A tool for drawing alignments

Scott Schwartz

October 29, 1993

Introduction.

This document describes a tool called *laps* which is used to generate annotated dotplot-like representations of DNA or protein alignments. *laps* is constructed in two layers. The outer layer is a command line interface that accepts source files that describe an alignment. The inner layer is a set of PostScript subroutines that provide a convenient framework for rendering alignments, along with a driver program called *dotplot* that sets defaults and emits the output. The glue between these layers is a Unix pipeline, the composition of several smaller programs, yielding a flexible and convenient way to extend the capabilities of the system. This tool has continuously evolved for more than two years to meet our changing needs.

The *laps* input language is designed to be conveniently parsed by text handling tools like *awk* or *perl*. It consists of a header line, a series of labeled blocks, and an optional footer line. The header is of the form

```
#:lav
```

while the footer is of the form

```
#:end
```

or *end of file*. Intervening blocks are of the form

```
keyword {
        ...
}
```

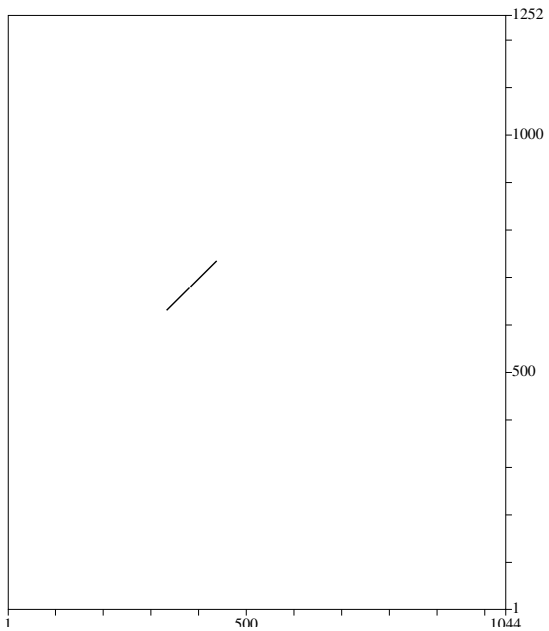
where valid keywords are `description`, `sequence`, `alignment`. The `description` record is a textual string documenting the provenance of the alignment. The `sequence` record describes the sequences that were used to construct the alignment. The record consists of a number of lines of the form "*filename*" *origin end*. Finally, the `alignment` records consist of a number of lines describing individual components of the alignment. Much of the substructure of the `alignment` record is optional, but at a minimum it will contain `line` records representing matches within that record. All of these may be abbreviated up to uniqueness. Thus a valid alignment file would look like:

```
#:lav
d {
  "SIM output with parameters:
  Used PAM200 matrix
  O = 12, E = 4"
}
s {
  "seq1" 1 1044
  "seq2" 1 1252
}
a {
  s 120.0
  b 333 631
  e 438 735
  l 333 631 381 679 26.5
  l 383 680 438 735 19.6
}
#:end
```

The alignment block describes a single, possibly disjoint, alignment. The internal structure shown above describes the score, beginning, ending, and weighted matches of that alignment.

Given a file of this form, one could use *laps* to generate output that would look like this:

/export/roke1/schwartz/align/laps.doc
SIM output with parameters: Used PAM200 matrix O = 12, E = 4 Tue Sep 7 15:50:41 EDT 1993



General structure.

Generating a picture using *laps* internally consists of two steps. First, the alignment file is translated into a more concrete format, enumerating the individual line segments that are to be rendered. This proceedure is handled by a utility program called *align2dot*, which emits calls to *PostScript* procedures that will actually render the alignments. For the input shown above, *align2dot* would generate something like

```
/description (SIM output with parameters:
Used PAM200 matrix
O = 12, E = 4) def
/x0 1 def /x1 1044 def /y0 1 def /y1 1252 def
plotdata
333 631 381 679 1.000000 0 Line
383 680 438 735 1.000000 0 Line
```

Notice that in addition to line drawing commands, *align2dot* has also computed the bounding coordinates of the plot, and recorded other information in *PostScript* variables for ready access.

