Modelling control in generation*

Roger Evans†, David Weir‡, John Carroll‡, Daniel Paiva‡, Anja Belz†

†**University of Brighton**
Brighton, UK

‡**University of Sussex**
Brighton, UK

Abstract

In this paper we present a view of natural language generation in which the control structure of the generator is clearly separated from the content decisions made during generation, allowing us to explore and compare different control strategies in a systematic way. Our approach factors control into two components, a 'generation tree' which maps out the relationships between different decisions, and an algorithm for traversing such a tree which determines which choices are actually made. We illustrate the approach with examples of stylistic control and automatic text revision using both generative and empirical techniques. We argue that this approach provides a useful basis for the theoretical study of control in generation, and a framework for implementing generators with a range of control strategies. We also suggest that this approach can be developed into tool for analysing and adapting control aspects of other advanced wide-coverage generation systems.

1 Introduction

Natural Language Generation (NLG) has traditionally been most successful when applied to restricted domains using hand-crafted, carefully tuned systems[1] In such scenarios, the range of outputs to be generated is generally quite small, and controlling the generator is often trivial (for example, if the process is deterministic) or unimportant (if all solutions are equally good). Moving beyond such systems, however, requires control issues to be taken more seriously, in order to efficiently generate suitable output from among a range of options, some of which may not be appropriate. Delivering re-usable generation components, or components with sufficient breadth to be empirically tuned, requires a better understanding of control in generation.

The problem is exacerbated by the complexity of the generation process, and correspondingly of generation systems. The simple pipeline model of Reiter and Dale (2000) actually conceals a wide range of underlying approaches to generation (cf. (Mellish et al., 2006, section 2.1)), and a recent initiative to provide a 'reference architecture' for such systems (involving some of the present authors) abandoned any attempt to harmonise control aspects of the systems it studied (Mellish et al., 2006). In such a situation, it is difficult to see how any general statements, results or techniques relating to controlling generation can be developed, although Paiva and Evans (2005) report an approach that has the potential for wider applicability.

In the present paper, we introduce a view of the generation process which abstracts away from specific generation systems or architectures, to a point at which it is possible to separate control from content decisions in the generation process. This allows us to explore systematically different control strategies for constructing the same content (mapping from an input to the same output), and examine different approaches (e.g. generative, empirical) to the problem of controlling generation. The approach we develop is quite abstract, but we see potential for using it in at least three ways: as a theoretical framework for studying and understanding control; as a directly implementable generation framework in its own right, at least for small scale systems, and with suitable care over data representations, larger scenarios too; and as a tool for modelling the control behaviour of other generation systems, perhaps with a view to tuning them to a particular domain.

The paper is organised as follows. Section 2 introduces the model and discusses in general terms

* This research was part of COGENT (Controlled Generation of Text) project, supported by the EPSRC under grants GR/S24480/01 (Brighton) and GR/S24497/01 (Sussex).

[1] See, for example, Paiva (1998) for a survey of such systems.

the ways it can be used for studying control. Section 3 describes two examples of the model being used for practical generation research: using the pCRU system for exploring stylistic variation in weather forecasts, and developing an automatic text revision system for medical leaflets. Section 4 explores some wider issues and summarises the contribution of this work and possible future directions.

2 The control model

We start with a very general view of the generation process[2]. Generation takes an input and produces an output, which is a 'more linguistically instantiated' representation of the input (but we will not say precisely what that means — cf. (Evans et al., 2002; McDonald, 1993). In the process of doing this, the generator makes various decisions about the content of its output — it reaches a choice-point at which several options are possible and selects one to follow, and then reaches another choice point, and so on. In fact, this is all any generation algorithm does: visit choice points one after another and make a decision relating to the content of output at each one. Each decision may be constrained by the input, constrained by other decisions already made, or determined by the generation algorithm itself. These constraints may not reduce the number of choices at a choice point to exactly one, in which case the algorithm may be capable of returning several solutions, all deemed to be equally good, or no solutions at all. A simple way to think of this behaviour is as a tree structure, where each node corresponds to a choice-point, and each branch corresponds to a choice (leading to the next choice-point etc.).

