Generating and selecting grammatical paraphrases

Claire Gardent
CNRS/LORIA
Nancy, France
claire.gardent@loria.fr

Eric Kow
INRIA/LORIA
Nancy, France
eric.kow@loria.fr

Abstract

Natural language has a high paraphrastic power yet not all paraphrases are appropriate for all contexts. In this paper, we present a TAG based surface realiser which supports both the generation and the selection of paraphrases. To deal with the combinarial explosion typical of such an NP-complete task, we introduce a number of new optimisations in a tabular, bottom-up surface realisation algorithm. We then show that one of these optimisations supports paraphrase selection.

1 Introduction

As is well known, natural language has a very high paraphrastic power so that the same core meaning can be expressed in many different ways [Gross, 1975; Mel'čuk, 1988]. Yet not all paraphrases are appropriate for all contexts. So for instance, a sentence and its converse (1a) express the same core meaning and so can be considered paraphrases of each other. Yet as example (1b) illustrates, they are not interchangeable in the context of a control verb:

(1) a. John borrowed a book from Mary.
≡ Mary lent a book to John

b. Peter persuaded John to borrow a book from Mary.
≢ Peter persuaded Mary to lend a book to John

Similarly, a canonical and a cleft sentence (2a) communicate the same core meaning yet a contrastive context (2b) only admits the cleft version.

(2) a. John looks at Mary.
≡ It is Mary that John looks at

b. * It is not Sarah, John looks at Mary.
It is not Sarah, it is Mary that John looks at

More generally, the anaphoric potential (that is, the discourse status of the entities being talked about) of the preceding discourse, its structure, the presence of an embedding verb or of a given subordinating or coordinating conjunction are all factors which may restrict the use of paraphrases. To preserve completeness, it is therefore important that a generator be able to produce paraphrases in a systematic fashion.

On the other hand, it is also well known that surface realisation (the task of producing the set of sentences associated by a grammar with a given semantic representation) is NP-complete [Brew, 1992].

In this paper, we present a TAG based surface realiser which supports both the generation and the selection of *grammatical* paraphrases (section 2 and 3). To deal with the resulting combinatorics, we introduce a number of new optimisations (section 4). We then show how one of these optimisations can be used to support the selection of contextually appropriate paraphrases (section 5). Finally, we relate our approach to similar proposals and show that it compares favorably in terms of efficiency (section 6 and 7).

2 The grammar

The grammar used by the surface realiser is Feature-based TAG, a unification based version of Tree Adjoining Grammar. Briefly[1], a Feature-based TAG consists of a set of (auxiliary or initial) elementary trees and of two tree composition operations: substitution and adjunction. Substitution inserts a tree onto a leaf node of another tree[2] while adjunction inserts an auxiliary tree into a derived tree (i.e., either an elementary tree or a tree resulting from the combination of two trees). In an FTAG, each tree node which is not a substitution node is associated with two feature structures called `top` and `bottom` and during derivation, the following unifications take place.

- The adjunction at some node X with `top` features t_X and `bottom` features b_X, of an auxiliary tree with root `top` features r and foot `bottom` features f entails the unification of t_X with r and of b_X with f.
- The substitution at some node X with `top` features t_X of a tree with root `top` features t entails the unification of t_X with t.
- At the end of a derivation, the `top` and `bottom` features of all nodes in the derived tree are unified.

[1] For more details on FTAG see [Vijay-Shanker and Joshi, 1988].

[2] Leaf nodes where substitution can take place are graphically distinguished by a downarrow.

In the FTAG used by the surface realisation algorithm, linguistic expressions are associated with semantic representations as advocated in [Gardent and Kallmeyer, 2003]. The semantic representations used are flat semantic representations in the sense of [Copestake *et al.*, 2001] and the **semantic parameters** (that is, the semantic indices representing the missing arguments of the semantic functors) are represented by unification variables.

Further, each elementary tree is associated with a semantic representation of the type just described and the appropriate nodes of the elementary trees are decorated with semantic indices or parameters.

More precisely, the substitution nodes of the tree associated with a semantic functor will be associated with semantic parameters whilst root nodes and certain adjunction nodes will be labelled with semantic indices. As trees are combined, semantic parameters and semantic indices are unified by the FTAG unification mechanism thus specifying which semantic index provides the value for which semantic parameter.

Generally, the idea is that the association between tree nodes and unification variables encodes the syntax/semantics interface: it specifies which node in the tree provides the value for which semantic parameter in the semantic representation of a semantic functor. So for instance, the trees for *John, loves* and *Mary* will be as given in Figure 1. The tree for *loves* is associated with a semantic representation including the two semantic parameters x and y. These parameters also label the subject and the object substitution nodes of this tree. Conversely, the root node of the tree for *John* is labelled with the semantic index j. If the string parsed is *John loves Mary*, this tree will be substituted at the subject substitution node of the *loves* tree thus instantiating the semantic parameter x to j. And similarly, for the *Mary* tree.

