Tree pattern matching programs

There are two well-known groups of tree pattern matching programs. One is the tgrep/tgrep2/tregex grouping. The other is TiGerSearch.

TiGerSearch is an excellent GUI program, written in Java, which you can run on any major operating system, with many output options, etc. It has a powerful but complex query language over feature-value pairs and a GUI. The home page is here:

http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERSearch/index.shtml

A tutorial covering TiGerSearch by Florian Jaeger and Roger Levy is here:

http://www.stanford.edu/dept/linguistics/corpora/material/X_tgrep+%20tigersearch_tutorial_020604.pdf

But I never got into using it. Many people somehow don't seem to.

The tgrep/tgrep2/tregex family are broadly compatible with each other, but with dialectal differences. Tgrep was the original tree-matching program that came with the Penn Treebank. It was written in very unportable C code, and has never successfully run on anything but a SUN. You can still run it on an old SUN at Stanford, but basically, forget it. Tgrep2 was a reimplementation with extensions by Dough Rohde. It is command-line only and only comes ready-to-go for Linux (but it's probably possible to compile it on other Unix operating systems, though not everyone has succeeded). It's homepage is here:

http://tedlab.mit.edu/~dr/Tgrep2/

and tutorials at Stanford that discuss it include:

http://www.stanford.edu/dept/linguistics/corpora/material/X_tgrep+%20tigersearch_tutorial_020604.pdf
http://www.stanford.edu/dept/linguistics/corpora/cas-tut-tgrep.html

Tregex was written at Stanford by Roger Levy and Galen Andrew. It is also in Java and runs on any major operating system (after you have installed Java: if you need Java for your machine, go to:

http://www.java.com/getjava/	OR
http://java.sun.com/javase/downloads/	if you want more choices

Thereafter, download tregex from:

http://nlp.stanford.edu/software/tregex.shtml

For tgrep2 vs. tregex, each has some advantages and disadvantages. But I'll be using tregex. The below table summarizes their relative merits in a hopefully not too biased way:

	Tgrep2	tregex
Available for OSes other than Linux	No	Yes

Has a GUI	No	Yes
Command-line use	Yes	Yes
Has a decent manual	Yes	No
Supports macros	Yes	No
Speed	Fast (preindexed)	Slow (isn’t)
Can run on output of previous search	No	Yes
Headship, constrained dominance, and variables in queries	No	Yes

Exercise

Some of this exercise is adapted from a tutorial by Liz Coppock, Florian Jaeger, and Neal Snider, which you can now find here:

http://www.bcs.rochester.edu/people/fjaeger/teaching/tutorials/TGrep2/LabSyntax-Tutorial.html

You may find it interesting to look at the original. But note that you can only use the material as is with tgrep2 not tregex. In particular, it extensively makes use of macros….

These searches look for complement clauses that either have a "that" complementizer or no complementizer at all. An example would be "Neal didn't think (that) I would mess around with this".

Unless you know a lot about Penn Treebank structures, the first thing to do is to get a sense of how the Penn Treebank codes some structures. The verb 'tell' takes a lot of that-complements, so search for sentences with 'told' in them:

```
told
```

This suggests that roughly what we want is a VP with an SBAR under it that has either an IN that is 'that' or a -NONE-. So ones with 'that' are:

```
VP < (SBAR < (IN < that))
```

And ones with no 'that' are:

```
VP < (SBAR < /^-NONE-$/)
```

Note that things that aren't just letters have to be written as regular expressions…..

The first match of this expression raises analytic questions. Do we want to have sentences ending in 'researchers reported' count? Should that construction be rather seen as a parenthetical? We might decide to stick to cases where the SBAR is really part of the VP. One way to do that seems to be to make sure that the S node is not a -NONE- coindexed with something else:

```
VP < (SBAR < /^-NONE-$/ !< (S < /^-NONE-$/))
```

For most situations it makes most sense to count total matches. (Tregex gives you this by default; for tgrep2, you need to look at the -a and -f flags.) You need to check enough trees to see that the output is sensible. In particular, you might want to keep going until you find a tree with 2 matches, to check that you do want to count that as 2. Seems reasonable to me.

1. Write a corresponding query for VPs with an overt 'that' SBAR complement.

Let's start with the most basic statistics:

```
Number of SBAR in VPs with ‘that’:                   3415
Number of SBAR in VPs with 0 complementizer:         6745
```

R

For statistics in this class, we will use R. You can get it from http://www.r-project.org/ (follow the download links).

