Journal of Experimental & Theoretical Artificial Intelligence
Vol. 00, No. 00, Month-Month 200x, 1–17

Using Genetic Algorithms to Create Meaningful Poetic Text

RULI MANURUNG[†], GRAEME RITCHIE[‡], and HENRY THOMPSON[§]

[†]Faculty of Computer Science, Universitas Indonesia, Depok 16424, Indonesia. (maruli@cs.ui.ac.id)

[‡]Department of Computing Science, University of Aberdeen, Aberdeen AB24 3UE, UK. (g.ritchie@abdn.ac.uk)

[§]School of Informatics, University of Edinburgh, Edinburgh EH8 9LW, UK. (ht@inf.ed.ac.uk)

(*Received 00 Month 200x; In final form 00 Month 200x*)

This paper presents a series of experiments in automatically generating poetic texts. We confined our attention to the generation of texts which are syntactically well-formed, meet certain pre-specified patterns of metre, and broadly convey some given meaning. Such aspects can be formally defined, thus avoiding the complications of imagery and interpretation that are central to assessing more free forms of verse. Our implemented system, McGonagall, applies the genetic algorithm to construct such texts. It uses a sophisticated linguistic formalism to represent its genomic information, from which can be computed the phenotypic information of both semantic representations and patterns of stress. The conducted experiments broadly indicated that relatively meaningful text could be produced if the constraints on metre were relaxed, and precise metric text was possible with loose semantic constraints, but it was difficult to produce text which was both semantically coherent and of high quality metrically.

Keywords: natural language generation, genetic algorithms, poetry, creative language

1 Introduction

The task of automatically generating texts that would be regarded as poetry is challenging, as it involves many aspects of language. Moreover, there is no well-defined route by which poems are created by human poets. However, it is possible to set down some properties that a poetic text should manifest, which suggests that the generation might be achieved by an algorithm that couples a suitably effective search process with a relatively separate evaluation of the end-product; stochastic search, and in particular evolutionary algorithms, operate in this way.

We start here with a brief survey of work on automatic poetry generation (Sect. 2), before setting out a restricted definition of poetry (Sect. 3.1) as a text that embodies *meaningfulness*, *grammaticality*, and *poeticness*, and a model of its generation as a stochastic search process (Sect. 4). We then describe McGonagall, our implemented system that adopts this model and uses genetic algorithms (Mitchell 1996) to generate texts that are syntactically well-formed (Sect. 5.1), meet certain pre-specified patterns of metre (Attridge 1995) (Sect. 5.3), and broadly convey some given meaning (Sect. 5.4). Finally, we present results of some experiments conducted with McGonagall (Sect. 6) and conclude with some discussion about future avenues of research.

2 A Brief Survey of Poetry Generators

In this section we will examine four particular poetry generators and discuss their relative merits, similarities, and differences. (Further reviews of this area can be found in Bailey (1974), Gervás (2002), Manurung (2003)).

2.1 *ALFRED the Agent*

ALFRED the Agent (Donald n.d.) is highly typical of a large number of poetry generators that randomly choose words from a handcrafted dictionary to fill gaps in some incomplete linguistic structure, defined either as a predefined template or via some phrase structure rules. Fig. 1 shows some ALFRED output.

Wheresoever amorphous – just barely the nightclub,
howsoever apostolic amidst a calamity,
a dragon will irrigate a Copernican currant – an emphysema.
His cowlick must have incinerated a housebroken revelry as per a melamine.
your inactive hydrocarbon could atone.

Figure 1. Sample output from ALFRED The Agent

Other systems of this type include Masterman's haiku generator (Boden 1990) and ALAMO "rimbaudelaires" (Gervás 2002). Similar systems are available on the web such as ELUAR, The Poetry Creator, and ADAM. Notable instances are RACTER and PROSE (Hartman 1996), which have the distinction of having their poetry published – the poetry anthology *"The Policeman's Beard is Half Constructed"* consisted solely of poems constructed by RACTER. Some of these systems employ certain heuristics to simulate the appearance of coherence and poeticness, such as assigning ad-hoc 'emotional categories', e.g. {*ethereality, philosophy, nature, love, dynamism*} in ELUAR, and choosing lexical items repetitively to give a false sense of coherence, as in RACTER.

2.2 *Ray Kurzweil's Cybernetic Poet*

Ray Kurzweil's Cybernetic Poet, or RKCP (Kurzweil 2001), is a commercial software package that generates well-formed texts based on rhythmic properties of words that are compiled in a statistical language model trained on existing poems. RKCP employs proprietary algorithms, unavailable for us to analyze.

Crazy moon child
Hide from your coffin
To spite your doom.

Figure 2. Sample output from Kurzweil's Cybernetic Poet

The RKCP text in Figure 2 is presented as a haiku, although it does not meet the formal requirements of a haiku, i.e. lines of five, seven, and five syllables. This is because RKCP can relax constraints when it is unable to write a poem. Unlike the systems listed in Sect. 2.1, RKCP attempts to conform to *metre*, a fundamental aspect of most poetry.

2.3 *COLIBRI*

COLIBRI (Diaz-Agudo et al. 2002) uses *case-based reasoning* (Aamodt & Plaza 1994) to generate formal Spanish poetry that satisfies the constraints of strophic forms such as *romances, cuartetos*, and *tercetos encadenados*. Its input is a list of keywords representing the meaning to be conveyed, and a specification of the particular strophic form required.

Figure 3 shows how COLIBRI works. In (a) we see keywords representing the required meaning for the poem. In (b) a case is retrieved from a corpus of human-authored poems with an appropriate strophic form. In (c) the keywords from (a), marked in boldface, have been placed in the text, while maintaining syntactic well-formedness. Finally, in (d) we see the result of a revision, where the words marked with * have been substituted in to ensure the metre and rhyme specified by the strophic form.

COLIBRI differs from the systems in the previous sections because, as well as aiming for syntactic well-formedness (cf. Sect. 2.1) and appropriate rhythm (like RKCP), its goal is a text that is 'about' a given message, even if this is approximated rather trivially. However, when COLIBRI revises a poem to maintain metre and rhyme it pays no heed to meaning. Modifications to the text at this stage may destroy whatever meaning has already been set up (cf. Sect. 4.1).

(a) *una boca ardiente pase techo y suelo*

(b) *no sólo en plata o viola truncada*
se vuelva mas tú y ello juntamente
en tierra en humo en polvo en sombra en nada

(c) *no sólo en* **boca y** *viola* **ardiente**
se **pase** *mas tú y ello juntamente*
en tierra en **techo** *en* **suelo** *en sombra en nada*

(d) *no sólo* **para* boca y** *viola* **ardiente**
se **pase** *mas tú y ello juntamente*
en **tía*** *en* **techo** *en* **suelo** *en sombra en* **serpiente***

Figure 3. COLIBRI: (**a**) input, (**b**) verse pattern, (**c**) substituted, and (**d**) revised

2.4 *Chart Generation of Rhythm-Patterned Text*

Our initial attempt at poetry generation (Manurung 1999) constructs a syntactically well-formed text that conveys a conjunction of atomic formulae and satisfies a given metre. A context-free grammar guides exhaustive search for phrases that match the semantic items, with *chart generation* (Kay 1996) used to reduce the computational complexity. Patterns of required rhythms act as filters for potential structures during generation. Figure 4 shows a generated text in (b) with the metre of a limerick and the semantics in (a).

(a) `cat(c)` $\wedge$ `dead(c)` $\wedge$ `eat(e,c,b)` $\wedge$ `past(e)` $\wedge$ `bread(b)` $\wedge$ `gone(b)`

(b) *The cat is the cat which is dead.*
The bread which is gone is the bread.
The cat which consumed
the bread is the cat
which gobbled the bread which is gone.

