Division of Informatics, University of Edinburgh

Institute for Communicating and Collaborative Systems

A Flexible Integrated Architecture For Generating Poetic Texts

by

Hisar Manurung, Graeme Ritchie, Henry Thompson

Informatics Research Report EDI-INF-RR-0016

Division of Informatics **May 2000**
http://www.informatics.ed.ac.uk/

A Flexible Integrated Architecture For Generating Poetic Texts

Hisar Manurung, Graeme Ritchie, Henry Thompson

Informatics Research Report EDI-INF-RR-0016

DIVISION *of* INFORMATICS
Institute for Communicating and Collaborative Systems

May 2000

Appears in Proceedings of the Fourth Symposium on Natural Language Processing (SNLP 2000), Chiang Mai, Thailand 10-12 May 2000

Abstract :

In this paper we describe a flexible approach to natural language generation that employs a stochastic hillclimbing search algorithm and an integrated architecture. We then discuss the benefits of this approach over existing, informative, goal-driven generation systems. We choose the generation of poetry as our research task domain, as it is a prime example of natural language that demands the degree of flexibility provided by our approach. Finally, we report and discuss results of our preliminary implementation work.

Keywords : NLG architectures, stochastic search, creativity, poetry generation

A FLEXIBLE INTEGRATED ARCHITECTURE FOR GENERATING POETIC TEXTS

Hisar Maruli Manurung, Graeme Ritchie, Henry Thompson
Institute for Communicating and Collaborative Systems, Division of Informatics, University of Edinburgh, 80 South Bridge Edinburgh EH1 1HN, Scotland, UK
hisarm@dai.ed.ac.uk, g.d.ritchie@ed.ac.uk, ht@cogsci.ed.ac.uk

In this paper we describe a flexible approach to natural language generation that employs a stochastic hillclimbing search algorithm and an integrated architecture. We then discuss the benefits of this approach over existing, *informative*,goal-driven generation systems. We choose the generation of poetry as our research task domain, as it is a prime example of natural language that demands the degree of flexibility provided by our approach. Finally, we report and discuss results of our preliminary implementation work.

Key words: NLG architectures, stochastic search, creativity, poetry generation.

1. MOTIVATION

Natural language generation (NLG) systems have typically been aimed at producing what might be termed *informative* texts: sentences, paragraphs or documents which attempt to convey precise facts about some situation or events. The generation system is considered successful if it can produce well-formed and lucid text that conveys a certain communicative goal. This goal is given as some form of non-linguistic input, and the system must find the linguistic structure and form that conveys it. Even where the requirement that the text must encode the semantics exactly is relaxed (as in the *approximate generation* of Nicolov (1998)), the aim is still to inform the audience as accurately as possible.

With the broadening of AI into the production of cultural artefacts (e.g. stories, jokes, poetry), and also *affective computing* (cf. Picard (1997)), the task of a NLG system may be less straightforward. Such texts (sometimes classed as "creative") may have a more diffuse communicative goal, or may be attempting to achieve some emotional effect rather than to inform. This may involve a trade-off between semantic, syntactic, lexical and phonetic factors, rather than simply having semantics supply a specification which other levels must conform to. Some work on incorporating affect into NLG already exists, e.g. Kantrowitz and Bates (1992), Walker et al. (1997), DiMarco et al. (1993).

Conventional informative NLG systems commonly decompose the generation task into several stages: *content determination*, *sentence planning*, *surface generation*, *morphology* and *formatting* (Reiter 1994). Furthermore, these stages are usually handled by different modules arranged in a *pipelined* or *interleaved* architecture (De Smedt et al. 1996). This often introduces a problem of architectural rigidity: the possible interactions and mutual constraints between the various linguistic levels may not be permitted.

