

Partial Emulation of `tree-dvips.sty` in PSTricks (`pst-node` module)

Avery D Andrews
Jan 2006

The PSTricks package `pst-node` is far more powerful than Emma Pease's `tree-dvips`, but the latter strikes me as being frequently more convenient for tasks that arise in linguistics. So this is a partial emulation of `tree-dvips` built on the `pst-node` module of PSTricks. Most of the original `tree-dvips` commands work unaltered (`\(a)nodecurve` being the most important exception), although some have been extended to take advantage of the facilities of PSTricks.

So in this emulation, some `tree-dvips` facilities are extended to support additional possibilities afforded by `pst-node` (and its parent package PSTricks), while others will require (modest) changes in the source text to run. And there are a few completely new facilities which seem to me to be useful and in the spirit of `tree-dvips`, and easy to provide with `pst-node` available as the basis.

One of the advantages of `pst-node` is that many `.dvi` viewers understand its Postscript specials, so that lines will tend to be drawn. However not always quite correctly: in Yap, for example (the Postscript viewer that comes with MiKTeX), lines aren't drawn when they cross over nodes, resulting in wrong-looking displays of arrows leading into f-structures, for example. And some of the positioning commands don't work either. So if things look wrong, check the postscript output before attempting serious debugging of your line-drawing code.

It might also be worth mentioning that the PSTricks error-messages aren't always very illuminating, and the `tree-dvips` emulation layer does nothing to improve this situation.

The treatment is divided into basics, which can be used without serious knowledge of PSTricks, and advanced. It assumes previous knowledge of `tree-dvips`.

1 Basic Commands

1.1 `\node(point)`

The `\node` command looks just like it does in `tree-dvips`:

(1) `\node{label}{stuff}`

`stuff` can be anything, even a regular PSTricks node.

`\nodepoint` is slightly extended with an option for the final argument to be an angle, in parentheses:

(2) `\nodepoint{label}[horizontal displacement][vertical displacement]`
`\nodepoint{label}[displacement](angle)`

where the displacements are dimensions, and the angle is a number, 0 by default, so sole displacement is always horizontal to the right.

1.2 `\treelinewidth`

This is a dimension, setting the default width of the lines produced by the node connection commands. Its default is .4pt, rather thinner than default PSTricks lines. So if we want the thicken up the node connection lines a bit, this will do the trick:

(3) `\treelinewidth=.6pt`

The `linewidth` graphics parameter in individual line-drawing commands overrides this.


1.3 `\(a)nodeconnect`

`\nodeconnect` takes the same arguments as in `tree-dvips`, with a final optional `[]` argument taking PSTricks graphics parameters:

(4) `\nodeconnect[from loc]{from node}[to loc]{to node}[gr. params]`

The locations are the usual edges and corners, `t`, `tr`, `r`, `br`, `b`, `bl`, `l`, and `tl`, with the obvious interpretations.

So here's an example:

(5) Some fairly  explained
silly stuff

which is produced by this input:

(6) `\node{a}{\begin{tabular}{c}Some fairly\\silly stuff\end{tabular}}\hskip 3em`
`\node{b}{explained}`
`\nodeconnect[r]a[l]b[linewidth=1pt, nodesepB=.5ex]`

The final (new) [] argument allows many further properties of the line to be set; for example we get this:

(7)
$$\begin{array}{l} \text{Some fairly} \\ \text{silly stuff} \end{array} \longrightarrow \text{explained}$$

from this:

(8) `\node{a}{\begin{tabular}{c}Some fairly\\silly stuff\end{tabular}}\hskip 3em
\ode{b}{explained}
\nodeconnect[r]a[1]b[linewidth=1pt, nodesepB=.5ex, arrows=->>]`

For the full assortment of graphics parameters, see the PSTricks documentation, but here are some that strike me as useful in the present context:

- (9) a. `nodesep=Xdim`, `nodesepA=Xdim`, `nodesepB=Xdim`: the distance between the edge of the node and the endpoints of the connection line; `nodesep` applies to beginning and the end, `nodesepA` only to the beginning, and `nodesepB` only to the end.
- b. `arrows=ARR`: the style of arrow head, stated iconically (value of - gives no arrowhead)
- c. `linewidth=Xdim`
- d. `dashed`: (no value)
- e. `dash=Xdim Ydim`: `Xdim` is the length of the blank portion, `Ydim` of the dashes
- f. `dotted`: (no value)

`\anodeconnect` takes all the same arguments as `\nodeconnect`, but draws an arrowhead at the end of the connection. So from this:

(10) `\node{a}{\begin{tabular}{c}Some fairly\\silly stuff\end{tabular}}\hskip 3em
\ode{b}{explained}
\nodeconnect[r]a[1]b[nodesepB=.5ex]`

We get this:

(11)
$$\begin{array}{l} \text{Some fairly} \\ \text{silly stuff} \end{array} \longrightarrow \text{explained}$$

You can of course specify an `arrows` parameter with `anodeconnect`, but if you do, plain old `nodeconnect` would do the job as well.

1.4 `\arrow(width|length)`

These aren't implemented, since the PSTricks scheme is so different. Use:

```
(12) \psset{arrowsize=Xdim Y, arrowlength=Z. arrowinset=W}
```

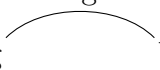
`pst-tree-dvips-emu.sty` sets defaults which look decent to me.