Figure 4. Sample (**a**) input semantics and (**b**) output limerick from our chart generator

Like COLIBRI, this system takes account of syntax, rhythm, and meaning, but its account of meaning is much more sophisticated. In particular, it conveys the predicate-argument structure of the input. Also, COLIBRI handles meaning and metre separately, but our chart generator maintains syntactic, semantic, and rhythmic well-formedness at every step. Unfortunately, due to its exhaustive search, it is computationally very expensive, even in our tests on a tiny 'toy' grammar and lexicon. It is also not very flexible, as it can generate *only* immaculate results – if the linguistic resources are insufficient to produce the required output, it fails completely. We believe that a poetry generator should be *robust*, in the sense that it produces the best possible output with the resources at its disposal.

3 Defining Poetry

There is no universally accepted definition of what counts as poetry, but we will argue for using a particular restricted definition in our investigations.

3.1 *The Unity of Poetry*

> "Poetry is a literary form in which language is used in a concentrated blend of sound and imagery to create an emotional response." (Levin 1962)

This points to a strong interaction, or *unity*, between the form and content of a poem. The diction and grammatical construction of a poem affects the message that it conveys to the reader over and above its

obvious denotative meaning. This contrasts with the commonly-held linguistic notion that a text serves as a medium for conveying its semantic content.

3.2 *Abusing Poetic License*

Boden claims, in Boden (1990), that:

> "Readers of [poetry] are prepared to do considerable interpretative work. ... In general, the more the audience is prepared to contribute in responding to a work of art, the more chance there is that a computer's performance (or, for that matter, a human artist's) may be acknowledged as aesthetically valuable ... Much of the beauty, might one say, is in the eye of the beholder ... Hence poetry, especially poetry of a highly minimalist type (such as haiku), is more tolerant of programmed production than prose is."

The implication here is that the automatic generation of poetry is relatively easy because the burden of assigning meaning to a text is carried by the reader. We believe that this argument abuses *poetic license*[1], and that by taking it to the extreme, one can potentially justify any randomly created text as being poetry. This would seriously undermine any research on poetry generation. When appraising the output of a computer program that is claimed to generate poetry, to what extent can we allow deviations (e.g. semantic incoherence) in the text, by invoking 'poetic license'?

We believe the opposite of Boden's claim to be the case: automatic generation of poetry is *harder* than that of prose. Poetry is further constrained by rules of form that prose need not adhere to. Using true poetic license to turn a phrase in an effective way requires even greater mastery of the language than that for producing prose. Deviations from the rules and norms must have some purpose, and not be random.

3.3 *A Restricted Definition of Poetry*

We must have a definition of poetry that renders *falsifiable* the claim that a program successfully generates poetry. Our proposed definition is: a poem is a natural language artifact that simultaneously satisfies the constraints of *grammaticality*, *meaningfulness*, and *poeticness*.

- **Grammaticality:** A poem must be syntactically well-formed. This might seem obvious for natural language texts, but it must be stated explicitly in the context of poetry. Syntactic well-formedness in poetry may well be different from that of ordinary texts (e.g. figurative language), but it is still governed by rules.
- **Meaningfulness:** A poem must intentionally convey some conceptual message that is meaningful under some interpretation. By stipulating this constraint, we are more critical of what we consider *not* to be poetry, e.g. the output of ALFRED (Fig. 1), RKCP (Fig. 2), and to a lesser extent, COLIBRI (Fig. 3).
- **Poeticness:** A poem must exhibit features that distinguishes it from non-poetic text. In our research, we concentrate on concretely observable aspects, specifically the rhythmic patterns, or *metre*, of a poem.

This characterization suggests a 'classical' account of poetry, i.e. one where adherence to regular patterns in form is paramount. This avoids some of the complications of imagery or interpretation that are central to assessing more free forms of verse, and ensures that many of the important aspects of a text can be defined formally. It is an approximate formalisation of the sort of verse that was normal within British culture until the mid-20th century and is still commonplace today, and includes established forms such as limericks and sonnets (although we have not yet tackled the important issue of rhyme). Figure 5 exhibits Hillaire Belloc's *"The Lion"*, an example of this genre – and one that we will revisit throughout this paper.

Although our definition does impose constraints, it says nothing about abstract notions such as imagery and metaphor, which are beyond the scope of our current work. As the field advances, these might become further requirements, or be incorporated under the poeticness constraint.

Table 1 shows how the generators in Sect. 2 fit our restricted definition.

[1] According to the Oxford English Dictionary, poetic license is the "deviation from recognized form or rule, indulged in by a writer or artist for the sake of effect".

The Lion, the Lion, he dwells in the waste,
He has a big head and a very small waist;
But his shoulders are stark, and his jaws they are grim,
And a good little child will not play with him.

Figure 5. Hillaire Belloc's *"The Lion"*

Table 1. Poetry generators (Sect. 2) and how they address our constraints

System	Grammaticality	Poeticness	Meaningfulness
ALFRED	Yes	No	No
RKCP	Yes	Yes	No
COLIBRI	Yes	Yes	Yes
Chart generation	Yes	Yes	Yes

4 Evolving Poetry

We turn now to the question of how to automatically generate texts that satisfy the three constraints of grammaticality, meaningfulness, and poeticness.

4.1 *The Problem with Unity*

Poetry generation can be viewed as a specialized instance of *natural language generation*, or NLG, a subfield of artificial intelligence that focuses on the development of computer systems that can produce understandable texts in a human language, starting from some nonlinguistic representation of a *communicative goal* as input (Reiter & Dale 2000). A considerable amount of research has been done in NLG on the so-called "generation gap" problem, where interdependent decisions must be made across various levels of linguistic representation (Meteer 1991). This problem is exacerbated by the unity of poetry (Sect. 3.1). For example, with regards to metre, every single linguistic decision potentially determines the success of a poem in fitting a regular defined pattern. Various solutions have been proposed, such as: using alternative system architectures, using better search methods, and employing a new paradigm of *overgeneration*. We have drawn upon this research to propose our model of poetry generation as stochastic search, which we present in the following section.

4.2 *Poetry Generation as Stochastic Search*

Poetry generation can be viewed as a state space search problem, where a state in the search space is a possible text with all its underlying representation, from semantics all the way down to phonetics. A goal state satisfies the three constraints of meaningfulness, grammaticality, and poeticness. Such a search space is undoubtedly immense, even – given the recursive structures in natural language – infinite. Our proposed solution is to employ a form of *stochastic search*, a heuristic search strategy that relies on random traversal of a search space with a bias towards more promising solutions. It has become increasingly popular for solving computationally hard combinatorial problems from AI domains such as planning, scheduling, and constraint satisfaction, and has been shown to outperform deterministic approaches in a number of domains.

The particular search methodology that we use is the *genetic algorithm*, which is essentially an iteration of two phases, *evaluation* and *evolution*, applied to an ordered set, called the *population*, of candidate solutions (Mitchell 1996).

4.3 *Satisfying Grammaticality, Meaningfulness, and Poeticness*

How do we design our genetic algorithm to find solutions that satisfy the three constraints described in Sect. 3.3? There are two ways that constraints can be implemented: either invalid solutions are totally excluded from the search space, or they are admitted to the space but with a bias against them (Davis & Steenstrup 1987). Our three constraints are not independent. Within a generator, we must be able to

relate syntactic structure and semantic content systematically, probably by some variant of the widely accepted notion of *compositional semantics*. This would be hard to arrange if we allowed syntactically ill-formed structures. Also, the kind of poeticness we are concentrating on (metre), can be imposed regardless of grammaticality, but (as we shall see when examining *The Lion*), grammaticality may sometimes take precedence. We have therefore decided to treat grammaticality as a prerequisite for meaningfulness and poeticness; that is, we will implement grammaticality as an absolute constraint on the items admitted to the search space, through the design of representation and genetic operators. In doing this, we can call on the immense amount of linguistic work on mechanisms for ensuring grammaticality.