The second step is to generate a complete *PostScript* program from the preprocessed alignment data and some user specified templates, using a utility program called *dotplot*.

Annotating the plot.

Annotations may be placed around the borders or inside of the plot. Hereinafter, the files containing these annotations will be refered to as *tag* files. Tag files contain stylised descriptions of the graphical or textual elements to be rendered and will contain calls to a library of *PostScript* routines which permit arbitrarily complex graphical additions to the basic plot. Tag files will generally be specified on the *laps* command line, as described below.

The structure of a tag file is as follows. First, a line with either `horizontal`, `vertical`, or `dotplot` indicates where these tags will be drawn. The `horizontal` and `vertical` operators signal that the subsequent annotations will be placed along the horizontal or verical edges of the diagram. The `dotplot` operator signals that the annotations will be drawn in the body of the diagram. Second, for the annotations on the edges, a line like `0 setrow` selects which row will containg the annotations. Row numbers start at zero and there is room for about four of them on a standard sized page. Third, several lines of the form

start end label feature

Here, *start* and *end* are addresses spanned by the feature. Labels are described using some notation: In *PostScript*, strings are delimited by parentheses, and lists are delimited by square brackets. Font names always start with a slash. Since we sometimes want to use several fonts in a label—for Greek letters or special symbols—we supply a list of lists, where each inner list contains a string, a font, and optionally a point-size and superscript (Sup) or subscript (Sub) indicator.

`[[ (` *string* `) /font option ] ...]`

Features are names like `Box, Arrow, ArrowBox, ThickArrow, Triangle`. They may be prefixed with the adjective `Left` or `Right`, and with the colors `Black, DarkGray, Gray, LightGray, White` or the patterns `Bars, Bars0, Bars1, Bars2`. Colors stay in effect until the next color change, so defaults are easy to establish. Comments extend from `%`-characters to the end of line. Normally the label runs parallel and just above the feature it names, but in the case of `Arrows` the label will be perpendicular to the arrow. A simple example looks like this:

```
% liverwort chloroplast genome
vertical
 0 setrow
  3679  3758 [[(trnL) /Times-Italic]] Right Arrow
  5859  9056 [[(rpoB) /Times-Italic]] Right Arrow
  9087 15971 [[(rpoC) /Times-Italic]] Right Arrow
 2 setrow
  White  81096  91153 [[(IR-B) /Times-Roman]] Right ThickArrow
  White 110976 121024 [[(IR-A) /Times-Roman]] Left ThickArrow
 3 setrow
  [[(liverwort chloroplast genome:) /Times-Roman]] Banner
```

You can use *PostScript* language features to add new names for objects. For example, the definitions

```
/Gene { White Box } def
/Exon { Null Black Box } def
/Alu { Null Black Triangle } def
/L1  { Null White ArrowBox } def
/InnerL1 { 0.8 setsize Null White ArrowBox } def
```

might be used as follows

```
23136 31154 Left L1
25903 28005 Left InnerL1
32425 32729 Right Alu
33004 33119 Left L1
39432 41003 [[(A) /Times-Roman 8 Sup] [(g) /Symbol]] Gene
39432 39576 Exon
```

Optional arguments.

The optional arguments to *laps* control details about how the page is laid out. These arguments are passed to *dotplot* for interpretation. They come in two versions. The first variety is supplied on the command line; these include the following:

```
-inv          generate a fourth quadrant plot
-ip           generate " " " " of the alignments only
-if           generate " " " " of the features only
-in           generate " " " " of the index only
-nc           don't clip to the plot boundaries
-nb           don't generate banner along the top edge
-nh           don't generate horizontal tics
-nv           don't generate vertical tics
-w#           set width to #
-h#           set height to #
-ppi#         set points-per-inch to #
-ht#          set horizontal tic spacing to #
-vt#          set horizontal tic spacing to #
-x#           set horizontal offset to #
-y#           set vertical offset to #
-noshowpage   don't emit a page at the end of this plot
-landscape    plot in landscape mode
-portrait     plot in portrait mode
x0 x1 y0 y1   bounding box of plot
```

