated text is presented.

These are theories upon which natural language generation tools, such as simpleNLG[21], have been designed. There are others, such as grammar generation implemented by NLTK (similar to the one used in MCGONNAGAL) or the constraint programming technique used by Toivanen et al. explained in section 2.2.2, but we find that this process is not granular enough, jumping from the goal to the surface text without enough consideration for the individual words used.

The inputs into this system can vary. One popular problem is raw numerical data into a textual summary. In our case, however, we look at symbolic representations of content in the form of a DRS. Basile and Bos argue that DRSs are appropriate for all stages of the Bateman and Zock example[4]. Gardent and Kow have also proposed a symbolic approach to the Surface Realisation phase, working from first-order logic as the input form[20].

In this paper, we will look at using DRSs to guide the content, i.e. the Macro Planning stage. However, we feel it is restrictive to use it for Micro Planning, since it gives no scope the use of symbolic imagery, or Surface Realisation, as it forces grammatical perfection.

Chapter 3

Poem Analysis

The first phase of implementation is to write a suite of algorithms to analyse a single poem in terms of all the features mentioned in section 2.1.2. The aim is to analyse a large collection of poems in such depth to learn the usage patterns of poetic features for that collection of poems. This is in line with the ultimate aim of avoiding hard-coded rules for different types of poems.

The algorithms will cover the detection of the use of:

- Rhyme and internal rhyme
- Rhythm, including meter and syllable count
- Alliteration, assonance, consonance, onomatopoeia and other sound devices
- Structure, tense, point of view and repetition

Other algorithms will attempt understand the context of the poem to extract:

- Characters
- Objects
- Locations
- Descriptions
- Relationships
- Actions
- Symbolism including metaphors, similes and personification

The output of this phase is a full analysis of a single poem. In lieu of this, we will walk through the implementations of each of these algorithms using the poems in Figures 3.1 and 3.2 as case studies.

There once was a big brown cat
That liked to eat a lot of mice
He got all round and fat
Because they tasted so nice

Figure 3.1: A rhyming quatrain often used in teaching poetry

The limerick packs laughs anatomical
Into space that is quite economical.
But the good ones I've seen,
So seldom are clean,
And the clean ones so seldom are comical.

Figure 3.2: A limerick about limericks

[['DH', 'EH1', 'R']]
[['W', 'AH1', 'N', 'S']]
[['W', 'AA1', 'Z'], ['W', 'AH1', 'Z'], ['W', 'AH0', 'Z'], ['W', 'AO1', 'Z']]
[['AH0'], ['EY1']]
[['B', 'IH1', 'G']]
[['B', 'R', 'AW1', 'N']]
[['K', 'AE1', 'T']]

Figure 3.3: The different ways of pronouncing the first line of the cat poem

3.1 Obtaining Phonetic Structure

When poets choose words they take the sound of the words when spoken out loud into account as well as their (literal or symbolic) meaning. As explained in section 2.4.1, we can use CMU Pronounciation Dictionary (CMUPD) to get around the difficulty of determining phonetic structure by spelling. A word in the CMUPD is mapped to a list of different pronunciations for the same word. Each pronunciation is a list of phonemes that make up that particular pronunciation of that word, including indication of emphasis on the syllables as explained in section 2.4.1.

We want to convert the poems into their phoneme lists for use by the detection algorithms. This needs to be done word by word, so we first need to *tokenise* the sentence. Tokenisation involves splitting each sentence into a list of its basic components; words and punctuation. Once we have done that, we can simply iterate through the list and run each word through the CMUPD. Some words have multiple pronunciations, so we consider each possible permutation of pronouncing each line of the poem. Each of the pronunciations of the first line of Figure 3.1 is shown in Figure 3.3.

Unfortunately, the CMU Pronunciation Dictionary only has about 133,000 words. This means that occasionally we will fail to translate to the phonetic structure, particularly in Shakespearean poems. We get around this by temporarily converting the word into its closest match that exists in the dictionary and returning the phonetic structure in its place.

Python difflib provides a function to find the closest matches of a word to a list of words, based on the Ratcliff/Obershelp pattern recognition algorithm. The complexity of this algorithm is quadratic in the average case and cubic in the worst case. However, difflib's implementation avoids this using a 'junk' heuristic and hashing so the worst case becomes quadratic and the best case linear. The behaviour is based on how many sequences have in common. Since we are only dealing with single words at a time, we have a higher chance of the best case. On average it takes X MILLISECONDS to look up a 4-letter word and Y MILLISECONDS to look up an 8-letter word, which is easily fast enough given that speed is not a priority for this phase since it is a pre-processing phase, not a time-sensitive one.

We could use a variety of other techniques to get around this problem other than string matching:

- Break many syllable words into likely part-words, e.g. 'thrift' and 'less' instead of 'thriftless'
- Try all combinations of stress and syllables
- Train a finite-state transducer model as in [**dobrivsek2010towards**]

The first option only works in a limited number of situations that are reasonably handled by the string matching solution. For example, 'thriftless' would become 'shiftless', which has an identical phonetic structure. The second option can result in poor performance and more false negatives or false positives than would be worth the added processing.

The final option is the most viable and would be used if the generation phase depended on perfect readings of the poems, as the stochastic n-gram methodology of RKCP would. However, as we only use this as an approximation for abstraction in later phases, we can afford to use a more naive implementation.

3.2 Rhyme

We want to detect end-line and internal rhyme, as described in section 2.1.2.1. Along with detecting it, we want to be able to build a normalised rhyme scheme representation for easy analysis in the abstraction phase.

First we collect the phoneme set of the words that we wish to analyse for rhyme. For end-line, we collect the last word of each line of the poem. For internal rhyme, we collect the all the words in a particular stanza.

Once we have our list, we can run it through Algorithm BLAH.

1 for each word in the list: 2 for each pronuciation of the word: 3 for each phoneme in the pronunciation: 4 if this phoneme is stressed: 5 get all the rhyme phoneme pattern from this phoneme onwards in this pronunciation 6 if we have seen this pattern before: 7 assign it corresponding rhyme token 8 else: 9 map this pattern to a new rhyme token 10 11 append this token to a set of possible tokens for this word 12 append the set possible tokens for this word to the unzipped rhyme scheme 13 build all possible permutations of rhyme scheme from they unzipped rhyme scheme 14 normalise the rhyme schemes

There are a few tricks to this algorithm, namely obtaining the rhyme phoneme pattern on line 5 and the process of building and normalising the rhyme scheme in lines 11 to 14.