The remaining constraints of meaningfulness and poeticness will be implemented as preferences via the evaluation functions of the genetic algorithm.

5 McGonagall, an Instance of an Evolutionary Poet

McGonagall is our implementation of poetry generation as stochastic search. In this section we will describe how McGonagall ensures grammaticality through its representation (Sect. 5.1) and genetic operators (Sect. 5.2), and how it optimizes poeticness (Sect. 5.3) and meaningfulness (Sect. 5.4).

5.1 *Linguistic Representation*

Linguistic structures are represented in McGonagall using *lexicalized tree-adjoining grammar*, or LTAG (Schabes 1990), augmented with the use of feature structures (Vijay-Shanker & Joshi 1988). These grammars are based on the composition of *elementary trees*, of which there are two types, i.e. initial trees and auxiliary trees. These trees represent minimal linguistic structures in the sense that they account for all and only the arguments of the head of the syntactic constituent they represent, e.g. a sentential structure would contain a verb and all its complements. There are two types of operations, *substitution* of initial trees (typically labelled α) and *adjunction* of auxiliary trees (typically labelled β), that can be used to compose larger tree structures.

In LTAG, the *derivation tree* is a data structure that records the composition of elementary trees using substitution and adjunction. The *derived tree*, on the other hand, is the phrase structure tree created by performing all the operations specified within the deriviation tree; in a sense, the derived tree is the result of generation, whereas the derivation tree is a trace of how it was built up. Figure 6 gives an example of this. The derived tree in (c) is obtained by a series of operations on the elementary trees in (a). These operations are recorded by the derivation tree in (b). Each node in the derivation tree specifies an elementary tree. Furthermore, each node aside from the root node also specifies the Gorn address[1] of the node in its parent's elementary tree where the operation occurs, and is indicated by the label of the edge connecting a node to its parent. For example, α_{book} subtitutes into α_{read} at address 2.2, the object NP node. The process of composing the elementary trees is illustrated in (d).

The derivation tree can therefore be seen as the basic formal object that is constructed during sentence generation from a semantic representation (Joshi 1987), and is the appropriate data structure on which to perform nonmonotonic operations in our stochastic generation framework (Sect. 5.2). Essentially, the LTAG derivation tree forms the *genotypic* representation of our candidate solutions, from which we can compute the *phenotypic* information of semantic (Sect. 5.4) and prosodic (Sect. 5.3) features via the derived tree.

We adopt a simple 'flat' semantic representation (Hobbs 1985) that is often used in NLG (Koller & Striegnitz 2002). A *semantic expression* is a set of first order logic literals, which is logically interpreted as a conjunction of all its members. The arguments of these literals represent concepts in the domain such as objects and events, while the functors state relations between these concepts. For example, the representation of the semantics of the sentence *"John loves Mary"* is $\{john(j), mary(m), love(l, j, m)\}$,

[1] A Gorn address is a list of integers that indicates the position of a node within a tree. The Gorn address of the i-th child of node n is the Gorn address of n with i appended to the end of the list. The Gorn address of a root node is ϵ, or sometimes 0 for convenience.

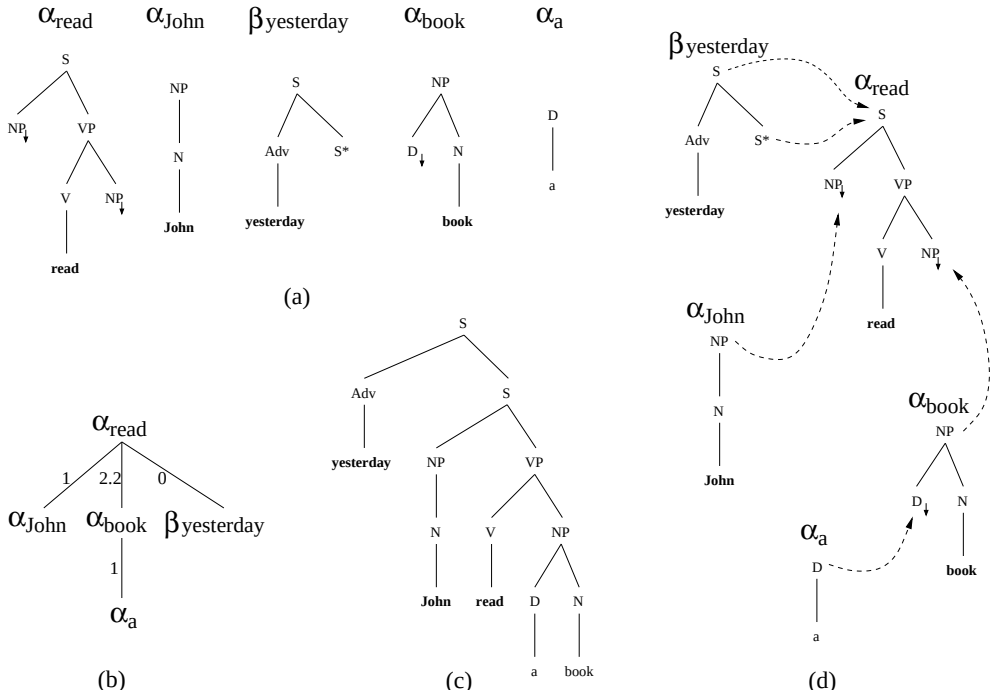

Figure 6. A derivation tree captures the process of composition of elementary trees

where l is the event of j, who has the property of 'being John', loving m, who has the property of 'being Mary'.

The semantic expression of a tree is the union of the semantic expressions of its constituent elementary trees, with appropriate binding of variables during substitution and adjunction through the use of *semantic signatures* to control predicate-argument structure; cf. Stone & Doran (1997), Kallmeyer (2002). Ultimately semantic information originates from the lexicon, where words are annotated with the meanings they convey. Figure 7 shows an example of the substitution of two noun phrases as the subject and object of a transitive tree. During the substitution of α_{John} at the subject NP position, the signatures $\boxed{\text{Y}}$ and $\boxed{\text{A}}$ are unified. Likewise, for the substitution of α_{Mary} at the object NP position, the signatures $\boxed{\text{Z}}$ and $\boxed{\text{B}}$ are unified. From this, we can recover the semantics of the derivation $S = \{love(X, Y, Z), john(\_, Y), mary(\_, Z)\}$.

Each word is associated with its phonetic spelling, taken from the CMU pronouncing dictionary (Weide 1996). Vowels are marked for lexical stress, with `0` for no stress, `1` for primary stress, and `2` for secondary stress. For example, the spelling of *'dictionary'* is [`D,IH1,K,SH,AH0,N,EH2,R,IY0`]. For monosyllables, closed class words (e.g. *the*) receive no stress, and open class words (e.g. *cat*) primary stress.

5.2 *Genetic Operators for Linguistic Structure Building*

We have implemented baseline ('blind') mutation operators that randomly *add* or *delete* subtrees of a derivation tree. In addition, we have also implemented a subtree *swapping* operator, which randomly swaps two derivation tree subtrees. When these subtrees belong to the same derivation tree, it behaves as a mutator, otherwise, it is used as a crossover operator. These nonmonotonic tree-based operators are similar to those commonly found in genetic programming (Angeline 1996).

We also implemented semantically motivated ('smart') genetic operators that exploit domain knowledge and attempt to maximize meaningfulness by conveying the *target semantics* (see Sect. 5.4), whilst still maintaining grammaticality. This is motivated by NLG systems such as PROTECTOR (Nicolov et al. 1995) and SPUD (Stone & Doran 1997). When adding content, these operators attempt to convey the *unrealized* semantics, i.e. the portion of the target semantics that has not yet been realized by the candidate solution. Conversely, when deleting content, operators try to remove *extraneous* items, i.e. semantics realized by the candidate solution but not present in the target semantics.

