Non-Contiguous Tree Parsing

Mark Dras
Centre for Language Technology
Macquarie University
madras@ics.mq.edu.au

Chung-hye Han
Department of Linguistics
Simon Fraser University
chunghye@sfu.ca

Abstract

Pairing structural descriptions in MT, syntax-semantics interfaces and so on becomes more difficult the more structurally different are the languages involved; there is, implicitly or explicitly, a process of 'tree parsing', where a structural description is split into component smaller trees for transfer rules to be applied. Recent work has looked at the construction of transfer rules, using both symbolic and statistical approaches, that require the pairing of groups of several contiguous nodes in structural descriptions. We look at the case where pairings of groups of non-contiguous nodes are necessary, and present an efficient dynamic programming algorithm based on TAG and drawing on compiler theory for a decomposition into appropriate groupings. We then examine the formal properties of this algorithm, and show that it is linear in the number of nodes in the tree and has the same complexity as existing algorithms requiring only groupings of contiguous nodes.

1 Introduction

There are many situations in which it is necessary to relate two sets of structures: machine translation, paraphrase, mapping between syntax and semantics, and so on. Often these are trees, and often structural divergences are significant. Dorr (1994) presents a classification of divergences in MT, including the structural, and uses the extent of the divergences to argue for an explicit semantic representation.

Because of structural differences, it is necessary to use some transformation operation in the pairing of the trees. In some cases this is dealt with in an ad hoc manner, although there are several different models for dealing algorithmically with these structural differences that have been proposed. For example, in the structure-pairing formalism based on context-free derivations proposed for MT by Wu (1997), re-ordering of righthand sides in context-free grammar rules is allowed in order to represent differences in structure; more recently, Eisner (2003) has used a model of Synchronous Tree Substitution Grammars (S-TSGs) as the basis for a stochastic mapping induction system. Broadly, this takes a group of nodes in each tree and treats them as a single unit in order to be able to pair trees of different structure.

Abeillé et al. (1990), in presenting Synchronous Tree Adjoining Grammar (S-TAG) as a formalism for representing MT, note that the extent of the divergences and consequent restructuring will depend on the formalism chosen: with a formalism such as S-TAG, with its extended domain of locality which incorporates predicate–argument structure into the elementary units of the grammar, there are fewer divergences. Even with this minimisation of divergence through choice of representation, it is not the case that the structures to paired are isomorphic: the redefinition of S-TAG in Shieber (1994) which requires isomorphic (i.e. node-to-node) derivations is extended in that paper to include also the pairing of groups of nodes in trees.

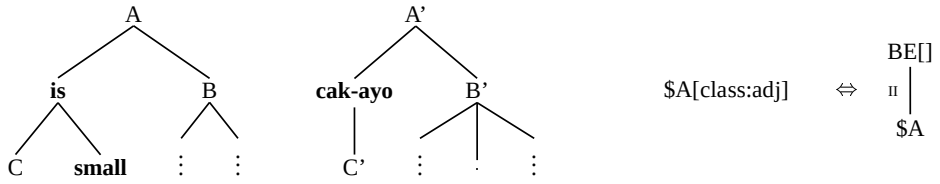

Figure 1: Paired non-isomorphic structures, with transfer rule

Current models of tree transformation, however, allow only the grouping of contiguous nodes for the purpose of pairing, and there are situations where groupings of non-contiguous nodes—but not just any arbitrary groups of non-contiguous nodes—are required. 'Parsing' a tree with a grammar based on some formalism other than CFGs or TSGs will then permit the mapping of such groupings; this can be viewed as applying to the tree a meta-level grammar along the lines of Dras (1999). For cases of parsing trees with groupings of contiguous nodes, there are standard efficient algorithms in compiler theory; however, these do not exist for pairing of groupings of non-contiguous nodes, and this would at first glance appear to require more powerful and slower mechanisms.

In this paper we use Tree Adjoining Grammar as the formalism for capturing non-contiguous groupings of nodes required by pairings; it has properties that, given certain conditions, allow an efficient tree parsing algorithm. In Section 2 we examine some examples of the types of groupings required; in Section 3 we give a brief overview of TAG; and in Section 4, we present a dynamic programming algorithm that allows tree mappings with groupings of non-contiguous nodes, which is linear in the number of nodes in the tree and hence as efficient as that for the contiguous case, followed by some discussion of more general questions related to the notion of tree parsing.

2 Pairing Structural Descriptions

The aim of this work is to decompose trees into groupings of non-contiguous nodes that have been identified as being in a correspondence for a transfer-based translation. The starting point for the process is thus a tree assigned independently as the input to the transfer, typically by a parser; whether it is a dependency tree, TAG derivation tree, or other, is immaterial.

