Connectivity in Bag Generation

Arturo Trujillo and Simon Berry*
School of Computer and Mathematical Sciences
The Robert Gordon University, St Andrew Street
Aberdeen AB1 1HG
Scotland
{iat,cs5sby}@scms.rgu.ac.uk

Abstract

This paper presents a pruning technique which can be used to reduce the number of paths searched in rule-based bag generators of the type proposed by (Poznański et al., 1995) and (Popowich, 1995). Pruning the search space in these generators is important given the computational cost of bag generation. The technique relies on a connectivity constraint between the semantic indices associated with each lexical sign in a bag. Testing the algorithm on a range of sentences shows reductions in the generation time and the number of edges constructed.

1 Introduction

Bag generation is a form of natural language generation in which the input is a bag (also known as a multiset: a set in which repeated elements are significant) of lexical elements and the output is a grammatical sentence or a statistically most probable permutation with respect to some language model.

Bag generation has been considered within the statistical and rule-based paradigms of computational linguistics, and each has handled this problem differently (Chen and Lee, 1994; Whitelock, 1994; Popowich, 1995; Trujillo, 1995). This paper only considers rule based approaches to this problem.

Bag generation has received particular attention in lexicalist approaches to MT, as exemplified by Shake-and-Bake generation (Beaven, 1992; Whitelock, 1994). One can also envisage applications of bag generation to generation from minimally recursive semantic representations (Copestake et al., 1995) and other semantic frameworks which separate scoping from content information (Reyle, 1995). In these frameworks, the unordered nature of predicate or relation sets makes the application of bag generation techniques attractive.

A notational convention used in the paper is that items such as 'dog$_1$' stand for simplified lexical signs of the form (Shieber, 1986):

$$\begin{bmatrix} \text{CAT} = \mathbf{N} \\ \text{SEM} = \begin{bmatrix} \text{RELN} = \mathbf{dog} \\ \text{ARG1} = \mathbf{1} \end{bmatrix} \end{bmatrix}$$

In such signs, the semantic argument will be referred to as an 'index' and will be shown as a subscript to a lexeme; in the above example, the index has been given the unique type **1**.

The term index is borrowed from HPSG (Pollard and Sag, 1994) where indices are used as arguments to relations; however these indices may also be equated with discourse referents in DRT (Kamp and Reyle, 1993). As with most lexicalist generators, semantic variables must be distinguished in order to disallow translationally incorrect permutations of the target bag. We distinguish variables by uniquely typing them.

Two assumptions are made regarding lexical-semantic indexing.

Assumption 1 *All lexical signs must be indexed, including functional and nonpredicative elements (Calder et al., 1989).*

Assumption 2 *All lexical signs must be connected to each other. Two lexical signs are connected if they are directly connected; furthermore, the connectivity relation is transitive.*

Definition 1 *Two signs, A, B, are directly connected if there exist at least two paths, PathA, PathB, such that A:PathA is token identical with B:PathB.*

The indices involved in determining connectivity are specified as parameters for a particular formalism. For example, in HPSG,

*Now at SHARP Laboratories of Europe, Oxford Science Park, Oxford OX4 4GA. E-mail: simon@sharp.co.uk

play a major role in preventing the generation of incorrect translations.

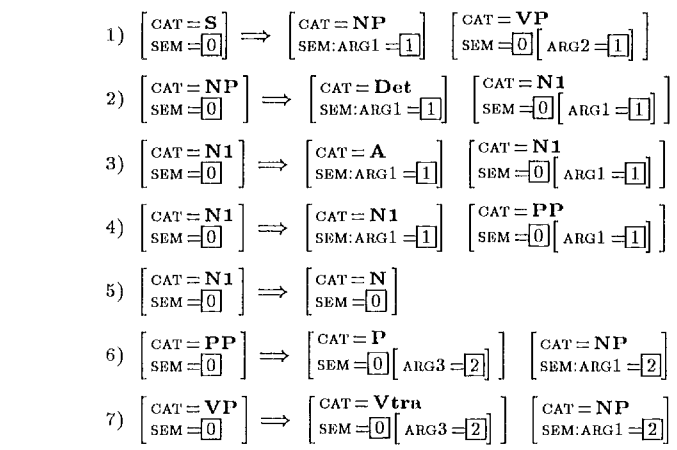


Figure 1: Simple unification grammar.

