PoeTryMe: Towards Meaningful Poetry Generation

Hugo Gonçalo Oliveira
hroliv@dei.uc.pt
CISUC
Universidade de Coimbra
Portugal

Abstract

PoeTryMe is a poetry generation platform under development that intends to help the automatic generation of meaningful poetry according to a given semantics. It has a versatile architecture that provides a high level of customisation where the user can define features that go from the base semantics and sentence templates to the generation strategy and the poem configuration. A prototype using PoeTryMe was implemented to generate Portuguese poetry. The results are interesting but there is still a long way for improvement, so we devise ideas for future work.

1 Introduction

After an overview on poetry generation systems, presented in [5], we started to develop a platform aiming to help the automatic generation of grammatically correct and meaningful poetry. PoeTryMe generates sentences according to relational triples and generation grammars that are used by a chart parser in the opposite direction to achieve chart generation [7]. It has a versatile architecture that provides a high level of customisation and can be used as the base of poetry generation systems that can be built on the top of it. Everything can be changed: the base semantics, represented as relations triples and relation templates; the templates of the generated sentences, included in the generation grammars; the generation strategies, that select the verses to include in the poem; as well as the poem configuration.

We start by introducing the external resources that are used in PoeTryMe and in the prototype (Section 2). After briefly describing each module of PoeTryMe's architecture (Section 3), we present the first prototype we have implemented on the top of PoeTryMe, for Portuguese poetry generation (Section 4) and show an example of a generated poem. Before discussing some further work (Section 6) we categorise our system according to its approaches and techniques and also according to its goals (Section 5).

```
RAIZ ::= REGRA
RAIZ ::= REGRA <&> OUTRAREGRA

REGRA ::= terminal
OUTRAREGRA ::= outroterminal
OUTRAREGRA ::= outroterminal <&> OUTRAREGRA
```

Figure 1. PEN example rules.

2 External resources

In the development of PoeTryMe's, and also in the prototype, external existing resources, presented in this section, were used to complete the whole system. The first resource is PEN, a Java implementation of the Earley [1] chart-parsing algorithm that analyses sentences according to grammars given as input. These grammars are text files, and thus easily editable, where each line contains a rule and its body. In order to differentiate rule tokens from terminals, all characters in rule names are upper case. An example of a very simple rule set is shown in Figure 1.

Jspell [10] is a morphological analyser for Portuguese. Given a word, Jspell identifies all possible analysis for that word, consisting of a lemma, a grammatical category, and other morphological information like the number, gender or verb tense. An example of the analysis of Jspell is shown in Figure 2.

PAPEL [6] is a lexical resource that consists of a set of approximately 200,000 relations between words, extracted semi-automatically from a general Portuguese dictionary, and can be used to build a semantic network. The relation set includes relations of synonymy, hypernymy/is-a, meronymy/part-of, causation, producer, purpose and property. Relations are represented as triples (*arg1 type arg2*) and the grammatical category of the arguments can be inferred from their type, if a special template description file

```
* uma 0 :lex(um, [CAT=art,CLA=indef,N=s,G=m], [], [G=f], []),
   lex(um, [CAT=card,N=s,G=m], [], [G=f], []),
   lex(um, [CAT=pind,G=m,N=s], [], [G=f], [])
* pessoa 4 :lex(pessoa, [CAT=nc,G=f,N=s], [], [], [])
* feliz 11 :lex(feliz, [CAT=adj,N=s,G=_], [], [], [])
* cantou 17 :lex(cantar, [CAT=v,T=inf,TR=_], [],
   [P=3,N=s,T=pp], [])
```