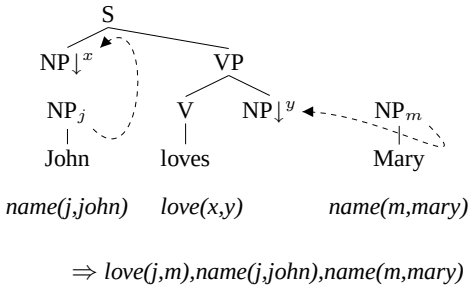

Figure 1: *John loves Mary*

Coverage. The grammar used describes a core fragment for French and contains around 4 000 trees. It covers some 35 basic subcategorisation frames and for each of these frames, the set of argument redistributions (active, passive, middle, neutre, reflexivisation, impersonnal, passive impersonnal) and of argument realisations (cliticisation, extraction, omission, permutations, etc.) possible for this frame. As a result, it captures most grammatical paraphrases that is, paraphrases due to diverging argument realisations or to different meaning preserving alternation (e.g., active/passive or clefted/non clefted sentence).

3 The basic algorithm

The basic surface realisation algorithm used is summarised in Figure 1 (appendix). It is a bottom up, tabular algorithm [Kay, 1996] optimised for TAGs. Its workings can be illustrated by the following example. Suppose that the input semantics is the following :

```
{camp(s,j),john(j),in(s,l),paris(l)}
```

Then the algorithm proceeds as follows. In a first step (**lexical selection**), the elementary trees whose semantics subsumes[3] part of the input semantics are retrieved and added to the agenda. In our simple example, the selected trees are the trees for *Jean, campe, dans* and *paris*.

The second step (the **substitution phase**) consists in systematically exploring the possibility of combining two trees by substitution. It is summarised for our example by the table in figure 2 where each line corresponds to a processing step. The words in each column indicate the trees present at each step in the chart, the agenda and the agenda for auxiliary trees (AgendaA). The combination column indicates which tree combines with which tree by means of which operation ($\downarrow$ indicates a substitution, $\star$ an adjunction). The trees resulting from such a combination are represented using the concatenation of the names of the combined trees (jeanCampe is the tree resulting from the combination of the tree anchored by *Jean* with that anchored by *campe*). Thus, the first line indicates that the trees anchored by *Jean*, *campe*, *dans* et *Paris* are in the agenda and that the chart is empty. The second line shows that the next state is a state where the tree anchored by *Jean* has been retrieved from the agenda and added to the chart. The third line indicates that when the trees anchored by *campe* and *Jean* are in the chart, they can be combined using substitution. The result is added to the agenda etc.

More generally, the items are retrieved one by one from the agenda to be added either to the chart or to the auxiliary agenda (in the case of an auxiliary tree devoid of substitution node). For each item added to the chart, all possible substitutions are carried out and the resulting derived trees are added to the agenda. The loop ends when the agenda is empty.

At this stage, all the items containing an empty substitution node are erased from the chart (here, the trees anchored by *campe* and *dans* are erased). The agenda is then reinitialised to the content of the chart and the chart to the content of the auxiliary agenda. The third step (the **adjunction phase**) occurs then in which all possible adjunctions are performed (figure 3). Finally (**retrieval phase**), the strings labelling the items in the chart whose semantics is the input semantics are printed

[3]Subsumption is here taken to denote term unification. Hence lexical selection is done on a very "syntactic" basis: only these lexical entries whose semantics representation matches part of the input semantics are selected. This is partly alleviated by taking lexical synonymy into account while developing the grammar so that two (intra- or inter-categorial) synonyms are assigned the same semantic representation. A more complete treatment would require the integration either of a richer lexical semantics or of a lexical selection module permitting inference so that for instance "adult(x) male(x) human(x)" can be inferred to be denoted by the word "man".

Agenda	Chart	Combination	AgendaA
Jean,campe,dans,Paris			
campe,dans,Paris	Jean		
dans,Paris	campe,Jean	↓(campe,Jean)	
Paris,JeanCampe	campe,Jean,dans		
JeanCampe	campe,Jean,dans,Paris	↓(dans,Paris)	
dansParis	campe,Jean,dans,Paris,JeanCampe		
	campe,Jean,dans,Paris,JeanCampe		dansParis

Figure 2: Sample run of the substitution phase

out, which in this case yields the sentence *Jean campe dans Paris*.

4 Optimisations

Surface realisation is NP complete [Brew, 1992]. Moreover the paraphrastic power of natural language is enormous [Gross, 1975; Mel'čuk, 1988]. Hence optimisation is a key issue and so is the possibility to select a given paraphrase on demand. We now present a number of optimisations we added to the algorithm just described in order to reduce the combinatorics.

4.1 Tabulation and ordered combinations

Tabulation serves to avoid redundant computations. In analysis, the use of the chart to store intermediate constituents and avoid multiple computation of the same structure renders an exponential task polynomial. In generation however, tabulation increases efficiency by avoiding duplicate computations but the complexity remains exponential because in particular of multiple modifiers [Brew, 1992]. Suppose for instance that the input semantic representation is the following:

`fierce(x),little(x),cat(x),black(x)`

For this input, a naive bottom-up realisation algorithm will generate all intermediate structures that is, $n!$ intermediate structures with n the number of modifiers. These $n!$ structures will furthermore be multiplied by the context so that for instance given the input for *the fierce little black cat runs*, the following structures will all be generated.