It will be shown that it is possible to exploit the connectivity Assumption 2 above in order to achieve a reduction in the number of redundant wfss constructed by both types of generator described in section 2.

3.1 Using Connectivity for Pruning

Take the following bag:

Ex. 2 $\{dog_1, the_1, brown_1, big_1\}$

(corresponding to 'the big brown dog'). Assume that the next wfss to be constructed by the generator is the NP 'the dog'. Given the grammar in Figure 1, it is possible to deduce that 'brown' can never be part of a complete NP constructed from such a substring. This can be determined as follows. If this adjective were part of such a sentence, 'brown' would have to appear as a leaf in some constituent that combines with 'the dog' or with a constituent containing 'the dog'. From the grammar, the only constituents that can combine with 'dog' are VP, Vtra and P. However, none of these constituents can have 'brown$_1$' as a leaf: in the case of P and Vtra this is trivial, since they are both categories of a different lexical type. In the case of the VP, 'brown$_1$' cannot appear as a leaf either because expansions of the VP are restricted to NP complements with 2 as their semantic index, which in turn would also require adjectives within them to have this index. Furthermore, 'brown$_1$' cannot occur as a leaf in a deeper constituent in the VP because such an occurrence would be associated with a different index. In such cases 'brown' would modify a different noun with a different index:

Ex. 3 $\{the_1, dog_1, with_{1,2}, the_2, brown_2, collar_2\}$

A naive implementation of this deduction would attempt to expand the VP depth-first, left to right, in order to accommodate 'brown' in a complete derivation. Since this would not be possible, the NP 'the dog' would be discarded. This approach is grossly inefficient however. What is required is a more tractable algorithm which, given a wfss and its associated sign, will be able to determine whether all remaining lexical elements can ever form part of a complete sentence which includes that wfss.

Note that deciding whether a lexical sign can appear outside a phrase is determined purely by the grammar, and not by whether the lexical elements share the same index or not. Thus, a more complex grammar would allow 'the man' from the bag

Ex. 4 $\{the_1, man_1, shaves_{e,1,1}, himself_1\}$

even though 'himself' has the same index as 'the man'.

3.2 Outer Domains

The approach introduced here compiles the relevant information offline from the grammar and uses it to check for connectivity during bag generation. The compilation process results in a set of (Sign,Lex,Bindings) triples called *outer domains*. This set is based on a unification-based phrase structure grammar defined as follows:

Definition 2 *A grammar is a tuple (N,T,P,S), where P is a set of productions $\alpha \Rightarrow \beta$, α is a sign, β is a list of signs, N is the set of all α, T is the set of all signs appearing as elements of β which unify with lexical entries, and S is the start sign.*

Outer domains are defined as follow:

Definition 3 $\{$ *(Sign,Lex,Binds)* | *Sign* $\in N \cup T$, *Lex* $\in T$ *and there exists a derivation* $\alpha \stackrel{*}{\Rightarrow} \beta_1 Sign' \beta_2 Lex' \beta_3$ *or* $\alpha \stackrel{*}{\Rightarrow} \beta_1 Lex' \beta_2 Sign' \beta_3$, *and Sign' a unifier for Sign, Lex' a unifier for Lex, and Binds the set of all path pairs <SignPath,LexPath> such that Sign':SignPath is token identical with Lex':LexPath*$\}$

Intuitively, the outer domains indicate that preterminal category Lex can appear in a complete sentence with subconstituent Sign, such that Lex is not a leaf of Sign. Using ideas from data flow analysis (Kennedy, 1981), predictive parser constructions (Aho et al., 1986) and feature grammar compilation (Trujillo, 1994) it is possible to construct such a set of triples. Outer domains thus represent elements which may lie outside a subtree of category Sign in a complete sentential

they would be indicated through paths such as SYNSEM:LOCAL:CONTENT:INDEX.