3.2.1 Obtaining the Rhyme Phoneme Pattern

Two words rhyme when:

- the last phoneme of each word match
- the vowel sounds after and including the first stressed syllable match **in order**

The algorithm finds the first stressed syllable in line 4. At that point we only need to iterate through the rest of the pronunciation and find the vowel sounds and the last phoneme check that the last phonemes. Putting them together in order gives the unique rhyme phoneme pattern.

For example strategy and tragedy, kite and height - colourful diagrams!

3.2.2 Building and Normalising the Rhyme Scheme

Each unique rhyme phoneme pattern is represented by a single capital letter starting with 'A'. This is matches with the standard convention for rhyme schemes used in theory.

The complication comes in the various ways of pronouncing a single word. Therefore, we create a list for every possible rhyme pattern for a each word. The list of these lists is what the algorithm refers to as the 'unzipped' rhyme scheme on line 12 and 13. Figure BLAH shows the unzipped rhyme scheme for the limerick in Figure BLAH. Not that the word 'anatomical' has two pronunciations that affect the rhyme pattern, according to the CMUPD.

FIGURE OF THE UNZIPPED RHYME SCHEME

We then build every possible permutation of this list, which then gives us a list of the possible rhyme schemes for these given words. We leave this list as it is and do not make a claim for any rhyme scheme to be more likely than any other at this stage.

3.3 Rhythm

We attempt to recognise all three types of rhythm described in 2.1.2.2.

3.3.1 Detecting Syllabic Rhythm

The number of syllables in a word is equal to the number of vowel phonemes it has. We know that all vowel phonemes have a stress marker appended to it so we can just count the number of stress markers in the word.

However, Syllabic Rhythm is done on a line-by-line basis. Therefore we tokenise each line and aggregate the vowel phonemes across all words in the line to get the number of syllables for that line.

If we take one permutation of the line in Figure 3.3, we can see that each word has one vowel phoneme, so each word in that sentence is monosyllabic. Therefore the number of syllables in the line is 7.

The full syllabic rhythm for the poems in Figure 3.1 and 3.2 are 7-8-6-7 and 11-6-5-11-11 respectively. We represent these as a list of integers. We create a separate list for each stanza.

3.3.2 Detecting Quantitative and Accentual Rhythm

While theory counts rhythm in terms of metre and feet, as described in 2.1.2.2, some poems might have rhythm without following one of these pre-defined popular styles. Instead of searching for them, we will extract the pattern of stressed and unstressed syllables for each line without making a claim on how it matches to theory yet.

We choose to do this mainly because there could be multiple possibilities for the stress pattern of a line due to variations in emphasis for particular words, e.g. '**ob**ject' and 'ob**ject**'. The variability of possible stress patterns is compounded by some limitations of the CMUPD:

- Some words are restricted to a single stress pattern even though it does not lose or change the meaning of the word if it was emphasised differently. For example, '**quan**tity', 'quan**ti**ty' and 'quanti**ty**' are all valid, but the CMUPD only recognises the first because the other two are unusual in normal speech (but not for poetry).

- Similarly, the CMUPD fixes monosyllabic words to one of stressed or unstressed as in Figure 3.3. However, they can be either so we need to allow for both possibilities.

- Stresses on some words have changed over time. For example **prov**ed was pronounced prov**ed** in Shakespeare's era

- Though not necessarily a limitation, we do not give importance to light or secondary stress in a word (the '2' stress marker). We therefore assume it could be either '1' or '0'.

- Poetic license often allows words to be pronounced with an entire extra syllable. For example *de-served* could be pronounced as *de-ser-ved*.

We account for these by increasing the number of possible readings. However, this will lead to a lot of possibilities. For example the line in Figure 3.3 will give us any combination of '1's and '0's since they are all monosyllabic and by the second point above we want to allow for either.

We narrow this down initially by taking away any occurrences of the same stress three or more times in a row in the same line, because no one will ever read a poem in this way. Like with rhyme, we do not make a claim for any stress pattern to be more likely than any other at this stage.

3.4 Sound Devices

We aim to detect each of the sound devices described in 2.1.2.3.

3.4.1 Detecting Consecutive Sounds

RETHINK THIS IMPLEMENTATION

For assonance, consonance and alliteration, we only care about whether or not they are being used, not necessarily which sounds are repeated or where in the poem it occurs.

3.4.2 Onomatopoeia

Some onomatopoeia are recognised in traditional dictionaries, while some are not. There are also many variations of the same sound, e.g. 'Aah' versus 'Aaah'. New onomatopoeia can be created at any point depending on the sound the writer wants to portray. This range of variation and lack of consistency makes it difficult to detect all occurrences of onomatopoeia.

However, we can use the WrittenSound.com dictionary of onomatopoeia to recognise the most common ones. We can check each word in the poem for existence in the dictionary or for a very similar words to those in the dictionary. To do this we can use the Levenshtein distance, a measure of the number of changes required to turn one string into another. If the Levenshtien distance between a word and an entry in the dictionary is less than or equal to 1, not including extensions to the head or tail of the onomatopoeia, then we consider it to be onomatopoeic.

Furthermore, WrittenSound has mappings between the onomatopoeia and the type of sound they represent. For example, *'ding'* represents a 'hard hit', probably of a 'metal' object. This can be useful somehow.

3.5 Form and Structure

We implement some basic detectors for the form and structure of the poem. This does not include analysis of words and phrases themselves as that will be done in the abstraction phase, as described in section BLAH.

The algorithms for detecting the features of structure described in section 2.1.2.4 are fairly simple:

- **Number of stanzas** = Number of blank lines + 1
- **Number of lines per stanza** = Number of new line characters per stanza
- **Number of repeated lines** = Length of the list of lines - Size of the set of lines (sets only hold unique elements)
- **Number of distinct sentences** = The number of sentences returned when parsed using the CLiPS *parsetree* function. More lines than sentences indicates enjambment.