(3) a. *fierce cat, fierce black cat, little cat,little black cat, fierce little cat, black cat*
b. **the** *fierce cat, the fierce black cat, the little cat, the little black cat, the fierce little cat, the black cat*
c. **the** *fierce cat* **runs**, *the fierce black cat runs, the little cat runs, the little black cat runs, the fierce little cat runs, the black cat runs*

To minimise the impact of multiple modifiers, the algorithm presented here performs all substitutions before considering adjunctions. In effect, this means that adjunction only applies to syntactically complete trees and so that the many intermediate structures induced by the modifiers do not multiply out with other incomplete structures. In the above example for instance, (3c) will be computed but neither (3a) nor (3b).

4.2 Avoiding spurious derivations

Categorial grammars often allow so called spurious derivations in that one and the same syntactic structure can be derived in several different ways [Hepple, 1991]. TAGs also induce spurious derivations due to the fact that substitutions and adjunctions on different nodes of the same tree can be carried out in different relative orders all of which result in one and the same structure. Thus for instance, given the trees `np(Marie)`, `np(Jean)`, `s(np↓, v(aime), np↓)` and the semantic `aime(j,m),jean(j), marie(m)`, two derivations are possibles, one where `np(Jean)` is first substituted in `s(np↓, v(aime), np↓)` before the tree for `np(Marie)` is ; and the other where `np(Marie)` is first substituted before `np(Jean)` is added. More generally, for a tree containing n substitution nodes, there will be $n!$ possible derivations. For instance given the sentence

(4) Jean persuade Marie de promettre à Claire de donner un livre à Marie.
Jean persuades Mary to promise Claire to give Mary a book

there will be $3! \times 2! \times 2! = 24$ possible derivations all of them produce the same syntactic tree and hence the same sentence.

Adjunction suffers from the same shortcoming. Given a TAG tree and n auxiliary trees that can adjoin to different nodes of that tree, there are $n!$ possible ways of deriving the tree resulting from these n adjunctions.

To avoid these spurious derivations, we impose a unique order (from left to right) on the sequences of substitutions and adjunctions done within a given tree. Because the algorithm systematically examines all pairs of items, this restriction does not affect completeness : the unique derivation supported by the imposed constraints will be taken into consideration by the algorithm and will therefore be produced.

A third source of spurious derivations come from the possibility of having multiple adjunctions on the same node of a given tree for instance in the case of the *little black cat*. The auxiliary trees anchored by *little* and *black* can adjoin in two different orders on the tree anchored by *cat*: either *little* is adjoined to *cat* and *black* to the foot node of *little* in the resulting tree or *black* is adjoined to *cat* and *little* to the root node of the resulting derived tree. To eliminate this type of spurious derivations, adjunction on a foot node is ruled out – which is usual in TAG parsers.

Agenda	Chart	Combination	AgendaA
Jean,Paris,JeanCampe	dansParis		
Paris,JeanCampe	dansParis,Jean		
JeanCampe	dansParis,Jean,Paris		
	dansParis,Jean,Paris,JeanCampe	⋆(JeanCampe,dansParis)	
JeanCampeDansParis	dansParis,Jean,Paris,JeanCampe		
	dansParis,Jean,Paris,JeanCampe, JeanCampeDansParis		

Figure 3: Sample run of the adjunction phase

4.3 Filtering of valid lexical sequences

The most efficient optimisation takes place between the lexical selection phase and that of combination by substitution and adjunction. At this stage, the number of combinations that are *a priori* possible is $\prod_{1 \leq i \leq n} a_i$ with a_i the degree of lexical ambiguity of the i-th literal and n, the number of literals in the input semantic. That is, the search space is exponential in the number of literals. To reduce the combinatorics, we use a technique introduced for parsing by [Perrier, 2003] called *polarity based filtering*.

Polarity based filtering is based on the observation that many of the combinations of lexical items which cover the input semantics are in fact syntactically invalid either because a syntactic requirement is not fulfilled or because a syntactic resource is not used. Accordingly, polarity based filtering detects and eliminates such combinations by:

1. assigning each lexical item a polarity signature reflecting its syntactic requirements and resources
2. computing for each possible combination of lexical items the net sum of its syntactic requirements and resources and
3. eliminating all combinations of lexical items that do not have a net sum of zero (because such combinations cannot possibly lead to a syntactically valid sentence)

As we shall see below, polarity based filtering is implemented using finite state techniques.

Let us see how this works by running through a simple example. Suppose that the input semantic representation is:

(5) `buy(e,t,j), annoying(e), field(t),john(j)`

and that the TAG trees selected for this input are the ones given in Figure 8 (appendix).

