Querying and updating treebanks: A critical survey and requirements analysis

Querying and Updating Treebanks: A Critical Survey and Requirements Analysis

Catherine Lai and Steven Bird
Department of Computer Science and Software Engineering
University of Melbourne, Victoria 3010, Australia
{clai,sb}@csse.unimelb.edu.au

Abstract

Language technology makes extensive use of hierarchically annotated text and speech data. These databases are stored in flat files and manipulated using corpus-specific query tools or special-purpose scripts. While the size of these databases and the range of applications has grown rapidly in recent years, neither method for managing the data has led to reusable, scalable software. The formal properties of the query languages are not well understood. Hence established methods for indexing tree data and optimizing tree queries cannot be employed. We analyze a range of existing linguistic query languages, and adduce a set of requirements for a reusable, scalable linguistic query language.

1 Introduction

Corpora form the backbone of language technology. However corpora usually need to be annotated with structural information describing, for example, syntax or phonology. Query languages are necessary to extract useful information from these massive data sets. Moreover, annotated corpora require thousands of hours of manual annotation to create, revise and maintain. Query languages are also useful during this process. For example, queries can be used to find parse errors or to transform annotations into different schemes.

However, the query languages currently available for annotated corpora suffer from several problems. Firstly, updates are not supported as query languages focus on the needs of linguists searching for syntactic constructions. Secondly, their relationship to existing database query languages is poorly understood, making it difficult to apply standard database indexing and query optimization techniques. As a consequence they do not scale well. Finally, linguistic annotations have both a sequential and a hierarchical organization. Query languages must support queries that refer to both of these types of structure simultaneously. Such hybrid queries should have a concise syntax. The interplay between these factors has resulted in a variety of mutually-inconsistent approaches.

This paper aims to describe the query language requirements for navigating and modifying structurally annotated corpora. We focus on tree structured annotation. This entails sequential structure.

This paper is organized as follows. Section 2 surveys six linguistic tree querying languages and where appropriate the data models they are based on. The survey is the basis of the linguistic tree query requirements presented in Section 3. We conclude in Section 4 with suggested areas for further work.

2 Tree Models and Query Languages

The prototypical hierarchical linguistic annotation is the syntax tree, an ordered tree where terminals (leaves) contain the text of a sentence being analyzed. Non-terminals represent syntactic categories, and the hierarchical organization represents constituency. The leaf level is usually considered immutable. The queries in Figure 1 have been chosen to highlight the expressive capabilities of current tree query languages.

A query language must be able to accurately specify which subtrees to match in a corpus. This means quantifying the existence of nodes and succintly stating the relationships between them. Q1 is a simple query based on the dominance relation inherent in trees. As mentioned earlier, however, the sequential ordering of nodes is also an important factor. Q3 and Q4 demonstrate the precision that is available for describing subtrees constrained by both dominance and precedence relations.

It is also desirable to specify subtrees by what they do not contain. This requires some form of negation. Q2 is a simple example of this type of

Q1. Find sentences that include the word ‘saw’.
Q2. Find sentences that do not include the word ‘saw’.
Q3. Find noun phrases whose rightmost child is a noun.
Q4. Find verb phrases that contain a verb immediately followed by a noun phrase that is immediately followed by a prepositional phrase.
Q5. Find the first common ancestor of sequences of a noun phrase followed by a verb phrase.
Q6. Find a noun phrase which dominates a word *dark* that is dominated by an intermediate phrase that bears an L-tone.
Q7. Find an noun phrase dominated by a verb phrase. Return the subtree dominated by that noun phrase only.

Figure 1: Syntactic Queries for Comparing Tree Query Languages

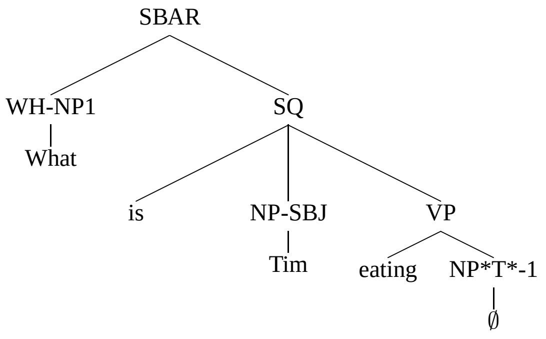


Figure 2: Trace elements in Penn Treebank

query. Q5 contains an implicit negation: we need to select the common ancestor that has no descendant that is also a common ancestor of the nodes in question. These queries also explore the interaction between quantification and negation of nodes and subtrees.

