VisualTregex: A Graphical Tool for Querying Parse Trees

Mason Smith

***Abstract*—Searching parse tree databases is an important tool for traditional and computational linguists. We present a tool for constructing graphical queries on these corpora. We analyze the tool's design and suggestion possible extensions.**

I. Introduction

Treebanks are a valuable tool for analyzing parse structures in a given language. Many tools have been created for searching these treebanks. These tools vary greatly in scope, implementation, and power. However, the majority of theses search tools involve writing text-based queries, in a style reminiscent of regular expressions or SQL queries. However, trees have a natural non-flat structure. As a result, queries in many of these tools bear no resembles to the structures they are attempting to find. (Figure 2 is a good example of this).

We extend an existing tool to allow a user to construct queries graphically, rather than with text. Our intuition suggests that the queries created using this tool will be easier to construct and interpret than their textual counterparts.

II. Related Work

As the result of increasingly available structured and annotated corpora, many tools have been created to search through these sentence databases. Until now, these tools have all largely been text-based. We review the suite of tools on which our novel tool is based. These tools, however, represent only a small slice of the search tools produced over the years. See [1] for a comprehensive review of all of these tools. We also briefly discuss a graphical tool focused on tree creation and annotation, rather than searching, as well as TigerSEARCH, the only graphical tool for constructing queries.

A. Tgrep/Tgrep2

Tgrep was written in 1993 by Richard Pito. Based on the Unix-utility 'grep', tgrep allows searches on the command line from a preprocessed database of Penn Treebank data files. tgrep searches on basic relationships, such as domination, sibling, and precedence relations, as well as their transitive closures. tgrep2 provides a large number of improvements, allowing for significantly more powerful queries. Written by Doug Rohde, tgrep2 allows the user to specify full boolean relationships for every given relation. tgrep also permits the use of node labels which can be labeled and referenced later in the query. Possibly at the cost of clarity, this allows for a fuller range of graph-structured searches, rather than the limited tree-structured searches permitted by tgrep. Lai and Bird note in [1] that some of the non-primitive operators, such as leftmost descendant (<<,), have non-transitive closure equivalent and that a general closure operator would be a useful addition.

```
tregex:
NP <– NN

TIGERSearch:
#n1:[cat="NP"] & #n2:[pos="NN"]
& (#n1 >* #n2) & (#n1 >@r #n3)
& (#n2 >* #n4)
```

Figure 2. Query for finding a noun phrase whose rightmost child is a singular common noun. In tregex, there is a special operator (<-) for this 'rightmost child' relation, so this particular query is simple. In TIGERSearch, one needs to specify multiple dummy nodes to relate to the actual nodes in question.

B. Tregex

Tregex was created in 2006 by Roger Levy and Galen Andrew [2]. Tregex was designed to be largely syntax-compatible with tgrep2. However, tregex extends functionality through variable groups, headship, and constrained dominance and precendence relations. As an example of the latter, once can specify in tregex that an `NP` node must descend from an `S` node by a series of `VP` nodes of unspecified number, using the constrained dominance operator `S <+(VP) NP`. This can be done for a single `VP` node in tgrep2 by `S < (VP < NP)`, but a similar query would need to be specified explicitly for 0 or 2 or more intermediate nodes.

Unlike tgrep2, tregex is a Java-based tool and gives the user a simple GUI for performing queries and storing recent searches and results (Figure 1). It also provides a classic command-line interface. tregex also requires no preprocessing, though this results in slower searching than through tgrep2. Our tool is implemented as an extension to tregex.

C. TIGERSearch

TIGERSearch was produced by König and Lezius in 2001. Like tregex, TIGERSearch is a GUI-based Java tool for searching corpora. The TIGER language used by TIGERSearch uses the concept of 'syntax graphs', yielding queries that are rather unlike the queries written in tregex. Unfortunately, the resulting queries tend to be much longer using TIGERSearch (Figure 2). Lai and Bird also argue in [1] that TIGERSearch is less powerful than Tregex.