In this figure, the literals following the tree name give the polarities that are automatically assigned to each of these trees on the basis of their root and substitution nodes (for instance, the *v_achete* has polarity $(+p - 2n)$ meaning that it provides a sentence and requires two NPs). Since in a TAG, substitution nodes indicates syntactic requirements whilst an initial tree permits fulfilling a syntactic requirement, polarity signatures can be automatically computed as follows:

- a polarity of the form $+C$ is added to the tree polarity signature of each initial tree with root node category C.
- a polarity of the form $-S$ is added to the tree polarity signature of each initial tree for each substitution node with category S in that tree.

Now we need to compute the polarity of all possible combinations of lexical items. This is done by:

1. building a polarity automaton for each polarity category occurring in the set of possible combinations (in this case, n and s),
2. computing the intersection of these automata and
3. minimising the resulting automaton.

In the final automaton, only the combinations that have a null polarity are represented. These will be the combinations actually explored by the realiser.

For the above example, the final automaton is that given in figure 9 where each state is labelled with the cumulated polarity of the path(s) leading to that state and where the transitions are labelled with the lexical item covered. As can be seen, the combinations that are syntactically invalid (in grey in the automaton) have been eliminated. Thus in particular, the combination of the predicative tree *n0Vadj* with the verb *achète* and its two complements is ruled out (as the n requirement of *n0Vadj* cannot be satisfied) and conversely, the combination of the predicative tree *p0Vadj* with the relational noun *achat* (because the p requirement of *p0Vadj* cannot be satisfied)[4] .

4.4 Combining polarity based filtering and tabulation

To preserve the factorisation supported by the use of a chart, polarity filtering must furthermore be integrated with the realisation algorithm. Indeed, each path through a polarity automaton represents a combination of lexical items whose total semantics is the input semantics and which may lead to a syntactically valid expression. But some of these paths may share some subpath(s). To avoid computing these shared subpaths several times, each selected elementary tree is annotated with

[4]For lack of space, we ignore here functional words (determiners, prepositions). In the full algorithm, their treatment is implemented either by means of co-anchors (a verb whose complément requires a given preposition for instance, will be assigned a tree with multiple anchors, one for the verb, the other for the preposition) or by means of a richer semantic (contrary to what is shown here, a quantifier will have a non nul semantics). Note further that lexical items with multi-literal semantics are also handled as well as items whose semantics is reduced to an index (pronouns, control verb subject, modifiers, etc.).

the set of automaton paths it occurs in. During realisation, two items will be compared only if the intersection of their path sets is not empty (they appear in the same automaton path). The result of a combination is labelled with the intersection of the labels of the combined constituents. In this way, the elementary items appearing in several paths of the automaton are only introduced once in the chart and the factorisation of both elementary and derived items that are common to several automaton path is ensured.

5 Paraphrase selection

As pointed out in the introduction, not all paraphrases are appropriate in all contexts. To test the ability to generate contextually appropriate sentences, we augmented the realiser with a paraphrase selection mechanism based on the polarity filtering system described in section (4.3). For instance, it is possible to select from among the possible realisations for `regarde(j,m), jean(j), marie(m)`, the variant where *jean* is verbalised as a cleft subject namely, *C'est Jean qui regarde Marie (It is John who is looking at Mary)*.

More generally, the selection constraints allowed are syntactico-semantic constraints of the form `Synt:SemIndex` where `Synt` is a morpho-syntactic feature (declarative, interrogative, cleft, pronoun, etc.) and `SemIndex` is an index occurring in the input semantics.

Intuitively, a selection constraint supports the selection, for a given semantic index, of the variant(s) obeying the syntactico-semantic constraint set by the selection constraint for that index.

Formally, these constraints are imposed during the polarity filtering phase as follows. The syntactic **properties** supported by the selection constraints are automatically associated during grammar compilation to the elementary trees of the grammar by means of so-called hypertags [Kinyon, 2000]. This is made possible by the fact that the TAG used is derived from a metagrammar [Crabbé and Duchier, 2004] that is, from a highly factorised way of representing the linguistic concepts encoded in the TAG trees. Roughly, the metagrammar formalism is used (i) to define abstractions over these concepts and (ii) to combine these abstractions so as to produce the elementary trees of a TAG. During the metagrammar compilation process, a so-called *hypertag* is built for each tree which records the abstractions used to produce that tree. Thus hypertags contain detailed information about the linguistic content of the TAG elementary trees. In particular, the hypertag of the tree with clefted subject of the n0vn1 family (i.e., the set of verbs taking two nominal arguments) will contain the property `+cleft:X` where `X` is the semantic index associated with the subject node.

During lexical selection, this index is instantiated by unification with the input so that the selected elementary tree for *regarde* will have the property `+cleft:j`.

Conversely, a **restrictor** is a property that a lexical item intervening in the production of the generated paraphrases *must* have. In the above example, the restrictor is `-cleft:j` meaning that the `j` index must be realised by a clefted structure.