Linguistic query languages, in general, need to be able to deal with heterogeneous data. Many interesting queries fall on the interface of linguistic fields. For example, Q6 requires both syntactic and phonological data. This can be represented as two trees that intersect at the word level (Cassidy and Harrington, 2001).

Finally, trees can be large, and it is often undesirable to return whole trees for which the query tree matches a tiny fragment. Thus, we need a way to specify what part of a query simply constrains context, and what part should be returned. Q7 is a simple test of a query language’s ability to control output.

2.1 Penn Treebank and Tgrep2

Penn Treebank contains approximately 50,000 parse trees of Wall Street Journal text (Marcus et al., 1994). Each parse is represented as an ordered tree, and syntactic dependencies are indicated using

```
Q1.  S << saw
Q2.  S !<< saw
Q3.  NP <- N
Q4   VP=vp << (V . (N >> =vp . PP >> =vp))
Q5.  *=p << (NP=n .. (VP=v >> =p
               !>> (* << =n >> =p)))
Q6.* Not expressible
Q7.  VP << ‘NP
```

Figure 3: Tgrep2 Queries; Q5 is taken from (Rohde, 2001)

zero-width *trace* elements co-indexed with a full noun phrase (cf. Figure 2).

Tgrep2 is a *grep*-like tool for this database (Rohde, 2001), and all example queries can be specified correctly except Q6 (cf. Figure 3). Queries are nested expressions involving nodes and relationships between nodes. Nodes are specified using strings or regular expressions. Tgrep2 supports a large number of node relationships, including immediate precedence: `A` immediately precedes `B` (ie `A . B`) if the right corner of `A` immediately precedes the left corner of `B` in the sentence on which the tree is built. A wildcard (`*`) can be used for a node when there is no constraint on its name. Node identifiers allow multiple sub-expressions to refer to the same node (e.g. Q4). Tgrep2 can specify non-inclusion or non-existence (Q2, Q5).

Query execution uses a binary file representation of the data, including an index on the words in the trees. The top node in a query is first matched and the rest of the tree is matched recursively, constrained by node relations. The output is a set of human-readable subtrees. The subtree matched by the first node mentioned in the query is returned by default, but another return subtree can be specified by the user as shown in Q7. This output cannot be queried, since it must first be converted into the binary format; i.e. Tgrep2 is not compositional.

Not all node relationships are primitive (e.g. sibling relation (`$`)). However, dominates type relations such as leftmost descendant (`<<,`) cannot be derived. The depth of such a descendant is unknown. Thus, the language could be greatly simplified with a closure operator.

2.2 The TIGER Corpus and TIGERSearch

The TIGER corpus contains syntactically annotated German newspaper text (Brants et al., 2002). The syntax of German permits *discontinuous constituents*, which are represented in the corpus as trees with crossing edges. TIGERSearch is a logic and constraint programming approach to querying

```
Q1.   #s:[cat="S"] & #l:[lex="saw"]
      & #s >* #l
Q2.*  #s:[cat="S"] & #l:[lex="saw"]
      & #s !>* #l
Q3.   #n1:[cat="NP"] & #n2:[pos="N"]
      & (#n1 >* #n2) & (#n1 >@r #n3)
      & (#n2 >* #n4)
Q4    #vp:[cat="VP"] & #v: [pos="V"]
      & #np:[cat="NP"] & #pp:[cat="PP"]
      & #vp >* #v & #vp >* #np
      & #vp >* #pp & #v >@r #vr
      & #np >@l #npl & #vr .1 #npl
      & pp >@l ppl & npl .1 ppl
Q5.*  #vp:[cat="VP"] & #np:[cat="NP"]
      & (#x >* #v) & (#x >* #np)
      & (#v .* #np)
Q6.   Not expressible
Q7.*  #vp:[cat="VP"] & #np:[cat="NP"]
      $ (#vp >* #np)
```

Figure 4: TigerSearch Queries; (*) Queries are approximations and may not produce the correct output

these so-called “syntax graphs” (König and Lezius, 2001). Query graphs are built using two main relations: immediate dominance (>) and immediate precedence (`.`). Closures of these relations (`>*` and `.*` respectively) and other node relations such sibling (`$`) and left and right corners (`>@l`,`>@r`) increase expressiveness. However, intersecting hierarchies are not supported (Q6).