In another window fire up R. You can do arithmetic in R. It’s easier to do it in a calculator program, but it’s one place to start….

```
> 3415 / 6745
[1] 0.506301
> 6745 / (6745+3415)
[1] 0.663878
```

Deletion occurs almost exactly two-thirds of the time.

A simple question is whether that rate is true for particular verbs. Let’s try ‘tell’ in its various forms:

```
VP < (/^VB/ < tell|tells|told|telling) < (SBAR < /^-NONE-$/ !< (S < /^-NONE-$/))
```

2. Write the corresponding query for VPs with an overt ‘that’ SBAR complement.

We get the following statistics:

```
VP[tell] with SBAR[that]:      103
VP[tell] with SBAR[none]:       64
```

Less deletions of ‘that’ with ‘tell’!

```
> 64 / (103+64)
[1] 0.3832335
```

What about for ‘say’:

```
VP[say] with SBAR[that]:        669
VP[say] with SBAR[none]:       4965
```

Worrying fact: note that:

```
> (669 + 4965)/(3415 + 6745)
[1] 0.5545276
```

But the percentage of that-deletion is very different:

```
> 4965/(4965+669)
[1] 0.8812567
```

i.e., over half of all instances of that-clause complements in the corpus are with *say.* Such highly skewed distributions are unfortunately all too common with language.

Okay, so why don't we try it with a rarer verb, say *remark*:

VP[remark] with SBAR[that]:	3
VP[remark] with SBAR[none]:	1

Ah, that's more typical of what you get with language data, if worrying in a very different way….

The percentage of that-deletion here is easy to do in your head: 0.25.

And one more verb *know:*

VP[know] with SBAR[that]:	33
VP[know] with SBAR[none]:	62

Now we'll get to use R to do something other than be a calculator. A significance test. But first we'll do some stuff with Excel, to understand what is going on….

We'll make the 2x2 contingency table, and then run some tests on it…

```
> tell.say <- matrix(c(103, 64, 669, 4965), 2)
> tell.remark <- matrix(c(103, 64, 3, 1), 2)
> know.remark <- matrix(c(33, 62, 3, 1), 2)
> tell.say
     [,1] [,2]
[1,]  103  669
[2,]   64 4965
> chisq.test(tell.say)

        Pearson's Chi-squared test with Yates' continuity correction

data:  tell.say
X-squared = 344.384, df = 1, p-value < 2.2e-16

> chisq.test(tell.say, correct=F)

        Pearson's Chi-squared test

data:  tell.say
X-squared = 348.6874, df = 1, p-value < 2.2e-16

> chisq.test(know.remark, correct=F)

        Pearson's Chi-squared test

data:  know.remark
X-squared = 2.689, df = 1, p-value = 0.1010

Warning message:
```

```
Chi-squared approximation may be incorrect in: chisq.test(know.remark, correct = F)
> chisq.test(know.remark)

        Pearson's Chi-squared test with Yates' continuity correction

data:  know.remark
X-squared = 1.2305, df = 1, p-value = 0.2673

Warning message:
Chi-squared approximation may be incorrect in: chisq.test(know.remark)
> fisher.test(know.remark)

        Fisher's Exact Test for Count Data

data:  know.remark
p-value = 0.1351
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.00333321 2.35026666
sample estimates:
odds ratio
 0.1806426
```

The differences looked at above are for patterns of different verbs. It's important to realize that there are often lexical differences, but there are other hypotheses that are often rather more interesting to linguists. Here are two more interesting hypotheses to collect counts on.

A. Definiteness hierarchy

> *THAT-omission (the rate of ZERO complementizers) is higher for CCs with pronoun subjects than for CCs with definite NP subjects. For the current purpose we treat all and only NPs that start with "the" as definite NPs.*

The SBAR directly governs a clausal node /^S/ which directly governs a subject node /^NP-SBJ/. You can look up the relevant tags for determiner and for pronoun in a list of Penn Treebank tags or just ask about a word like *the* or *she.*

B. Adjacency

THAT-omission is lower for CCs that are not adjacent to the embedding verb.

Our current pattern finds *that* clauses anywhere in a VP. We might differentiate ones that are usually next to the verb from ones where there is intervening material. Rather than using the dominance operators (<, >), look at the precedence operators (.. . , ,, etc.). You can match any node with two underscores __. Think about how you can express this! One skill to learn is how to get clever at expressing ideas in tgrep language.

What types of interveners do you find in practice? Do you think there would be a difference between the different types of interveners and their effect on *that* omission? (Speculate).