Paraphrase selection is implemented by parameterising the realiser with a restrictor (for instance, `-cleft:j`). This restrictor is then used to initialise the polarity automaton and eliminate (by polarity filtering) all these combinations which do not contain the `+cleft:j` charge (since the negative charge introduced during initialisation must be cancelled). As a result, the realiser will only produce the variant:

(6) *C'est Jean qui regarde Marie*.

More generally, the polarity mechanism permits selecting paraphrases on the basis of the information contained in the grammar hypertags or in the TAG tree features. This information, which is decided upon by the grammar writer, can be both fine grained and of different natures.

Feature values can be used to control the feature values associated with the root node of the constructed tree, typically requiring that it is of interrogative, declarative or imperative mood.

Hypertags can be used more generally to control the selection of the lexical items entering in the generated construct. Importantly, the information they contain can be specified both at the grammar and at the lexical level so that paraphrase selection can then operate both on features determined by syntax and on lexically determined caracteristics (level of speech, modality, type of semantic relation, thematic and fore/backgrounding structure, etc;).

6 Implementation and Experimentation

The realiser described here has been implemented in Haskell. It includes a graphical interface as well as a debugger so that the user can inspect the content of the chart and of the agenda at each step of the algorithm. It also supports batch processing thus permitting a systematic evaluation of the impact of various optimisations combinations. In what follows, we discuss the effect of polarity filtering and of paraphrase selection in that system.

6.1 The effect of polarity filtering

To get an estimate of how our realiser compares with existing published results, we revisited the test cases discussed in [Carroll *et al.*, 1999] and [Koller and Striegnitz, 2002] by producing similar sentences in French namely (7a) and (7b).

(7) a. Le directeur de ce bureau auditionne un nouveau consultant d'Allemagne (*The manager in that office interviews a new consultant from Germany*)

b. Le directeur organise un nouveau seminaire d'equipe hebdomadaire special (*The manager organizes an unusual additional weekly departmental conference*).

The grammar used contains 2063 trees. In this grammar, the verb *organiser* is associated with 107 trees and adjectives with 8 trees. For the purpose of efficiency testing, we furthermore treated the PP *d'équipe* as an adjective. As a result, there are 107 x 8 (856) combinations of lexical items covering the input semantics for example (7a) while for example (7b), this number is 107 x 8^4. The effect of polarity filtering for these two examples is summarised in the following table.

That is, polarity filtering reduces the number of lexical items combinations actually explored from 856 to 55 in the first case and from 438 272 to 232 in the second.

	Example 7a	Example 7b
Possible combinations	856	438 272
Combinations explored	55	232
Sentences (w/o selection)	9	216

Figure 4: Filtering out combinations

Note furthermore that despite the overhead introduced by the construction of the polarity automaton, polarity filtering reduces both chart size and processing times (cf. Figure 5).

	Example 7a		Example 7b	
Optimisations	Chart sz	Time	Chart sz	Time
none	420	14.8 s	320	93.8 s
pol	559	0.8 s	1189	14.7 s
Carroll	n/a	1.8 s	n/a	4.3 s
Koller	n/a	1.4 s	n/a	0.8 s

Figure 5: Chart size and processing times

Thus, for the examples considered, chart size is reduced by 58% and 53% respectively, while processing times for (7a) compare favourably with those published for both the Carroll et al. and the Koller and Striegnitz realisers. The latter comparison is not all that meaningful, however, since we are using different grammars and significantly faster computers, a 3 Ghz Pentium IV to the 700 Mhz Pentium III in [Koller and Striegnitz, 2002].

Indeed, the poor performance of our surface realiser in example (7b) is directly related to the degree of lexical ambiguity in our grammar. As illustrated in section 4.1, input semantics with multiple modifiers pose a problem for surface realisers. Although performing adjunction separately from substitution prevents this problem from spilling over into incomplete structures, the fact remains that n translate to $n!$ structures. Further aggravating the situation is that our grammar provides 8 trees for every adjective, leading to $8^5 \times 5!$, or 3.9 million possible structures. When we modified our grammar to only have one tree per adjective, our realisation times dropped to 9s without filtering and 2.7s with. This example calls to attention the fact that polarity filtering does not account for lexical ambiguity in modifiers. In section 7, we suggest some potential mechanisms for dealing with modifiers, which we expect to be complementary to the filtering technique.

6.2 Paraphrase selection

Paraphrase selection permits reducing the combinatorics one step further. Thus introducing a cleft restrictor for examples (7a) and (7b), causes the generator to produce fewer results, 2 sentences instead of 9 in the first example, and 18 instead of 54 in the second.

These figures can be explained as follows. The grammar allows 9 syntactic structures for the input considered namely:

(8) a. C'est par le directeur de ce bureau qu'un nouveau consultant d'Allemagne est auditionné
b. C'est le directeur de ce bureau qui auditionne un nouveau consultant d'Allemagne
c. C'est un nouveau consultant d'Allemagne qu'auditionne le directeur de ce bureau
d. C'est un nouveau consultant d'Allemagne que le directeur de ce bureau auditionne
e. C'est un nouveau consultant d'Allemagne qui est auditionné par le directeur de ce bureau
f. Le directeur de ce bureau auditionne un nouveau consultant d'Allemagne
g. un nouveau consultant d'Allemagne est auditionné par le directeur de ce bureau

Since for the moment the grammar places no constraints on the respective order of modifiers, there are 9 possible realisations for example (7a) and $9 \times 3!$ for example (7b). With the object cleft restrictions on "consultant", these numbers drop to 2 for the first example and to $2 \times 3!$ for the second.