To ensure that only connected lexical signs are generated and analysed, the following assumption must also be made:

Assumption 3 *A grammar will only generate or analyse connected lexical signs.*

2 Bag Generation Algorithms

Two main types of rule-based bag generators have been proposed. The first type consists of a parser suitably relaxed to take into account the unordered character of the input (Whitelock, 1994; Popowich, 1995; Trujillo, 1995). For example, in generators based on a chart parser, the fundamental rule is applied only when the edges to be combined share no lexical leaves, in contrast to requiring that the two edges have source and target nodes in common. The other type of generator applies a greedy algorithm to an initial solution in order to find a grammatical sentence (Poznański et al., 1995).

2.1 Redundancy in Bag Generation

One disadvantage with the above generators is that they construct a number of structures which need not have been computed at all. In building these structures, the generator is effectively searching branches of the search space which never lead to a complete sentence. Consider the the following input bag:

$\{dog, barked, the, brown, big\}$

Previous researchers (Brew, 1992; Phillips, 1993) have noted that from such a bag, the following strings are generated but none can form part of a complete sentence (note that indices are omitted when there is no possibility of confusion; # indicates that the substring will never be part of a complete sentence):

Ex. 1 # *the dog*
the dog barked
the brown dog

For simple cases in chart based generators such unnecessary strings do not create many problems, but for longer sentences, each additional substring implies a further branch in the search tree to be considered.

Since the computational complexity of the greedy bag generator (Poznański et al., 1995) is polynomial (i.e. $\mathcal{O}(n^4)$), the effect of redundant substructures is not as detrimental as for parser based generators. Nevertheless, a certain amount of unnecessary work is performed. To show this, consider the test-rewrite sequence for Example 1:

Test: dog barked the brown big
Rewrite: __ barked the **dog** brown big
Test: barked (the dog) brown big
Rewrite: __ (the dog) **barked** brown big
Test: ((the dog) barked) brown big
Rewrite: the **brown** dog barked __ big
Test: ((the (brown dog)) barked) big
Rewrite: the **big** (brown dog) barked __
Test: ((the (big (brown dog))) barked) *(terminate)*

In this sequence double underscore (__) indicates the starting position of a moved constituent; the moved constituent itself is given in bold face; the bracketing indicates analysed constituents (for expository purposes the algorithm has been oversimplified, but the general idea remains the same).

Now consider the step where 'brown' is inserted between 'the' and 'dog'. This action causes the complete structure for 'the dog barked' to be discarded and replaced with that for 'the brown dog barked', which in turn is discarded and replaced by 'the big brown dog barked'.

2.2 Previous Work

A number of pruning techniques have been suggested to reduce the amount of redundancy in bag generators. Brew (1992) proposed a constraint propagation technique which eliminates branches during bag generation by considering the necessary functor-argument relationships that exist between the component basic signs of categorial signs. These relationships form a graph indicating the necessary conditions for a lexical item to form part of a complete sentence. Such graphs can be used to eliminate the substrings in Example 1. Unfortunately the technique exploits specific aspects of categorial grammars and it is not clear how they may be used with other formalisms.

Trujillo (1995) adapts some of Brew's ideas to phrase structure grammars by compiling Follow functions and constructing adjacency graphs. While this approach reduces the size of the search space, it does not prune it sufficiently for certain classes of modifiers.

Phillips (1993) proposes handling inefficiency at the expense of completeness. His idea is to maintain a queue of modifiable constituents (e.g. N1s) in order to delay their combination with other constituents until modifiers (e.g. PPs) have been analysed. While practical, this approach can lead to alternative valid sentences not being generated.