The poem in Figure 3.1 has one stanza, four lines, no repeated lines and two distinct sentences, indicating enjambment.

We can also determine the point of view of the speaker, i.e. whether it is in first or third person. We look for the word 'I' anywhere in the poem and as long as it is outside speech marks, we say that the entire poem is in first person. Otherwise we default to third person. Figure 3.1 is written in third person but Figure 3.2 is in first person.

The tense of each line can be found by analysing the verb in that sentence. CLiPS pattern library provides a very easy method of doing this using their 'tenses' algorithm. It also gives us the *aspect*, e.g. perfect, progressive. However, tests found this to be less accurate so we will settle for just the tense for this stage.

We record the tense of each line as well as the overall tense of the poem. EXAMPLES

3.6 Context and Pragmatics

Here we describe the approach to the difficult challenge of determining the context of the poem in terms of its characters and their relations, descriptions etc. as described in section 2.1.2.6. The aim is to build a representation of the characters much like a human reader would in their mind. This will then be compared in the abstraction section to find a correlation between these structures and types of poetry.

3.6.1 ConceptNet Relations

ConceptNet is a semantic network of '*general knowledge*'. Each node in the network is a natural language word or phrase called a *concept*. Edges in the network represent a relationship between two concepts. These relationships come in various types including, but not limited to:

MadeOf

What is it made of?
E.g. tree - MadeOf → wood

IsA

What kind of thing is it?
E.g. banana - IsA → fruit

AtLocation

Where would you find it?
E.g. priest - AtLocation → church

CapableOf
What can it do?
bird - CapableOf → sing

Desires
What does it want?
banker - Desires → his loan to be repay

HasA
What does it have in its possession?
old person - HasA → white hair

HasProperty
What properties does it have?
doctor - HasProperty → smart

PartOf
What is it part of?
player - PartOf → team

ReceivesAction
What can you do to it?
book - ReceivesAction → read

CreatedBy
How do you bring it into existence?
sound - CreatedBy → vibration

UsedFor
What do you use it for?
guitar - UsedFor → make music

MotivatedByGoal
Why would you do it?
learn - MotivatedByGoal → knowledge

Each of these relationships also has its inverse, e.g. NotIsA. We can use these relationships to build a cohesive story about particular characters during the generation phase as well as a few of our own:

Believes

What does the character perceive to be true?

Little Red Riding Hood - Believes → Wolf is her grandma

SendMessage

What did the character say or write?

Little Red Riding Hood - SendMessage → What a big mouth you have

ReceiveMessage

What did the character hear or read?

Little Red Riding Hood - ReceiveMessage → The better to eat you with!

TakesAction

What did the character do?

Wolf - TakesAction → Swallow Little Red Riding Hood

Named

What is the name of the character?

Gotham Vigilante - Named → Batman

We also have the corresponding inverses for these relations.

In keeping with the theme of this paper we cannot simply hard-code or randomise the choice of relations in building a poem. Therefore, we need to be able to analyse these relationships in existing poems during the abstraction phase to find correlations. For example, we may find that descriptive poems have a high number of *HasProperty* relations and very few characters.

To do this, we need to be able to extract relationships between concepts in a particular poem in lieu of ConceptNet relations. For example:

Give the cat example

3.6.2 Semantic Labelling using Noah's ARK

It would be very difficult to determine these relationships from a syntactical parse alone. This is due to the complex nature of the English language, in particular verb usage. For example, the phrase *tasted so nice* is a description rather than an action because the word *tasted* in this case is being used as a linking verb, where it is usually an action verb.

These complexities are further compounded by the fact that we cannot rely on correct grammar in poems. We therefore need a *semantic* parse. Noah's ARK is an informal research group run by Noah Smith at Carnegie Mellon University. They provide online API access to two tools for linguistic structure analysis, *SEMAFOR* and *TurboParser*. Both of these tools in conjunction are satisfactory for our aim to extract ConceptNet relations from natural language.

3.6.2.1 FrameNet Semantic Role Labelling using SEMAFOR

Semantic Role Labelling (SRL) is best described with an example. Take the sentence *"The shopkeeper told the hoodlum to go away"*, we wish to recognise the verb *'to tell'* as the target, *'the shopkeeper'* as the speaker, *'go away'* as the message and *'the hoodlum'* as the addressee. This can be seen clearly in Figure BLAH.

This is independent of the syntactic structure of the sentence and does require grammatical correctness. The SRL for the *'Yoda-speak'* equivalent will remain the same, as shown in Figure BLAH.

These labels are retrieved through the SEMAFOR tool, which was trained on FrameNet data to determine the frame-semantic structure of the text. FrameNet is a lexical database of *semantic frames*, which describe the meaning of a word based on the words that typically participate with it, known as frame elements. It is based on the Frame Semantics Theory, introduced by Charles J.Fillmore et al.

We can use these to derive the ConceptNet relations by looking for the occurrence of frames, and elements thereof, that correspond to the ConceptNet relation. The manually chosen list of the ConceptNet relations with the frames and corresponding elements that translate directly into that relation is given in the APPENDIX.

However, each list may not be exhaustive for its corresponding ConceptNet relation and there are some relations that will not be picked up by this method.

3.6.2.2 Semantic Dependency Relations using TurboParser

The Stanford Dependencies is another representation based around the relationships between words. All dependency relations are strictly binary and come in various types depending on the participants (called the *governor* and the *dependent*).

The semantic dependency tree for the previous shopkeeper example can be seen in Figure BLAH.

This representation fills the gaps left by the frame-semantic parse using the following heuristics:

PSEUDO-CODE HEURISTICS HERE

Together, these tools provide a fairly complete coverage of the ConceptNet relations.

3.6.3 Extracting and Binding Relations to Characters

At this point, the derived ConceptNet relations are only a set of abstracted FrameNet frames and matched dependencies. This alone does not give us much more information than if we were to use the frame-semantic parse on its own. The true usefulness of this approach only arises if the relations can be bound to characters of the poem. For the shopkeeper example, we would recognise *the shopkeeper* and *the hoodlum* as objects with SendMessage and ReceiveMessage relations bound to each of them respectively with respect to the *go away* message.