	Example 7a	Example 7b
Sentences (w/o selection)	9	54
Sentences (with selection)	2	18

Figure 6: Selection

Accordingly, the chart size drops by 90% and 95% with respect to simple polarities (cf. Figure 7).

	Example 7a		Example 7b	
Optimisations	Chart sz	Time	Chart sz	Time
none	420	14.8 s	320	93.8 s
pol	559	0.8 s	1189	14.7 s
pol + select	18	0.3 s	8	1.8 s

Figure 7: Polarity + Selection

7 Related approaches

Several recent papers focus on improving the efficiency of surface realisation. In this section, we relate our approach to the HPSG based approach presented in [Carroll *et al.*, 1999], to the statistical and semi-statistical strategies used in [Bangalore and Rambow, 2000] and in [White, 2004] and to the constraint based approach described in [Koller and Striegnitz, 2002]. We also briefly relate it to the greedy strategy used in [Stone *et al.*, 2003].

7.1 Copestake et al.'s HPSG approach

As mentioned in section 4.1, multiple modifiers may trigger an exponential number of intermediate structures. The "adjunction after substitution" idea is inspired from the proposal made in [Carroll *et al.*, 1999] that a complete syntactic skeleton be built before modifiers be inserted into that tree. Because the Carroll et al. proposal is set within the HPSG framework however, extracted modifiers as in *Which office did work in?* need specific treatment. In contrast, in TAG, all modifiers are treated using adjunction so that no specific treatment is required. All that is needed is that adjunction only be applied after all possible substitutions have been carried out. A second, more meaningful difference is that no such *global* optimisation as polarity filtering is proposed to

filter out on the basis of global information about the sets of possible combinations, a priori invalid ones.

7.2 Statistical approaches

Interestingly, [White, 2004] proposes a treatment of modifiers which is in some sense the converse of the "adjunction after substitution" treatment and where complete NPs are first built before they are combined with the verb. This second option is also feasible in TAG (adjunction would then apply on specific sets of lexical entries and the results combined with the verb) and it would be interesting to experiment and compare the relative efficiency of both approaches within the TAG framework.

Both approaches isolate the addition of modifiers to a constituent, thereby avoiding spurious combinations with unrelated constituents; but neither directly address the fact that are still an exponential $n!$ ways to combine any n modifiers for a single constituent. [White, 2004] and [Bangalore and Rambow, 2000] propose statistical solutions to this problem based on a linear n-gram language model. In White's approach the statistical knowledge is used to prune the chart of identical edges representing different modifier permutations, e.g., to choose between *fierce black cat* and *black fierce cat*. Bangalore assumes a single derivation tree that encodes a word lattice (*a* {*fierce black, black fierce*} *cat*), and uses statistical knowledge to select the best linearilisation. Our framework does not currently implement either approach, but we hope to adopt an approach similar to Bangalore's. Rather than directly performing adjunction, we associate each node with the set of auxiliary trees (modifiers) that are to be adjoined to that node. The order in which these modifiers are adjoined can be decided through statistical methods.

There are three other uses for probalistic techniques: for lexical selection, optimisation and ranking. Such techniques are useful for guiding the surface realiser towards a single best result (or a relatively small number thereof). On the other hand, we aim to produce *all* possible paraphrases, that is explore the entire search space of syntactic variants, and so with the exception of modifier ordering, we eschew the use of probabilities in favour of an "exact method" [G. Bonfante, 2004]. While Bangalore uses a tree model to produce a single most probable lexical selection, we use polarities to filter out all the definitely impossible ones. While in White's system, the best paraphrase is determined on the basis of n-gram scores that is, on the basis of frequency, in our approach "best" means "most contextually appropriate". Indeed, the restrictors we use to select a paraphrase, although they are here given by hand, could equally be set by the context and so permit modeling the effect of contextual constraints on paraphrases. We believe that our approach, modulo statistical handling of modifiers, would be roughly equivalent to White's with anytime-searching disabled.

7.3 Koller et al.'s constraint-based approach

Finally, our approach has interesting connections to the constraint-based approach proposed by [Koller and Striegnitz, 2002]. In this approach, the subset of the TAG grammar which is used for a given realisation task is translated into a set of lexical entries in a dependency grammar defining well formed TAG derivation trees. This set of entries is then parsed by an efficient constraint-based dependency parser thus producing the derivation trees associated by the grammar with the set of input lexical entries. A post processing phase produces the derived trees on the basis of the derivation trees output by the first step.