3 Connectivity Restrictions

In searching for a mechanism that eliminates unnecessary wfss, it will be possible to use indices in lexical signs. As mentioned earlier, these indices

derivation. The following definition specifies how outer domains are used:

Definition 4 *A lexical sign Lex′ is in the outer domain of Sign′ iff there is a triple (Sign,Lex,Binds) in outer domains such that Sign and Lex unify with Sign′ and Lex′ respectively, and there is at least one pair <PathS,PathL> ∈ Binds such that Sign′:PathS unifies with Lex′:PathL.*

In compiling outer domains, *inner domains* are used to facilitate computation. Inner domains are defined as follows:

Definition 5 { *(Sign,Lex,Binds)* | *Sign* $\in N \cup T$, *Lex* $\in T$ *and there exists a derivation* $\alpha \stackrel{*}{\Rightarrow} \beta_1 Lex' \beta_2$, *with Sign′ a unifier for Sign, Lex′ a unifier for Lex, and Binds the set of all path pairs <SignPath,LexPath> such that Sign′:SignPath is token identical with Lex′:LexPath*}

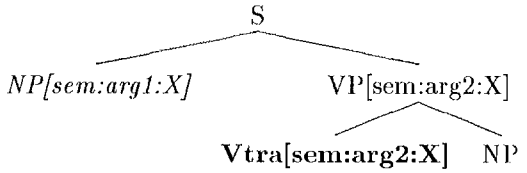
The inner domains thus express all the possible terminal categories which may be derived from each nonterminal in the grammar.

To be able to exploit connectivity during generation, inner and outer domains contain only triples in which Binds has at least one element. In this way, only those lexical categories which are directly connected to the sign are taken into account; the implication of this will become clearer later.

As an example, the outer domain of NP as derived from the above grammar is:

(NP[sem:arg1:X],Vtra[sem:arg2:Y],
{<sem:arg1,sem:arg2>})
(NP[sem:arg1:X],Vtra[sem:arg3:Y],
{<sem:arg1,sem:arg3>})
(NP[sem:arg1:X],P[sem:arg3:Y],
{<sem:arg1,sem:arg3>})

This set indicates that for any NP, the only terminal categories not contained in the subtree with root NP, and with which the NP shares a semantic index, are Vtra and P. For instance, the first triple arises from the following tree:



3.3 Pruning through Outer Domains and Connectivity

The pruning technique developed here operates on grammars whose analyses result in connected leaves.

Consider some wfss W constructed from a bag B and with category C; this category, in the form of a sign, will include syntactic and lexical-semantic information. Such a wfss will have been constructed during the bag generation process. Now, either W includes all the input elements as leaves, in which case W constitutes a complete sentence, or there are elements in the input bag which are not part of W. In the latter case, for bags obeying Assumption 2, the following condition holds for any W that can form part of a complete sentence:

Condition 1 *Let L be the set of leaves appearing in W, let G be the graph (V,E), where V* = {*C*} ∪ *B* − *L, and E* = { {*x,y*} | *x,y* ∈ *V and y is in the outer domain of x*}. *Then G is connected.*

To show that this condition indeed holds, consider a grammatical ordering of some input bag B, represented as the string W:

$\alpha..\gamma\delta..\omega$

By Assumption 2, the lexical elements in the bag, and therefore in any grammatical ordering of it, are connected. Now consider reducing this string using the production rule:

$D \Rightarrow \gamma\delta$

to give the string W′:

$\alpha..D..\omega$

In this case, the signs in W′ will also be connected. This can be shown by contradiction:

Proof 1 *Assume that there is some sign ζ in W′ to which D is not connected. Then grammar G would allow disconnected strings to be generated, contrary to Assumption 3. This is because D would not be able to rewrite* $\gamma_1\delta_1$ *in such a way that both daughters were connected to ζ, leading to a disconnected string.*

The situation in string W′ is analogous to that in Condition 1. By identifying signs which are directly connected in E, it is possible to determine whether E is connected and consequently whether C can form part of a complete derivation. Instead of simply comparing the value of index paths, it is more restrictive to use outer domains since they give us precisely those elements which are directly connected to a sign and are in its outer domain.