We can represent the desired output as a set of 'hubs', with each character at the centre of the hub as in Figure BLAH. This will also improve our knowledge for the generation phase in that we will know the *number* of characters typical for a poem type, the *number and type* of relations associated with *each* character, as well as allow us to find commonalities in the types of characters themselves.

To reach the desired representation, we execute Algorithm BLAH:

1 for each sentence in the poem: 2 get the dependency relations and frame semantic parse 3 collapse loose leaves of the dependency relations 4 find and create possible characters 5 find candidate ConceptNet relations from the frame-semantic parse 6 for each character: 7 get all associated dependency relations 8 for each associated

Field Number	Field Name	Description
1	ID	Token counter, starting at 1 for each new sentence.
2	FORM	Word text form or punctuation symbol
3	LEMMA	Lemma or stem of FORM
4	CPOSTAG	Coarse-grained part-of-speech tag
5	POSTAG	Fine-grained part-of-speech tag
6	FEATS	Unordered set of syntactic and/or morphological features
7	HEAD	ID of the parent of the current token ('0' if root)
8	DEPREL	Dependency relation to the HEAD
9	PHEAD	Projective head of current token
10	PDEPREL	Dependency relation to the PHEAD

Table 3.1: The CoNLL data format output by TurboParser.

dependency relation: 9 if the dependent is involved in a candidate relation from the frame-semantic parse: 10 bind the relation to the current character and continue 11 else: 12 use the heuristics for dependency relations to find possible ConceptNet relations 13 bind any that are found to the current character and continue

Here we describe each step in detail.

3.6.3.1 Obtaining the Semantic Dependency Relations and Frame-Semantic Parse from Noah's ARK

Noah's ARK provides parse data from both SEMAFOR and TurboParser in a JSON format that can be accessed by sending a request to their demo API. However, there is a cap of 20 seconds processing time for any request so we only send one sentence at a time to be safe. Some basic experimentation has shown that this does not hinder accuracy.

SEMAFOR returns the Frame-Semantic parse in JSON format, an example of which can be seen in Figure BLAH. We leave it in this format because we only need to access this data once when finding the candidate relations on line 5 of Algorithm BLAH.

TurboParser returns the Semantic Dependency Parse in CoNLL data format, whose structure can be seen in Table 3.1. An example of the output can be seen in Figure BLAH.

We parse this into a simple dependencies dictionary format shown in Table 3.2 for easy

ID	FORM	CPOSTAG	POSTAG	HEAD	DEPREL
1	There	EX	EX	3	expl
2	once	RB	RB	3	advmod
3	was	VB	VBD	0	null
4	a	DT	DT	7	det
5	big	JJ	JJ	7	amod
6	brown	JJ	JJ	7	amod
7	cat	NN	NN	3	nsubj
8	who	WP	WP	9	nsubj
9	liked.	VB	VBD	7	rcmod
10	to	TO	TO	11	aux
11	eat	VB	VB	9	xcomp
12	a	DT	DT	13	det
13	lot	NN	NN	11	dobj
14	of	IN	IN	13	prep
15	mice	NN	NNS	14	pobj
16	.	.	.	3	punct

Table 3.2: The dependencies dictionary data structure for the first sentence of the cat poem.

access and processing downstream.

TurboParser and SEMAFOR are resource heavy (SEMAFOR requires minimum 8GB RAM), so it makes sense to run it on a server. Since the output from Noah's ARK is exactly what we want and the request is fairly quick (typically BLAH milliseconds for simple sentences, BLAH milliseconds for complex ones) and speed is not a priority for this project, there is no need to run our own copy on our own server.

3.6.3.2 Collapsing Loose Leaves of the Semantic Dependency Relations

In a lot of cases, the character or relation that we look for is a phrase, not just a word. For example in Figure BLAH the character should be *a lot of mice* rather than just *mice*. In fact if we skipped this step, we would get two characters; *a lot* and *mice*, which is obviously wrong.

Another example is the phrase *tasted so nice*. If we did not collapse the tree, we would have that *they* took the action of *tasted* and can be described as *nice*, both of which are wrong again.

The non-demo version of TurboParser provides different styles of dependency representation. This includes collapsed dependencies but not exactly in the way we want it done. First, we manually select the dependencies that we want to collapse (APPENDIX).

By default, the parent of the dependency keeps all its attributes unchanged except for merging its form with that of the leaf, e.g. *of* and *mice* becomes *of mice*. However, there are some cases where we wish to retain the part-of-speech (POS) tag of the leaf and overwrite the parent's part of speech tag. For example, *of mice* should retain the 'NNS' POS tag of *mice* rather than keep the 'PRP' POS tag of *of*. The full list of these conditions are in the APPENDIX.

- If the leaf is an adjective, the parent is a verb and the dependency relation is *'dep'* then we retain the adjective POS. These conditions usually imply a *linking verb* generally used to describe a property, not an action. E.g. 'tasted so nice' is an adjectival phrase despite the use of a verb.
- The *'acomp'* dependency relation is also evidence of a linking verb.
- The *'pobj'* and *'prep'* dependency relations are evidence of the preposition example described above.

Then we follow Algorithm 2: 1 Get a list of all the leaves in the graph. 2 For each leaf in the reverse of this list: 3 If the dependency relation of this leaf (i.e. from its parent) is collapsable: 4 Get the parent 5 Merge the form of the parent and the leaf 6 If necessary, the parent retains the part-of-speech tag of the leaf 7 The leaf is destroyed and the parent becomes the leaf. 8 Loop back to line 3.

Figure BLAH shows a dry run of this algorithm on the sentence *"There once was a big brown cat who liked to eat a lot of mice."*

3.6.3.3 Finding and Creating Characters

A naive method for finding characters would be to look for nouns and pronouns to recognise characters in a poem. This would actually be sufficient, but we can do better. Notice in the poem in Figure BLAH, the cat character is represented by the words *'a cat'* and *'he'*. Similarly, *'a lot of mice'* and *'they'* represent the same character.

In Computational Linguistics, this problem is called **Anaphora Resolution**. It is an ongoing research area and I will present a new potential solution in section 3.6.4.

Part of the solution is gathering some basic semantic information about the character. We would like to determine whether the character:

1. is plural or singular.
2. is male, female, neutral or unknown.
3. the character is an animate/living object, an inanimate physical object or a non-object.

