Towards A Computational Model Of Poetry Generation

by

Hisar Manurung, Graeme Ritchie, Henry Thompson

Informatics Research Report EDI-INF-RR-0015

Division of Informatics **May 2000**

http://www.informatics.ed.ac.uk/

Towards A Computational Model of Poetry Generation

Hisar Maruli Manurung, Graeme Ritchie, Henry Thompson
Division of Informatics
University of Edinburgh
80 South Bridge Edinburgh EH1 1HN
hisarm@dai.ed.ac.uk, g.d.ritchie@ed.ac.uk, ht@cogsci.ed.ac.uk

Abstract

In this paper we describe the difficulties of poetry generation, particularly in contrast to traditional *informative* natural language generation. We then point out deficiencies of previous attempts at poetry generation, and propose a stochastic hillclimbing search model which addresses these deficiencies. We present both conceptual and implemented details of the most important aspects of such a model, the evaluation and evolution functions. Finally, we report and discuss results of our preliminary implementation work.

1 Motivation

Poetry is a unique artifact of human natural language production, with the distinctive feature of having a strong unity between its content and its form. The creation of poetry is a task that requires intelligence, expert mastery over world and linguistic knowledge, and creativity. Although some research work has been devoted towards creative language such as story generation, poetry writing has not been afforded the same attention. It is the aim of this research to fill that gap, and to shed some light on what often seems to be the most enigmatic and mysterious forms of artistic expression.

Furthermore, poetry possesses certain characteristics that render traditional natural language generation (NLG) systems, which are geared towards a strictly *informative* goal, unsuitable due to architectural rigidness.

Lastly, although not readily obvious, there are potential applications for computer generated poetry, such as the increasingly large industry of electronic entertainment and interactive fiction, the commercial greeting card poetry genre, and perhaps even the odd pop music lyric or two.

2 Poetry Generation

2.1 What Is Poetry?

Regarding poetry the artifact, Levin (1962) states that "In poetry the form of the discourse and its meaning are fused into a higher unity." This definition highlights the point of a strong interaction between semantics, syntax and lexis. Boulton (1982) reiterates this point, claiming that it is misleading to separate the physical and intellectual form of a poem so far as to ask, "What does it mean?". The poem means itself.

These are rather esoteric quotations, and one would expect these quotes to be referring to "high-brow" poetry. However, we claim that this unity is inherent in simpler forms of poetry, for example, Hillaire Belloc's *The Lion* (Daly, 1984):

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.

Essentially unity here means that the poem "works" due to a combination of features at the surface level (the rhyming of *waste* and *waist*, *grim* and *him*, the repetition of *The lion*, *the lion* to fit the rhythm), and semantics (description of a lion as told to a child).

Of the many special characteristics that poetic form possesses, among the most essential are: rhythm, rhyme, and figurative language.

As for the process of writing poetry, it is often claimed to proceed in a much more flexible manner than other writing processes. There is often no well-defined communicative goal, save for a few vague concepts such as "wintery weather" or "a scary lion". Furthermore, a human could begin writing a poem inspired by a particular concept, or scenario, but end up writing a poem about an altogether different topic.

This specification of loose constraints fits with (Sharples, 1996) and (Boden, 1990), who claim that while a writer needs to accept the constraints of goals, plans, and schemas, *creative* writing requires the breaking of these constraints. Yet these constraints are still necessary, as they allow for the recognition and exploiting of opportunities.

Sharples (1996) models the writing process as that of creative design, involving a cycle of analysis, known as *reflection*, and synthesis, known as *engagement*. This process is analogous to our iterative process of evaluation and evolution, and ties in with the concept of unity between content and form: during the reflection phase, when looking at an intermediate draft of the poem on paper, a poet may come to realize the opportunities of surface features that can be exploited, which enables further content to be explored upon subsequent engagement phases.

2.2 Previous Attempts

Most previous attempts at poetry generation are "hobbyist experiments" that are available on the World Wide Web, such as The Poetry Creator, ELUAR, and Pujangga, with the two exceptions that exist in publication being RACTER and PROSE (Hartman, 1996). RACTER is also the only computer program with a published poetry anthology, "The Policeman's Beard is Half Constructed" in 1984.