TIGERSearch is novel in that it seems to be the only current tool for constructing queries visually. Nodes are constructed as small-ish windows. Local information about the node, such as the matched pattern and the node's arity restriction, is displayed around the node's 'window'. Relations are encoded as edges between the nodes. The type of relationship is encoded positionally, where the edge endpoints are attached

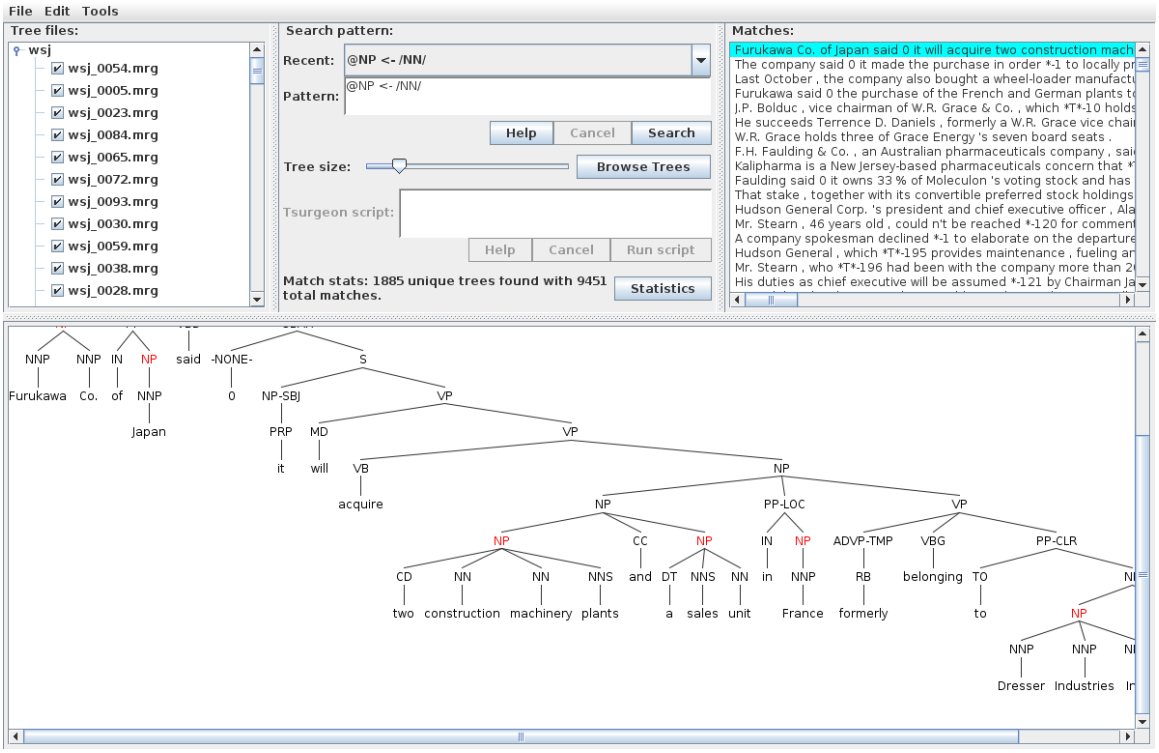

Figure 1. Screenshot of tregex.

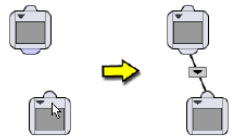

Figure 3. Dominance relationship in TIGERSearch.

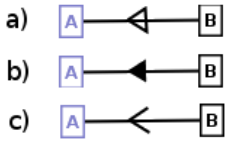

Figure 5. The basic symbols for (a) siblings, (b) dominance, and (c) precedence respectively.

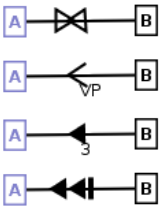

Figure 6. A sample of the relationship types represented as edges in VisualTregex. The depicted relations are

with respect to the nodes. For instance, two nodes exhibit a dominance relationship when one endpoint is attached to the bottom plug of the first endpoint and the other endpoint is attached to the top plug of the other node (Figure 3).

Internally, TIGERSearch translates these graphical queries into text queries and uses the results to perform the search. The reverse operations has not been implemented. Unfortunately, the tool has not been maintained since 2006, according to its website.

III. Methods

We create a tool called VisualTregex as an extension to tregex to construct queries graphically, similar to TIGERSearch. An example query is shown in Figure 4. The extension makes no changes to the tregex library, and only a menu addition to the tregex GUI.