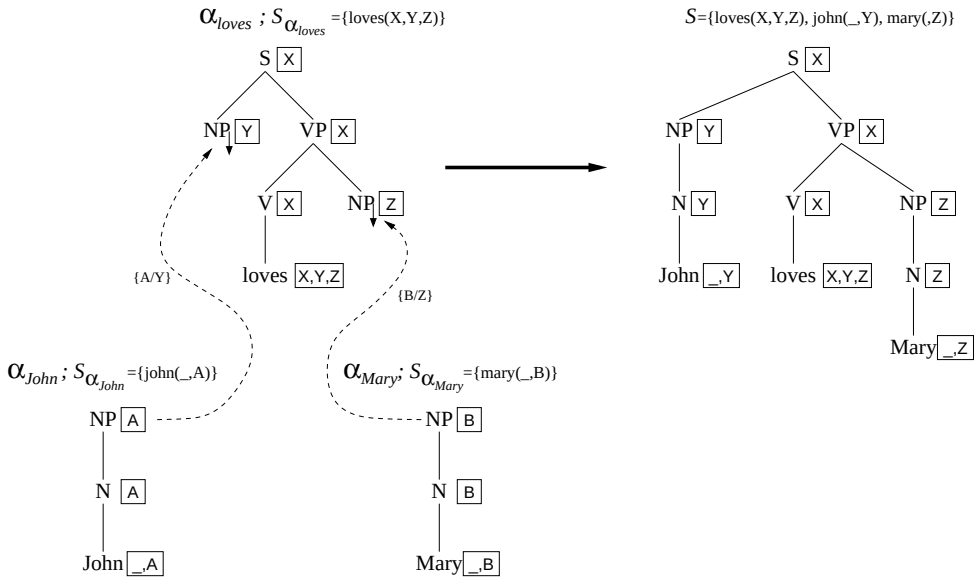

Figure 7. Compositional semantics through unification of signatures

5.3 *Evaluating Poeticness*

In McGonagall, poeticness is taken to be the (well-defined and objectively observable) *metre* – regular patterns in the rhythm of the lines (Attridge 1995). Figure 8 shows the metre of Belloc's *"The Lion"*, with stressed syllables in bold type, unstressed syllables in normal type, syllables extraneous to the underlying metre in italics, and • indicating a 'missing' syllable. Each line has the same number of stressed syllables, with a regular pattern of 'beats' at intervals of two unstressed syllables.

The	**Li**on, the	**Li**on, he	**dwells** in the	**waste**,
He	**has** a big	**head** and a	**ve**ry small	**waist**;
But his	**shoul**ders are	**stark**, and his	**jaws** they are	**grim**,
And a	**good** little	**child** • will	**not** play with	**him**.

Figure 8. Metre pattern of Hillaire Belloc's *"The Lion"*

McGonagall tries to maximize the similarity between the *target form*, a specification of the required metrical constraints, and the *candidate form*, the stress pattern of a candidate solution.

The target form is encoded as a list of *target syllables*, notated as follows: `w` (*'weak'*) is an unstressed syllable, `s` (*'strong'*) is a stressed syllable, `x` (*'wildcard'*) is any syllable, and `b` indicates a linebreak. Figure 9 shows example target forms for a limerick, a haiku, and *"The Lion"* (formatted into lines for readability purposes).

```
[w,s,w,w,s,w,w,s,b,
w,s,w,w,s,w,w,s,b,
   w,s,w,w,s,b,
   w,s,w,w,s,b,
w,s,w,w,s,w,w,s,b]
```
(a)

```
[x,x,x,x,x,b,
x,x,x,x,x,x,x,b,
   x,x,x,x,x,b]
```
(b)

```
[w,s,w,w,s,w,w,s,w,w,s,b,
w,s,w,w,s,w,w,s,w,w,s,b,
w,s,w,w,s,w,w,s,w,w,s,b,
w,s,w,w,s,w,w,s,w,w,s,b]
```
(c)

Figure 9. Target forms for (**a**) a limerick, (**b**) a haiku, and (**c**) *"The Lion"*

The candidate form, a representation of the metre exhibited by a candidate solution, is encoded as a list of *candidate syllables* obtained from a derived tree by concatenating the phonetic spellings at its leaf nodes (Sect. 5.1). Notation is as follows: 0 is an unstressed syllable, 1 is a primary stressed syllable, 2 is a

secondary stressed syllable, and b is a potential site for a natural line break. In addition, candidate syllables 0, 1, and 2 have a subscript index of either 1 or n, indicating whether they belong to a monosyllabic or polysyllabic word respectively. Figure 10 shows the derived tree representing the text *"The lion has a big head"*, and how its stress pattern is obtained.

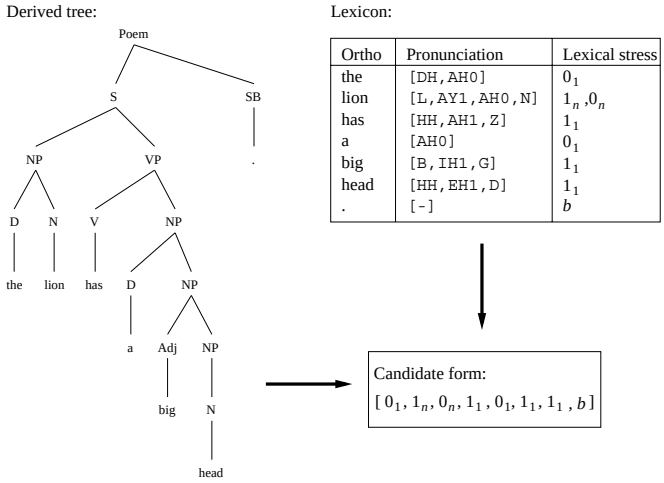

Figure 10. The candidate form of a text is a concatenation of lexical stress

Having explicit linebreaks in both target and candidate allows for a flexible handling of linebreaking anywhere in the text, in contrast to a hierarchical model of metrical forms consisting of lines of syllables (e.g. in Gervás (2000), Diaz-Agudo et al. (2002)).

To compute the similarity between the target and candidate forms, we use the *minimum edit distance* (Sankoff & Kruskal 1983), in which the distance between two strings is the minimal sum of costs of operations (symbol insertion, deletion, and substitution) that transform one string into another. The minimum edit distance can be efficiently computed (Jurafsky & Martin 2000) in a way that produces a pairwise syllable alignment between candidate and target, thus indicating the operations that yield the minimum cost. The costs are shown in Table 2(a)–(c), which reflect our intuitions, as follows:

(i) There should be no penalty for aligning candidate unstressed, stressed or linebreak syllables with target syllables of that same class, nor where non-linebreak candidate syllables align with a wildcard in the target.
(ii) Linebreaks cannot align with non-linebreaks, in either direction.
(iii) Having a linebreak in the candidate where none exists in the target (*enjambment*, the running on of a sentence from one line of a poem to the next) should be relatively expensive. Allocating no penalty for deletion of a candidate linebreak lets a line consist of more than one clause.
(iv) All other operations should incur some penalty. Insertion and deletion should be costlier for stressed syllables than for unstressed syllables.

Table 2. Operation costs for (**a**) substitution, (**b**) insertion, and (**c**) deletion

(a)

Cost	`w`	`s`	`x`	`b`
0_1	0	2	0	∞
0_n	0	2	0	∞
1_1	3	0	0	∞
1_n	3	0	0	∞
2_n	1	1	0	∞
b	∞	∞	∞	0

(b)

	Cost
`w`	1
`s`	3
`x`	1
`b`	10

(c)

	Cost
0_1	1
0_n	1
1_1	3
1_n	3
2_n	2
b	0

Our candidate forms indicate lexical stress patterns as if the words were pronounced in isolation. Within poetic text, context can affect stress. To compensate for this, the system iterates over the minimum edit distance alignment, detecting certain patterns and adjusting the similarity value. Two types of patterns

are implemented: two consecutive deletions, or two consecutive insertions, of non-linebreaks, increases the cost by 1; the destressing of a stressed candidate syllable adjacent to a stressed target syllable, or the stressing of an unstressed candidate syllable adjacent to an unstressed target syllable, decreases the cost by 1.