But now let us abstract this process a little more. Consider first the input. It is common in generation to distinguish between different kinds of information that an algorithm may appeal to. Reiter and Dale (2000, p.43) specify four kinds of information in their generic generator definition: knowledge source, communicative goal, user model and discourse history. But we shall take a 'black box' view of the generator's input – the input is simply a collection of constraints on the process of producing the output. On this view, the generator does not 'transform' an input into an output, but creates an output 'from nowhere', guided in some way by the input information.

Turning now to the output, we assume this takes the form of some kind of data structure, for example a string of characters, a list of words, a parse tree or even a forest representing multiple realisations. We further assume that such a structure is in general constructed incrementally through applying a sequence of operations, which we shall call *content operations*, and that there may be some flexibility in how these operations are applied to produce 'the same' output (thinking algebraically, content operations may be commutative, or may combine together in various ways). The potential for applying these operations in different orders to produce the same output gives rise to one source of variation of control. We are not interested in what outputs are, but only how they are constructed, and so we think of them purely in terms of the content operations that give rise to them.

Adopting this very abstract view, we can conceptualise a generator as having the following principal components:

1. a set of *content operations* – these are possible operations that can be taken with respect to an output (for example, in a feature-based approach, a content operation might be to set a feature in the data-structure under construction to a particular value);
2. a way of organising content operations into a *generation tree* – each node in the tree corresponds to a choice-point for the generator, each branch is labelled with a set of content operations, and selecting that branch corresponds to choosing those operations on the output, and then traversing the branch to a new choice-point node;
3. an *input constraint algorithm* – at each choice point, the tree provides a set of possible options, and this algorithm applies constraints imposed by the input to reduce these options (possibly removing some or even all of them);
4. a *traversal algorithm* for the generation tree – at each choice point, this algorithm selects which branches to follow from among those still available after the intervention of the input constraint algorithm.

This view of generation may seem a little unusual, but it is, we believe, uncontroversial, in the sense that most implemented generators could in principle be 'unwrapped' into such a structure,

[2] In principle, this view applies far more generally than language generation, but we only consider NLG here.

though it may be very large, or even in principle unbounded. Its value from the present perspective is that it allows us to separate out control aspects from content in a uniform way. Specifically, we maintain that control resides in (a) the structure of the generation tree (that is, the grouping of content operations into choice point options, and the ordering of choice points in the tree) and (b) the traversal algorithm. In contrast, the content aspects of the algorithm arise from its input/output specification: the form of the input gives rise to the input constraint algorithm (independently of the control context, that is, the position in a generation tree at which the algorithm is applied), and the set of outputs, viewed as content operation combinations, gives rise to the set of content operations.

Our interest in these control components arises because they represent the *process* of generation, rather than, for example, the linguistic structure of the sentence generated. Understanding and controlling generation can thus be expressed and studied in terms of generation tree structures and traversal algorithms independently from actual systems and their linguistic representations. In the following subsections, we discuss these control components in more detail.

2.1 Generation trees

A **generation tree** is an unordered tree whose branches are labelled with sets of content operations, and whose leaves are labelled with output structures. Although we do not stipluate what content operations or output structures are, we do intend that the output structure is in some sense the result of applying content operations, and also allow that not all content operations are compatible with each other (for example, some may be mutually exclusive). This allows us to place a general consistency requirement on generation trees: a generation tree is **consistent** if each output structure labelling a leaf is compatible with the sequence of content operation sets labelling the path from the leaf to the root.

To make this definition more concrete, here are two possible interpretations:

- Consider a simple generator which uses a context-free grammar to specify its output language. Generation consists of starting at the initial category (S) and using grammar rules to expand non-terminals, until no non-terminals are left. One possible generation tree representation would have each node corresponding to a sentential form of the grammar (a string of terminals and nonterminals which could expand to a sentence in the grammar), with the root node corresponding to S, and each branch corresponding to the expansion of one or more non-terminals in the sentential form to produce a new sentential form. Here, the content operations are the individual rule expansions, and there is some flexibility in how they are applied to produce a particular output. Different generation trees correspond to different control strategies for grammar expansion – for example, always expanding just the left-most non-terminal would correspond to a left-to-right depth-first strategy, expanding all non-terminals simultaneously would result in a breadth-first strategy.