The first point is easiest:

- If it is a noun and the part-of-speech tag ends in an 'S', it is plural.
- If it is a pronoun, check for membership in the manually built list of plural pronouns (e.g. 'they', 'them').

The pronoun case for the second point is similar; we check for membership in the manually built list of male pronouns (e.g. 'he', 'him') and the separate list of female pronouns (e.g. 'she', 'her').

The noun case is trickier. First we need to find the *synset* of the noun. Synsets are sets of cognitive synonyms, e.g. *car* and *automobile* are in the same synset, but not *cable car*. Once we have this, we can use the hypernym relationships between synsets.

A hypernym of a synset is a **type-of** relation. For example, the synset *mammal* is a hypernym of *cat* because cat is a type of mammal. Similarly, *animal* is also a hypernym of cat, as well as being a hypernym of mammal. The full hypernym tree for *cat* can be seen in Figure BLAH.

The *direct hypernym* of a synset is the synset directly above it in the hypernym tree; *feline* in the case of the cat. The *inherited hypernym* of a word refers to any word that appears in its hypernym tree. WordNet, a lexical database of the cognitive relations between words provides all of these resources.

Now, finally, we can check for whether a noun is male or female by looking for the existence of particular synsets that imply a gender. For example, the *'female'* synset would be an inherited hypernym of the synset *'cow'*. Therefore we know that cows are female. Similarly, *'maharaja'* has the hypernym *'prince'*, which we know to be male.

We can extend this practice to the final point of determining the animation of the character by checking if the *'living thing'* and *'physical object'* synset is an inherited hypernym of the synset of the noun concerned.

3.6.3.4 Extracting Candidate ConceptNet Relations from the Frame-Semantic Parse

As mentioned in section 3.6.2.1, we have a manually built map between FrameNet frames and ConceptNet relations. Given this, we can carry out Algorithm BLAH.

1 for each frame found in the frame-semantic parse: 2 look up the target of the frame in the map 3 if it can help build a ConceptNet relation: 4 retrieve the frame elements 5 replace the frame elements with the corresponding text from the poem 6 if we cannot find all the elements we are looking for, we leave it blank 7 add mapping from the text of the frame target to the newly built relation

Let us take the shopkeeper example from earlier. If it exists, we then look for the specific frame elements that will allow us to build the ConceptNet relation.

GO THROUGH THE SHOPKEEPER EXAMPLE

3.6.3.5 Obtaining the Associated Dependency Relations for a Particular Character

This step breaks up the semantic dependency tree and flatten it into the character-centric hubs as shown in Figure BLAH.

First, we want to identify the character in the dependency tree. All of the relations going out from it are naturally related dependencies of the character, so they get added to the list, see Figure BLAH.

The single relation coming into this character is also a related dependency, so we added that to the hub and reverse the direction of the branch, see Figure BLAH.

Now we have a tree with the character as the root. The next step is to flatten it so that everything is related to the character. We do this by recursively converting all the grandchildren of the character root node to a direct child node until there are no more grandchildren. This process is illustrated in Figure BLAH.

We repeat this process for each character starting from the leftmost character in the sentence. However, we quickly notice that there will be duplication; the same dependency will be associated to more than one character. To avoid this, we can prune these hubs by working backwards through the list of characters (i.e. starting from the rightmost in the sentence) and removing duplications in earlier nodes, as shown in Figure BLAH.

This works because the head of the last character will be lower down in the tree than the head of any other character. This has the effect of removing the duplicated dependencies such that the last one standing is only associated to the one it was closest to in the original tree.

3.6.3.6 Binding ConceptNet Relations to Characters

Now that we have our hub data structure for all characters, we now need to look at each branch and of each hub and decide if it can be converted into a ConceptNet relation.

First, we check if the text of the child maps to one of the candidate relations we extracted from the frame-semantic parse in section 3.6.3.4. If it does, then we accept that and move on to the next child without checking the dependency relation since it is likely to be the more accurate. Sometimes the dependant of the relation will be blank because we could not find the right frame element as explained earlier. In this case, we just assume that this is a relation to the next character in the characters list. This is a fair assumption given the limited number of characters per sentence, the general forward reference structure of sentences and the fact that most of the relations we look to build are between characters.

If there is no candidate relation for this child, we then look at the type of the dependency between it and the character. We use the heuristics described in section 3.6.2.2 to convert it into a ConceptNet relation.

If we cannot find a relation through either of the above methods then it is likely there isn't one, so we remove it from the hub.

In all of those steps, we need to look out for negative adverbs such as 'not', 'seldom', 'rarely' etc. which we use to negate the ConceptNet relation found. We also need to look out for antonyms in the target word in the frame-semantic parse. For example, the *Experiencer-focus* frame helps us find *Desires* and *NotDesires* ConceptNet relations depending on whether the target word is synonymous with 'love' or with 'hate', for example.

We can see this final step being applied to our ongoing example in Figure BLAH.

3.6.4 Anaphora Resolution

We can now clearly see the anaphora problem as described in section 3.6.3.3. We must match up the *'a cat'* character and *'he'* character, and the *'they'* character with the *'a lot of mice'* character.

In this particular example, we can simply match the plurals and singulars together. However, we may not always be so lucky. We may also run into a situation where the cat could be referred to as *'the feline'* instead of *'he'*.

Section 2.4.5 gives an overview of the state of the art solutions. Our solution is also pretty darn simple at the moment, so I am going to wait to see if I can do some of the more complicated things and then write this section up.

3.6.5 Peeks at Future Development of Pragmatic Language Analysis

If I get time...

3.7 Symbolism and Imagery

This particular poetic technique can be very subtle and difficult to detect. Of the ones listed in section 2.1.2.5, we are able to recognise the use of two; similes and personification.

3.7.1 Personification

We used the animation state of characters for anaphora resolution. We can take this idea further to detect the use of personification in poetry.

The following relations are symptomatic of a sentient being:

- 'Named' and 'NotNamed'
- 'Desires' and 'NotDesires'
- 'Believes' and 'NotBelieves'
- 'SendMessage' and 'NotSendMessage'
- 'ReceiveMessage' and 'NotReceiveMessage'

Therefore, if a character that is marked as an inanimate physical object has any of these relations, it is likely that the author of the poem was using personification to give the object human-like behaviour.