3.4 Example

Consider Example 2. To eliminate the wfss 'the dog' from further consideration, a connected graph of lexical signs is constructed before generation is started (Figure 2). This graph is built by using the outer domain of each lexical element to decide which of the remaining elements could possibly share an index with it in a complete sentence.

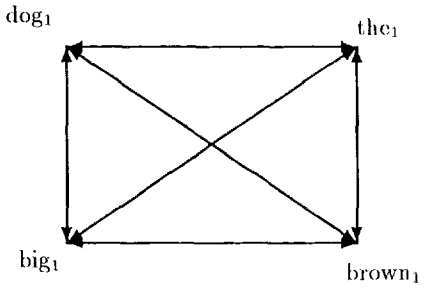


Figure 2: Initial connected graph.

When a new wfss is constructed during generation, say by application of the modified fundamental rule or during the rewrite phase in a greedy algorithm, this initial graph is updated and tested for connectivity. If the updated graph is not connected then the proposed wfss cannot form part of a complete sentence. Updating the graph involves three steps. Firstly every node in the graph which is a leaf of the new wfss is deleted, together with its associated arcs. Secondly, a new node corresponding to the new wfss is added to the graph. Finally, a new arc is added to the graph between the new node and every other node lying in its outer domain. The updated (disconnected) graph that ensues after constructing 'the dog' is shown in Figure 3; this NP is therefore rejected.

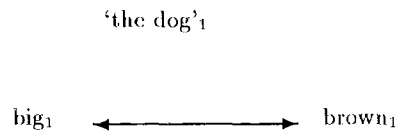


Figure 3: Updated disconnected graph after the wfss 'the dog' is constructed.

4 Compiling Connectivity Domains

For reasons of space, the computation of outer domains cannot be described fully here. The broad outline, however, is as follows. First, the inner domains of the grammar are calculated. This involves the calculation of the fixed point of set equations, analogous to those used in the construction of First sets for predictive parsers (Aho et al., 1986; Trujillo, 1994). Given the inner domains of each category in the grammar, the construction of the outer domains involves the computation of the fixed point of set equations relating the outer domain of a category to the inner domain of its sisters and to the outer domain of its mother, in a manner analogous to the computation of Follow sets.

During computation, the set of Binds is monotonically increased as different ways of directly connecting sign and lexeme are found.

5 Results

The above pruning technique has been tested on bags of different sizes including different combinations of modifiers. Sentences were generated using two versions of a modified chart parser. In one, every inactive edge constructed was added to the chart. In the other, every inactive edge was tested to see if it led to a disconnected graph; if it did, then the edge was discarded. The results of the experiment are shown in Table 1. The implementation was in Prolog on a Sun SparcStation 10; the generation timings do not include garbage collection time. The grammar used for the experiment consisted of simplified, feature-based versions of the ID rules in GPSG; there were 18 rules and 50 lexical entries. Compilation of the outer domains for these rules took approximately 37 minutes, and the resulting set occupies 40K of memory. In the general case, however, the size of the outer domains is $\mathcal{O}(n^2)$, where n is the number of distinct signs; this number can be controlled by employing equivalence classes of different levels of specificity for pre-terminal and non-terminal signs.

	Chart Gen.		*+ Pruning*	
Bag size	*Time*	*Edges*	*Time*	*Edges*
2	0.1	15	0.1	15
4	0.3	37	0.4	36
7	1.5	103	2.0	99
7	0.9	72	1.0	67
11	5.1	213	3.9	138
12	2.6	133	3.4	123
15	9.0	294	7.2	186
15	17.6	448	11.1	253
17	2.3	126	2.6	105

Table 1: Effect of pruning (times in secs).

Only one reading was generated for each bag, corresponding to one attachment site for PPs. The table shows that the technique can yield reductions in the number of edges (both active and inactive) and time taken, especially for longer sentences, while retaining the overheads at an acceptable level.