All of these attempts were essentially "party trick"-type programs, in the mould of ELIZA (Weizenbaum, 1966). Typically, the generation process simply consisted of randomly choosing words from a hand-crafted lexicon to fill in the gaps provided by a template-based grammar. However, several clever tricks and heuristics were employed on top of the randomness to give the appearance of coherence and poeticness, such as: (1) assigning ad-hoc "emotional categories", e.g. {ethereality, philosophy, nature, love, dynamism} in ELUAR, and {romantic, patriotic, wacky, moderate} in Pujangga, (2) choosing lexical items repetitively to give a false sense of coherence, e.g. RACTER, (3) constructing highly elaborate sentence templates, often to the point that the resulting poetry would have to be attributed more to the human writer than to the program.

This is a representative output from ELUAR:

Sparkles of whiteness fly in my eyes,
The moan of stars swang branches of trees,
The heart of time sings in the snowy night.
Seconds of Eternity fly in grass,
The Clock of rain turns,
Death of the Apples,
The Equinox penetrates the words.

The two great deficiencies of these attempts were that they took no account whatsoever of semantics (the systems were not trying to convey any message) nor of poetic form, e.g. rhythm, rhyme, and figurative language.

2.3 What Makes It Difficult?

If we are to develop a poetry generation system which overcomes the two deficiencies mentioned above, what difficulties do we run into, particularly in comparison to conventional NLG systems?

1. In conventional, informative NLG systems, the starting point is a given message, or communicative goal, and the goal is to produce a string of text that conveys that message according to the linguistic resources available. In poetry, however, there may not be a well-defined message to be conveyed (see section 2.1).

2. The generation process is commonly decomposed into stages of content determination, text planning, and surface realisation (Reiter, 1994). We claim this approach is unsuitable for the task of poetry generation because it introduces problems of architectural rigidness (cf. De Smedt et al. (1996)) which are exacerbated by the unity of poetry, where interdependencies between semantics, syntax, and lexis are at their strongest.

3. If our poetry generator is to create texts which satisfy the multitude of phonetic, syntactic and semantic constraints, it must have a very rich supply of resources, namely: a wide coverage grammar which allows for paraphrasing, a rich lexicon which supplies phonetic information, and a knowledge-base if we hope to produce coherent poems.

4. One of the main difficulties lies in the objective evaluation of the output text. The question of measuring text quality arises for existing NLG systems, but is much more pronounced in evaluating poetry: how does one objectively evaluate if something is a poem or not.

   When writing about Masterman's haiku producer, Boden (1990) states that readers of poetry are prepared to do considerable interpretative work, and 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 may be acknowledged as aesthetically valuable. Hence readers of computer-generated text will be more tolerant in their assessment of poetry than of prose.

   This sounds encouraging for doing work in poetry generation. However, this observation also implies that it could be *too* easy to program the computer production of poetry: precisely because poetry readers are prepared to do interpretative work, it would be all too easy to pass off random word-salad output as Truly Genuine Poetry (whatever that may be).

The first three points mentioned above are of a more technical nature, while the last one is more conceptual, perhaps even philosophical. Currently, we do not have much to say on this last point, except that we hope to adopt an objective and empirical evaluation methodology similar to that of Binsted et al. (1997).

2.4 Limiting our Poetry

The main characteristics which we look for in our target generated poetry are a highly regular occurrence of syntactic and phonetic patterns, such as metre, rhyme, and alliteration. These are easily identifiable, and one could say we are adopting a "classic" view of poetry. Furthermore, we will only offer a relatively simple and straightforward treatment of semantics (see section 4.3) and of constructing the poem's content. The verse in section 2.1 typifies these attributes.

3 A Stochastic Hillclimbing Model

In an attempt to address the difficulties raised in Section 2.3, we propose to model poetry generation as an explicit search, where a state in the search space is a possible text with all its underlying representation, and a "move" in the space can occur at any level of representation, from semantics all the way down to phonetics. This blurs the conventional divisions of content determination, text planning, and surface realisation, and is actually readopting what De Smedt et al. (1996) call an *integrated architecture*, which goes against recent developments in NLG, but seems a necessary decision when considering poetry.