`A` precedes `B` (i.e. `A .. B`) if the left corner of A precedes the left corner of B. Immediate precedence means the distance between left corners is 1. Queries requiring immediate precedence (e.g. Q4) will not be correctly described using this relation. Left and right corners can be used to define the query we want (cf. fig 4). However, this mimics the Tgrep2 precedence definition and may fail if the syntax graph has crossing branches.

Nodes are implicitly existentially quantified before the graph description. This means non-inclusion in Q2 will fail unless the negated node exists in the graph. The expression of Q5 is incorrect as the non-existence of a lower common ancestor cannot be established. The closest approximation will include all common ancestors. However, we can use the (somewhat unintuitive) fact that a rightmost child of a node must dominate the right corner of that node to formulate Q3.

The corpus of syntax graphs is indexed before querying. This index contains inferred facts about the corpus graphs with respect to TIGERSearch relations and predicates. In effect, the corpus data becomes a prolog fact database. The query processor attempts to unify elements of corpus graphs with

```
Q1.  [Syntax=S ^ Word=saw]
Q2.* Not expressible
Q3.  end(Syntax=NP, Syntax=N)=1
Q4.* [Syntax=VP ^ [Syntax=V ->
          [Syntax=NP -> Syntax PP] ]]
Q5.* [Syntax!=x ^ [Syntax=NP -> Syntax=VP]]
Q6.  [Syntax=NP ^
           [Word=dark ^ intermediate=L-]]
Q7.? [Syntax=VP ^ #Syntax=NP]
```

Figure 5: Emu Queries

the query graph. Exhaustive search is avoided using label based filtering and by re-ordering the query graph traversal.

Matching syntax graphs are returned in their entirety. This means output reduction of Q7 cannot be done.

2.3 Emu Query Language

The Emu speech database system (Cassidy and Harrington, 2001) defines an annotation scheme involving temporal constraints of precedence and overlap. Emu annotations are stratified into levels; each level is an interval structure, and elements on each level overlap those on the same and on other levels. The overlap relation is called dominance in the Emu documentation, but it is a reflexive, symmetric and non-transitive relation best understood as temporal overlap. Given this approach to dominance, nodes in an Emu structure can be “dominated” by multiple parent nodes. Hence Emu claims to support multiple intersecting hierarchies. Example Emu queries are given in Figure 5.

Built in functions `start()`, `end()`, `mid()` and `position()` allow expression of the positional constraint in Q3. However lack of precision in dominance and precedence relations is a problem when dealing with syntax. Immediate dominance is only expressible between the appropriate levels. However, syntax trees do not easily split into such identifiable levels. Precedence is only defined for nodes on the same level. There is no way to describe immediate precedence within a level. Negated relations are not possible hence Q2 type non-inclusion cannot be expressed either.

Each query only returns one node per match. The target node can be specified by the user (Q7). However, this is unhelpful if structure is of interest. It also prevents query composition. Query constraints are tested in all possible match positions in the structure. This is satisfactory for small data sets but does not scale up well.

```
Q1.  node: S
     query: saw exists
Q2.  node: S
     query: !saw exists
Q3.  node: NP
     query: NP iDomsLast1 N
Q4.? node: VP
     query: V iprecedes NP
        and NP iprecedes PP
Q5.* Not expressible
Q6.* Not expressible
Q7.  node: VP
     query: exists NP
     node: NP
     query: exists *
```

Figure 6: CorpusSearch Queries

2.4 CorpusSearch

CorpusSearch was developed for the Penn-Helsinki Parsed Corpus of Middle English, though it can be used with any corpus annotated in the Penn Treebank style. An interesting feature of this language is that queries are limited in scope to subtree rooted by a specific type of node. In Q1 `exists` asks that the word 'saw' exists in a subtree rooted with an 'S'. Surprisingly, the search function (relation) `dominates` has been discontinued. This means nested dominance queries cannot be expressed. Precedence is only defined among siblings which greatly restricts the number of queries possible. Q4, for example, will miss many hits.