These and other parameters are also controlled by a number of *PostScript* variables, which may be embedded in the input stream.

```
/inverted-dotplot    draw fourth quadrant plot
/inverted-features   draw features as in a fourth quadrant plot
/inverted-numbers    arrange axis labels for a fourth quadrant plot
/clip-features       clip the features to the dotplot bounding box
/override-width      set the plot width
/override-height     set the plot height
/override-scale      set the scale factor
/want-htic           turn on horizontal tic marks
/want-vtic           turn on vertical tic marks
/banner-line         turn on banner line
/htic-step           set horizontal tic increment
/vtic-step           set vertical tic increment
/x-offset            horizontal offset in inches on the page
/y-offset            vertical offset in inches on the page
/page-orientation    select landscape or portrait mode
/x0 /y0 /x1 /y1      bounding box of the plot
```

Thus, the operation of the *dotplot* program can be controlled by command line parameters, by input data, or a combination of the two. Our tools exploit this capability to automate certain operations. For example, a tool that generates alignment data in the internal format can include bounding box information, possibly for a subset of the data.

Here, inverted means "upside down". The option to suppress clipping is useful if you have features that extend slightly past the limits of the plot and you want them to be completely visible.

All of these options are available in *PostScript*, so you can embed them in, or in between, tag files if you want *laps* to vary its behavior during the run.

As it happens, *laps* is a simple shell script that calls two other programs to do the real work. Occasionally you may want to call them directly if there is a case the wrapper doesn't handle nicely. The first of these programs is called *align2dot*. It converts from alignment data format to the format expected by the next

program, which is *dotplot*. Now, *align2dot* takes an optional parameter giving the line width for the plot in units of printers points (1/72 inch), while *dotplot* takes the same optional arguments as *laps* plus an optional box bounding the coordinates to plot. Both programs read from standard input and write to standard output.

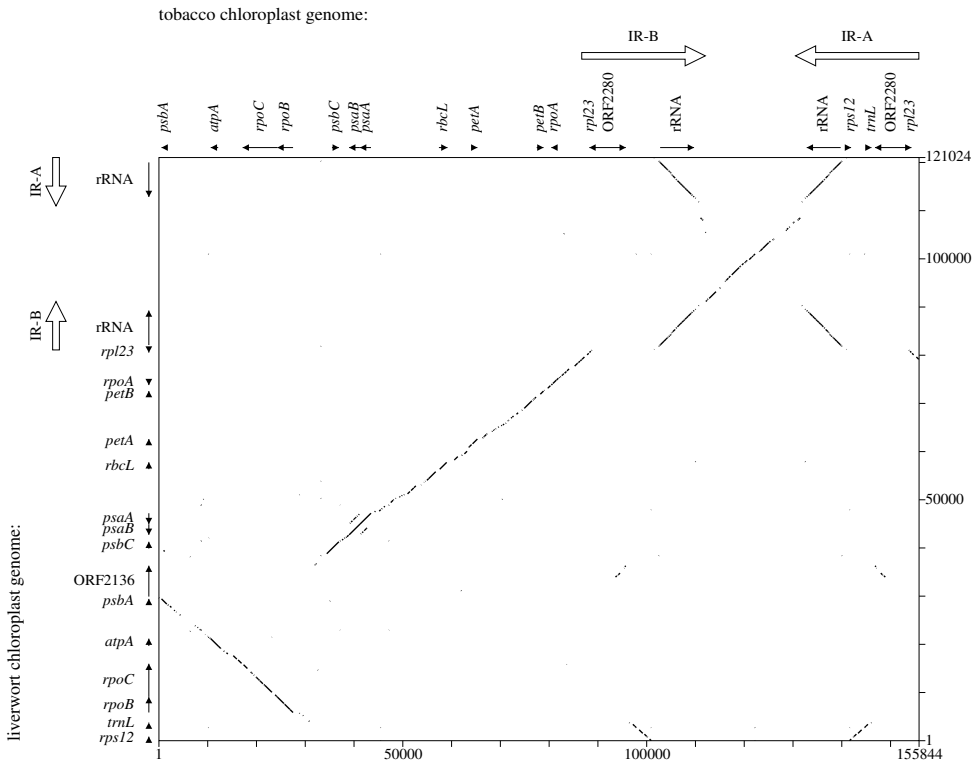
Some examples.