First, we will define more precisely what we mean by groupings of contiguous nodes (gCNs) and groupings of non-contiguous nodes (gNCNs). Taking nodes in a tree to be represented by Gorn addresses,[1] a gCN N is a set of nodes such that if two nodes with addresses p_1, p_2 are in N, and they have largest common prefix p_c, then all nodes with address p_i such that p_c is a prefix of p_i and p_i is a prefix of p_1 or p_2 must be in N. A gNCN is any set of nodes in a tree that is not a gCN.

In this section we will illustrate some of the situations where pairing of gNCNs is required. As an example of the standard case of groupings of gCNs, we give pair (1) from Korean-English MT.

(1) pang-un cak-ayo
room–TOP be-small–DECL
The room is small.

For predicative adjectives, English uses copula *be* plus the adjective, while Korean uses only a verb-like lexical item. Embedding the adjectival constructions from (1) within a larger context, paired struc-

[1]The Gorn address of the root is ϵ; the Gorn address of the jth child of node with address i is $i \cdot j$, $j \in \mathbb{N}_+$. For $p, q \in {\mathbb{N}_+}^*$, q is a PREFIX of p if and only if there exists an $r \in {\mathbb{N}_+}^*$ such that $p = q \cdot r$.

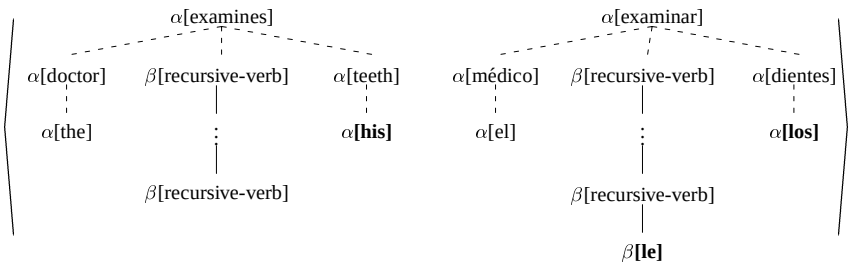

Figure 2: Pairs for (2)

tures might look like the lefthand side of Figure 1.[2] Here, we would need to treat the nodes for *is* and *small* as a single unit in order to pair it with *cakayo*, so the grouping of the nodes for *is* and *small* would be designated a gCN. If there were a need to group *is*, *small* and B, this would be a gNCN.

Paired TAGs In pairing two TAGs for MT, syntax-semantics mapping or paraphrase, under the redefinition of Synchronous TAG in Shieber (1994) there must be an isomorphism between the derivations of two strings to be paired. In TAG, for each DERIVED TREE derived from smaller elementary trees, there is a corresponding DERIVATION TREE which describes the history of the derivation. This derivation tree has a number of similarities to dependency trees, but is not exactly the same (Rambow and Joshi, 1997). In general there will not be an isomorphism between two such trees for any of the above applications, hence Shieber's proposed extension to allow "bounded subderivation" (which correspond to gCNs in the context of derivation trees). However, he also notes the possibility, further explored in Dras and Bleam (2000), that the pairing of gNCNs will be necessary. An example taken from the latter is in (2).

(2) El médico le quiero poder ... examinar los dientes.
The doctor him–DAT wants to-be-able ... to-examine the teeth.
The doctor wants to be able ... to examine his teeth.

In this Spanish-English example, the clitic can climb over an unlimited number of 'trigger verbs' (Aissen and Perlmutter, 1976) (indicated by the ellipses in the example), and for certain TAG grammars this can correspond to a pair of derivation trees as in Figure 2. In this pair of trees, *his* corresponds to both *los* and the clitic *le*. Both *his* and *los* are fixed in relation to the root of the tree, but *le* is an unbounded distance from it, so it is not possible to form a gCN in the Spanish tree for pairing without the unbounded and unrelated recursively-inserted verbs, hence requiring infinitely many transfer rules.

Paired dependency trees The system of Han et al. (2000) pairs two dependency trees based on a Deep Syntactic Structure (DSyntS) of Meaning Text Theory (MTT) (Mel'čuk, 1988), a dependency representation composed of nodes labeled by lexemes that correspond to meaning-bearing words (nouns, verbs, adjectives, adverbs) and directed arcs with dependency relation labels. Transfer rules are also represented by DSyntS trees, with variables.[3] The goal of this particular dependency representation is to minimise 'spurious' structural divergences, such as when a preposition in one language is

[2] We use the romanization of Han et al. (2000), for consistency with our later example.

[3] The subject is labeled as 'I', the direct object as 'II', the indirect object as 'III', and other oblique arguments as 'IV'; adjuncts are labeled as 'ATTR'. Function words such as determiners, semantically empty auxiliary verbs and grammatical morphology are represented through features on the node labels.