In pipelined systems, the module for the early conceptual stage commits to its decisions and passes its output along to the surface realisation module, hoping it will succeed in finding an appropriate linguistic structure. Committing to such decisions early on can often lead to awkward results, such as failing to exploit an opportunity to convey two or more subgoals with one linguistic expression, or worse, being unable to convey some subgoals at all. Meteer (1991) refers to this as the "generation gap" problem, and Kantrowitz and Bates (1992) refers to these systems as "talking themselves into a corner".

An interleaved system tries to remedy this by providing a feedback loop from the surface realiser back to the conceptual module. However, this does not elegantly handle certain multi-

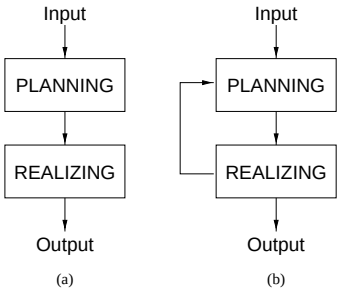

FIGURE 1. (a) Pipelined architecture (b) Interleaved architecture

level linguistic phenomena, which require a combination of many individual local effects that contribute to create a global effect.

These forms of inflexibility can be a serious issue when tackling more "artistic" (as opposed to "informative") types of text.

We are exploring ways of generating *poetry*, which provides a particularly extreme form of these problems. A poetic text must not only convey a communicative goal but also satisfy figurative, form, and phonetic constraints, and the *unity* of poetry means that interdependencies between semantics, syntax, lexis and phonetics are at their strongest.

2. THE TASK DOMAIN

2.1. Poetry generation

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 abstract, complex, "high-brow" poetry. However, we claim that this unity is inherent even in traditional forms of poetry, for example, the following stanza of Robert Burns' *A Red, Red Rose*:

O my Luve's like a red, red rose
That's newly sprung in June:
O my Luve's like the melodie
That's sweetly play'd in tune!

Essentially unity here means that the poem "works" due to a combination of features at the surface level (the rhyming of *June* and *tune*, the repetition of *red* to fit the rhythm), rhetorical level (*coupling* effect of repetitive forms in first and third lines, and second and fourth lines), and semantics level (describing love through analogies with red roses and music).

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 the proposals of 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 (see section 3 below), 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. The problems of poetic text

There are various ways in which poetic text makes particular demands on an NLG system, as indicated in section 1 above. More fully, these are:

1. The high occurrence of interdependent linguistic phenomena requires simultaneous consideration of semantics, syntax and lexis to produce a poem.
2. Conventional informative NLG systems require a given well-specified communicative goal as its starting point. In poetry, however, there may not be a well-defined message to be conveyed (see section 2.1).
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. This, in turn, leads to a problem of efficiently searching these resources for a text that satisfies all constraints.
4. One of the main difficulties lies in the objective evaluation of the output text (and hence the performance of the generation system). The question of measuring text quality arises for existing NLG systems, but is much more pronounced in evaluating poetry: it is unclear how one objectively evaluates if something is a poem or not.

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).

2.3. Limiting our aims

There is no clear definition of what counts as poetry, but the texts we are focussing on are those more structured verses used in relatively simple and traditional poetry within our own cultures, as typified by the verse in section 2.1. Such texts involve a highly regular occurrence of syntactic and phonetic patterns, such as metre, rhyme, and alliteration, and hence are a suitably demanding test for our proposed architecture.

3. INTEGRATION AND EVOLUTION

3.1. Overview

We have chosen a scheme in which all the levels of a sentence (semantic, syntactic, etc.) are represented simultaneously (section 3.2). Operators can then be applied to this structure randomly, to mutate or extend the linguistic content (section 3.3). The effects of these operators are evaluated at each step to assess whether the amended structure is an improvement (section 3.3). The cycle is then repeated (section 3.3).

3.2. Linguistic representation

Integrated structure. As mentioned above in section 2.2, the unity of poetry demands that semantic, syntactic, and lexical information are simultaneously considered at every step of decision making. Furthermore, the exploratory nature of the creative writing process would suggest that it should be possible to apply semantic, syntactic and lexical operations in any order. Both pipelined and interleaved architectures fail to provide for this.