Our metre evaluation function, $\mathcal{F}_{metre}$, takes the value computed by the minimum edit distance algorithm, adjusts it using our context-sensitive compensation scheme, and normalizes it to the interval [0,1].

Table 3 shows $\mathcal{F}_{metre}$ values for various candidate texts, against the target form in Fig. 9(c). The first is Belloc's actual poem, which itself contains some metrical imperfections; the second is a limerick by Edward Lear; the third is an extract from an academic text, containing roughly the correct number of syllables; the last is chosen for its inappropriateness. The $\mathcal{F}_{metre}$ scores do not conflict radically with our intuitions of poetic metre.

Table 3. Metre similarity fitness scores for various candidates

Candidate text	Candidate form	$\mathcal{F}_{metre}$
The Lion, the Lion, he dwells in the waste. He has a big head and a very small waist. But his shoulders are stark, and his jaws they are grim, and a good little child will not play with him.	$[0_1, 1_n, 0_n, b, 0_1, 1_n, 0_n, b, 0_1, 1_1, 0_1, 0_1, 1_1, b, 0_1, 1_1, 0_1, 1_1, 1_1, 0_1, 0_1, 1_n, 0_n, 1_1, 1_1, b, 0_1, 0_1, 1_n, 0_n, 0_1, 1_1, b, 0_1, 0_1, 1_1, 0_1, 0_1, 1_1, b, 0_1, 0_1, 1_1, 1_n, 0_n, 1_1, 0_1, 0_1, 1_1, 0_1, 0_1, b]$	0.787
There was an old man with a beard, who said, "it is just as i feared! two owls and a hen, four larks and a wren, have all built their nests in my beard!"	$[0_1, 0_1, 0_1, 1_1, 1_1, 0_1, 0_1, 1_1, b, 0_1, 1_1, b, 0_1, 0_1, 1_1, 0_1, 0_1, 1_1, b, 1_1, 1_1, 0_1, 0_1, 1_1, b, 1_1, 1_1, 0_1, 0_1, 1_1, b, 1_1, 1_1, 1_1, 0_1, 1_1, 0_1, 0_1, 1_1, b]$	0.686
Poetry is a unique artifact of the human language faculty, with its defining feature being a strong unity between content and form.	$[1_n, 0_n, 0_n, 0_1, 0_1, 0_n, 1_n, 1_n, 0_n, 2_n, 0_1, 0_1, 1_n, 0_n, 1_n, 0_n, 1_n, 0_n, 0_n, b, 0_1, 0_1, 0_n, 1_n, 0_n, 1_n, 0_n, 1_n, 0_n, 0_1, 1_1, 1_n, 0_n, 0_n, 0_n, 1_n, 1_n, 0_n, 0_1, 1_1, b]$	0.539
John loves Mary.	$[1_1, 1_1, 1_n, 0_n, b]$	0.264

5.4 *Evaluating Meaningfulness*

The McGonagall approach to meaningfulness is similar to the above approach to poeticness: try to maximize the similarity between the *target semantics*, a specification of the meaning an optimal solution should convey, and a *candidate semantics*, the meaning conveyed by a candidate solution. This requires a method for computing the similarity between two semantic expressions.

Following Love (2000), we propose two factors that must be considered: *structural similarity* and *conceptual similarity*. Structural similarity measures the degree of isomorphism between two semantic expressions. Conceptual similarity is a measure of relatedness between two concepts (logical literals). We simply use the following: two concepts are the same if and only if they share the same literal functor.

Computing a structural similarity mapping between two expressions is an instance of the NP-hard graph isomorphism problem. However, we have implemented a greedy algorithm that serves our purposes and runs in $O(N^3)$, based on Gentner's structure mapping theory (Falkenhainer et al. 1989). It takes two sets of logical literals, S_{target} and $S_{candidate}$, and attempts to 'align' the literals. In doing this, it creates various variable bindings and also two sets of 'dangling' literals that are left over (from S_{target} and $S_{candidate}$) as unmatched items.

We then apply a function $\mathcal{F}_{sem}$, normalised to [0,1], to compute a score based on various aspects of the best match that has been achieved; this is based on Love's computational model of similarity (Love 2000).

Table 4 shows an example of computing semantic similarity for a selection of candidate texts against a target semantics that represents the second line of Belloc's *"The Lion"*, i.e. "[The lion] has a big head and a very small waist". The target semantic expression is as follows:

$$S_{target} = \{lion(\_, L), own(\_, L, H), head(\_, H), big(\_, H), \\ own(\_, L, W), waist(\_, W), small(S, W), very(\_, S)\}$$

The first two texts convey a subset of the target; the third text conveys an altogether different fact about the lion; the fourth text is purposely inappropriate; and the last text, interestingly, conveys the semantics of the first text in its object to the verb *'love'*. As with our metre similarity function, we believe that the $\mathcal{F}_{sem}$ scores roughly approximate human intuitions.

Table 4. Semantic similarity fitness scores for various candidates

Candidate text	Candidate semantics	$\mathcal{F}_{sem}$
The lion has a big head.	{*lion(_,L), own(_,L,H), head(_,H), big(_,H)*}	0.525
The lion has a head and a waist	{*lion(_,L), own(_,L,H), head(_,H), own(_,L,W), waist(_,W)*}	0.598
The lion dwells in the waste	{*lion(_,L), dwell(D,L), inside(_,D,W), waste(_,W)*}	0.078
John loves Mary	{*john(_,J), love(_,J,M), mary(_,M)*}	0.0451
John and Mary love the lion's big head	{*john(_,J), love(_,J,H), mary(_,M), love(_,M,H), lion(_,L), own(_,L,H), head(_,H), big(_,H)*}	0.389

6 Experiments with McGonagall

We conducted five tests, which clustered into three stages of experiments:

- **Metrical generation.** We set the program to generate grammatical texts that satisfy a given target metre. This measures its ability to generate solutions that optimize poeticness without concern for meaning. The evaluation function used was $\mathcal{F}_{metre}$ (Sect. 5.3). The target form used was that of a limerick, as in Figure 9(a).
- **NLG test.** This test measured the ability of McGonagall to generate solutions that optimize meaningfulness, without concern for metre. The evaluation function was $\mathcal{F}_{sem}$ (Sect. 5.4). This effectively causes McGonagall to behave as a conventional NLG system, i.e. it attempts to produce a text that conveys a given input semantics. The target semantics used is a representation of the first two lines of *"The Lion"*, with a slight alteration where we have replaced the original opening noun phrase *"the lion, the lion"* with *"the african lion"*, i.e. *"The african lion, he dwells in the waste. He has a big head and a very small waist"*. The target semantic expression is as follows:

$$\begin{aligned} S_{target} = \{&lion(\_,l), african(\_,l), dwell(d,l), inside(\_,d,was), waste(\_,was),\\ &own(\_,l,h), head(\_,h), big(\_,h), own(\_,l,wai), waist(\_,wai),\\ &small(s,wai), very(\_,s)\} \end{aligned}$$

  We conducted two variants of this test, one with the syntactically-aware 'blind' operators and one with the syntactically and semantically-aware 'smart' operators (Sect. 5.2).
- **Poetry generation.** In this test we measured the ability of McGonagall to perform the task it was designed for, namely the generation of texts that simultaneously satisfy grammaticality, meaningfulness, and poeticness. We took a very simple approach to combining the metre similarity and semantic similarity functions – the arithmetic mean of their scores, i.e. $\mathcal{F}_{poetry} = (\mathcal{F}_{metre} + \mathcal{F}_{sem})/2$. We used the same target form and target semantics as in the previous tests. As in the previous test, two variants were conducted: with the 'blind' operators and with the 'smart' operators.