The problem, of course, is navigating the prohibitively large search space. Our proposed solution is to employ a stochastic hillclimbing search, not merely for its relatively efficient performance, but especially since the creative element of poetry generation seems perfectly suited to a process with some element of randomness to it.

Our stochastic hillclimbing search model is an *evolutionary* algorithm, which is basically an iteration of two phases, *evaluation* and *evolution*, applied to an ordered set (the *population*) of candidate solutions (the *individuals*).

This approach is quite analogous to (Mellish et al., 1998), an experiment in using stochastic search for text planning, but in our research we extend it to the whole NLG process.

3.1 Evaluation

Arguably the most crucial aspect of a stochastic search is the evaluation scheme which lets the system know what a desirable solution is. Below we present an informal description, not necessarily exhaustive, of the features that our evaluation functions must look for in a poem. A description of the actual evaluators in our currently implemented system can be found in Section 4.5.

1. **Phonetics**: One of the most obvious things to look for in a poem is the presence of a regular phonetic form, i.e. rhyme, metre, alliteration, etc. This information can be derived from a pronunciation dictionary.

   One possible evaluation method is to specify a "**target phonetic form**" as input, i.e. the ideal phonetic form that a candidate solution should possess, and to then score a candidate solution based on how closely it matches the target form.

   For example, we could provide the system with the following target form (here `w` means a syllable with weak stress, `s` a syllable with strong stress, and `(a)` and `(b)` would determine the rhyme scheme, e.g. *aabba*), which effectively means we are requesting it to generate a *limerick*:

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

   Alternatively, we could specify a set of these target forms, thus feeding the system with knowledge of existing poetry forms: the quintain, haiku, rondeau, sestina, etc., and allow the evaluation function to reward candidate solutions that are found gravitating closely towards one of those patterns. This is a more flexible alternative, but would probably not be as informed a heuristic, as the definition of the goal becomes less focussed.

2. **Linguistics**: Aside from phonetic patterns, there are other, more subtle, features to look for in a poem: **lexical choice**, where the evaluation could reward the usage of interesting collocations and words marked as "poetic", **syntax**, where reward could be given to usage of interesting syntactic constructs, e.g. inverse word and clause order, topicalization, and **rhetoric**, where evaluation would score the usage of figurative language constructs such as metonymy.

3. **Semantics**: Even more abstract would be a mechanism for evaluating the semantics of a certain candidate. Again, we could specify a "target semantics" and score a candidate's semantics relative to this target. Unlike conventional NLG, though, this target semantics is not viewed as a message that must be conveyed, but rather as a "pool of ideas", from which the system can draw inspiration. The system could choose to convey more or less than the given semantics (cf. *approximate generation* in Nicolov (1998)).

   Story generation issues such as narrative structure and interestingness are beyond the scope of this research.

Having analysed the three points above, it seems that to devise an evaluation function, the following 3 issues must be tackled:

- How to **identify** the presence of a feature: with the possible exception of figurative language, it is reasonably straightforward to observe the features. Most of them are represented directly in the data structure, e.g. phonetic form, lexical choice, syntactic structure, semantic interpretation.

- How to **quantify** a feature: yielding a numerical measure for the occurrence of a poetic feature sounds like a very naive idea. Nonetheless, we believe that it is the only way to mechanically and objectively guide the stochastic search to producing poem-like texts.

  Above we have mentioned a score-relative-to-target strategy for both phonetics and semantics. This seems to be the most concrete method of evaluation, and is what we have chosen to implement in our current system. Certain features, however, most notably those considered to be *preferences* as opposed to *constraints*, do not lend themselves easily towards this strategy. Furthermore, as mentioned above, we would sometimes like the flexibility of allowing the system to operate unguided by such a specific target.