6 Conclusion

A technique for pruning the search space of a bag generator has been implemented and its usefulness shown in the generation of different types of constructions. The technique relies on a connectivity constraint imposed on the semantic relationships

expressed in the input bag. In order to apply the algorithm, outer domains needed to be compiled from the grammar; these are used to discard wfss by ensuring lexical signs outside a wfss can indeed appear outside that string.

Exploratory work employing adjacency constraints during generation has yielded further improvements in execution time when applied in conjunction with the pruner. If extended appropriately, these constraints could prune the search space even further. This work will be reported at a later date.

Acknowledgments

Two anonymous reviewers provided very useful comments; we regret not being able to do justice to all their suggestions.

References

A. V. Aho, R. Sethi, and J. D. Ullman. 1986. *Compilers - Principles , Techniques, and Tools.* Addison Wesley, Reading, MA.

J. L. Beaven. 1992. *Lexicalist Unification Based Machine Translation.* Ph.D. thesis, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK.

C. Brew. 1992. Letting the cat out of the bag: Generation for Shake-and-Bake MT. In *Proceedings of the 14th COLING*, pages 610–16, Nantes, France, August.

J. Calder, M. Reape, and H. Zeevat. 1989. An algorithm for generation in unification categorial grammar. In *Proceedings of the Fourth European Conference of the ACL*, pages 233–40, Manchester, England, April.

H.H. Chen and Y.S. Lee. 1994. A corrective training algorithm for adaptive learning in bag generation. In *New Methods in Language Processing*, Manchester, UK.

A. Copestake, D. Flickinger, R. Malouf, S. Riehemann, and I. Sag. 1995. Translation using minimal recursion semantics. In *Proceedings of the 6th International Conference on Theoretical and Methodological Issues in Machine Translation*, Leuven, Belgium, July.

H. Kamp and U. Reyle. 1993. *From Discourse to Logic – Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*, volume 42 of *Studies in Linguistics and Philosophy.* Kluwer Academic, Dordrecht, The Netherlands.

Ken Kennedy. 1981. A survey of data flow analysis techniques. In Muchnick and Jones (1981), chapter 1, pages 5–54.

Steven S. Muchnick and Neil D. Jones, editors. 1981. *Program Flow Analysis: Theory and Applications.* Software. Prentice-Hall, Englewood Cliffs, NJ.

J. D. Phillips. 1993. Generation of text from logical formulae. *Machine Translation*, 8(4):209–35.

C. Pollard and I. Sag. 1994. *Head Driven Phrase Structure Grammar.* Chicago University Press, IL.

Fred Popowich. 1995. Improving the efficiency of a generation algorithm for Shake and Bake machine translation using Head-Driven Phrase Structure Grammar. In *Proceedings of Natural Language Understanding and Logic Programming V*, Lisbon, Portugal, May.

V. Poznański, J. L. Beaven, and P. Whitelock. 1995. An efficient generation algorithm for lexicalist MT. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, Boston, MA, June.

Uwe Reyle. 1995. On reasoning with ambiguities. In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics*, pages 1–15, Dublin, Ireland, March.

C. J. Rupp, M. A. Rosner, and R. L. Johnson, editors. 1994. *Constraints, Language and Computation.* Academic Press, London.

S. M. Shieber. 1986. *An Introduction to Unification-based Approaches to Grammar*, volume 4 of *CSLI Lecture Notes.* CSLI, Stanford, CA.

A. Trujillo. 1994. Computing FIRST and FOLLOW functions for Feature-Theoretic grammars. In *Proceedings of the 15th COLING*, pages 875–80, Kyoto, Japan, August.

A. Trujillo. 1995. *Lexicalist Machine Translation of Spatial Prepositions.* Ph.D. thesis, Computer Laboratory, University of Cambridge, April.

Pete Whitelock. 1994. Shake-and-bake translation. In Rupp et al. (1994), pages 339–59.