6.1 *GA Parameters*

Throughout our testing, we employed one of the simplest selection algorithms, *proportionate selection*, which assigns a distribution that accords parents a probability to reproduce that is proportional to its fitness (Bäck et al. 1997). Individuals are sampled from this distribution using *stochastic universal sampling*, which minimises chance fluctuations in sampling (Baker 1987). To reduce the chances of premature convergence or stagnation, we used an *elitist* population of 20% of the entire population, which was found to yield the best results during an initial experiment with different ratios. The population size was set to

Table 5. Statistical summary of test results

Test	Min	Max	Mean	Std.Dev
Stage 1:				
Metrical generation	0.86	1.00	0.93	0.05
Stage 2:				
NLG test (blind operators)	0.74	0.93	0.85	0.06
NLG test (smart operators)	0.93	0.99	0.97	0.03
Stage 3:				
Poetry generation test (blind operators)	0.60	0.81	0.69	0.06
Poetry generation test (smart operators)	0.75	0.83	0.79	0.02

Table 6. Best found solution for metrical generation test

Fitness score	1.00
Text	They **play**. An ex**pense** is a **waist**. A **lion**, he **dwells** in a **dish**. He **dwells** in a **skin**. A **sens**itive **child**, he **dwells** in a **child** with a **fish**.

40, inspired by Cheng (2002), who in turn points to empirical studies by Goldberg (1989). Each test was run five times, and each run lasted for 500 iterations.

The three mutation operators used, along with their probabilities, were creation (0.5), adjunction (0.3), and deletion (0.2). The choice of these probabilities is intended to account for the fact that the substitution operation typically introduces the essential linguistic structures, and that adjunction is for auxiliary constituents. For crossover, the subtree swapping operator was used. The probabilities of applying genetic operators were $p_{mutation} = 0.6$, $p_{crossover} = 0.4$, for both the blind and smart variants.

A handcrafted grammar and lexicon was used, with 33 elementary trees and 134 lexical items, 28 of which were closed class words. Most of the content words were taken from *"The Bad Child's Book of Beasts"* (Belloc 1991).

6.2 *Overview of Results and Analysis*

Table 5 shows a statistical summary of the best fitness scores obtained during the three stages of experiments. It shows the minimum, maximum, mean, and standard deviation of the best fitness scores obtained after the last iteration across the five runs of each test. From this table we can observe that the GA is able to find optimal solutions with fitness approaching the maximum of 1.0 during the first two stages of experiments, but is failing to do so for the difficult multi-objective optimization task of the third stage.

Figure 11 shows, over time, the maximum and average of the best fitness scores across all the GA runs for the metrical generation test, (a), NLG test with blind and smart operators, (b) and (c), and poetry generation test with blind and smart operators, (d) and (e).

6.3 *Metrical Generation*

Table 6 shows the highest-scoring candidate from the metrical generation test. As expected, the text is syntactically well-formed, being generated through a composition of LTAG elementary trees. The target metre is satisfied perfectly in terms of syllable count, stress placement, and linebreaking.

6.4 *NLG Test with Blind Operators*

Table 7 shows the highest-scoring candidate from the blind operators NLG test. It shows the fitness score, text, and the best mapping of S_{target} to $S_{candidate}$ in terms of matched and unmatched literals.

The generator achieves a near-optimal solution of 0.93. At a glance, we can note that the text is certainly more 'about' the concepts within the target semantics than the best solution from the metrical generation test (Table 6) – at the expense of metre. This shows the marked effect of changing the evaluation function from metre similarity to semantic similarity. In terms of the evaluated semantic mapping, the only suboptimal feature is the presence of the unmatched candidate literal $own(\_52, \_53, \_54)$, which represents

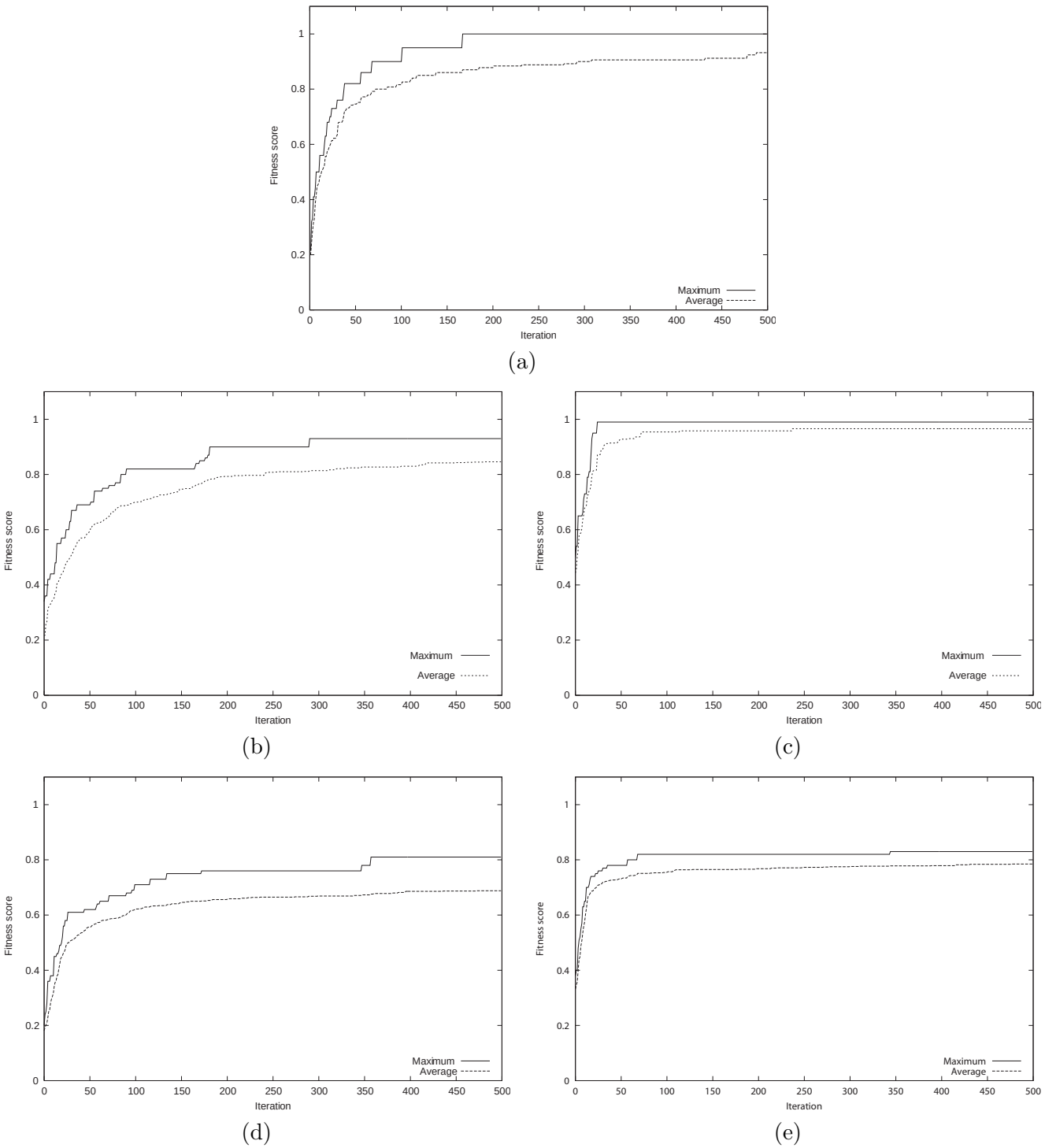

Figure 11. Maximum and average of best fitness scores for (**a**) metrical generation test, NLG test with (**b**) blind and (**c**) smart operators, poetry generation test with (**d**) blind and (**e**) smart operators

the fact that the *head* owns the *waist*. This is conveyed by the peculiar second sentence. If we observe the text itself, we can see that the facts that the *head* and *waist* belong to the *lion* are expressed by the genitive determiner *'its'*. There is also the spurious relative clause *"that is him"* that does not contribute to the semantics, but makes the text unnecessarily complex.