  A naive alternative scoring method is to maintain a tally of points for every occurrence of a feature encountered in a text. This resembles a greedy algorithm search heuristic. For example: applied to the feature of alliteration, if we scored positively for each word that appeared in a line starting with the same phoneme, the final output could become ridiculously riddled with redundant repetitions of rewordings. This might be good for generating tongue-twister-like sentences, but any literary critic would baulk at these results. However, at the moment this is how we implement evaluation of such features, and although we do not intend to go deep into literary theory, we hope to develop a more sophisticated approach.

  For now our aim is to facilitate a modular approach to the evaluation of features, so that each particular type of feature will have its own corresponding "evaluator function". This will allow for more sophisticated approaches and techniques to be easily added in the future.

  Apart from a modular approach, we also aim to parameterize the behaviour of these evaluation functions, e.g. allow a user to set the coefficients and weighting factors that determine the calculation of a certain score. A very interesting prospect is the interfacing of these coefficients with empirical data obtained from statistical literary analysis, or stylometry.

- **Weighting** across features: assuming we have obtained numerical scores for each of the features we are considering, how do we combine them? As in the previous point about parameterizing coefficients of a particular evaluator, we propose to treat the weighting across features in a similar fashion. This parameterization could possibly allow a choice between, say, a preference for rigidly structured poetry and a preference for a more contemporary content-driven poem.

3.2 Evolution

After evaluating a set of candidate solutions and choosing a subset of candidates with the best score, we must then create new variations of them through "mutation". This process can be seen as applying a collection of operators on the chosen candidates. We introduce here three conceptual types of operators, before describing our currently implemented operators in Section 4.6:

- **Add**: "John walked"→"John walked *to the store*"
- **Delete**: "John likes Jill and Mary"→"John likes Jill"
- **Change**: "John walked"→"John *lumbered*'

Due to our integrated architecture, these mutations may occur at different underlying levels of representation of the text. Because these different levels are all interdependent, the operators must take special care to preserve consistency when performing mutation. For example, if the addition of "*to the store*" is viewed mainly as a syntactic addition of a prepositional phrase, the operator would have to update the semantics to reflect this, for instance by adding `destination(w,shop)`. In contrast, if it is viewed primarily as a semantic addition, the operator would have to realize these semantics, one option being the use of a prepositional phrase. Our practice of introducing semantics via a flexible "semantic pool" addresses this issue (see Section 4.3).

Another issue is that these operators can perform non-monotonic modifications on the structures of the candidate solutions, hence our grammar formalism must allow for this.

As it is probably too optimistic to rely on pure random mutation to lead us to a decent poem, we would also like to introduce several heuristic-rich operators. These heuristics would be the encoding of "how to write poetry" guidelines, such as "use 'little' words to 'pad' sentences when trying to fit the metre", and "don't use words with few rhymes at the end of a line". These 'smarter' operators, however, seems to go against stochastic search traditions wherein the operators are deliberately knowledge-poor, relying on the stochastic nature to lead us to the solution. Here, we are adding informedness of heuristics to the whole stochastic process, somewhat analogous to **sampling bias** in stochastic search.

4 Implementation

We are currently in the process of implementing our stochastic search model in Java. In this section we will first briefly discuss our choice of representation for grammar, lexicon and semantics, before describing properties of the architecture and the current implementation of our evaluation and evolution functions.

4.1 Grammar Formalism

Our choice of representation is Lexicalized Tree Adjoining Grammar, or LTAG. For reasons of space, however, we will not explain this formalism in depth. See Joshi and Schabes (1992) for details.

In Tree Adjoining Grammar, a derivation tree is a kind of meta-level tree that records operations performed on elementary trees. In particular, nodes of a derivation tree do **not** signify phrase structure in any way, but rather the process of adjoining and substitution of elementary phrase structure trees. Nodes of a derivation tree are labelled by references to elementary trees, and edges are labelled by the address at which the elementary tree of the child node substitutes or adjoins into the elementary tree of the parent (see Figure 1).

The root node of the derivation tree would introduce the verb, and its two siblings would introduce the subject and object noun phrases. The edges would signify at which NP node the child gets substituted into, effectively informing which is the subject and which is the object.