Negation of dominance has the semantics required for Q2. However, `VP precedes !NP` will not match verb phrases that are rightmost amongst their siblings. Search functions such as `iDomsLast` exist to express this sort of positional constraint instead. Wildcards can be specified, however the lack of a dominance search function means Q5 cannot be expressed. Multiple domination, as in Q6, is not supported.

A striking feature of this language is that it treats regular expressions over node names as variable names. Thus `A iprecedes B*|C and B*|C iprecedes D` describes a sequence of three nodes, while `A iprecedes B*|C and C|B* iprecedes D` describes two unrelated sequences, each of two nodes.

CorpusSearch is compositional. This allows Q7 to be specified in two stages.

2.5 NiteQL

NiteQL extends the MATE workbench query language Q4M (McKelvie et al., 2001) and has been released as part of the NITE XML Toolkit (Heid et al., 2004). Queries consist of weakly typed variable declarations followed by match conditions. Matches are evaluated over attribute, structure and time constraints. Precedence can be defined by the application designer depending on the data model.

```
Q1. ($s syntax) ($w word):
       ($w@orth=="saw") && ($s@cat=="S")
        && ($s ^ $w)
Q2.*($s syntax) ($w word):
       ($w@orth=="saw") && ($s@cat=="S")
        && !($s ^ $w)
Q3. ($np cat) ($w word) :
       ($np@cat=="NP") && ($w@pos=="N")
        && ($np ^1[-1] $w)
Q4. ($vp syntax) ($v word)
    ($np syntax) ($pp syntax):
        ($v@pos=="V") && ($np@cat=="NP")
        && ($pp@cat=="PP")
        && ($vp@cat=="VP")
        && ($v <>1 $np) && ($np <>1 $pp)
        && ($vp ^ $v) && ($vp ^ $np)
        && ($vp ^ $pp)
Q5.*($vp syntax) ($np syntax) ($x syntax):
        ($vp@cat=="VP") && ($np@cat=="NP")
        && ($x ^ $vp) && ($x ^ $np)
        && ($np <> $vp)
Q6. ($s syntax) ($i intermediate)
    ($w word):
       ($s@cat=="NP") && ($i@tone=="L-")
        && ($w@orth=="dark")
        && ($s ^ $w) && ($i ^ $w)
Q7. (exists $vp syntax) ($np syntax):
       ($vp@cat=="VP") && ($np@cat=="NP")
        && ($vp ^ $np)
```

Figure 7: NiteQL Queries

The queries in Figure 7 assume the model used by Tgrep2. If crossing branches are permitted, and the left corners precedence definition is used, then NiteQL will behave like TIGERSearch instead. Dominance and precedence relations can take modifiers that provide more positional constraints. In Q3, `^1` indicates immediate dominance and `[-1]` indicates rightmost descendant. Any sibling position at any level can be specified.

Like TIGERSearch, variables are existentially quantified when declared so Q2 and Q5 cannot be correctly expressed. Quantification (`exists`, `forall`) can be used in variable declarations. However their main purpose is to suppress marked in query output as used in Q7.

The NITE project encourages storage in standoff XML. An XPath-like pointer relation enables the formation of secondary (non-tree) edges between

```
Q1.  /S[//_[@lex = 'saw']]
Q2.  /S[not //_[@lex = 'saw']]
Q3.  //NP{/N$}
Q4   //VP{V -> NP -> PP}
Q5.? //_{//NP --> VP}/
          ancestor-or-self::*[1]
Q6.* Not expressible
Q7.  //VP/NP
```

Figure 8: LPath Queries

nodes. These can also occur between separate hierarchies. The final output is an XML document listing pointers to matches in the corpus. This is useful for searches during the annotation process. NiteQL's type system allows queries on intersecting hierarchies as seen in Q6. Complex queries provide compositionality and can be used to structure results to some extent.

2.6 LPath

LPath is a path language for linguistic trees extending XPath, with with an immediate precedence and a scoping operator (Bird et al., 2004). LPath queries can be translated into SQL for efficient evaluation. The LPath versions of the example queries are shown in Figure 8. The LPath representation of Q5 follows from node set selection in XPath. The positional predicate `[1]` is applied as the ancestor axis is traversed. In this case, only the first node in *reverse document order* is selected. This is the first common ancestor required by the query.