Table 7. Best found solution for NLG test with blind operators

Fitness score	0.93
Text	An african lion, it dwells in a waste. Its big head, that is him, it has its very small waist.
Matched literals	$\{(inside(\_2, d, was), inside(\_47, \_45, \_48)), (lion(\_0, l), lion(\_50, \_46)),$ $(african(\_1, l), african(\_51, \_46)), (dwell(d, l), dwell(\_45, \_46)),$ $(waste(\_3, was), waste(\_49, \_48)), (head(\_5, h), head(\_59, \_53)),$ $(own(\_4, l, h), own(\_61, \_46, \_53)), (own(\_7, l, wai), own(\_56, \_46, \_54)),$ $(big(\_6, h), big(\_60, \_53)), (small(s, wai), small(\_57, \_54)),$ $(waist(\_8, wai), waist(\_55, \_54)), (very(\_9, s), very(\_58, \_57))\}$
Unmatched target	$\emptyset$
Unmatched candidate	$\{own(\_52, \_53, \_54)\}$

Table 8. Best found solution for NLG test with smart operators

Fitness score	0.99
Text	An african lion, who has a very small waist, it dwells in a waste. Its head, it is big.
Matched literals	$\{(inside(\_2, d, was), inside(\_45, \_43, \_46)), (lion(\_0, l), lion(\_48, \_44)),$ $(african(\_1, l), african(\_49, \_44)), (dwell(d, l), dwell(\_43, \_44)),$ $(waste(\_3, was), waste(\_47, \_46)), (head(\_5, h), head(\_57, \_56)),$ $(own(\_4, l, h), own(\_58, \_44, \_56)), (own(\_7, l, wai), own(\_50, \_44, \_51)),$ $(big(\_6, h), big(\_55, \_56)), (small(s, wai), small(\_53, \_51)),$ $(waist(\_8, wai), waist(\_52, \_51)), (very(\_9, s), very(\_54, \_53))\}$
Unmatched target	$\emptyset$
Unmatched candidate	$\emptyset$

6.5 *NLG Test with Smart Operators*

Table 8 shows the highest-scoring candidate from the smart operator NLG test. It shows that an optimal solution was found; i.e. a text that conveys the semantics without any unmatched literals (the score of 0.99 and not 1 is caused by rounding off in the scaling function). Our own subjective analysis of the text itself seems to confirm this: the text is clearly understandable, if somewhat verbose.

Comparing Figure 11(b) to (c) we can see that the semantically smart operators lead to these optimal solutions much more rapidly and consistently.

Although McGonagall successfully found an optimal solution in this instance, some further tests indicated that it struggles to do so when the size of S_{target} is roughly double the size of the one used in this test.

6.6 *Poetry Generation with Blind Operators*

Table 9 shows the highest-scoring candidate from the blind operator poetry generation test. It shows the fitness score, text, and semantic mapping of S_{target} to $S_{candidate}$. In contrast to the previous tests, no optimal solution was found to this much more difficult multi-objective optimization task.

The text is metrically perfect. However, in terms of conveying S_{target}, it performs suboptimally. The unmatched target literals show that the text is failing to convey three concepts from S_{target}, i.e. that the lion is $african$, that its head is *big*, and that the waist is *very* small. The text contains three instances of the phrase *'will be rare'*, which is unrelated to the semantics of S_{target}. While this phrase contributes positively to achieving the target metre, it leads to unmatched candidate literals. Also, the second line is a repeat of the first. Although it contributes positively to the metre, our mapping algorithm is unable to match the semantics it conveys, resulting in unmatched candidate literals that penalizes the semantic similarity score. This does not seem to reflect the intuition that most humans would have reading the text, i.e. that it is simply a repetition of the first line.

The original text that conveys S_{target} has 22 syllables, whereas a limerick has 34 syllables. This may explain why the generator had to "pad out" the remaining syllables by adding unrelated content such as *"will be rare"* and repeating the first line, to the detriment of semantic similarity.

Table 9. Best found solution for poetry generation test with blind operators

Fitness score	0.81
Text	A **li**on, it **dwells** in a **waste**. A **li**on, it **dwells** in a **waste**. A **waste** will be **rare**. Its **head** will be **rare**. Its **waist**, that is **small**, will be **rare**.
Matched literals	$\{(head(\_5, h), head(\_53, \_52)), (own(\_4, l, h), own(\_54, \_28, \_52)),$ $(own(\_7, l, wai), own(\_48, \_28, \_45)), (lion(\_0, l), lion(\_29, \_28)),$ $(dwell(d, l), dwell(\_27, \_28)), (inside(\_2, d, was), inside(\_30, \_27, \_31)),$ $(waste(\_3, was), waste(\_32, \_31)), (small(s, wai), small(\_47, \_45)),$ $(waist(\_8, wai), waist(\_46, \_45))\}$
Unmatched target	$\{african(\_1, l), big(\_6, h), very(\_9, s)\}$
Unmatched candidate	$\{rare(\_33, \_34), will(\_35, \_36), waste(\_37, \_34), dwell(\_38, \_39),$ $lion(\_40, \_39), inside(\_41, \_38, \_42), waste(\_43, \_42), rare(\_44, \_45),$ $will(\_49, \_50), rare(\_51, \_52), will(\_55, \_56)\}$

Table 10. Best found solution for poetry generation test with smart operators

Fitness score	0.83
Text	A **ve**ry • **af**rican **li***on*, *who* is **af**rican, **dwells** in a **waste**. Its **head**, that is **big**, is **ve**ry • **big**. A **waist**, *that* is its **waist**, it is **small**.
Matched literals	$\{(inside(\_2, d, was), inside(\_150, \_147, \_149)), (lion(\_0, l), lion(\_151, \_140)),$ $(dwell(d, l), dwell(\_147, \_140)), (waste(\_3, was), waste(\_148, \_149)),$ $(head(\_5, h), head(\_138, \_136)), (own(\_4, l, h), own(\_139, \_140, \_136)),$ $(own(\_7, l, wai), own(\_146, \_140, \_143)), (small(s, wai), small(\_142, \_143)),$ $(african(\_1, l), african(\_152, \_140)), (big(\_6, h), big(\_135, \_136)),$ $(waist(\_8, wai), waist(\_144, \_143))\}$
Unmatched target	$\{very(\_9, s)\}$
Unmatched candidate	$\{very(\_137, \_135), big(\_141, \_136), waist(\_145, \_143), very(\_153, \_152),$ $african(\_154, \_140)\}$

6.7 *Poetry Generation with Smart Operators*

Table 10 shows the highest-scoring candidate from the smart operators poetry generation test. Again, no optimal solution is found. However, although the fitness score is very similar to the one in Table 9, the characteristics of the text are markedly different.

It is interesting to see the detrimental effect that the smart operators, which increase bias towards semantics, have on the metre. Unlike the metrically perfect limerick in Table 9, this text requires several edit operations (2 insertions and 2 deletions). (Even Belloc's original poem contains similar rhythmic imperfections – see Fig. 8). However, the text in Table 10 conveys S_{target} much better than that in Table 9. The only aspect that it fails to convey is the fact that the waist is *very* small. Three of the unmatched candidate literals are actually semantically accurate duplicates of the literals mapped in the proper matches, and not extraneous semantics. Unfortunately our mapping algorithm does not account for these in a satisfactory manner, just as it wrongly penalized the solution in Table 9 for the repetition of the first line.