We can also take advantage of the *Semantic Type* attribute of FrameNet elements. FrameNet marks frame elements with a semantic precondition if they are to be of a certain type, for example only *'Sentient'* semantic types can be the *'Abuser'* and the *Victim* in the *'Abusing'* frame. By looking up these semantic types in FrameNet as we are building the ConceptNet relations, we are able to flag characters as evidence for personification.

3.7.2 Simile

Similes are relatively easy to detect because they often use the phrases 'like a' or 'as a' and 'than'. They also follow certain syntactical patterns in that it is usually a noun followed by a verb or adjective, then one of the aforementioned phrases, followed by a noun or an adjective.

A particular symptom of simile use is that the aforementioned phrases will be involved in a prepositional noun phrase, or 'PNP Chunks' to use the parser terminology.

We could take this further by using ConceptNet btw. As fast as a cheetah + cheetah HasProperty fast -¿ simile.

3.7.3 Metaphor

We need to do something here. Can't just leave it...

Chapter 4

Evaluation Plan

We will employ four methods of evaluating this project.

4.1 Comparison of Analysis Results to Theory

The Analysis and Abstraction phases attempt to discover the common features of any type of poetry. Current theory of these common features have been derived and documented by poets through their own analysis. We wish to investigate whether this system is able to find everything documented that has been documented by poets. It is likely that the system could only find a subset but it will be interesting to see which points were missed. A full comparison will be made and explanation for any missing features will be attempted, as well as possible algorithms to detect them that could have been implemented.

A particularly interesting result would be if the system manages to find a common feature of a type of poetry that poets have not yet found. If even one case of this occurs then this would make a strong case for the ability of computers to understand, interpret and find patterns in natural language. Further investigation would surely be warranted into how much a computer system would be able to find that human readers have overlooked.

Note that this section does not expect the system to infer the effects of any of the poetry features on the reader, just simply detect the use of the features.

4.2 Turing-style Tests

The simplest way to check the quality of poems produced would be to show people a poem either generated by this system or written by a human without telling them and asking them to determine whether the author was man or machine.

However, this is highly dependent on the reader's fluency of English, understanding of poetry and imagination. A poor English speaker with little to no understanding of poetry and with a far reaching imagination could easily be fooled into thinking that a piece of text was written by a human rather than a computer.

Therefore, we propose a survey that asks for these three measures to be told truthfully. It then randomly shows a poem and asks whether it was written by a human or computer. This way, we can get a demographic of those who are fooled and those who are not so that we can see at what level of poetry literacy this system is.

Further, we plan to approach real poets from literary institutions and university departments and ask them to do this survey in person. If they incorrectly believe even one poem to be written by a human and not a computer, then this project is a definite success. For the cases where this does not arise, having an in-person survey will enable us to ask for feedback on what gave us away, which can be used to improve the system and future attempts at poetry generation.

However, Pease and Colton[34] make several arguments that Turing-style tests are not appropriate for poetry generation. We believe that it has its place because ultimately this project is for user consumption as well as research and experimentation. Furthermore, poems attempt to create an emotional connection with the reader, something that cannot be determined other than with a human reader.

Having said that, Colton, Charnley and Pease[14] described the FACE and IDEA Descriptive Models for evaluating computational creativity projects. We believe these models provide a useful evaluation methodology alongside Turing-style tests and so are just as much part of the evaluation of this system as described in Section 4.3 and 4.4. It will also be interesting to evaluate them as an evaluation method as they have only just been proposed.

4.3 FACE Descriptive Model

A full FACE model has four symptoms: examples, concepts, aesthetics and framing information.

- *Examples* will be showcased by the templates generated by the Analysis and Abstraction phase.
- *Concepts* are of the form of the algorithm described for the Generation phase as it takes input from the user or online, the results from the Abstraction phase and several third party libraries to output a poem.
- *Aesthetics* are assessed by running the Analysis phase over the poem again. In fact, this happens several times during the creation of the poem. Any faults are reported back to the next iteration of that poem.
- *Framing Information* is the poem created.

Therefore, we can see that this system should fully abide by the FACE Descriptive Model. We will evaluate the results mathematically as per Colton, Charnley and Pease.

4.4 IDEA Descriptive Model

A full IDEA model has six stages to which the software can reach. We want our software to be in the, fourth or *Discovery stage*.

- *Developmental stage*: this system has a full Abstraction phase to avoid the case that all creative acts undertaken by this system are purely based on inspiring examples. So this system will have surpassed this stage.
- *Fine-tuning stage*: the Abstraction phase only looks for a limited number of overlapping features to provide the template, leading to higher level abstraction. For example, it does not use part-of-speech tags from previous examples or any low level abstractions. We believe the system should be able to surpass this level.

- *Re-invention stage*: the system is able to work off a template provided by the Abstraction phase, but also able to mutate the templates and add or remove restrictions both automatically and guided by the user. Therefore, the creative acts are not restricted only to those that are known and should be able to surpass this stage as well.
- *Discovery stage*: the ability to work off templates derived from Analysis and Abstraction imply that the system is able to generate works that are sufficiently similar to be assessed with current contexts. However, given the flexibility of the mutation and user-guidance ability, it can also produce works that are significantly dissimilar. We believe the system to be able to reach this stage.
- *Disruption and Disorientation stages*: Since templates and constraints on the creative work that are imposed are the results of analysing and abstracting existing works, it is not the case that this system solely produces poetry that is too dissimilar to those known by theory.

Therefore, we can see that this system should reach the desired *Discovery stage* of the IDEA Descriptive Model. We will evaluate the results mathematically as per Colton, Charnley and Pease.

Appendix A

A.1 Dry Run of Generation Phase with Commentary

Input seed: Limerick about a computer that is bored with data and finds poetry fun

1. *There once was (a/an)* _____ *named* ____*[A]*
________________________________*[A]*
_____________________*[B]*
_____________________*[B]*
________________________________*[A]*

This is a limerick template straight out of the generation phase.