Figure 2. Jspell analysis of *uma pessoa feliz cantou***.**

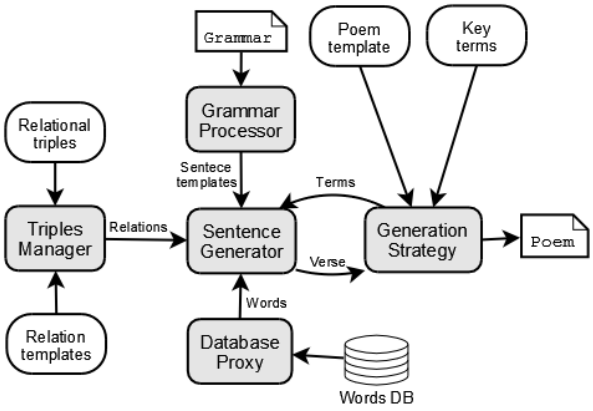

Figure 3. Prototype architecture

is used.

SilabasPT[1] is a Java API that performs syllabic division and stress identification for Portuguese words. It was developed to help generating text based on rhythm [4] in the project Tra-La-Lyrics [9], but it is an API that can be used by any other Portuguese language application.

3 Architecture

PoeTryMe relies on a modular architecture (see Figure 3) where each module has a simple and well defined task, enabling the independent improvement of each one of them. This architecture intends to be versatile enough to provide a high level of customisation, depending on the needs and ideas of the user. Many features are easily customised: it is possible to define the semantics to be used, the sentence templates in the generation grammar, the generation strategy and the configuration of the poem. In this section, the modules, their inputs and interactions are presented.

3.1 Sentence Generator

The Sentence Generator is the core module of PoeTryMe's architecture and is used to generate meaningful sentences with the help of:

- a semantic graph, managed by the Triples Manager;
- generation grammars written as if they were to be used by PEN and processed by the Grammar Processor;
- a database with words and their morphological information, result of Jspell's analysis, accessed via a Database Proxy.

The generation of a sentence/verse starts with a set of key terms that are used to select a sub-graph from the main semantic graph. All the relations that cannot be mapped to rules in the grammars are removed and all the relations involving the given terms are included in the sub-graph. After selecting a random term from the sub-graph, all the relations where it is involved are obtained. One of those relations is then randomly selected and mapped to a set of generation rules, included in the grammars. Out of the possible sentences for the chosen relation, one is randomly selected and completed (if necessary) with the result of database requests.

3.2 Triples Manager

This module uses the triples, given has input, to build a semantic graph where terms are related with other terms by means of semantic relations. It is used to obtain sub-graphs only with relations where the key terms or terms related with them are involved.

A file with a the template for each relation type is used in order to differentiate equal terms with ambiguous grammatical categories (e.g. *duck* can either be a noun or a verb).

3.3 Grammar Processor

Partly inspired by Manurung [7] we used a chart-parser, PEN, in the opposite direction, in order to perform chart generation. The generation grammars are written as if they were to be used by PEN and can be validated by PEN's validator, so the Grammar Processor is exactly the same module PEN uses for processing grammars. The head rules in the grammars should have the name of semantic relations present in the semantic graph, so that they can be later associated with those relations. When a grammar is processed, the Sentence Generator uses the Grammar Processor to get the name of all the head rules. The relations whose name is not the name of at least one head rule are not included in the semantic graph when the Triples Manager is creating it.

The body of the rules should consist of natural language renderings of semantic relations. Besides the simple terminal tokens, that will be present in the poem without any change, the Grammar Processor supports special terminal tokens, that are mapped by the Sentence Generator into database requests. These requests can either be

[1] http://code.google.com/p/silabaspt/

```
estrofe{verso(10);verso(10);verso(10);verso(10)}
estrofe{verso(10);verso(10);verso(10);verso(10)}
estrofe{verso(10);verso(10);verso(10)}
estrofe{verso(10);verso(10);verso(10)}
```

Figure 4. The structure of a sonet.

a simple token to be replaced by the argument of a relation (<arg1> and <arg2>) or it can be a request for a word like a noun (<n.lemma.gender.number>) or a verb (<v.lemma.tense.person.number>), with optional parameters (e.g. <v.cause.?.3.?> is a request for a form of the verb *to cause* in any tense and third person of any number).