We therefore propose to model poetry generation as a process in which there is, at every stage, a full representation of *all* the levels of the text (semantic, syntactic, phonetic). This representation can then be operated on in various ways, at *any* of the available levels, and in any order. The applications of the operators are viewed as moves in an explicit search space. This is actually readopting what De Smedt et al. (1996) call an *integrated architecture*, where there is no explicit modular decomposition of the generation process.

Syntax. Our central idea of an integrated representation and an evolutionary processing method (section 3.3 below) are not dependent on any particular linguistic representation. However, to allow concrete testing it was necessary to select a grammatical formalism. We have opted for Lexicalized Tree Adjoining Grammar (LTAG). For reasons of space, however, we will not explain this formalism in depth; see Joshi and Schabes (1992) for details.

Ordinary syntax trees are conventionally used to represent the complete constituent structure of a sentence (or phrase). In Tree Adjoining Grammar, there is, in addition to such *elementary trees*, a *derivation tree*, which 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 2).

In our example, the root node of the derivation tree introduces the verb, and its two siblings then introduce the subject and object noun phrases. The edges signify at which NP node the child gets substituted into, effectively indicating which is the subject and which is the object.

The common way to deal with TAG trees is by repeatedly performing adjunction and substitution (creating a *derived* tree), while having a derivation tree as a record-keeping "roadmap" of how the derived tree is created. However, the derived tree alone does not allow us to change or delete portions of our text (see section 3.3 below), since there is no way to "un-adjoin" subtrees. We must always refer back to the derivation tree. For our purposes, 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.

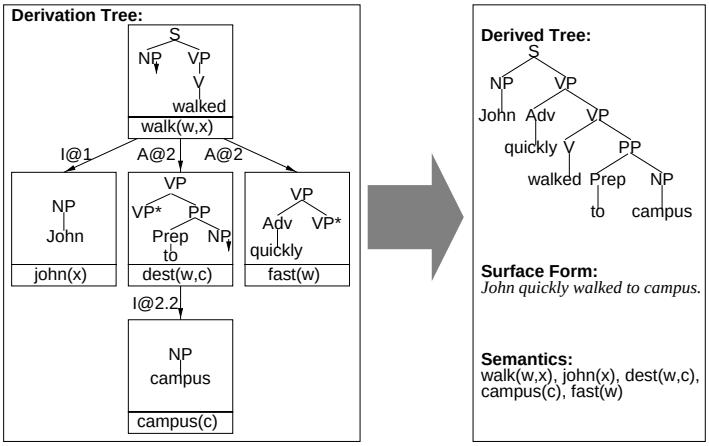

FIGURE 2. The LTAG derivation tree as the main data structure

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).
- The derivation tree provides an elegant mechanism for deletions and amendments. 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.

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.

The generation process maintains a "semantic pool", which is simply a collection of propositions. The generator is under no commitment to realize these semantics (unlike a traditional NLG system with strict communicative goals). The relationship between the semantic pool and the derivation tree's semantics is very flexible. In particular, there is no subsumption relationship either way between them.

In the beginning (the initialization phase) the semantic pool is initialised with a copy of the target semantics. This is ultimately what we hope our resulting poem to "be about". But as time progresses, the generation can evolve and mutate the semantic pool, thereby (probably) moving away from the initial target.

3.3. Stochastic Hillclimbing Search

If we are representing the generation process as a search in which operators are applied to a structure, the problem is then to find a way to navigate through the prohibitively large search space. Our proposed solution is to employ a *stochastic hillclimbing* search, not merely for its relatively efficient performance, but especially because the existing models of creative writing (section 2.1) advocate an iterative, incremental process of refining and rewriting. Also, a process with some element of randomness to it seems perfectly suited to the creative aspect of poetry.

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.

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 4.2.