1.5 `\(a)nodecurve`

Not a full implementation, because the way in which PSTricks handles curvature of links between nodes is quite different from the way in which `tree-dvips` does (and, I'd say, is better). `tree-dvips` requires a final argument which specifies the curvature in terms of a dimension, with no sensible default, while PSTricks uses a number that does have a sensible default. The current `\anodecurve` command simply doesn't read this final argument, so it will be dumped into the text, so that an the input:

```
(13) \node{a}{the dog}\hskip 5em\node{b}{barked}  
      \nodecurve[tr]a[t1]b{2ex}
```

produces this wrong result

```
(14) the dog  barked 2ex
```

What we have instead to specify curvature is an optional argument in parentheses, containing one or two numbers, the bigger the number, the bigger the 'pull' of the curve away from its beginning or final endpoint. So to get this:

```
(15) the dog  barked
```

we write this:

```
(16) \node{a}{the dog}\hskip 5em\node{b}{barked}  
      \nodecurve[tr]a[t1]b(1.3)
```

And this:

```
(17) \node{a}{the dog}\hskip 5em\node{b}{barked}  
      \nodecurve[tr]a[t1]b(2,.4)
```

yields this:

(18) the dog  barked


And finally we get all the usual graphics parameters in square brackets at the end. Of these, there are two that are specifically important for curves, `angleA` and `angleB`. These specify the angle at which the line goes from its source and to its target node. If nothing is specified, this is determined by the position just as in `tree-dvips`. So we can ‘flatten’ the curve above by writing for example:

(19) the dog  barked

An angle of 0 degrees points to the right edge of the page.

1.6 `\(a)barnodeconnect`

As in `tree-dvips.sty`, except with final graphics parameters. So we get this:

(20) connect some words with a `barnodeconnect` command



from this:

(21) `\node{a}{connect}` some words with a
 `\node{b}{\tt barnodeconnect}` command
 `\barnodeconnect {a}{b}`

A useful set of options is `offset(A|B)`, which shifts the connection points to the left or right, so that:

(22) `\node{a}{connect}` some words with the
 `\node{b}{barnodeconnect}` command
 `\barnodeconnect [2ex]{a}{b} [offsetB=3em]`

produces:

(23) connect some words with the `barnodeconnect` command


And if we want an arrow at the end, we can use `\abarnodeconnect`, which takes all the same options.

1.7 `\nodetriangle`

`\nodetriangle{a}{b}[gr. params]` draws a triangle with apex at bottom of a, and base the top side of b (for ‘triangles of laziness’) in syntax trees).

1.8 `\node{circle|oval|box}`

Circle and oval require dimensions to be specified, but since I have hopes of eventually overcoming this limitations, the specifications are in parentheses even though they’re obligatory. For the circle there is one dimension, the radius, for the oval, two: (`height,width`). Here’s some sample input:

```
(24) \makespace{2em}\node{a}{A}
      \nodecircle{a}(1em)[linewidth=1pt, linestyle=dotted]\\[4ex]
      \makespace{2em}\node{a}{A}
      \nodeoval{a}(2em, 1em)\\[4ex]
```

These commands differ from the PSTricks `ovalnode`, etc, in that they don’t create nodes (ovalnodes containing text are bigger than a plain node containing the same text), but just draw lines around a pre-existing node.

The `\nodebox{node}` box works as in `tree-dvips`, drawing the box at a distance of `\nodemargin` around its content. Put as usual, there’s an optional square-bracketed graphics parameter argument, which can be used to specify the corner-rounding `framearc` parameter:

```
(25) a. Some Stuff
      b. \node{a}{Some Stuff$\/$}
         \nodebox{a}[framearc=.3]
```

2 More advanced

These commands depend on a bit more of a knowledge of PSTricks than the previous ones.

2.1 `\putpoint`, `\putstuff`

`\putpoint` one puts a PSTricks point node at a position relative to a `tree-dvips` node center or corner. The usage is:

(26) `\putpoint{new node name}{distance}(angle)[corner]{old node name}`

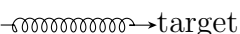
Where `distance` can be either a single dimension (distance), or a pair of dimensions `{horizontal, vertical}`, in which case the angle is ignored. For examples, see the file `treetest.txp`, where it is used in conjunction with `\psccurve` to put loops around contiguous sets of phrase-structure nodes.

`\putstuff` takes the same arguments, except that the first one is any regular text or L^AT_EX code, rather than a label for a point node.

2.2 <command>(options) arguments to `\nodeconnect`

In addition to the regular arguments, `\nodeconnect` can take an initial argument in angle-brackets, to specify the actual node connection command to be used, followed by graphics parameters in parentheses. This makes it easy to define variants of the node connection commands, such as `\coilconnect` below:

(27) a. `\def\coilconnect{\nodeconnect<\nccoil>(coilwidth=1ex,coilarmA=1ex,
coilarmB=2ex,arrows=->)}
\node{a}{source}\hskip 5em\node{b}{target}
\coilconnect[r]a[l]b`

b. source-target

Only `\pc...` commands will work here, their `\nc...` commands will produce enigmatic errors, such as ! Argument of `\nodeloc` has an extra `}`.

2.3 `\nodeloc`

This command delivers locations of nodes, taking the corner/edge as the first, argument, the node name as the second. These locations can only be used when `\SpecialCoor` is active, and must be enclosed in parentheses to work. For examples, look at some of the definitions in `pst-tree-dvips-emu.sty`.