- Consider a generator that produces a complex feature structure (such as an HPSG sign (Pollard and Sag, 1994), or an MRS representation (Copestake et al., 2005)), by taking an underspecified input and further instantiating it. Here, content operations are feature specifications (of the form 'the value of feature f is v'), the output structures are feature structures and an output structure is compatible with a feature specification if it instantiates the feature with the specified value. Then a generation tree is consistent if each feature structure labelling a leaf node instantiates all the features specified on the path from the root to that leaf. Thinking more algorithmically, as we move down from the root towards a leaf, each decision instantiates some additional features in the output structure, until the structure is completed at the leaf.

Generation trees describe the process of generating outputs. Each path from root to leaf corresponds to a different 'run' of the generator leading to an output. Thus the structure of the generation tree bears no necessary relationship to the structure of the representations returned, or to any underlying linguistic formalism used to specify them. In the context-free generator described above, it is possible to read off parse trees associated with each leaf, but the generation tree itself is not directly related to the generator's grammar – in general there is no requirement that a generation tree is itself a derivation tree according to some grammar, or has any other formal property.

This notion of a generation tree has its roots in two earlier ideas. Paiva (2004) introduces 'tracing trees' to analyse the behaviour of a generation algorithm and discusses two variants, a tree representation of the generation of an individual sentence, and a single tree representation of all the sentences that can be generated. In that work and subsequent publications (Paiva and Evans, 2004; Paiva and Evans, 2005), the first variant is used, but the second variant is very similar in structure to generation trees as defined here. Belz (2004) introduces generation trees which are a direct precursor to the present definition, representing a generation algorithm directly as traversal of a tree mapping from partially specified to more specified content (this system is the basis of example 3.1, discussed below).

2.2 The structure of generation trees

Generation trees provide a framework for organising the decisions made by the generator. The structure of the generation tree is a key component in the control structure of the algorithm, because it captures two important factors:

- how content operations group together into choice-points – each node in the tree has a number of branches leading from it, and each branch is labelled with a set of content operations; selecting a branch commits to the content operations which label it as a group, and in preference to the content operations of other branches (although nothing prevents those operations occurring again further down the selected branch);
- the order in which choice-points are visited – each branch leads to another node and hence another choice;

The primary content of a generation tree is the set of output structures labelling leaves, and these are 'constructed' from the content operations on the path from the tree root. But where there is scope for varying how content operations are applied to produce the same output, it is possible to alter the structure of the tree without changing the output structures, in other words, changing the control structure without changing the content. As we saw in the context-free generator, above, depth-first and breadth-first strategies may have radically different trees, but result in the same output sentences.

A range of control considerations may come into play when we think about what the structure of the tree should be. Examples that can be directly measured in the tree structure include how balanced the tree should be, how many daughters a node may have, or how many content operations may label a single branch. These will each correspond to aspects of the processing, for example, how much variation in processing time there is for different outputs or how much the system commits in one step. Other aspects of structure may only be observable when the tree is combined with actual inputs, for example, how often this branch results in dead ends (choice-points with no admissible choices remaining). There may also be interactions with the traversal algorithm, since the tree shape provides the domain for making choices and the branch labels provide the way of discriminating choices. In the examples below, we shall consider traversal algorithms trained on corpus data – this will only work well if the choices available at a choice point correspond to 'events' in a corpus which can be effectively measured.