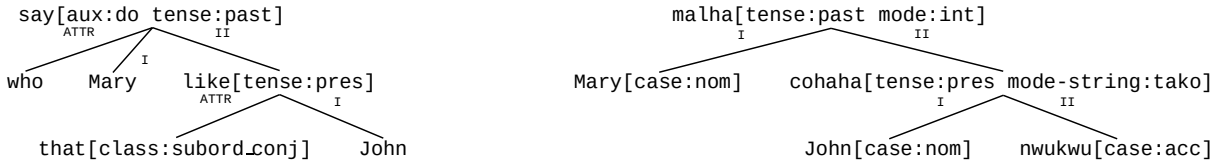

Figure 3: Paired DSyntS for (3)

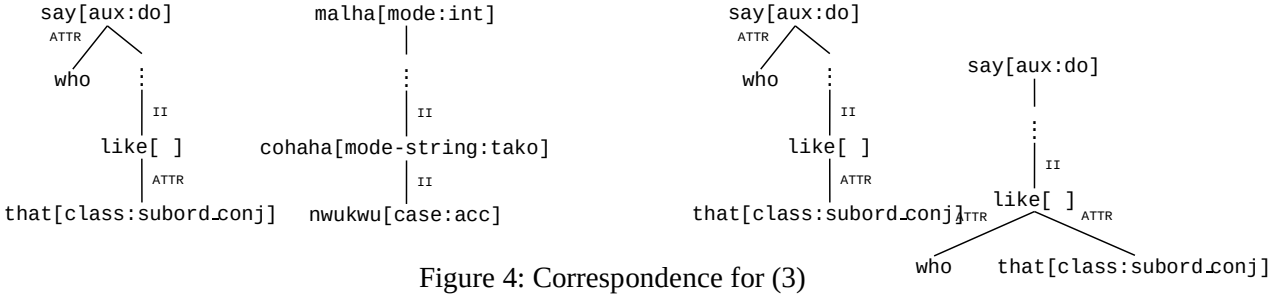

Figure 4: Correspondence for (3)

represented by a verbal inflection in the other. However, some divergences still occur, as in (1). The transfer rule then requires that the two nodes *is* and *small* pair with the single node *cakayo*: a transfer rule for Figure 1, treating them as a gCN, would be as in the righthand side of that figure.[4] However, there are constructions which cannot be handled in such a way. Consider the translation pair in (3).

(3) Mary-ka John-i nwukwu-lul cohaha-n-tako malha-yess-ni?
Mary–NOM John–NOM who–ACC like–PRES–COMP say–PAST–Q
Who$_i$ did Mary say that John likes t_i?

Syntactically, *who* is dependent on the matrix clause verb, *did* in English, while semantically it is an argument of the subordinate verb *likes*, a case of long distrance extraction (see Figure 3). In the DSyntS, *did* becomes part of *say* as a feature on the *say* node. Further, *who* is dependent on *say* and can only be labeled as ATTR since it is not an argument of *say*. In the Korean however, *nwukwu* ('who') is still an object of *cohaha* ('like') with its dependency arc labeled as II. So, a transfer rule covering long distance extracted *who* would need to include matrix and embedded verbs, as in the lefthand pair of Figure 4. But, because long distance extraction is in principle unbounded, we would need to specify all the possible cases, giving an infinite number of transfer rules. Moreover, in the English DSyntS, there is no way to represent the fact that *who* is a semantic argument of *likes*, unless additional features are used to track their relation.

Again, the key element in this problem is that nodes that are contiguous in the English tree (*say*, *who*) are not contiguous in the corresponding Korean tree (*malha*, *nwukwu*); this, along with the TAG example, can be seen as a case of intervening material breaking what should be contiguous.

It can of course be argued that an alternative representation would be more appropriate for MT, where *who* depends from *likes* in the tree. We have used the system of Han et al. (2000) to illustrate this point because it is a system that has the goal of exploring the feasibility of a plug-and-play architecture: that is, necessary components such as a parser are obtained from elsewhere, with a given output structure that it is necessary to use. Given this, gNCNs are required either directly or indirectly. The case of the direct relation, using these structures as the basis for a transfer component, is illustrated already in the

[4] $A is a variable slot for an adjective.