A. Interface

The main interface for creating the query appears in a window outside of tregex. Nodes are represented as rectangles containing the pattern to be matched. When a label is provided for the node, the label is also displayed within the rectangular section in brackets. These nodes are manually positioned by the user. Relationships between nodes are displayed as edges. The type of the edge is displayed using a symbol in the middle of the edge. All of the implemented relations are categorized into three classes: dominance, precedence, and sibling. Each of the categories is represented by a basic symbol, as shown in Figure 5. From these basic symbols, the full set of relationship types is formed by modifying and annotating these symbols (Figure 6).

Properties of edges and nodes are modified in a side panel.

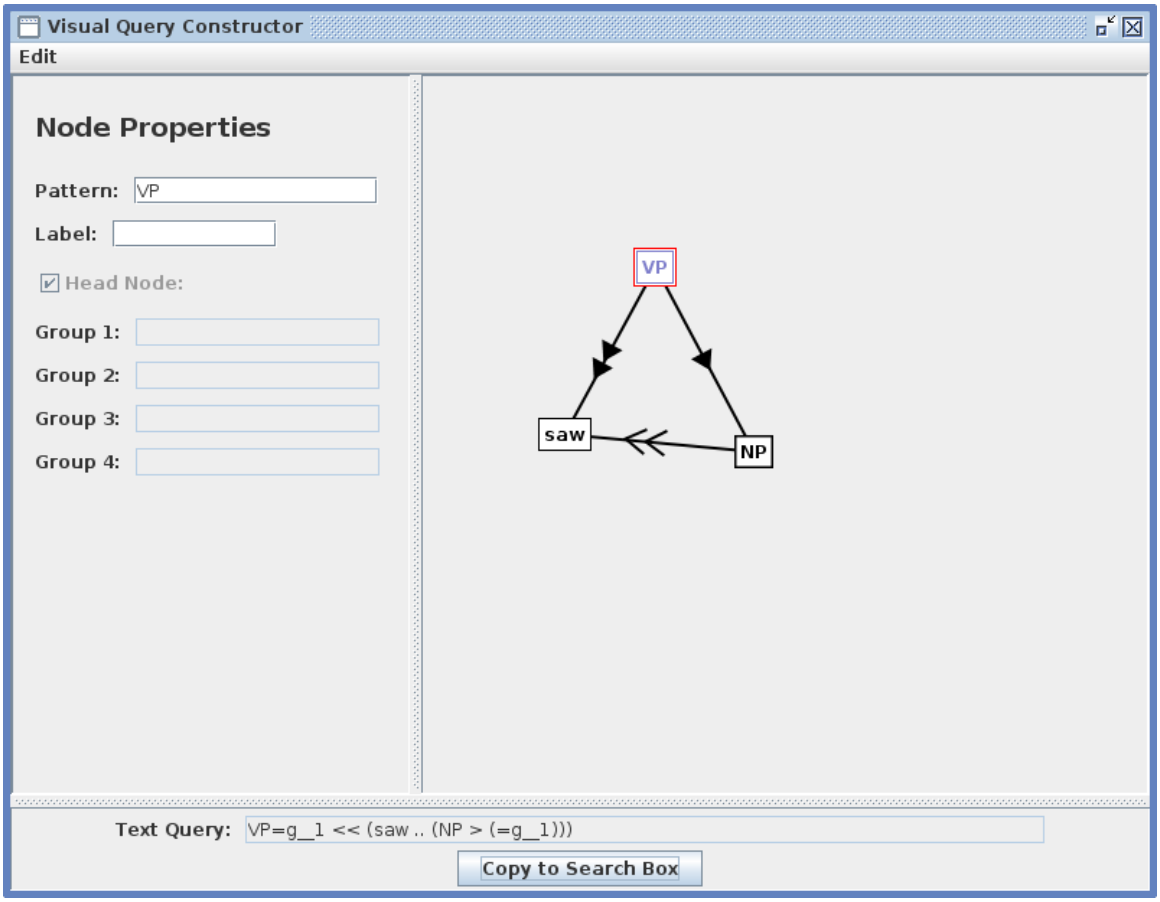

Figure 4. An example query constructed using VisualTregex.