We believe that this resulting text is comparable to the results obtained from our chart generation system, where we adopted an exhaustive search approach to poetry generation (see Sect. 2.4).

7 Summary

We have presented a model of poetry generation as stochastic search, where a goal state is a text that satisfies the constraints of meaningfulness, grammaticality, and poeticness. Previous attempts at poetry generation (Section 2) only properly addressed a subset of these constraints, with the exception of our initial attempt at poetry generation using chart generation (Manurung 1999). Unfortunately, this approach was computationally very expensive and was not very robust, i.e. if the system was unable to perfectly satisfy any of the constraints, it would fail to produce a single solution.

We reported on McGonagall, our implemented instance of this model, which operationalizes the three constraints by focusing on texts that are syntactically well-formed, meet certain pre-specified patterns of metre, and broadly convey some pre-specified meaning. It satisfies grammaticality through the use of lexicalized tree-adjoining grammar (LTAG) and optimizes meaningfulness and poeticness by maximizing evaluation functions that measure the degree of isomorphism between the phenotypic features exhibited by a candidate solution and the target semantics and metre. Experiments showed that it managed to find optimal solutions for moderately sized target semantics and metre forms in isolation. However, it encountered difficulties when simultaneously considering both evaluation functions.

There is still a lot of interesting research to be carried out in this area, particularly in exploring the different ways in which grammaticality, meaningfulness and poeticness can be defined. The notion of grammaticality can be augmented through the use of figurative language rules (Quinn 1982). With regards to meaningfulness, we note that our measure of isomorphism to a target semantics actually embodies the narrower notion of *faithfulness* to a specific meaning rather than the general concept of meaningfulness proposed in Sect. 3.3. A more ambitious account could consider aspects such as coherence, relevance, and interestingness. As for poeticness, whilst there are other phonetic features such as rhyme and alliteration that can be explored, there are many more features to be considered, such as figurative language and metaphor.

Acknowledgements

The work described was carried out while the authors were at the School of Informatics, University of Edinburgh. We would also like to thank members of the University of Aberdeen NLG group and the anonymous reviewers for their comments in preparing this article.

References

Aamodt, A. & Plaza, E. (1994), 'Case-based reasoning: Foundational issues, methodological variations, and system approaches', *AI Communications* **7**(1), 39–59.

Angeline, P. J. (1996), Genetic programming's continued evolution, *in* P. J. Angeline & K. E. Kinnear, eds, 'Advances in Genetic Programming', Vol. 2, MIT Press, Cambridge, USA, pp. 89–110.

Attridge, D. (1995), *Poetic Rhythm: an Introduction*, Cambridge University Press, Cambridge, UK.

Bäck, T., Fogel, D. & Michalewicz, Z., eds (1997), *Handbook of Evolutionary Computation*, Oxford University Press and Institute of Physics Publishing.

Bailey, R. W. (1974), Computer-assisted poetry: the writing machine is for everybody, *in* J. L. Mitchell, ed., 'Computers in the Humanities', Edinburgh University Press, Edinburgh, UK, pp. 283–295.

Baker, J. E. (1987), Reducing bias and inefficiency in the selection algorithm, *in* J. J. Grefenstette, ed., 'Proceedings of the Second International Conference on Genetic Algorithms', Lawrence Erlbaum Associates, Cambridge, USA, pp. 14–21.

Belloc, J. H. P. (1991), *The bad child's book of beasts*, Jonathan Cape, London, UK.

Boden, M. A. (1990), *The Creative Mind: Myths and Mechanisms*, Weidenfeld and Nicolson, London, UK.

Cheng, H. (2002), Modelling Aggregation Motivated Interactions in Descriptive Text Generation, PhD thesis, Division of Informatics, University of Edinburgh, Edinburgh, UK.

Davis, L. & Steenstrup, M. (1987), Genetic algorithms and simulated annealing: An overview, *in* L. Davis, ed., 'Genetic Algorithms and Simulated Annealing', Pitman, London, UK, pp. 1–11.

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

Donald, B. R. (n.d.), 'Dartmouth computer poetry homepage', `http://www.cs.dartmouth.edu/brd/alfred`.

Falkenhainer, B., Forbus, K. D. & Gentner, D. (1989), 'The structure-mapping engine: Algorithm and examples', *Artificial Intelligence* **41**, 1–63.

Gervás, P. (2000), WASP: Evaluation of different strategies for the automatic generation of spanish verse,

in 'Proceedings of the AISB'00 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science', AISB, Birmingham, UK.

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

Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, USA.

Hartman, C. O. (1996), *Virtual Muse: Experiments in Computer Poetry*, Wesleyan University Press.

Hobbs, J. (1985), Ontological promiscuity, *in* 'Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics', The Association for Computational Linguistics, Chicago, USA, pp. 61–69.

Joshi, A. K. (1987), The relevance of tree adjoining grammars to generation, *in* G. Kempen, ed., 'Natural Language Generation: New Results in Artificial Intellligence', Martinus Nijhoff Press, Dordrecht, The Netherlands, pp. 233–252.

Jurafsky, D. S. & Martin, J. H. (2000), *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice-Hall.

Kallmeyer, L. (2002), Using an enriched TAG derivation structure as basis for semantics, *in* 'Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)', Universita di Venezia, pp. 101–110.

Kay, M. (1996), Chart generation, *in* 'Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics', ACL, Santa Cruz, USA, pp. 200–204.

Koller, A. & Striegnitz, K. (2002), Generation as dependency parsing, *in* 'Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics', Philadelphia, USA.

Kurzweil, R. (2001), Ray Kurzweil's Cybernetic Poet. `http://www.kurzweilcyberart.com/poetry`.

Levin, S. R. (1962), *Linguistic Structures in Poetry*, number 23 *in* 'Janua Linguarum', Mouton, 's-Gravenhage.

Love, B. C. (2000), A computational level theory of similarity, *in* 'Proceedings of the 22nd Annual Meeting of the Cognitive Science Society', Philadelphia, USA, pp. 316–321.

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

Manurung, H. M. (2003), An Evolutionary Algorithm Approach to Poetry Generation, PhD thesis, School of Informatics, University of Edinburgh.

Meteer, M. (1991), 'Bridging the generation gap between text planning and linguistic realisation', *Computational Intelligence* **7**(4), 296–304.

Mitchell, M. (1996), *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, USA.

Nicolov, N., Mellish, C. & Ritchie, G. (1995), Sentence generation from conceptual graphs, *in* 'Proceedings of the International Conference on Conceptual Structures', number 954 *in* 'Lecture Notes in Artificial Intelligence', Springer-Verlag, Santa Cruz, USA.

Quinn, A. (1982), *Figures of Speech: 60 Ways to Turn a Phrase*, Hermagoras Press.

Reiter, E. & Dale, R. (2000), *Building Natural Language Generation Systems*, first edn, Cambridge University Press, Cambridge, UK.

Sankoff, D. & Kruskal, J. B., eds (1983), *Time warps, string edits and macromolecules: the theory and practice of sequence comparison*, Addison-Wesley, Reading, USA.

Schabes, Y. (1990), Mathematical and Computational Aspects of Lexicalized Grammars, PhD thesis, Computer Science Department, University of Pennsylvania.

Stone, M. & Doran, C. (1997), Sentence planning as description using tree adjoining grammar, *in* 'Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics', The Association for Computational Linguistics, Madrid, Spain, pp. 198–205.

Vijay-Shanker, K. & Joshi, A. K. (1988), Feature structure based tree adjoining grammars, *in* 'Proceedings of 12th International Conference of Computational Linguistics', Budapest, Hungary, pp. 714–720.

Weide, R. L. (1996), 'Carnegie Mellon University pronouncing dictionary', `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`.