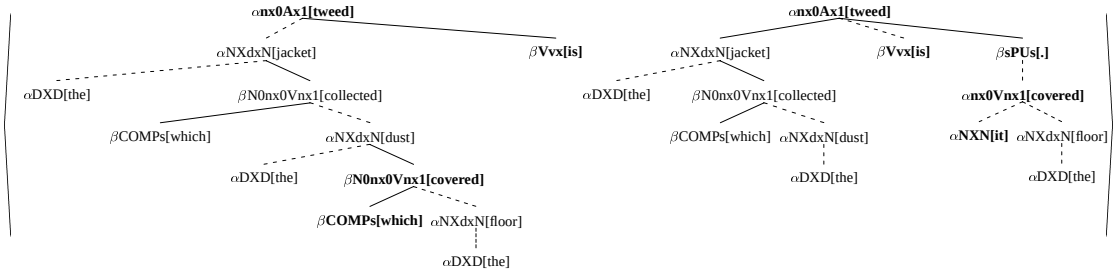

Figure 5: Derivation tree pair for example (4)

lefthand pair of Figure 4; a pairing indirectly involving gNCNs would be required in transforming the syntactic representation into a deeper semantic one (the one used in translation), as in the righthand pair of Figure 4. This latter is the sort of relation that may need to be specified, then, in a formalism with multiple levels, such as MTT.

In some cases it may be possible to know which representation will be structurally the most suitable for a particular application like MT and a particular pairing of languages, and to be able to specify for example the parser output representation, or to modify the parser (although this might be undesirable for reasons of modularity). However, this is not always the case, as we discuss in the next example.

Paraphrase Here we use an example, (4), from Dras (1999), where paraphrases are represented by pairing TAG derivation trees (Figure 5). This is again similar to the previous MT examples: in order to define a paraphrase where the most embedded clause becomes a separate sentence, it is necessary to form a gNCN (those nodes in bold in Figure 5).

(4) The jacket which collected the dust which covered the floor was tweed.

The jacket which collected the dust was tweed. The dust covered the floor.

Here, all other nodes correspond one-to-one in the trees, so the gNCNs are clear. This will be the case in paraphrase for many different types of representation: if the tree on the left has the most embedded clause represented by the most embedded subtree, there will still be this problem of fixed relation to the root vs unbounded relation; if the clause order is represented in the tree in reverse, with the most embedded clause the one closest to the root, there will be a parallel problem with a paraphrase where the least embedded non-matrix clause becomes a separate sentence. And unlike the case of *who* above, which representation is best is in general only a function of the pairing of the trees, not something innate to the grammar which generates an individual tree.

Thus there are a number of situations in which gCNs are not sufficient. Given that gCNs can be represented by Tree Substitution Grammars, as in Eisner (2003), which are in fact TAGs that do not allow precisely the kind of unbounded phenomena described by TAGs, this would suggest that using a TAG grammar to describe the gNCNs in order to decompose the trees would be feasible; and this is further an interesting question for theoretical reasons described below.

3 TAG Overview

TAG is a grammar formalism based on trees rather than context free rules (Joshi, 1987). Elementary trees are of two types, initial trees and auxiliary trees. Auxiliary trees have a designated foot node,

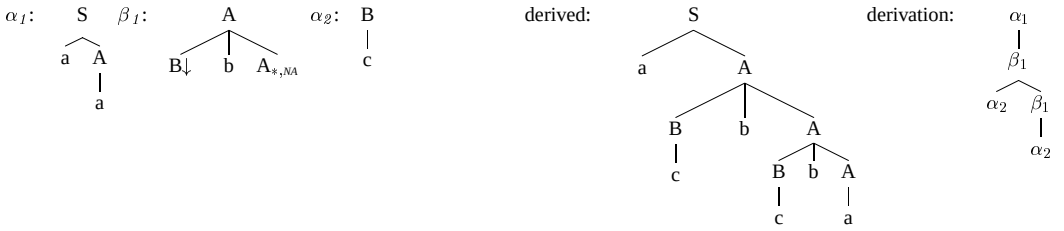

Figure 6: Elementary, derived and derivation trees

marked with a *, whose label is the same as that of the root. In Figure 6, α_1 and α_2 are initial trees; β_1 is an auxiliary tree. The trees are combined together by two operations, substitution and adjunction. Under substitution, a node marked for substitution[5] in a tree γ is replaced by an initial tree with the same label at the root; under adjunction, an internal node in a tree γ is 'split apart', replaced by an auxiliary tree with the same label at the root and foot. In the DERIVED TREE for the string $acbcba$, in Figure 6, copies of β_1 have been adjoined either at the root node labelled A of other nodes β_1 or ultimately at the A node of α_1; an α_2 tree has been substituted into each β_1 tree at the node labelled B. The derivation history is recorded in the DERIVATION TREE (Figure 6). It can be seen that the TAG property of an 'extended domain of locality' can allow the two as in the generated string to be separated by an abitrary amount of intervening material; this characteristic is used for representation of, for example, WH- phenomena when TAG derived trees are used for a linguistic representation. Of more interest for us than the derived string is the nature of the derived tree: the branches containing the a nodes in the derived tree are also separated by an arbitary distance.