3 Requirements for Tree Query

Tree query languages need to be able to express node relationships succinctly. The languages we have surveyed needed more than dominance and precedence relations to express the specialized relations invoked in the example queries. A reasonable definition of immediate precedence is clearly necessary. However, more is required than positive descriptions. Non-inclusion queries such as Q2 and Q5 could not be expressed in all languages.

3.1 Simple Navigation

Subtree Matching. A vital part of subtree matching is accurate specification of the query tree. An inventory of subtree description types is given in Figure 9. Subtree description may also be facilitated with a graphical interface that maps tree diagrams to expressions in a query language. This is an attractive option for non-computer scientists and already exists in tools such as TIGERSearch.

Immediate dominance:
A dominates B, A may dominate other nodes

Positional constraint:
A dominates B, and B is the first (last) child of A

Positional constraint with respect to a label:
A dominates B, and B is the last B child of A

Multiple dominance:
A dominates both B and C, but the order of B and C is unspecified.

Sibling precedence:
A dominates both B and C, B precedes C; A dominates both B and C, B immediately precedes C

Complete description:
A dominates B and C, in that order, and nothing else

Multiple copies:
A dominates B and B, and the two Bs are different instances

Negation:
A does not dominate node with label B

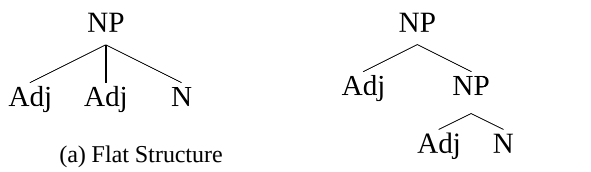
Figure 9: Subtree Matching Queries

Returning subtrees. The query languages had some capacity to constrain the output of a query as was shown in example query Q7. However, only NiteQL could choose specific nodes (as opposed to subtree roots) to output. This sort of precise reduction needs to be further supported.

Reverse navigation. Query specifications tend to reflect top down, left to right tree navigation. On the other hand, context can occur in any direction from a node. This is a problem for languages such as Tgrep2 where graph description focuses on one particular node at a time. Reverse relations, such as 'follows', are implemented in Tgrep2 and are necessary for queries such as Q5. However, reverse relations have less value in a language like TIGERSearch where the ordering of graph description is not important. In terms of expressiveness, necessity of these relations depends on the query structure.

However, reverse navigation is relevant for developing a matching strategy. For example, matching an ancestor only requires nodes on the path from the current node to the root. Strategies that allow reduction of the search space need to be employed.

Non-tree navigation. Queries are not always described in terms of edge traversal structure. Queries are often specified over the sequence of terminals (i.e. the text), regardless of hierarchical organization. Example queries have shown that this notion of sequential navigation needs to be extended to non-terminal nodes, e.g. to permit searching for sequences of one or more adjective



(a) Flat Structure

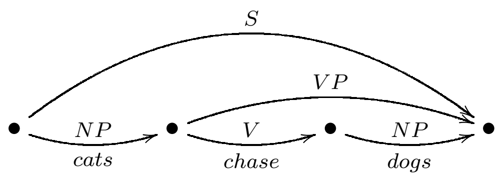
(b) Chomsky Adjunction

Figure 10: Two Representations for Optional, Repeatable Constituents

S → NP VP	cats chase mice
VP → V NP	NP chase mice
NP → cats	cats V mice
NP → mice	cats chase NP
V → chase	NP V mice
	NP chase NP
	cats V NP
	cats VP
	NP VP
	NP V NP
	S

(a) Phrase Structure Grammar

(b) Proper Analyses



(c) Syntactic Chart

Figure 11: Immediate Precedence in Linguistic Trees

followed by a noun, regardless of the internal organization of the noun phrase (cf. Figure 10). All surveyed languages support some notion of precedence though not all allow for immediate precedence. Immediate precedence in a tree can be conceived in terms of the "proper analyses" of early generative grammar (Chomsky, 1963) or syntactic charts, as shown in Figure 11.

3.2 Closures

The surveyed languages include closures of basic relations such as dominance, precedence and sibling precedence. These are necessary as distance between nodes of interest can be arbitrarily large. These are generally represented as separate relations in tree querying languages. However, closures are required of more complicated structures that are not handled currently.