The visual representation is updated in real-time as properties are changed in the side panel. In a bottom panel, the text translation of the current visual query is displayed. This translation is similarly updated in real time as the graphical query is modified. Once the user has finished the query, he can copy it over to the usual tregex search box, where can be run like any other text query.

B. Implementation

VisualTregex was implemented with roughly 2.2k lines of code, including comments and blank lines.

1) Text Query Translation: The graphical query is translated into a text query via a depth-first search algorithm. Starting at the marked head node, the graph is traversed depth-first. Whenever a new edge is discovered, the relation between the two nodes is recorded. Since every relationship in tregex has a 'passive' equivalent, we can ignore the direction of the relationship when traversing the graph. so that every edge is covered. When a vertex is discovered, it is recorded with its variable groups (if any) and label (whether user-provided or automatically generated prior to traversal). Whenever a vertex is encountered that has previously been visited, it is recorded only with the label instead. Once the initial text query is formed as above, the query is post-processed to remove redundant (auto-generated) labels.

IV. Discussion / Shortcomings

VisualTregex has a number of problems unique to its design. Some of them are correctable, but others are more fundamental to VisualTregex's design.

A. Ambiguity in Query Translation

The presence of negative and optional operators poses a potential problem for translation. In particular, the presence of a negative or optional edge in a cycle allows for multiple interpretations of the cycle, using the above algorithm implemented in this paper. An instance of this ambiguity is given in Figure 7. One solution to this is to allow the user to mark edges by priority. Instead of choosing randomly, the search algorithm would choose edges with higher priority first.

B. Unimplemented Operations

Some features of tregex are difficult to incorporate into a graphical tool. For instance, tregex allows for Boolean-OR'ing multiple relationships. This is currently unimplemented in VisualTregex. One could make multiple queries, for each OR possibility, but this is obviously quite tedious. The ideal method would be to group edges together by the OR relationship, perhaps using color as an encoding. In any case, this potentially poses the same type of ambiguity problem as in IV-A.

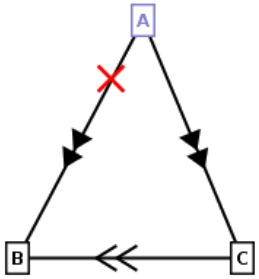

Figure 7. This query has an ambiguous interpretation. For instance, if the A, B edge is translated first, we get "A=a1 !>> (B .. (C << =a1))", which implies the existence of an A node that doesn't dominate a B node. However, if we translate the A, C edge first, we get "A=a1 >> (C „ (B !<< =a1))". This query searches for a node A that dominates a node C. C must be preceded by a node B that is not dominated by A.

V. Future Work

A number of potential improvements could be made to VisualTregex.

A. Automatic Layout

Currently, nodes need to be manually positioned by the user. Part of the appeal of using graphical tools is that resulting queries are easier to understand at a glance. Requiring the user to create this structure should be unnecessary. Due to the varying semantics of the different edge types, typical graph layout or hierarchical graph layout algorithms are most likely insufficient. One would prefer, for instance, that sibling relationships were more horizontal or that the leftmost descendant relationship were mostly vertical but slight left-leaning. Another useful feature would be an incremental layout, so the nodes don't pop into completely different positions when an edge is created or deleted. A force-directed approach, with desired angles or angle ranges for edges imposed by local 'magnetic fields', seems like a promising approach.

B. Automatic Validation

Currently, VisualTregex has minimal validation. For instance, the pattern specified for a given node must be a valid regular expression. However, the user has no indication as to whether a particular query represents an impossible structure. This is also a limit of the original tregex tool. Adding sufficient validation might possible involve modifying significant portions of the tregex search engine, sacrificing the modularity of VisualTregex's design.

C. Increased Visual Encoding of Semantic Information

Some easily visualizable features of the resulting query have yet to be displayed using VisualTregex. One simple addition might be connecting nodes that share a variable group by some visually distinct, automatically created edge.

References

[1] Catherine Lai and Steven Bird. Querying and updating treebanks: A critical survey and requirements analysis. In *In Proceedings of the Australasian Language Technology Workshop*, pages 139–146, 2004.

[2] Roger Levy and Galen Andrew. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *5th International conference on Language Resources and Evaluation*, 2006.