In general, for linguistic representation it is the derived tree that is used as the primary structure of representation, so the labels a, b, c would represent words in a typical lexicalised grammar and the trees α_1, α_2 and β_1 would represent argument structure of these words. However, we will use a TAG grammar as a way of characterising other sorts of trees, such as TAG derivation trees or dependency trees; this is thus in a sense an extension of the notion of the meta-level grammar of Dras (1999). The idea is then to use a TAG grammar to break down some tree representation—which may be a dependency tree, a TAG derivation tree,[6] or other—into component trees possibly representing non-contiguous groupings. The aim is not to describe every decomposition into non-contiguous groupings, only those such as the language-related cases presented in Section 2; and the use of TAG as representation allows for the complexity results below. We now present an algorithm for the decomposition in Section 4.

4 A Tree Parsing Algorithm

4.1 Pattern Trees and Compilers

The process of breaking down an input abstract syntax tree (AST) into component pattern trees, in order to generate an instruction set, is a standard one in compilers. The standard technique involves a bottom-up rewriting system (BURS), with the optimal instruction set constructed by the dynamic programming algorithm of Proebsting (1995); see for example Grune et al. (2000). Because of the nature

[5] Substitution sites are conventionally marked with ↓.

[6] Note that in MT based on TAG, it is *derivation* trees that are paired for transfer, rather than *derived* trees, and it is this derivation tree that must be decomposed; that is, the process of decomposition is not just a side-effect of the parsing to obtain the initial derived tree representation. Rather, it can be thought of as a TAG grammar sitting on another TAG grammar, a meta-level grammar in the sense of Dras (1999).

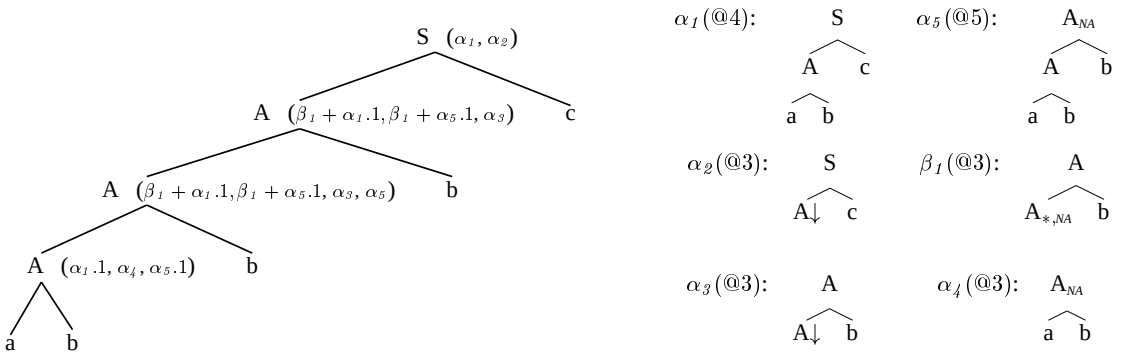

Figure 7: Abstract Syntax Tree and pattern trees

of programming languages, the sort of pattern trees that are allowed are only groupings of contiguous nodes; in effect, tree parsing is allowed with a tree grammar consisting of trees of possibly multiple levels and allowing only concatenation: this is equivalent to a TSG. Consider an AST in Figure 7 (ignoring the annotations on the nodes, in parentheses); and take for pattern trees only those initial trees of Figure 7 ($\alpha_1, \ldots, \alpha_5$). It can be seen that the AST can be decomposed in several ways, for example by the set of pattern trees $\{\alpha_2, \alpha_3, \alpha_3, \alpha_4\}$ or the set $\{\alpha_2, \alpha_3, \alpha_5\}$. If the numbers in parentheses after the labels (@c) are considered as costs, an optimal decomposition can be determined (here, $\{\alpha_2, \alpha_3, \alpha_5\}$).

Now in Section 4.2 we develop an algorithm based on this which allows an input AST (for us, a derivation or dependency structure, for example) to be broken into component non-contiguous 'trees' efficiently. From a theoretical point of view this is interesting, as the expectation would be that some more complex mechanism would be necessary, in much the same way that allowing stretching of paired characters in strings (say, in the language of nested strings $\{a^n b^n \mid n \geq 0\}$, where the ith a is matched with the $(n - i + 1)$th b) cannot be performed by a finite state automaton but requires a pushdown automaton through the addition of a stack; here, it might be expected that a stack is similarly necessary to keep track of the unbounded elements.

4.2 Generalizing to Restricted Non-Contiguity

As a first step, we consider only cases where at any node during the tree traversal in the BURS, there is only potentially one gNCN at a time: that is, it is not possible to embed or overlap these gNCNs. In order to explain this, consider first the example below. The input AST (ignoring the annotations on the nodes) is in Figure 7; pattern trees, in the form of a TAG grammar (with associated costs still indicated by @c), are also in Figure 7. The algorithm we use for bottom-up pattern matching, adapted from that of Grune et al. (2000), is in Figure 8.