The common way to deal with TAG trees is by repeatedly performing adjunction and substitution, while having a derivation tree as a record-keeping ’roadmap’ of how the tree is derived. However, since we can change and delete portions of our text, the derived tree is problematic since there is no way to “un-adjoin” subtrees. We must always refer back to the derivation tree. In effect, there is no point in maintaining the derived tree throughout the generation process. Instead, the derivation tree becomes our primary data structure, and everything else can be derived from it on demand.

When our operators are said to perform adjunction and / or substitution, they are simply recording the operation in the derivation tree, not actually performing it.

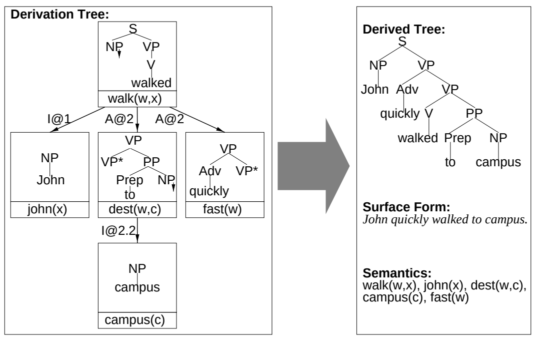

Figure 1: Making the LTAG derivation tree our main data structure

LTAG has the following advantages for our work:

- The *adjunction* operation in LTAG allows flexible incremental generation, such as subsequent insertion of modifiers to further refine the message. This is required as the system builds texts incrementally through an iterative process.
- LTAG provides an extended domain of locality which allows for predicate-argument structures and feature agreements over a structured span of text that need not be contiguous at the surface. This potentially allows for the coupling of poetic features such as rhyming across lines.
- We also adopt an extension to the formalism, Synchronous Tree Adjoining Grammar (STAG), which has been proven useful for paraphrasing purposes Dras (1999).
- LTAG provides an elegant mechanism for our non-monotonicity requirement through the use of the derivation tree. It keeps all syntax and semantics locally integrated at each node, and allows non-monotonic modification of content simply by deleting or replacing the corresponding node.

4.2 Linguistic Resources

At the moment we are still using a very small hand-crafted grammar and lexicon. Like most TAG-based systems, the grammar is a collection of elementary trees, and the lexicon is a collection of words that specify which elementary trees they can anchor. The lexicon also provides phonetic information and lexicon stress, which is extracted from the CMU Pronunciation Dictionary.

A typical lexical entry looks something like this:

```
Orthography:fried
Elementary Tree(s):ITV
Signature:F,Frier,Fried
Semantics:fry(F,Frier,Fried)
Phonetic Spelling:f,r,ay1,d
```

whereas a typical grammar entry looks something like this:

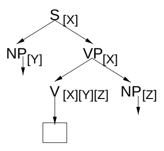

Figure 2: Grammar entry ITV: Intransitive Verb

When binding a lexical entry to an elementary tree, argument structure is preserved by unifying the *signature* of a lexical entry with the signature of its preterminal node. In the above case: (X=F, Y=Frier, Z=Fried).

4.3 Semantics

For our semantics, we follow Stone and Doran (1997) in using an ontologically promiscuous flat-semantics (Hobbs, 1985). The semantics of a given individual is simply the conjunction of all the semantics introduced by each lexical item, given adjustments for unification of argument structure.

Each individual is associated with a "semantic pool", which is simply a collection of propositions. Unlike the communicative goals of a traditional NLG system, the generator is under no commitment to realize these semantics. The relationship between an individual's semantic pool and its derivation tree's semantics is very flexible. In particular, there is no subsumption relationship either way between them.

Furthermore, in the beginning (the initialization phase) all semantic pools are initialised with a copy of the target semantics. This is ultimately what we hope our resulting poem to "be about". But as time progresses, each individual in a population can evolve and mutate its own semantic pool. Therefore each individual not only differs in form, but also in content.

4.4 Integrated, Incremental Generation

As mentioned above, unity of poetry demands that semantic, syntactic, and lexical information are available at every step of decision making. This calls for an integrated architecture, where there is no explicit decomposition of the process. In our implementation, this is reflected by the fact that individuals are complete structures which maintain all this information, and the semantic, syntactic and lexical operation functions can be applied in any order.