2.3 The traversal algorithm

The second control component in our model is the tree traversal algorithm. This gets applied once all other constraints (those imposed by the tree structure and the input) have been applied. So in effect, if there are still choices remaining at this point, they are 'free choices' as far as the generator tree and input structure are concerned. The simplest traversal algorithm is to accept all the remaining choices as valid, and pursue them all to return potentially multiple solutions. Serialised variants on this theme include pursuing any one choice, chosen at random (on the basis that they are all equally suitable), or imposing an order on the tree and then pursing a left-to-right depth-first traversal. In either of these cases, the possibility of dead-ends (no solutions at a descendant node) needs to be accommodated, perhaps by backtracking.

A more interesting scenario is where the traversal algorithm does not treat all options as the same, but imposes a preferential order on them. A familiar example of this would be where choices have probabilities associated with them, learned from some corpus, with the intention that the generator is attempting to generate sentences most like those that occur in the corpus. We will see several examples of this approach in section 3. We note here that this probability information is not part of the input, nor is it part of the generation tree as we have defined it above. However the traversal algorithm associates

these probabilities with individual choices-points in the tree, so that in effect it views the tree as being annotated with these probabilities. We refer to such a tree as a **trained** generation tree, while remembering that strictly speaking, according to our model the training parameters are part of the traversal algorithm, not the tree.

3 Examples of this control model

3.1 Example 1 – pCRU

pCRU (Belz, 2006; Belz, 2007) is a probabilistic language generation framework for creating NLG systems that contain a probabilistic model of the entire generation space, represented by a context-free underspecification grammar. The basic idea is to view all generation rules as context-free rules and to estimate a single probabilistic model from a corpus of texts to guide the generation process. In non-probabilistic mode, the generator operates by taking any sentential form of the grammar as an input and expanding it using the grammar to all possible fully specified forms, which are the outputs. Thus a pCRU grammar looks rather like a conventional grammar for syntax, except that it is used to model deep generation as well as surface realisation.

The probabilistic version of pCRU introduces a probability distribution over the generator decisions. This is achieved by using **treebank training**, that is, estimating a distribution over the expansion rules that encode the generation space from a corpus using two steps[3]:

1. *Convert corpus into multi-treebank:* use the underlying grammar to parse the corpus and annotate strings with the derivation trees obtained
2. *Train the generator:* Obtain frequency counts for each individual generation rule from the multi-treebank; convert into probability distributions over alternative rules

The resulting probability distribution is used in one of three ways to control generation.

1. *Viterbi generation:* undertake a Viterbi search of the generation forest for a given input, which maximises the joint likelihood of all decisions taken in the generation process. This selects the most likely generation process, but is considerably more expensive than the greedy modes.
2. *Greedy generation:* make the single most likely decision at each choice point (rule expansion) in a generation process. This is not guaranteed to result in the most likely generation process, but the computational cost is very low.
3. *Greedy roulette-wheel generation:* use a non-uniform random distribution proportional to the likelihoods of alternatives.

Belz (2006) describes an application of the system to weather forecast generation, and compares the different control techniques with human generation and a more traditional generate-and-test probabilistic architecture (Langkilde-Geary, 2002).

pCRU can be interpreted using the model introduce here in the following way (cf. the first example given in section 2.1). The **generation tree** is defined by all the possible derivations according to the grammar. Each node corresponds to a sentential form and each child node is the result of rewriting a single non-terminal using a grammar rule. Thus the **content operations** are grammar rules. The **content structures** are sentences in the grammar. The input is itself a sentential form which identifies where in the complete tree to start generating from, and the **input constraint algorithm** does nothing (alternatively, the input constraint algorithm only admits paths that pass through nodes associated with the input sentential form). Treebank training associates probabilities with all the possible expansion rules at a given choice-point, and three **traversal algorithms** Viterbi, greedy, and greedy roulette-wheel are defined.

3.2 Example 2 – Automatic Text Revision

Our second example of the generation model is in a system currently under development for Automatic Text Revision (ATR). The core idea here is that all the leaves in a generation tree are in a sense equivalent to each other (modulo any leaves rules out by particular input constraints). Hence they are, in a sense, paraphrases of each other. If they are text strings, the paraphrasing relationship may be obvious, but even generation trees over more abstract representations have this paraphrasing quality. Hence they support a notion of **generalised paraphrasing**: substituting one data structure used during generation for another one, whose equivalence

[3] See Belz (2006) for a more complete description.

is licensed by a generation tree.