The main similarity between this and our approach is that they both use a *global* mechanism for filtering out combinations of lexical entries that cannot possibly lead to a syntactically valid sequences. In the Koller et al. approach, this filtering is based on well formed derivation trees (only these combinations of lexical entries that form a valid derivation tree are considered) whereas in ours, it is based on polarities and on the cancelling out of syntactic resources and requirements. As a preliminary evaluation shows, such a global optimisation is very efficient in pruning the search space.

There are differences though. In particular, while Koller et al. explicitly ignores feature information, our algorithm handles a TAG with fully specified feature structures. Further while in our approach, the processing of the valid combinations is done using a tabular algorithm optimised to avoid spurious derivations, the postprocessing step producing derived trees from derivation trees is left undefined in the Koller et al. approach. Finally, while the Koller et al. approach is based on constraint propagation, ours is based on finite state techniques. These differences open up the door for interesting comparisons and combinations. It would be interesting for instance to combine the Koller et al approach with the tabular surface realisation algorithm presented in this paper, or to compare run times once feature structures are taken into account.

7.4 Stone's greedy approach

[Stone *et al.*, 2003] presents a greedy approach to TAG based surface realisation. The greedy search applies iteratively to update a *single* state in the search space. On each iteration, all neighbors of the current state are produced but only one state is chosen at the next current state, based on a heuristic evaluation.

[Stone *et al.*, 2003]'s search strategy is therefore markedly different from ours. While we explore the entire search space and use polarities to control the combinatorics, Stone's greedy strategy is a best first strategy which incrementally trims the search space using heuristics. In terms of efficiency, the greedy strategy is of course better. The goals behind the two approaches are distinct however. Thus while Stone's approach aims at modelling the interaction of the various mechanisms involved in microplanning, the present proposal is directed towards generating and selecting paraphrases. In particular, we are interested in using the realiser to debug a paraphrastic grammar that is, a grammar which alleviates the inference task by assigning paraphrases the same semantics – this can only be done by adopting an exhaustive search strategy. More generally, "exhaustive surface realisation" provides a natural way to debug grammars and reduce their degree of overgeneration. Since the combinatorics is not only theoretically (worse case analysis) but also practically very high, it is worth investigating ways of optimising surface realisers which perform an exhaustive search.

8 Conclusion

We have presented a surface realiser for TAG which is optimised to support the generation of grammatical paraphrases while also permitting the selection, on the basis of syntactico semantic constraints, of a particular paraphrase. The most efficient optimisation proposed concerns polarity filtering, a global technique that permits the elimination of combinations of lexical items which cannot possibly lead to a syntactically valid sentence. While used here for generating with TAG, the technique is fully general and can be used for parsing [Perrier, 2003] but also for generating with other grammatical frameworks.

Future work will concentrate on extending the grammar and the lexicon to other types of paraphrases (in particular, morphoderivational or cross categorial paraphrases), on providing a systematic evaluation of the paraphrase selection mechanism and on using the realiser for the debugging of an existing TAG for French.

References

[Bangalore and Rambow, 2000] S. Bangalore and O. Rambow. Using TAGs, a tree model and a language model for generation. In *Proceedings of TAG+5*, Paris, France, 2000.

[Brew, 1992] C. Brew. Letting the cat out of the bag: Generation for shake-and-bake MT. In *Proceedings of COLING '92*, Nantes, France, 1992.

[Carroll *et al.*, 1999] J. Carroll, A. Copestake, D. Flickinger, and V. Paznański. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of EWNLG '99*, 1999.

[Copestake *et al.*, 2001] A. Copestake, A. Lascarides, and D. Flickinger. An algebra for semantic construction in constraint-based grammars. In *Proceedings of the 39th ACL*, Toulouse, France, 2001.

[Crabbé and Duchier, 2004] B. Crabbé and D. Duchier. Metagrammar redux. In *International Workshop on Constraint Solving and Language Processing - CSLP 2004*, *Copenhagen*, 2004.

[G. Bonfante, 2004] G. Perrier G. Bonfante, B. Guillaume. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *Proceedings of CoLing 2004*, 2004.

[Gardent and Kallmeyer, 2003] C. Gardent and L. Kallmeyer. Semantic construction in ftag. In *Proceedings of the 10th EACL*, Budapest, Hungary, 2003.

[Gross, 1975] M. Gross. *Méthodes en syntaxe*. Masson, Paris, 1975.

[Hepple, 1991] M. Hepple. Efficient incremental processing with categorial grammar. In *Proceedings of the 29th ACL*, Berkeley, 1991.

[Kay, 1996] M. Kay. Chart Generation. In *34th ACL*, pages 200–204, Santa Cruz, California, 1996.

[Kinyon, 2000] A. Kinyon. Hypertags. In *Proceedings COLING*, Sarrebruck, 2000.

[Koller and Striegnitz, 2002] A. Koller and K. Striegnitz. Generation as dependency parsing. In *Proceedings of the 40th ACL*, Philadelphia, 2002.

[Mel'čuk, 1988] I. Mel'čuk. Paraphrase et lexique dans la théorie linguistique sens-texte. *Lexique*, 6:13–54, 1988.