For explanatory purposes, we first look at the bottom up pattern matching aspect of the algorithm. First, we notionally split the pattern trees into a set of single-level trees, the SPLIT TREE SET, given labels based on Gorn address. So, for example, α_1 is considered as two trees, α_1 (for the top half) and $\alpha_1.1$ (for the bottom). Further, each node in these trees is given a `type`, indicating which others it can join with. This can be a single value for trees that were originally split (so the A node in the split would have the type $\alpha_1.1$), or one of four values `sub`, `adj`, `both`, `none`. For leaves of split trees not marked by single values, nodes labelled with terminals are of type `none`, nodes marked for substitution are marked `sub`, and foot nodes are marked `adj`. For roots of split trees, roots of auxiliary trees are marked `adj`, null adjunction nodes marked `sub`, and others marked `both`.

```
PROCEDURE bu-match (Node)
  IF Node has non-terminal label
    bu-match (Node.left)
    bu-match (Node.right)
    SET Node.annot-set TO get-annot-set (Node)
  ELSE
    SET Node.annot-set TO { (Node.type = none) }

FUNCTION get-annot-set (Node)
  SET a-set TO Empty-set
  FOR EACH tree IN split tree set
    FOR EACH l-annot IN Node.left.annot-set
      FOR EACH r-annot IN Node.right.annot-set
        IF tree.label = Node.label
          AND matches (tree.left, l-annot)
          AND matches (tree.right, r-annot)
            Insert new-annot (tree, l-annot, r-annot) into a-set
  RETURN a-set
```

```
FUNCTION matches (tree, annot)
  IF tree.label = annot.label
    IF tree.type IN annot.label
      OR annot.type = both
      OR tree.type = annot.type
        RETURN true
  RETURN false

FUNCTION new-annot (tree, l-annot, r-annot)
  IF tree.left.type = adj
    foot = l-annot
  IF tree.right.type = adj
    foot = r-annot
  IF tree.cat = auxiliary
    RETURN t + foot
  ELSE
    RETURN t
```

Figure 8: Bottom-up pattern matching

We then traverse the AST bottom up, annotating the nodes with those parts of pattern trees that can apply, taking into account both labels and types of nodes. (Ignore, at this stage, the costs indicated by $@c$.) The lowest A node and its immediate children a and b could result from the application of pattern tree α_4; equally, it could be the lower half of trees α_1 or α_5 (i.e. $\alpha_1.1$ or $\alpha_5.1$). The next higher A node with its children A and b could result from α_3; or from α_5 (since the left child A is annotated with $\alpha_5.1$, indicating that the subtree from that point contains the remainder of α_5); or from β_1. Here there are the additional annotations $+\alpha_1.1i$ and $+\alpha_5.1i$: this is because β_1 represents material that has split α_1 or α_5 into gNCNs (the role of auxiliary trees in TAG), and so $\alpha_1.1$ and $\alpha_5.1$ are percolated up the tree as a record of the lower potential gNCNs. It is necessary for this to be attached to the annotation of an auxiliary tree, as auxiliary trees are the only valid intervening material. At the next higher A the same situation holds. Finally, the root S node can either result from the application of α_2, or of α_1 with interposed material (indicated by the left child of S having the label $\alpha_1.1$).

For the dynamic programming algorithm, costs are taken into account. In compilers, this value is related to the cost of the instructions corresponding to the pattern tree. For this example, the costs are not a function of anything external; they do, however, capture the preference of larger pattern trees over combinations of smaller trees, which is desireable; see Estival et al. (1990). Tracing through the example again, then, this time with costs, at the lowest A node the annotation α_4 has cost 3; the other two annotations $\alpha_1.1$ and $\alpha_5.1$, being partial pattern trees, have no cost. At the next higher A node, the annotations $\beta_1 + \alpha_1.1$ and $\beta_1 + \alpha_5.1$ have cost 3; α_3 has cost 6 (3 for the pattern tree α_3, and 3 for the left child as annotated in the previous step); α_5 has cost 5. As both α alternatives span the same subtree from this A node down, and have the same return type (`sub`), it is possible to discard the annotation α_3, as it will always be cheaper to use α_5 at this point, regardless of what happens further up the tree. At the next higher A node, the annotations $\beta_1 + \alpha_1.1$ and $\beta_1 + \alpha_5.1$ have cost 6, and α_3 has cost 8. Finally, at the top S node, α_2 has cost 13 (5 for the pattern tree, 8 for the left child: as the pattern tree can only accept an initial tree as the left child, only α_3 is a suitable candidate); but α_1 has cost 10 (4 for the pattern tree, 6 for the intervening auxiliary trees). The algorithm in Figure 8 is modified so that any annotation in an annotation set with the same `type` but non-minimal cost is discarded. Thus the derivation of the optimal tree parse, top-down, would be α_1 with an adjunction of β_1 which in turn has an adjunction of β_1.