We are using this type of generation tree to experiment with a model of ATR, intended for applications such as automatic improvement of drafts, or texts written by non-native writers, editorial adjustment to a house style, stylistic modification of a finished document for a different audience, or smoothing out stylistic differences in a multi-authored document.

Our ATR system uses generation trees whose output structures are Minimal Recursion Semantics (MRS) representations (Copestake et al., 2005), and whose content operations are instantiations of particular features in MRS structures (cf. example 2 in section 2.1). Thus a conventional use of such a generation tree might be to walk down the tree, instantiating MRS features at each choice-point guided by some input information, until a completely specified MRS realisation is reached at one of the leaves. However, for ATR we use the trees somewhat differently: the input to the process is one of the content structures labelling a leaf of the tree. The objective is to locate another output structure (a 'paraphrase') by walking up the tree from the input leaf (*degenerating*), and then taking alternative decisions to come back down to a different leaf (*re-generating*). In the absence of an 'input', the generation tree cannot distinguish between its output structures, and so the result might be a random alternative leaf. To overcome this, we need to add control in the sense introduced above.

The control questions we are exploring in this scenario are:

- How can we train a generation tree so that it is possible to locate 'better' paraphrases than the input?
- Can we structure a tree so that 'better' paraphrases are close to the input (that is, limited de-generation will always result in significant improvement)?

We address the first question using probabilistic training against a corpus of texts like the ones we want to paraphrase into (the 'target' corpus). In a simlar way to the pCRU example, we train the tree by estimating frequency distributions for different branches at each choice-point in the tree. In this case the branches are labelled with sets of feature instantiations. We train by parsing a corpus of target texts to MRS structures and then counting instances of the sets of feature instantiations on each branch, smoothing and normalising. In effect, we are using MRS features to characterise the 'style' of the corpus. Once we have done this, each leaf MRS can be assigned an overall probability score, and each interior node can compare the incoming score (from the branch leading to the input MRS) with scores for MRS's on the other branches, deciding whether to de-generate higher (in the hope of more improvement, but at greater cost) or start re-generating from this point.

The second question is interesting both from a computational complexity point of view (can we structure the tree to make the ATR more efficient?) and also because our goal is *minimal* paraphrasing: in many ATR contexts, it is bad to change the text unnecessarily, and so we are looking for minimal changes that improve the style. Our approach to this problem is to induce the generation trees themselves from a pool of MRS structures. We automatically construct different decision trees which gather together choices about groups of feature assignments in MRS's in different ways and different orders. This results in a set of generation trees with different structures, but all covering the same set of MRS leaves (and all with the same very abstract specification at their root – the fragment of MRS structure shared by *all* the MRS's). We then experiment with these trees to find which trees are more efficient for ATR from one text domain to another. Our long-term aim is to use the difference between our two corpora (for example pharmaceutical leaflet written for doctor and or for patients) to allow us to construct or select efficient generation trees automatically.

The description of this system makes reference to concepts in the generation model introduced here, but it may be useful to be more explicit, as we did with the pCRU example. The **output structures** are MRS's (more accurately, de-lexicalised MRS's) and the **generation trees** are labelled with abstractions over MRS's. The **content operations** are instantiations of MRS features, and typically there is more than one content operation on each branch in a generation tree (where features tend to co-occur with each other). The input is one of the leaf MRS's, **the input constraint algorithm** does nothing (as the influence of the input is entirely captured by its position in the tree) and the **traversal algorithm** uses the probability distributions to locate the best (or a better) paraphrase, relative to the training corpus.