3.4 Database Proxy

The Database Proxy works as an interface for the words database and transforms requests made by the Sentence Generator into SQL queries whose result is returned in the form of "filled" requests, that add to the original request the information obtained from the database.

The database contains word forms, that can be used in the poem, associated with morphological information provided by Jspell. In order to cover all the words in the semantic graph, all semantic nodes should be used in the database creation.

Considering nouns, since the relation triples usually have lemmatised arguments, we may want to replace them with a different form of the argument, in order to make the poem a little bit more interesting. As for verbs, they can appear as predicates in sentences and the database is used to give us different verb forms in different tenses and persons.

3.5 Generation strategies

A generation strategy is basically a module that takes advantage of the Sentence Generator to obtain verses and build up a poem. The poem is generated according to a set of key terms, used to get sentences from the Sentence Generator, and a template, which is a file with the poem's configuration – the number of strophes, the number of verses per strophe and the number of syllables of each verse (see Figure 4). A generation strategy can do nothing more than fill verses with obtained sentences but it can include some procedure to find the better sentences for each verse, considering features like the metrics, the rhyme, coherence between verses or other, depending on the poem's purpose.

4 Prototype

To test our platform and observe the first results we have put some resources together and implemented a basic generation strategy to generate meaningful verses and include a structure that can be viewed as a poem. We aimed the generation of Portuguese poems, so we used the relational triples of PAPEL as the source semantics and all the words included in PAPEL were analysed by Jspell to create the words database. In order to have all the possible forms of each verb, we used an online verb conjugator for Portuguese[2] to complete the database.

Although the poems had meaningful verses (according to the given semantics), the notion of metrics was lacking, so we decided to implement two more generation strategies that still take advantage of the generation of verses but additionally attempt to select sentences with a length as close as possible to the verses in the configuration template.

Type	arg1, arg2	Quant.	Example
HIPERONIMO_DE	noun,noun	63455	(*planta*, *salva*)
PARTE_DE	noun,noun	14453	(*cauda*, *cometa*)
CAUSADOR_DE	noun,noun	1125	(*fricção*, *assadura*)
ACCAO_QUE_CAUSA	verb,noun	6424	(*limpar*, *purgação*)
PRODUTOR_DE	noun,noun	932	(*romãzeira*, *romã*)
FINALIDADE_DE	noun,noun	2095	(*avaliação*, *exame*)
ACCAO_FINALIDADE_DE	verb,noun	5640	(*fazer_rir*, *comédia*)
LOCAL_ORIGEM_DE	noun,noun	768	(*Japão*, *japonês*)

Table 1. Relations of PAPEL used in the prototype.

Type	Example template
HIPERONIMO_DE	<arg2> é <arg1>[3]
PARTE_DE	<arg2> tem <arg1>
CAUSADOR_DE	<arg2> por causa de <arg1>
ACCAO_QUE_CAUSA	<arg1> leva a <arg2>
PRODUTOR_DE	<arg1> produz <arg2>
FINALIDADE_DE	<arg2> serve para obter <arg1>
ACCAO_FINALIDADE_DE	<arg2> para <arg1>
LOCAL_ORIGEM_DE	<arg2> vem de <arg1>

Table 2. Renderings in the prototype grammars.

4.1 Relations and renderings

To keep it simpler, we have opted to use only relations between two nouns or a verb and a noun, all of them represented in Table 1. For each relation type, several sentences that corresponded to natural language renderings of the relations were included in a generation grammar. In Table 2 simple examples of these sentences are shown. During the generation process, the tokens <arg1> and <arg2> are replaced by words that hold the relation, for instance, the words in the examples of Table 1.

In addition to the simple templates shown in Table 2, some more complex sentences were added. For example, we added conditional sentences to express a HIPER-

[2] http://linguistica.insite.com.br/cgi-bin/conjugue

ONIMO_DE or PARTE_DE relation (e.g. `se ele for <arg1> tal como <arg2>`).