2. *There once was a computer named* ____ *[data]*
That was bored with data [data]
It finds poetry fun [fun]
_____________________*[fun]*
_____________________________ *[data]*

This step introduced the computer, the fact that it is bored with data and finds poetry fun. It does not take any structure into account when adding these.

3. *There once was a computer named Zeta*
That was bored to death with data
It finds poetry fun
___________ *sun*
___________________ *beta*

This step filled in the name Zeta since it needed to rhyme with data. It rephrased 'bored' with 'bored to death' to fit the rhythm without losing meaning. Found third line to be complete. Added sun to end of next line since it is an association with fun and follows rhyme scheme. Added beta to end of last line since it rhymes with data (and Zeta)

4. *There once was a computer named Zeta*
That was bored to death with data
It finds poetry fun
Like the summer sun
The revolutionary new system goes into beta

This step added a simile to compare the fun of poetry to the sun since it fit the rhythm structure (alternatively, “like the scorching/setting/sinking sun”, but summer sun has alliteration on ’su’ rather than just ’s’). A phrase was found that ends with beta and was arbitrarily added.

5. *There once was a computer named Zeta*
That was bored to death with data
It finds poetry fun
Like the summer sun
The new system goes into beta

Redundant adjective removed to fit structure in last line. Poem finished.

Note too that we added the input data to the first three lines. Separating them out will make it more coherent, especially in longer poems. Then the associations could be found with surrounding sentences, not just the previous one.

Further rephrasing of lifted sentences, such as the last one, could be beneficial to adding randomness and creativity.

Phoneme	Example	Tranlsation
AA	odd	AA D
AE	at	AE T
AH	hut	HH AH T
AO	ought	AO T
AW	cow	K AW
AY	hide	HH AY D
B	be	B IY
CH	cheese	CH IY Z
D	dee	D IY
DH	thee	DH IY
EH	Ed	EH D
ER	hurt	HH ER T
EY	ate	EY T
F	fee	F IY
G	green	G R IY N
HH	he	HH IY
IH	it	IH T
IY	eat	IY T
JH	gee	JH IY
K	key	K IY
L	lee	L IY
M	me	M IY
N	knee	N IY
NG	ping	P IH NG
OW	oat	OW T
OY	toy	T OY
P	pee	P IY
R	read	R IY D
S	sea	S IY
SH	she	SH IY
T	tea	T IY
TH	theta	TH EY T AH
UH	hood	HH UH D
UW	two	T UW
V	vee	V IY
W	we	W IY
Y	yield	Y IY LD
Z	zee	Z IY
ZH	seizure	S IY ZH ER

Table 1: ARPAbet phoneme set with corresponding examples

Tag	
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Table 2: Penn Treebank Tagset in alphabetical order

A.2 CMU Pronunciation Dictionary ARPABET Phoneme Set

A.3 Penn Treebank Tagset

A.4 FrameNet Frames to ConceptNet Relations

PartOf

Frame	**Character Frame Element**	**Whole Frame Element**
Architectural_part	Part	Whole
Building_subparts	Building_part	Whole
Clothing_parts	Subpart	Clothing
Observable_body_parts	Body_part	Possessor
Part_edge	Part	Whole
Part_inner_outer	Part	Whole
Part_ordered_segments	Part	Whole
Part_orientational	Part	Whole
Part_piece	Piece	Substance
Part_whole	Part	Whole
Shaped_part	Part	Whole
Vehicle_subpart	Part	Whole
Wholes_and_parts	Part	Whole
Inclusion	Part	Whole

CreatedBy

Frame	**Character Frame Element**	**Creator Frame Element**
Creating	Created_Entity	Creator
Cooking_creation	Produced_food	Cook
Intentionally_create	Created_Entity	Creator
Text_creation	Text	Author
Manufacturing	Product	Producer/Factory

MadeOf

Frame	Character Frame Element	Substance Frame Element
Substance	[Noun After]	Substance

Causes

Frame	Character Frame Element	Effect Frame Element
Cause_to_start	Cause	Effect
Causation	Actor	Affected
Causation	Cause	Effect
Condition_Symptom_Relation	Medical_condition	Symptom
Cognitive_connection	Concept_1	Concept_2

Desire

Frame	Character Frame Element	Desirable Frame Element
Desiring	Experiencer	Event
Experiencer_focus	Experiencer	Content

CapableOf

Frame	Character Frame Element	Activity Frame Element
Capability	Entity	Event

UsedFor

Frame	Character Frame Element	Purpose Frame Element
Expend_resource	Resource	Purpose
Using	Instrument	Purpose
Tool_purpose	Tool	Purpose
Ingest_substance	Substance	Purpose
Using_resource	Resource	Purpose
Usefulness	Entity	Purpose

MotivatedByGoal

Frame	Character Frame Element	Goal Frame Element
Purpose	Means	Goal/Value

Has

Frame	Character Frame Element	Possession Frame Element
Possession	Owner	Possession
Have_associated	Topical_entity	Entity
Inclusion	Total	Part
Containers	Container	Contents

Frame	Character Frame Element	Possession Frame Element
Used_up	[Character in view]	Resource
Expend_resource	[Character in view]	Resource
Abandonment	Agent	Theme

NotHas

Named

Frame	Character Frame Element	Name Frame Element
Being_named	Entity	Name

Believes

Frame	Character Frame Element	Belief Frame Element
Awareness	Cognizer	Content
Certainty	Cognizer	Content
Religious_belief	Believer	Content
Trust	Cognizer	Information
Opinion	Cognizer	Opinion

SendMessage and ReceiveMessage

Frame	Speaker Character Frame Element	Message Frame Element	Addresse
Communication	Communicator	Message/Topic	Addressee
Telling	Speaker	Message	Addressee
Request	Speaker	Message	Addressee
Speak_on_topic	Speaker	Topic	Audience
Statement	Speaker	Message/Topic	Addressee
Prevarication	Speaker	Topic	Addressee
Reporting	Informer	Behaviour/Wrongdoer	Authoritie
Text_creation	Author	Text	Addressee
Chatting	Interlocutor_1	Topic	Interlocuto

A.5 List of collapsable dependencies

- acomp
- advmod
- aux
- cop
- dep
- det
- measure
- neg
- nn
- num
- number
- preconj
- predet
- prep
- pobj
- quantmod

References

[1] URL: http://www.wordnik.com/about.