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)
```

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:

Identifying 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.

Quantifying 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.
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.

Evolution. 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 3.3, it then creates a collection of individuals, each corresponding to a minimally complete utterance that *more or less* conveys the target semantics. This initial semantic pool gives rise to an LTAG derivation tree as a result of application of any operators which map semantic propositions to tree structure (e.g. the *semantic realizer* described in section 4.3 below).

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 4.3:

- **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 3.2).

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.

As mentioned earlier, there is an initialisation phase in which each generation process (individual in our evolutionary population) is given an initial semantic pool of propositions which will guide its choice of content; as the evolution progresses, the various individuals may diverge in their semantic content.

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 evaluators analyses each individual for its surface form, phonetic information, and semantics, and assigns a score (see section 4.2).
- After every individual has been scored, the set is sorted by score in a descending order.
- During the evolution phase, the N highest ranking individuals spawn "children", which are mutations of themselves. These children replace the N lowest ranking individuals, thus maintaining the set size. N is a modifiable parameter.

4. THE IMPLEMENTATION

We are currently in the process of implementing our stochastic search model in Java. In this section we will first briefly discuss our chosen linguistic resources, e.g. lexicon and grammar, before describing the current implementation of our evaluation and evolution functions.

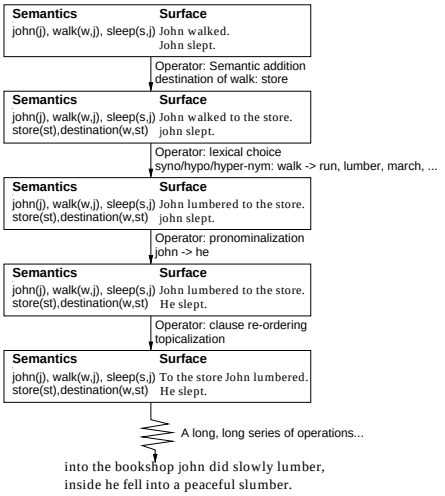

FIGURE 3. Idealized diagram of a stochastic search

4.1. 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 Figure 4.

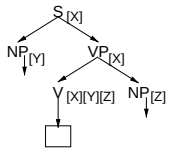

FIGURE 4. Grammar entry `ITV`: Intransitive Verb

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).

4.2. Evaluators

At the moment we have only implemented an evaluator for rhythm: the *metre evaluator*. Its function is to provide a quantitative measure of how well an individual satisfies the target form constraints.

For example, let us take as the target form the first line of the limerick specification in section 3.3: `w,s,w,w,s,w,w,s`, and compare it with the line "There /**once** was a /**man** from Ja/**kar**ta", which has a stress pattern of `w,s,w,w,s,w,w,s,w`.

Once we consider pairs of very different stress patterns, using a pattern-matching process on such flat lists does not give as good a measure as with pairs of *feet*-structured representation. For our purposes, a foot is a rhythmic unit consisting of a strong syllable followed by zero or more weak syllables, with the exception of the collection of weak syllables at the beginning of a line, known as the *upbeat*. Upbeats are not preceded by strong syllables. This definition of a foot corresponds to *descending* rhythm or *falling* rhythm. Note that the grouping of these feet units need not coincide with groupings of phrase structure.

Using this definition of feet, we can represent the previous two stress patterns as:
Target form feet: `(w),(s,w,w),(s,w,w),(s)`
Individual feet: `(w),(s,w,w),(s,w,w),(s,w)`

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**

4.3. 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

Although 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. The examples below should *not* be assessed for their poetic nature, as they barely rate as verse of any kind. What they show is that our architecture does indeed, as claimed, allow various types of linguistic constraints to control the stochastic search, resulting in texts which, as intended, are related to the semantic input and are close to the metric form stipulated.

In Example 1 below, we have provided a simple set of semantic predicates as the semantic pool and asked the system to produce a "poem" with the form similar to the first two lines of a limerick. The resulting score is obtained from the metre evaluator (section 4.2), and is out of a maximum score of 1.0. There are two particular points of interest from this example:

1. There is an appearance of the noun *poppies*, despite it 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.
2. The target metre is not precisely followed (the last foot in the second line is lacking a weak syllable), but the resulting form is arguably comparable with what a human might produce given the same task.