By observation, and just as the standard algorithm, this extension is also $O(n)$ time and space complexity in the number of nodes. This is not surprising, as the restriction on non-embedding of gNCNs occurs if a TAG grammar is restricted to the normal form of Rogers (1994), so that the tree set is rec-

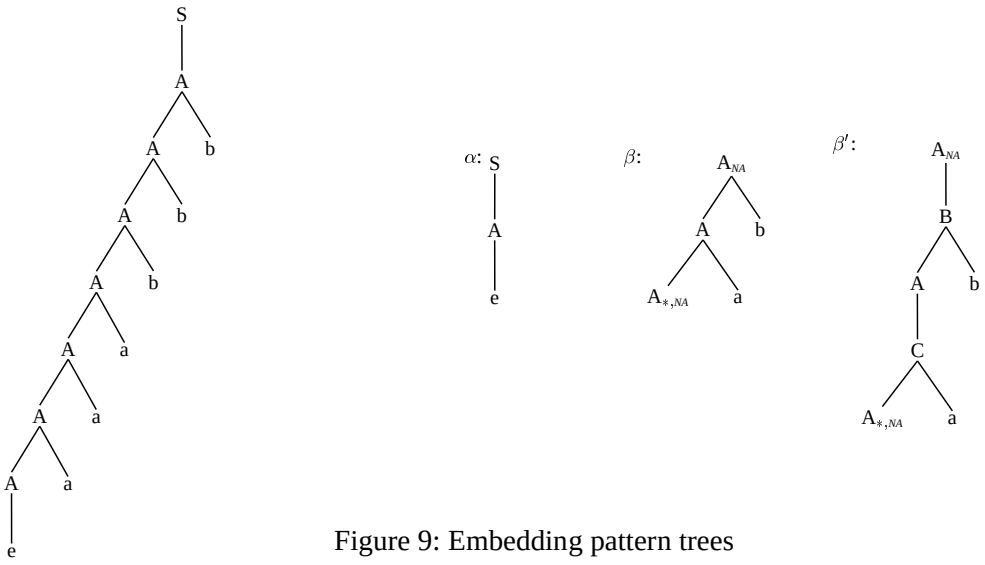

Figure 9: Embedding pattern trees

ognizable: in brief, in this normal form auxiliary trees cannot embed, and so the grammar is in effect an equivalent but variant form of CFG whose syntax allows a limited degree of non-local behavior. (In the given example, it can be seen that it is not possible to embed recursive material, as all the auxiliary trees are only of height 1.)

However, despite linear complexity in the number of nodes, the work done and space used are proportional to the number of pattern trees. A standard technique is to precompile all sets of annotations; as there is a finite set of pattern trees, there will be a finite set of annotations—in the case of the percolated annotations attached to β annotations representing gNCNs, this is still true—it is also possible here. The algorithm is then an implementation of a finite-state tree automaton.

4.3 A Further Generalization

If embedding is allowed, the algorithm is more complex. Consider the AST in Figure 9 and the pattern trees in Figure 9. Starting from the lowest A label, the annotations would be $\alpha.1$, $\beta+\alpha.1$, $\beta+\beta.1+\alpha.1$; for an AST of arbitrary depth, the annotation would be $\beta+\beta.1+\ldots+\beta.1+\alpha.1$. Clearly, a finite-state tree automaton is not an appropriate model: it is not possible to precompile the complete annotation set.

If the pattern tree we want to complete is only the most embedded—that is, it is not possible to overlap gNCNs—this corresponds to the operation of unrestricted TAG adjoining. That is, from the example, only the last $\beta.1$ annotation is accessible, so the obvious model is a stack. The procedure is then an implementation of some form of bottom-up tree pushdown automaton (buTPDA) (Schimpf and Gallier, 1985), a tree automaton augmented with a stack, in the same way a pushdown automaton (PDA) is a a finite-state automaton (FSA) plus a stack.

A standard buTPDA is not quite the right model. Schimpf and Gallier (1985) prove that TPDAs are necessary for operating on tree sets with context-free path languages.[7] But they also prove that the yield of the class of tree languages accepted by buTPDAs is the indexed languages. For the nature of gNCNs presented in this paper, the string language should be within the mildly context-sensitive languages (MCSLs); thus this type of TPDA is too powerful.