We may wish to find parts of a corpus that fit a particular grammatical theory. For example, Chomsky adjunction in Figure 10 can be described by the productions: `NP` → `Adj NP`, `NP` → `N`. This translates to an LPath-like expression `(/NP[/Adj])*/N`. Here closure is atomic and each repetition involves a single step along some axis. In general we would like to be able to express any self-recursive rule $\mathtt{A} \rightarrow \mathtt{L}_1 \cdots \mathtt{L}_n\mathtt{AR}_1 \cdots \mathtt{R}_n$ using a closure. In LPath this might possibly be specified by `(/A[<=Ln...<=L1, =>R1...=>Rn])+`.

Closures involving more than one step are also required. For example, a path consisting of alternating `VP` and `S`'s: `(/VP/S)+`. Outside of tree navigation we wish to find regular sequences such as consonants and vowels: `(->C->V)+`. Moreover some structure may best be described using nested closures, e.g. `((->C)+->V)+`.

Q5 represents a class of queries that ask for the first common ancestor of tree fragment. However, negation semantics means this can only be specified in Tgrep2 and LPath. This problem may be addressed more generally with the greedy matching approach of regular expression processing. That is, signal that we want the first match only. However, this is not compositional.

3.3 Beyond ordered trees

Queries may need to extend beyond sentence boundaries. For example, anaphoric arguments may occur in previous sentences (Prasad et al., 2004). If trees represent sentences and querying is restricted to subtree matching this is a problem. One solution is to include multiple sentences in trees. However, this drastically increases the size of trees. Query trees are generally very small (if spread widely) so massive trees decrease filter effectiveness during query processing and have a bad effect on matching algorithms.

This presents a good case for querying over ordered forests. In fact this is necessary when querying the Verbmobil treebanks of spontaneous speech (Hinrichs et al., 2000). Here discourse turns are modelled to include repetitions, interjections, disfluencies and sentence fragments. These are represented as trees disconnected from surrounding well-formed sentences. Trees can occur wrapped in other trees as seen in Figure 12. VIQTORYA (Steiner and Kallmeyer, 2002) is a query language developed for these treebanks. However, this can be considered a subset of the TIGERSearch language so was not discussed in the survey.

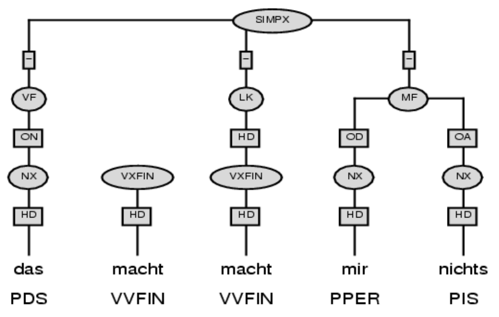


Figure 12: Forest representation of the Verbmobil corpus (Steiner and Kallmeyer, 2002)

There is a general need to move beyond single tree searches and integrate different types of linguistic data. Querying intersecting hierarchies has been well motivated by the workbenches such as Emu and the NITE project. There is also a need to query over relational and structural data. (e.g. Switchboard Treebank). We may want to match subtrees depending on the attributes of a word stored elsewhere (e.g. verb class in dictionary). Scope for these types of queries needs to be included in query language development.

Beyond this, there is a need to query non-tree structure. For example, Penn Treebank and the TIGER corpus includes secondary edges. It would be useful to navigate these links to extract information about the long range phenomena stored there. This means a definite move from tree based models which needs to be explored further.

3.4 Update

Curating a corpus of trees requires frequent updates. Tree edits often describe restructuring of constituents. For example, transforming structure to (resp. from) a small clause representation involves insertion (resp. deletion) of a common parent. Changing annotation style to reflect X-bar theory involves relabelling certain `NP` nodes to `N'`. Another useful transform is to reattach a phrasal adjunct to a higher level node, which calls for a notion of subtree movement.

Insertion, deletion and relabelling nodes are standard tree editing operations. However, linguistic trees are more constrained than general trees. Freedom of movement of constituents almost always depends on preserving the base text. Subtree deletion is not allowed (except zero-width elements) nor is re-ordering of leaves. Any subtree can only legally move to a limited number of locations without perturbing the text.