[Perrier, 2003] G. Perrier. Les grammaires d'interaction, 2003. Habilitation à diriger les recherches en informatique, université Nancy 2.

[Stone *et al.*, 2003] M. Stone, C. Doran, B. Webber, T. Bleam, and M. Palmer. Microplanning with communicative intentions: the SPUD system. *Computational Intelligence*, 19(4):311–381, 2003.

[Vijay-Shanker and Joshi, 1988] K. Vijay-Shanker and A. Joshi. Feature based tags. In *Proceedings of the 12th ACL*, pages 573–577, Budapest, 1988.

[White, 2004] M. White. Reining in CCG chart realization. In *INLG*, pages 182–191, 2004.

A Appendix

Algorithm 1 The GenI algorithm

procedure GENERATE($Gram$,Sem)
  $AgendaA \leftarrow \emptyset$; $Agenda \leftarrow \emptyset$; $Chart \leftarrow \emptyset$
  for all trees $t \in Gram$ such that t's semantics subsumes Sem **do**
    $Agenda \leftarrow Agenda + t$
  end for
  while $Agenda \neq \emptyset$ **do**
    $t \leftarrow$ any tree $\in Agenda$
    delete t from $Agenda$
    if t has a foot node and no substitution nodes **then**
      $AgendaA \leftarrow AgendaA + t$
    else
      for all trees $c \in Chart$ which can combine with t via substitution into a new tree ct **do**
        $Agenda \leftarrow Agenda + ct$
      end for
      $Chart \leftarrow Chart + t$
    end if
  end while
  delete from $Chart$ any tree with a substitution node
  $Agenda \leftarrow Chart$
  $Chart \leftarrow AgendaA$
  while $Agenda \neq \emptyset$ **do**
    $t \leftarrow$ any tree $\in Agenda$
    delete t from $Agenda$
    if t's semantics is Sem **then**
      return the string corresponding to t
    else
      for all trees $c \in Chart$ which can combine with t via adjunction into a new tree ct **do**
        $Agenda \leftarrow Agenda + ct$
      end for
    end if
  end while
end procedure

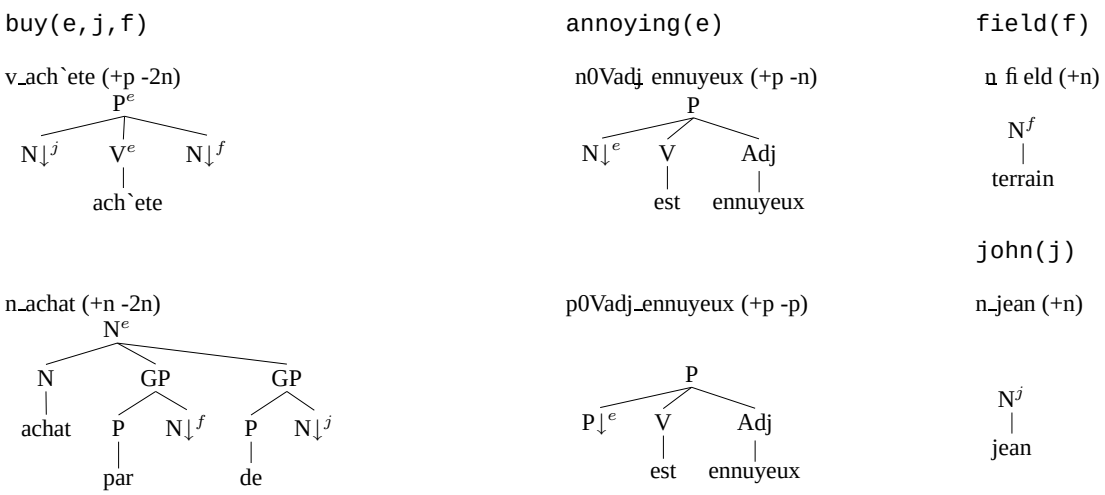

Figure 8: Grammar for example 5

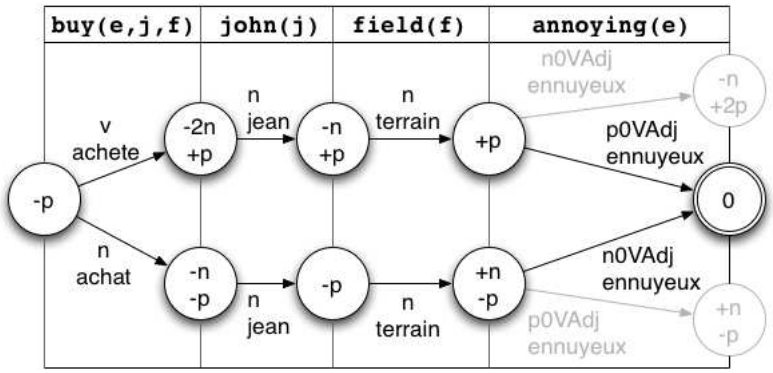

Figure 9: A minimised polarity automaton