However, it is possible to restrict the power of a TPDA so that the string language accepted by the automaton is within the MCSLs. A proof is beyond the scope of this paper, but a sketch follows. TPDAs as currently defined allow the stack to be accessible at any point during the operation of the

[7]The path language for ASTs of the form in Figure 9 is $\{SA^*\}$ which is regular. But it is clear that the path language for the grammar, $\{SA^*\} \cup \{SA^*BA^*B \ldots A^*CA^*CA^* \mid \text{number of As and Bs is equal}\}$, is context-free.

automaton. Thus it is possible for the stack to be accessed on different paths; and so it is possible for paths to be dependent (e.g. one path in the tree is A^n, another is B^n). Grammars that generate MCSLs cannot have dependent paths (Weir, 1988). But if access to the stack is restricted to a single path—in the same manner that restricting stack passing to a single non-terminal child in an indexed grammar produces a linear indexed grammar (Gazdar, 1988), which generates MCSLs—the power of the TPDA is suitably restricted. The idea is related to the Embedded Pushdown Automaton (EPDA) of Vijay-Shanker (1987), although this is of course a string automaton rather than a tree automaton. Regardless of this, it is still not possible to precompile the annotation set, in the same way a PDA cannot be compiled out like an FSA; so the algorithm is still $O(n)$ time and space complexity in the number of nodes, but is also proportional to the size of the grammar.

5 Conclusion

In this paper we have given examples of situations in the mapping of trees where it is necessary to pair groups of non-contiguous nodes. We have shown how some types of non-contiguity can be represented formally using the idea of a grammar to group nodes in the tree; and then, treating this as a set of pattern trees in the sense of a bottom-up rewriting system in compiler theory, we have developed an efficient algorithm for this tree decomposition. Future work will involve looking at various practical aspects: how in the BURS costs can be determined, beyond the general notion of preferring larger pattern trees over smaller; how best to represent precompilation of annotations in the BURS algorithm; and so on.

References

A. Abeillé, Y. Schabes, and A. Joshi. 1990. Using lexicalized Tree Adjoining Grammars for Machine Translation. In *Proc. of COLING '90*.

J. Aissen and D. Perlmutter. 1976. Clause Reduction in Spanish. In *Proc. of the 2nd Annual Mtg of Berkeley Ling. Soc.*

B. Dorr. 1994. Machine translation divergences: A formal description and proposed solution. *Comp. Ling.*, 20(4):597–633.

M. Dras and T. Bleam. 2000. How Problematic are Clitics for S-TAG Translation? In *Proc. of TAG+5*, pages 241–244.

M. Dras. 1999. A meta-level grammar: redefining Synchronous TAG for translation and paraphrase. In *Proc. of ACL '99*, pages 80–87.

J. Eisner. 2003. Learning Non-Isomorphic Tree Mappings for Machine Translation. In *Proc. of ACL '03*.

D. Estival, A. Ballim, G. Russell, and S. Warwick. 1990. A syntax and semantics for feature-structure transfer. In *Proc. of TMI '90*, pages 131–144.

G. Gazdar. 1988. Applicability of indexed grammars to natural languages. In Uwe Reyle and Christian Rohrer, editors, *Natural Language Parsing and Linguistic Theories*. D. Reidel Publishing Company, Dordrecht, Holland.

D. Grune, H. Bal, C. Jacobs, and K. Langendoen. 2000. *Modern Compiler Design*. John Wiley, Chichester, UK.

C-h. Han, B. Lavoie, M. Palmer, O. Rambow, R. Kittredge, T. Korelsky, N. Kim, and M. Kim. 2000. Handling Structural Divergences and Recovering Dropped Arguments in a Korean/English Machine Translation System. In *Envisioning Machine Translation in the Information Future*, pages 40–53. Springer-Verlag.

A. Joshi. 1987. An introduction to Tree Adjoining Grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins.

I. Mel'čuk. 1988. *Dependency syntax: theory and practice.* State University of NY Press.

T. Proebsting. 1995. BURS automata generation. *ACM Trans. Programming Languages and Systems*, 17(3):461–486.

O. Rambow and A. Joshi. 1997. A Formal Look at Dependency Grammars and Phrase-Structure Grammars, with Special Consideration of Word-Order Phenomena. In Leo Wanner, editor, *Current Issues in Meaning-Text Theory*. Pinter, London.

J. Rogers. 1994. Capturing CFLs with Tree Adjoining Grammars. In *Proc. of ACL '94*, pages 155–162.

K. Schimpf and J. Gallier. 1985. Tree pushdown automata. *J. of Comp. and System Sciences*, 30:25–40.

S. Shieber. 1994. Restricting the Weak-Generative Capacity of Synchronous Tree Adjoining Grammars. *Computational Intelligence*, 10(4).

K. Vijay-Shanker. 1987. *A study of tree adjoining grammars*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

D. Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, Dept. of Comp. and Info. Science, Univ. of Pennsylvania.

D. Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Comp. Ling.*, 23(3):377–404.