Subtree movement can be described in terms of node insertion and deletion. However, this will be extremely tedious for the user to specify as subtrees may be extremely large. Thus subtree movement should appear as a basic operation. (Cotton and Bird, 2002) present a tree edit operations all in terms of node movement of a distinguished node. The direction and surrounding structure determines where the node is reattached. Further operations are required to deal correctly with empty constituents. All update operations should have inverses so edits can be reversed.

Syntactically annotated corpora are often annotated with respect to a particular grammar. These grammars may be updated and annotations need to be changed to reflect this. However, it is inefficient to reannotate the entire corpus every time this happens. A useful update mechanism should be able to compare grammars and then implement changes only where necessary. The closures described previously will be useful here.

4 Conclusion

Several linguistic tree languages have been proposed in earlier work, and we have investigated their expressiveness and conciseness for a range of practical queries. Our survey has led us to propose a number of requirements for any general-purpose linguistic tree query language. They should permit hierarchical and sequential navigation, including an immediate precedence relation which cuts across the hierarchy. They should go beyond simple subtree matching to support a range of closures which correspond to grammar fragments, and positive and negative constraints on context. Whether as intersecting hierarchies or ordered forests, multiple tree querying must also be developed further. Requirements for a tree update language derive from a need to maintain the underlying text and present natural edit operations to the user.

We have broached several topics which require further investigation. Query languages incorporating variables, quantification, negation, and closures need to be better understood. This can be done by manually translating such queries to a language of first order logic or modal logic, exploring the kinds of nested quantification required, and consequences for implementation. The existing, well-understood relational and semi-structured query languages and finite automata could play a similar role.

Query expressions which can be mapped to the forwards and downwards subset of tree navigations are amenable to implementation in a streaming processor, opening the way for a true tree-grep tool which is able to function in a pipeline mode on unpreprocessed treebank files. Other areas of further work include an exploration of an appropriate typing system as used to navigate intersecting hierarchies; investigation of boundaries for contextual search; and the interaction of indexing and updates.

Acknowledgements

This research has been supported by an Australian Postgraduate Award (Lai), and a US National Science Foundation grant number 0317826 *Querying Linguistic Databases*, to the University of Pennsylvania (Bird).

References

S. Bird, Y. Chen, S. Davidson, H. Lee, and Y. Zheng. 2004. LPath: A path language for linguistic trees. Unpublished manuscript.

S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. 2002. The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories Sozopol*.

S. Cassidy and J. Harrington. 2001. Multi-level annotation in the Emu speech database management system. *Speech Communication*, 33(1-2):61–77.

N. Chomsky. 1963. Formal properties of grammars. In D. Luce, R. Bush, and E. Galanter, editors, *Handbook of Mathematical Psychology*, volume 2, pages 323–418. New York: Wiley and Sons.

S. Cotton and S. Bird. 2002. An Integrated Framework for Treebanks and Multilayer Annotations. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 1670–1677. ELRA.

U. Heid, H. Voormann, J-T Milde, U. Gut, K. Erk, and S. Pado. 2004. Querying both time-aligned and hierarchical corpora with nxt search. In *Fourth Language Resources and Evaluation Conference, Lisbon, Portugal*.

E. W. Hinrichs, J. Bartels, Y. Kawata, and V. Kordoni. 2000. The VERBMOBIL Treebanks. In *KONVENS 2000 Sprachkommunikation, ITG-Fachbericht 161*, pages 107–112. VDE Verlag.

E. König and W. Lezius. 2001. The TIGER language - a description language for syntax graphs. Part 1: User's guidelines. Technical report, University of Stuttgart, Stuttgart, Germany.

M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*.

D. McKelvie, A. Isard, A. Mengel, M. B. Moller, M. Gross, and M. Klein. 2001. The MATE workbench — an annotation tool for XML coded speech corpora. *Speech Communication*, 33(1-2):97–112.

R. Prasad, E. Miltsakaki, A. Joshi, and B. Webber. 2004. Annotation and Data Mining of the Penn Discourse TreeBank. In *Proceedings of the ACL Workshop on Discourse Annotation Barcelona, Spain*.

D. Rohde. 2001. Tgrep2 user manual.

I. Steiner and L. Kallmeyer. 2002. VIQTORYA – A Visual Query Tool for Syntactically Annotated Corpora. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 1704–1711. ELRA.