At present, agreement of arguments between semantic predicates is not considered, hence the assignment of arbitrary symbols to the target semantics' predicate arguments. For example, compare the target semantics of Example 1 and the target semantics of the example in section 6, where the subject of the verb is clearly the cat and the object is the bread. On one hand this limits the treatment of semantics to a rather trivial account, but on the other hand it allows for more flexibility in exploring the semantics search space.

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: 0.991)	
Surface: *a bottle was loved by a dog* *for poppies. the bottle smiled.*	Stress: w,s,w,w,s,w,w,s, w,s,w,w,s,w,s

EXAMPLE 1: Sample output given both semantic and form constraints.

As a baseline comparison, we also provide some purely random output of the system. To achieve this, we disabled the sorting process that normally occurs immediately after the evolution process, effectively crippling the hillclimbing heuristic. This is intended to simulate how a "blind", purely random traversal through the search space would fare in trying to satisfy the given constraints.

Using the exact same target semantics and target form from Example 1, we conducted 5 test runs, leaving the system to run for 15, 30, 60, 90, 120, and 150 seconds before terminating the process. Before each test run the system was always reinitialised. We also increased the population size to 100. The table below shows the average score of the population after termination along with the score and surface form of the best scoring individual.

What we can see is that without the hillclimbing heuristic, the blind random mutation does not produce output as good as the near-perfect score in Example 1.

Also, it is interesting to see how the system behaved when given more time to explore the search space. Because the texts are incrementally generated, they gradually grew longer in length as time went on. This is also caused by the fact that we have not yet implemented a "delete" operator. This also affects the scoring: the best results occurred when the system was terminated after 60 seconds (presented again as Example 2). After that, the score started to decrease again as the text lengths exceeded the target form specification. The best output of the last test run, for example, contains eight strong syllables, whereas the target form only has six.

Seconds	Average Score	Best Score	Best surface output
15	0.340	0.558	*the bottle sweetly sweetly smiled.*
30	0.372	0.615	*the bottle loved slowly John at clouds.*
60	0.455	0.672	*smiled. a bottle loved Mary. Mary sat.*
90	0.398	0.627	*a dog warm close smiled. music swelled.*
120	0.402	0.615	*the bottle loved John soft. a lambs smiled slowly.*
150	0.431	0.546	*a last bottle loved Mary. a dog smiled loudly. moss matured.*

Notice that the first sentence of Example 2 is the ungrammatical "*smiled*". 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 "*smiled*", leaving its subject empty. The fragment "ran" appears in Example 3 for the same reason.

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: 0.672)	
Surface: *smiled. a bottle loved* *Mary. Mary sat.*	Stress: s,w,s,w,s, s,w,s,w,s

EXAMPLE 2: The best produced output from total randomness

In Example 3, we give the system no target semantic input at all, and ask it to produce an iambic pentameter couplet. The system still produces output, something that a conventional informative NLG system wouldn't. However, since the semantic explorer operator is still purely random, it resembles "word salad".

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

EXAMPLE 3: Sample output given only form constraints.

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. This success is clearly evident when Examples 1 and 3 are contrasted with the baseline results achieved by the purely random generation.

6. PREVIOUS WORK

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 limerick target form shown in section 3.3, but disregarding the rhyme scheme, the chart generator could produce, among others, the following:

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 4.3), 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.

Although the example output shown here seems to compare favourably with those in section 5 above, it should be borne in mind that the non-stochastic system has very limited possibilities for further development, as it does not have the flexibility to allow various forms of knowledge to contribute and for multiple-level constraints to interact. On the other hand, our newer, evolutionary system is at a very early stage, and the sample outputs show little more than the fact that the overall processing model does indeed run as intended. This system, nevertheless, has the potential for future development owing to its generality and flexibility.