4.2 Generation strategies

Three different generation strategies were developed in the prototype. While the first is very basic and was used almost only for obtaining quick results (sometimes useful for debugging) the others follow evolutionary approaches, somehow inspired by Manurung's [8] evolutionary algorithm for poetry generation.

In both of the two latter strategies there is an evaluation function that computes the absolute difference between the number of syllables the verse has in the template with the number of syllables in the generated sentence – the lower the evaluation, the better the sentence is. For obtaining the number of syllables in a verse, we used the SilabasPT API to count the syllables of each sentence and identify its last stress.

The algorithms involved in each one of the strategies are briefly described as follows:

- Basic: for each verse to be filled, a random sentence is generated using the key terms;
- Generate and test: for each verse to be filled, N random sentences are generated and the one with better evaluation is chosen. All unused sentences are indexed and can be used if a new verse needs exactly the same amount of syllables of the previously unused sentence.
- Evolutionary: an initial population of N poems is first generated using the basic strategy and then each poem is evaluated according to the aforementioned evaluation function. Each new generation will consist of the poems with the best evaluation, poems that are the result of crossing two random poems in the population and also some newly created poems. When two poems are crossed, a new poem is created with verses selected from both.

4.3 Example poem

For demonstration purposes we have generated a sonet (see configuration in Figure 4) with the nouns *planta* and *fruto* as key terms. The result (see Figure 5) was generated using an evolutionary strategy of 10 generations and an initial population of 10 poems. Each new population consisted of 20% of the best poems, 50% resulting from crossing and 30% new. The probability of swapping a verse was set to 50%.

As we can see, the poem contains meaningful sentences about either plants (*planta*) or fruits (*fruto*) and use terms

```
como vagem de feijoeiro
uns frutos vieram de três-em-prato
como o fruto venha de badiana
como Quenopodiáceas tenha mirabela

Sapotáceas procura uajar
como açaí produzirá os frutos
vagem de feijoeiro
espcie de baga de ingre

como hédera viria de planta
como produtor de frutos procura uvaia
Primuláceas tiver candelabro

jacatirão fizera parte de útil
lampadário entre candelabro
espécie de baga de ingre
```

Figure 5. Example of a generated poem.

that are related the key terms besides the key terms themselves. For example, *planta* is an hypernym of *feijoeiro* and *Quenopodiáceas* that are both terms used in the poem.

Considering the structure of the poem, despite having the correct number of verses the length of those is not always the same as in the template configuration, but close. The main reason for this to happen is that the grammar is not that big and varied, so even if the evolutionary algorithm had more generations there would always be issues for matching of the exact length of verses.

5 Categorisation

As presented in [5], poetry generation systems can be characterised according to their approaches and techniques (proposed by Gervás [3]) and also according to their goals (proposed by Manurung [8]). Gervás divides the possible approaches into four groups: template-based, generate and test, evolutionary and case-base reasoning. Although our approach uses sentence templates, it can follow different selection approaches, depending on the generation strategy. Bearing in mind the strategies implemented in the prototype, one is only template-based (basic strategy), another follows a generate and test approach and the third an evolutionary one.

As for their goals, Manurung divides the existing systems according to the properties that poetic texts must hold, namely meaningfulness, grammaticality and poeticness. Since we use a semantic network as input and we render information in it to natural language sentences, we can say that, if the network is well constructed and if the grammars generate grammatically correct sentences, our system holds both the property of meaninfulness and grammaticality. As for poeticness, our system supports different configurations of poems and two of the implemented strategies take the number of syllables of each verse into considera-

tion but, at the moment, these are the only poetic features and their occurrence is not even always guaranteed.

6 Conclusions and Further Work

A platform for the automatic generation of poetry was presented in this paper along with a first and still very simple prototype used to test the platform's capabilities. The presented work is preliminary but the platform is intended to be used with different external resources and customised in order to give rise to different and interesting types of poems, according to a predefined purpose. We believe that, in the future, PoeTryMe can be used as the starting for one (or more) poetry generation systems, eventually after taking a few possible directions for improvement that we did not take due to lack of time but we will now discuss.