Like most stochastic search algorithms, the process starts with an initialisation phase. Provided with the input of a target semantics and target phonetic form as mentioned in Section 3.1, it then creates a collection of individuals, each corresponding to a minimally complete utterance that *more or less* conveys the target semantics.

What follows is a process of incrementally modifying the utterance to satisfy the target phonetic form while simultaneously attempting to maintain an approximation of the target semantics, as follows:

- During the evaluation phase, a collection of separate evaluators will analyse each individual for its surface form, phonetic information, and semantics, and assign a score (see Section 4.5).

- After every individual has been scored, the set is sorted. The higher ranked individuals spawn "children", copies of themselves which are mutated during the next phase. These children replace the lower ranked individuals, thus maintaining the set size.

- During the evolution phase, a collection of separate operators will be applied randomly on the aforementioned children (see Section 4.6).

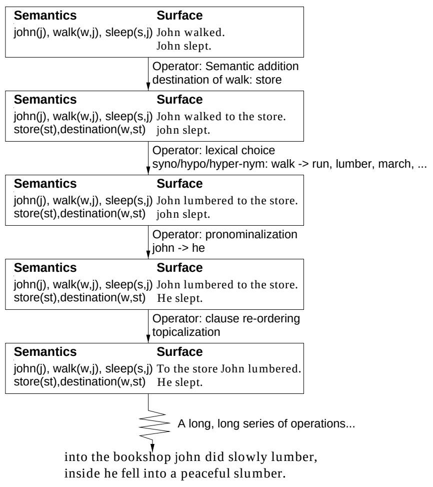
Semantics
Surface
john(j), walk(w,j), sleep(s,j)
John walked.
John slept.
Operator: Semantic addition
destination of walk: store
Semantics
Surface
john(j), walk(w,j), sleep(s,j)
store(st),destination(w,st)
John walked to the store.
john slept.
Operator: lexical choice
syno/hypo/hyper-nym: walk -> run, lumber, march, ...
Semantics
Surface
john(j), walk(w,j), sleep(s,j)
store(st),destination(w,st)
John lumbered to the store.
john slept.
Operator: pronominalization
john -> he
Semantics
Surface
john(j), walk(w,j), sleep(s,j)
store(st),destination(w,st)
John lumbered to the store.
He slept.
Operator: clause re-ordering
topicalization
Semantics
Surface
john(j), walk(w,j), sleep(s,j)
store(st),destination(w,st)
To the store John lumbered.
He slept.
A long, long series of operations...
into the bookshop john did slowly lumber,
inside he fell into a peaceful slumber.

Figure 3: Idealized diagram of a stochastic search

4.5 Evaluators

Unfortunately, at the moment we have only implemented an evaluator for rhythm: the metre evaluator. It works by first dividing the stress pattern of a given utterance into metrical feet of descending/falling rhythm. For instance, the line "There /**once** was a /**man** from Ma/**dras**", has a stress pattern of `(w,s,w,w,s,w,w,s)`. This can be divided into feet as `(w),(s,w,w),(s,w,w),(s)`. In other words, this line consists of a single upbeat (the weak syllable before the first strong syllable), followed by 2 *dactyls* (a classical poetry unit consisting of a strong syllable followed by two weak ones), and ended with a strong beat.

The evaluator compares the metrical configuration of an individual with the target phonetic form by first comparing their number of feet, penalizing those that are either too short or too long. Since each foot, with the exception of the upbeat, contains exactly one strong syllable, this effectively evaluates how close they match in number of strong syllables. It then compares the number of weak syllables between each corresponding foot, once again penalizing the discrepancies, but the penalty coefficient we impose here is less than that of the strong syllables. This provides a natural formalization of the heuristic that strong syllables dominate the effect of a line's metre, and a surplus or missing weak syllable here and there is quite acceptable. For example, (2) sounds more like (1) than (3) does:

(1) The /**cur**few /**tolls** the /**knell** of /**part**ing /**day**
(2) The /**cur**few /**tolls** the /**knell** of *the* /**part**ing /**day**
(3) The /**cur**few /**tolls** the /**knell** of /***long*** /**part**ing /**day**

4.6 Operators

We have currently implemented the following operators:

- Semantic explorer: this operator works with the semantic pool of an individual. Currently it just introduces random propositions into the pool, but with the help of a knowledge-base, it could introduce propositions that are conceptually related to what is already existing in the pool.