Our stochastic hillclimbing model is similar in some ways to other work, including the following:

NITROGEN This is 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 (Langkilde and Knight 1998). It is similar to our system in the sense that it generates many candidate solutions and then evaluates them, assigning each candidate solution a score, and finally sorts them based on this score. The difference is that the evaluation is a measure of the probability of a candidate solution with respect to some corpus data. Furthermore, there is no feedback or iterative process.

SPUD This is a TAG-based generator that exploits opportunity arising between syntax and semantics, allowing generation of collocations and idiomatic constructs (Stone and Doran 1996). In terms of representation, our approach is very similar to SPUD, as it also uses LTAG and maintains semantics as the conjunction of predicates found at each LTAG node. Moreover, SPUD also incrementally builds up a sentence by repeatedly adjoining or substituting new elementary trees. The difference, however, is that SPUD's algorithm is not stochastic, it does not generate several alternative candidate solutions. Instead, it employs a greedy algorithm, always adjoining or substituting in the "best" possible subscontituent, terminating when it has conveyed its input communicative goals. It also does not explicitly quantify scores over partial solutions.

GLINDA This is an integrated-architecture NLG system that is used in CMU's Oz Interactive Fiction project (Kantrowitz and Bates 1992). GLINDA produces narrative and dialogue from believable agents in the Oz storyworld, and therefore must consider issues of affect and style, differentiating it from conventional informative systems. Convicing believable agents must be able to express affect through its choice of word, e.g. expressing anger, sympathy, etc., and to achieve this GLINDA's planner must consider interdependent constraints at different levels of representation. In this respect, GLINDA employs an integrated architecture for the same reasons we do. Furthermore, (Kantrowitz and Bates 1992) also mentions poetic features as examples of texts requiring such an architecture, although it is not stated whether they handle this in their system. One difference from our approach is that GLINDA does not model the generation process as an explicit stochastic search, and its integrated architecture is more in terms of representation, where semantics and syntax are all represented using a uniform feature structure formalism.

7. CONCLUSION

If we are to tackle the generation of texts which are not straightforwardly informative, then we may have to consider a wider range of architectures and representations. We have outlined a very general and flexible approach, based on an integrated architecture and a stochastic hillclimbing search model. We believe this model allows the interaction of multiple constraints coupled with a less precise communicative goal. The main use we have had in mind for this system is the generation of simple poetry. Despite our implementation being at a very early stage, the sample output succeeds in showing how our method 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.

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.

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.

Chrysanne DiMarco, Graeme Hirst, and Manfred Stede. The semantic and stylistic differentiation of synonyms and near-synonyms. In Bonnie Dorr, editor, *Proceedings of the AAAI Spring Symposium on Building Lexicons for Machine Translation*, pages 114–121, Stanford, CA, USA, 1993. AAAI Press.

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

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.

Mark Kantrowitz and Joseph Bates. Integrated natural language systems. In Robert Dale, Eduard Hovy, Dietmar Rosner, and Oliviero Stock, editors, *Aspects of Automated Natural Language Generation: Proceedings of the Sixth International Workshop on Natural Language Generation*, number 587 in Springer Lecture Notes in Artificial Intelligence, pages 13–28, Trento, Italy, 1992. Springer-Verlag.

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.

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

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

Rosalind W. Picard. *Affective Computing.* MIT Press, Cambridge, MA, USA, 1997.

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.

Marilyn Walker, Janet Cahn, and Steve Whittaker. Improvising linguistic style: Social and affective bases of agent personality. In Jorg Muller, editor, *Proceedings of the First International Conference on Autonomous Agents*, pages 96–105, Marina del Rey, CA, USA, 1997. ACM SIGART.