[2] URL: http://www.writtensound.com/about.php.

[3] Collin F Baker, Charles J Fillmore, and John B Lowe. "The berkeley framenet project". In: *Proceedings of the 17th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics. 1998, pp. 86–90.

[4] Valerio Basile and Johan Bos. "Towards generating text from discourse representation structures". In: *Proceedings of the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics. 2011, pp. 145–150.

[5] Valerio Basile et al. "Developing a large semantically annotated corpus". In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*. Istanbul, Turkey, 2012, pp. 3196–3200.

[6] John Bateman and Michael Zock. *Oxford Handbook of Computational Linguistics*. Oxford University Press, 2003. Chap. 15 Natural Language Generation.

[7] Kim Binsted and Graeme Ritchie. "Computational rules for generating punning riddles". In: *Humor* 10.1 (1997), pp. 25–76.

[8] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. O'reilly, 2009.

[9] Patrick Blackburn and Johan Bos. "Computational semantics". In: *THEORIA. An International Journal for Theory, History and Foundations of Science* 18.1 (2008).

[10] William Chamberlain and Thomas Etter. “The Policeman’s Beard is Half-Constructed: Computer Prose and Poetry”. In: *Warner Software/Books, New York* (1984).

[11] Noam Chomsky. *Syntactic structures*. Walter de Gruyter, 2002.

[12] Simon Colton. “The painting fool: Stories from building an automated painter”. In: *Computers and creativity*. Springer, 2012, pp. 3–38.

[13] Simon Colton, Jacob Goodwin, and Tony Veale. “Full face poetry generation”. In: *Proceedings of the Third International Conference on Computational Creativity*. 2012, pp. 95–102.

[14] Simon Colton, A Pease, and J Charnley. “Computational creativity theory: The FACE and IDEA descriptive models”. In: *Proceedings of the Second International Conference on Computational Creativity*. 2011.

[15] Simon Colton and Geraint A Wiggins. “Computational Creativity: The Final Frontier?” In: *ECAI*. 2012, pp. 21–26.

[16] Jonathan Crowther, Sheila Dignen, and Diana Lea. *Oxford Collocations Dictionary: For Students of English*. Foreign Language Teaching and Research Press, 2003.

[17] Tom De Smedt and Walter Daelemans. “Pattern for python”. In: *The Journal of Machine Learning Research* 98888 (2012), pp. 2063–2067.

[18] Gustavo Diaz-Jerez. “Composing with Melomics: Delving into the Computational World for Musical Inspiration”. In: *Leonardo Music Journal* 21 (2011), pp. 13–14.

[19] George R Doddington et al. “The Automatic Content Extraction (ACE) Program-Tasks, Data, and Evaluation.” In: *LREC*. Citeseer. 2004.

[20] Claire Gardent, Eric Kow, et al. "A symbolic approach to near-deterministic surface realisation using tree adjoining grammar". In: *ACL*. Vol. 7. 2007, pp. 328–335.

[21] Albert Gatt and Ehud Reiter. "SimpleNLG: A realisation engine for practical applications". In: *Proceedings of the 12th European Workshop on Natural Language Generation*. Association for Computational Linguistics. 2009, pp. 90–93.

[22] Pablo Gervás. "Wasp: Evaluation of different strategies for the automatic generation of spanish verse". In: *Proceedings of the AISB-00 Symposium on Creative & Cultural Aspects of AI*. 2000, pp. 93–100.

[23] Dan Jurafsky et al. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Vol. 2. MIT Press, 2000.

[24] Hans Kamp and Uwe Reyle. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*. 42. Springer, 1993.

[25] Paul Kingsbury and Martha Palmer. "From TreeBank to PropBank." In: *LREC*. Citeseer. 2002.

[26] Peter Kolb. "Disco: A multilingual database of distributionally similar words". In: *Proceedings of KONVENS-2008, Berlin* (2008).

[27] Ray Kurzweil. "Ray kurzweil's cybernetic poet". In: *CyberArt Technologies* (1999).

[28] Hugo Liu and Push Singh. "ConceptNet—a practical commonsense reasoning tool-kit". In: *BT technology journal* 22.4 (2004), pp. 211–226.

[29] Hisar Manurung. "An evolutionary algorithm approach to poetry generation". In: (2004).

[30] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. "Building a large annotated corpus of English: The Penn Treebank". In: *Computational linguistics* 19.2 (1993), pp. 313–330.

[31] Adam Meyers et al. "The NomBank project: An interim report". In: *HLT-NAACL 2004 workshop: Frontiers in corpus annotation*. 2004, pp. 24–31.

[32] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[33] Douglas L Nelson, Cathy L McEvoy, and Thomas A Schreiber. "The University of South Florida free association, rhyme, and word fragment norms". In: *Behavior Research Methods, Instruments, & Computers* 36.3 (2004), pp. 402–407.

[34] Alison Pease and Simon Colton. "On impact and evaluation in computational creativity: A discussion of the Turing test and an alternative proposal". In: *Proceedings of the AISB*. Vol. 11. 2011, pp. 1–8.

[35] Ehud Reiter and Robert Dale. *Building natural language generation systems*. Vol. 152. MIT Press, 2000.

[36] Karin Kipper Schuler. "VerbNet: A broad-coverage, comprehensive verb lexicon". In: (2005).

[37] Jukka M Toivanen, Matti Järvisalo, and Hannu Toivonen. "Harnessing Constraint Programming for Poetry Composition". In: *Proceedings of the Fourth International Conference on Computational Creativity*. 2013, p. 160.

[38] Tony Veale. "Less Rhyme, More Reason: Knowledge-based Poetry Generation with Feeling, Insight and Wit". In: *Proceedings of the Fourth International Conference on Computational Creativity*. 2013, p. 152.

[39] Tony Veale and Guofu Li. "Specifying Viewpoint and Information Need with Affective Metaphors". In: ().

[40] R Weide. *The CMU pronunciation dictionary, release 0.6*. 1998.

[41] Geraint A Wiggins. "Searching for computational creativity". In: *New Generation Computing* 24.3 (2006), pp. 209–222.