- Semantic realizer: This operator is one of the most important ones: it interfaces between the semantic pool and the actual built structure. The semantic realizer will randomly select a proposition from the pool and attempt to realize it by:
  - Selecting all lexical items that can convey the proposition,
  - For each lexical item, selecting all elementary trees that can be anchored by it,
  - For each elementary tree, selecting all nodes in the derivation tree where it can be applied (either adjoined or substituted),
  - Building a list of all these possible nodes and choosing one at random, and inserting the new lexicalized elementary tree at that position.
- Syntactic paraphraser: This operator works by randomly selecting an elementary tree in an individual's derivation tree and trying to apply a suitable paraphrase pair in the manner of Dras (1999). Since all adjunction and substitution information is kept relative to one's parent node in the derivation tree, adjusting for paraphrases (i.e. changing an elementary tree at a certain derivation tree node) is a simple matter of replacing the elementary tree and updating the addressing of the children.

  For example, if paraphrasing a sentence from active to passive form, this would involve exchanging the "Active Transitive Verb" elementary tree at the root to "Passive Transitive Verb", and updating the substitution addresses of the subject and object noun phrases so that the subject now moves to the end of the verb and the object moves to the front.

5 Examples and Discussion

While our system is still in a very early stage of implementation, particularly in terms of evaluators, operators, and linguistic resources, we already have some sample output to present.

For comparison, we first describe our previous attempt at implementing poetry generation, reported in Manurung (1999). This was not a stochastic search model but exhaustively produced all possible paraphrases using chart generation, while simultaneously pruning portions of the search space which were deemed ill-formed from an early stage. It also worked with the target specification of both phonetic form and semantics.

As an example, given the target semantics {cat(c), dead(c), bread(b), gone(b), eat(e,c,b), past(e)} and the target form shown in Section 3.1, but disregarding the rhyme scheme, the chart generator could produce, among others, the following "limerick":

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

Since this system does not have the equivalent of a semantic explorer operation (Section 4.6), the output semantics is always subsumed by the target semantics. Moreover, because the chart generator rules out ill-formed subscontituents during the bottom-up construction, the output form always matches exactly the target form. There are no partial solutions or imperfect poems.

In Example 1 below of our stochastic model, there is an appearance of *Luke*, despite not being mentioned in the target semantics. This is due to the semantic explorer operator, which randomly introduces new predicates into the semantic pool. By and large, though, the output does approximate the target semantics. Unfortunately, predicate argument structure and agreement is currently not considered, and this limits the treatment of semantics to a rather trivial account.

The target metre is not precisely followed, but the resulting form is arguably comparable with what a human might produce given the same task.

The resulting score is obtained from the metre evaluator (Section 4.5), and is out of a maximum 1.0.

Example 1:

Input	
Target semantics: {john(1), mary(2), dog(3), bottle(4), love(5,6,7), slow(8), smile(9,10)}	Target form: w,s,w,w,s,w,w,s, w,s,w,w,s,w,w,s
Output (Score: 0.893)	
Surface: *the bottle was loved by Luke* *a bottle was loved by a dog*	Stress: w,s,w,w,s,w,s, w,s,w,w,s,w,w,s

In Example 2, given no semantic input at all, the system will still produce output, but since the semantic explorer operator is still purely random, it resembles word salad. Furthermore, the second sentence is the ungrammatical "ran". This is because this generation effort was terminated prematurely by manual intervention, and at the last iteration before termination the semantic realizer chose to introduce the verb "ran", leaving its subject empty.

Example 2:

Input	
Target semantics: none	Target form: w,s,w,s,w,s,w,s,w,s, w,s,w,s,w,s,w,s,w,s
Output (Score: 0.845)	
Surface: *a warm distinctive season humble mellow smiled refreshingly slowly. ran.*	Stress: w,s,w,s,w,s,w,s,w,s, w,s,w,s,w,w,s,w,s

Although these results can hardly be called poems, nevertheless they succeed in showing how the stochastic hillclimbing search model manages to produce text that satisfies the given constraints, something very difficult for a random word-salad generator to achieve.