In order to have a higher variation of text, the grammars can be improved according to the users will. Different linguistic constructions can be inserted so that the resulting poems are less predictable and have more variations. Besides, if, for the same relations, it is possible to generate sentences with very different lengths, it will also be easier to find exact matches (in terms of number of syllables) for verses in templates. Another interesting thing that could be tested in the grammars would be the use of weights that would change the probability of selecting each rule. PEN already supports weighted rules so it should not be a difficult thing to do.

In our implementation, we have used the Random class in the Java API for random number generation, which generates pseudo-random numbers, based on a seed. In the future a completely random number generator should be tested and the results compared.

The implemented prototype is only prepared to deal with relations with nouns and verbs, and can only request the database for words of both of these categories. There is no doubt that modifiers are very important in poetry and they should also take part in the poems, so another important improvement to enrich the resulting poems is the support for relations with adjectives and adverbs.

More generation strategies can be developed and the evolutionary strategy can be improved after testing different evaluation functions. For example, opposing to evaluating each verse independently, the evaluation could consider the whole poem and give a better evaluation to poems that have rhymes or poems that follow a predefined stress pattern.

As a system with a creative output, and despite the difficulty there is to evaluate this kind of systems, our results should be validated and evaluated. Ideas for validation include comparing the configuration of the generated poems with real poems with the same structure, while evaluation should be made by humans and take under consideration points like structure, metrics, novelty and, since we aim meaningful text generation, semantics.

Besides generating poetry, PoeTryMe can be used to assess the quality of semantic results. In the case of our prototype it can be used to accomplish an indirect evaluation of PAPEL. Although our prototype was created for Portuguese, we could adapt it to other languages but, in order to do that, we would have to create grammars in those languages, as well as a syllabic division algorithm. Additionally, a different semantic resource would have be needed (e.g. for English, WordNet [2] could be used) and the words database would have to be created with a specific morphological analyser.

References

[1] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 6(8):451–455, 1970. Reprinted in Grosz et al. (1986).

[2] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, May 1998.

[3] P. Gervás. Exploring quantitative evaluations of the creativity of automatic poets. In *Workshop on Creative Systems, Approaches to Creativity in Artificial Intelligence and Cognitive Science, 15th European Conference on Artificial Intelligence*, 2002.

[4] H. Gonçalo Oliveira. Geração de texto com base em ritmo. Master's thesis, University of Coimbra, 2007.

[5] H. Gonçalo Oliveira. Automatic generation of poetry: an overview. *1st Seminar of Art, Music, Creativity and Artificial Intelligence*, 2009.

[6] H. Gonçalo Oliveira, P. Gomes, D. Santos, and N. Seco. PAPEL: a dictionary-based lexical ontology for Portuguese. In A. Teixeira, V. L. S. de Lima, L. C. de Oliveira, and P. Quaresma, editors, *Computational Processing of the Portuguese Language, 8th Intl. Conference, Proceedings (PROPOR 2008)*, volume 5190, pages 31–40. Springer Verlag, 2008.

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

[8] H. Manurung. *An evolutionary algorithm approach to poetry generation*. PhD thesis, University of Edinburgh, 2004.

[9] H. R. Gonçalo Oliveira, F. A. Cardoso, and F. C. Pereira. Tra-la-lyrics: an approach to generate text based on rhythm. In *Cardoso, A. & Wiggins, G. (Ed.). Proceedings of the 4th. International Joint Workshop on Computational Creativity, London, UK*, 2007.

[10] A. M. Simões and J. Almeida. Jspell.pm – um módulo de análise morfológica para uso em processamento de linguagem natural. In *Actas do XVII Encontro da Associação Portuguesa de Linguística*, pages 485–495, Lisboa, 2002.