4 Discussion

The examples just discussed illustrate how the model introduced in this paper offers a uniform perspective which is still flexible. The model separates out control and content aspects of the generation process and makes it possible to reason about and compare different control strategies in different generation scenarios. In the examples described here, we have illustrated generation trees over two different kinds of structure (strings in a language and MRS's), created in two different ways (generatively from the sentential forms of a grammar, and induced from data), with two kinds of content operation (grammar expansion and feature instantiation), two notions of training and four different traversal algorithms, and two very different notions of input. And of course this does not exhaust the range. Additional sources for generation trees, for example, may include manually crafted trees (somewhat akin to systemic grammars, although concerned more with control than linguistics) or trees induced from implemented generation systems (in the manner used by Paiva (2004) for his tracing trees).

This last option highlights one of several potential uses for the model proposed here. On the one hand our aim is to promote a clearer theoretical perspective on control issues in generation, by introducing a model in which they can be represented and explored in a common framework. The two examples given above constitute direct implementations of the model. Although they are both quite small scale systems, they indicate the prospects for direct implementation in larger systems (and help identify where scaling up may be problematic). But we also envisage using this approach as an analytic tool for studying and tuning other generation systems – by mapping a system into this generation model it may become possible to understand how it might most effectively be empirically controlled, for example, to identify deficiencies in the design, or to reason about computational complexity and efficiency.

Although we have discussed generation trees as if they cover the entire generation process (from non-linguistic input, whatever that may be, to textual realisation), nothing in the definitions above forces this. In practice a generation tree might be used to model just some part of the generation process (such as content selection or realisation), and multiple generation trees might be used collectively in a more complex algorithm. We also believe that this approach suggests a way forward for hybrid symbolic/probabilistic systems. By making the control structures of the generator more explicit it becomes easier to see how (and in how many ways) one can introduce empirically informed processing into even quite complex symbolic systems.

Finally, the model presented here is not 'just' a classical search algorithm, because the process is in a sense constructive – content actions are intended to further specify the solution, rather than distinguish between fully-formed solutions, and the internal nodes of the tree are intended not to be solutions in themselves. A search perspective is also possible, by associating internal nodes with the set of realisations they dominate and construing the algorithm as a search for the right set of realisations generation trees can be embedded in. But such a view presupposes the enumeration of the set of realisations which we are keen to avoid. This may seem like a procedural nicety, and of course in a sense it is, but in addressing control issues at all we are committed to looking at procedural aspects of a system. In fact, ultimately what we are trying to achieve here is to provide a theoretical and formal foundation for exactly those aspects of a system generally regarded as procedural details.

References

A. Belz. 2004. Context-free representational underspecification for NLG. Technical Report ITRI-04-08, Information Technology Research Institute, University of Brighton.

A. Belz. 2006. Probabilistic generation using representational underspecification. Technical Report NLTG-06-01, Natural Language Technology Group, CMIS, University of Brighton.

A. Belz. 2007. Probabilistic generation of weather forecast texts. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2007)*.

A. Copestake, D. Flickinger, I. Sag, and C. Pollard. 2005. Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3(2/3):281–332.

R Evans, P. Piwek, and L. Cahill. 2002. What is NLG? In *Proceedings of International Conference on Natural Language Generation*.

I. Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of*

the 12th International Conference on Natural Language Generation, pages 17–24.

D. McDonald. 1993. Issues in the choice of a source for natural language generation. *Computational Linguistics*, 19(1):191–197.

C. Mellish, D. Scott, L. Cahill, R. Evans, D. Paiva, and M. Reape. 2006. A reference architecture for natural language generation systems. *Journal of Natural Language Engineering*, 12(1):1–34.

D. Paiva and R. Evans. 2004. A framework for stylistically controlled generation. In *Proceedings of the International Conference on Natural Language Generation*, pages 120–129.

D. Paiva and R. Evans. 2005. Empirically-based control of natural language generation. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics*, pages 58–65.

D. Paiva. 1998. A survey of applied natural language generation systems. Technical Report ITRI-98-03, ITRI, University of Brighton.

D. Paiva. 2004. *Using Stylistic Parameters to Control a Natural Language Generation System*. Ph.D. thesis, University of Brighton.

Carl J. Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

E. Reiter and R. Dale. 2000. *Building Natural-Language Generation Systems*. Cambridge University Press.