6 Related Work

The work reported in this paper is similar in some sense to the work of, among others: NITROGEN (Langkilde and Knight, 1998), a generator that employs a generate-and-test model, using a knowledge poor symbolic generator for producing candidate solutions and ranking them based on corpus-based information, and SPUD (Stone and Doran, 1996), a TAG-based generator that exploits opportunity arising between syntax and semantics, allowing generation of collocations and idiomatic constructs.

7 Conclusion

Poetry generation is different from traditional informative generation due to poetry's unity, which essentially means the satisfying of interdependent constraints on semantics, syntax and lexis. Despite our implementation being at a very early stage, the sample output succeeds in showing how the stochastic hillclimbing search model manages to produce text that satisfies these constraints.

Acknowledgements

The first author of this paper is being supported for his postgraduate studies by the World Bank QUE Project, Faculty of Computer Science, Universitas Indonesia. We would also like to thank the anonymous reviewers for their comments in preparing this paper.

References

Kim Binsted, Helen Pain, and Graeme Ritchie. Children's evaluation of computer-generated punning riddles. *Pragmatics and Cognition*, 5(2):309–358, 1997.

Margaret A. Boden. *The Creative Mind: Myths & Mechanisms*. Weidenfeld and Nicolson, London, 1990.

Marjorie Boulton. *The Anatomy of Poetry*. Routledge and Kegan Paul, London, 1982.

Audrey Daly. *Animal Poems*. Ladybird Books, Loughborough, 1984.

Koenraad De Smedt, Helmut Horacek, and Michael Zock. Architectures for natural language generation: Problems and perspectives. In Giovanni Adorni and Michael Zock, editors, *Trends in Natural Language Generation: An Artificial Intelligence Perspective*, number 1036 in Springer Lecture Notes in Artificial Intelligence, pages 17–46. Springer-Verlag, Berlin, 1996.

Mark Dras. *Tree Adjoining Grammar and the Reluctant Paraphrasing of Text*. PhD thesis, Macquarie University, Australia, 1999.

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

Jerry Hobbs. Ontological promiscuity. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 61–69, Chicago, Illinois, 1985. The Association for Computational Linguistics.

Aravind K. Joshi and Yves Schabes. Tree adjoining grammars and lexicalized grammars. In Maurice Nivat and Andreas Podelski, editors, *Tree Automata and Languages*. Elsevier Science, 1992.

Irene Langkilde and Kevin Knight. The practical value of n-grams in generation. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, Niagara-on-the-Lake, Ontario, 1998.

Samuel R. Levin. *Linguistic Structures in Poetry*. Number 23 in Janua Linguarum. 's-Gravenhage, 1962.

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

Chris Mellish, Alistair Knott, Jon Oberlander, and Mick O'Donnell. Experiments using stochastic search for text planning. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, Niagara-on-the-Lake, Ontario, 1998.

Nicolas Nicolov. *Approximate Text Generation from Non-Hierarchical Representations in a Declarative Framework*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1998.

Ehud Reiter. Has a consensus on NL generation appeared? and is it psycholinguistically plausible? In *Proceedings of the Seventh International Natural Language Generation Workshop*, pages 163–170, Kennebunkport, Maine, 1994. Springer-Verlag.

Mike Sharples. An account of writing as creative design. In Michael Levy and Sarah Ransdell, editors, *The Science of Writing: Theories, Methods, Individual Differences and Applications*. Lawrence Erlbaum, 1996.

Matthew Stone and Christine Doran. Paying heed to collocations. In *Proceedings of the Eighth International Workshop on Natural Language Generation*, pages 91–100, Brighton, 1996.

Matthew Stone and Christine Doran. Sentence planning as description using tree adjoining grammar. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 198–205, Madrid, Spain, 1997. The Association for Computational Linguistics.

Joseph Weizenbaum. Eliza - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.