To see how all this fits together, consider this example shell script (line numbers added for exposition):

An alignment and its reverse compliment.

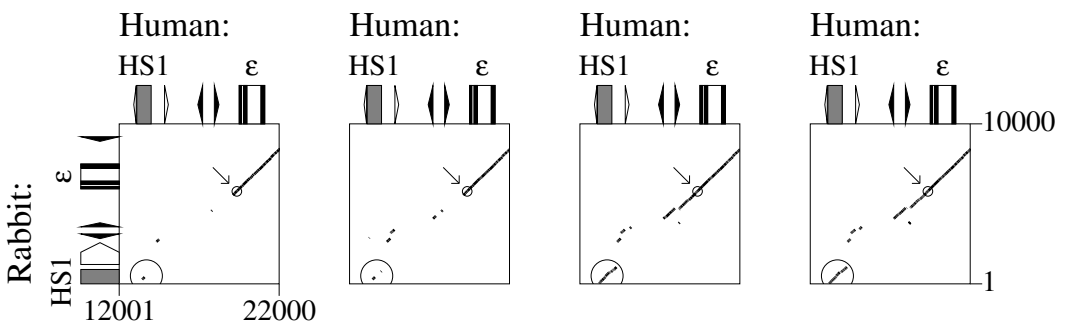
```
0   #!/bin/sh
1   ./revcomp liverwort.seq >liverwort.rev
2   { ./blast tobacco.seq liverwort.seq | align2dot 1
3     echo '/inverted-dotplot true def'
4     ./blast tobacco.seq liverwort.rev | align2dot 1
5     cat tobacco.tag liverwort.tag
6   } | dotplot | lpr -Plw0
```

Line zero tells the operating system which command interpreter we want to use; the Bourne shell in this case. Line one generates the reverse compliment of a DNA sequence. Line two computes an alignment between a tobacco sequence and a liverwort sequence, and sends it to *align2dot* for formatting into a one point wide plot. Line three echos a *PostScript* command that sets the `/inverted-dotplot` variable to be true. Line four computes another alignment, this time using the reverse compliment generated earlier, and formats it. Line five copies two tag files to standard output. In line six, the output from all the programs in lines two through five is grouped together and sent to *dotplot* for conversion to *PostScript*, and then to *lpr* for printing. All the items formatted in lines two through five will be processed in order by *dotplot*. In particular the `/inverted-dotplot` variable we set will be applied to the second plot generate (on line four), while the first one (on line two) will use the default settings. The output from the script looks like this:



Multiple plots per figure.

Sometimes one wishes to generate several related dotplots side by side, or stacked vertically. To illustrate that case, and a number of other features, consider the following plot:



That plot was constructed with the following shell script.

```
#!/bin/sh
{
 laps falign.8 -tag tag.style circles\
        -htag human.tag -vtag rabbit.tag\
        -a2d -clip 12001 22000 1 10000 \
        -dot -nc -nb -ppi12000 -nv -ht220000 -noshowpage\
```

```
         -portrait 12001 22000 1 10000
 laps falign.7 -tag tag.style circles\
        -htag human.tag\
        -a2d -clip 12001 22000 1 10000 \
        -dot -nc -nb -ppi12000 -nv -nh -x1.2 -noshowpage\
         -portrait 12001 22000 1 10000
 laps falign.6 -tag tag.style circles\
        -htag human.tag\
        -a2d -clip 12001 22000 1 10000 \
        -dot -nc -nb -ppi12000 -nv -nh -x2.4 -noshowpage\
         -portrait 12001 22000 1 10000
 laps keep.195.sim -tag tag.style circles\
        -htag human.tag\
        -a2d -clip 12001 22000 1 10000 \
        -dot -nc -nb -ppi12000 -vt10000 -nh -x3.6\
         -portrait 12001 22000 1 10000
} >4box.ps
```

The general structure of the script is simple: we call *laps* four times with varying parameters and collect the output in the file `4box.ps`. All four runs have a number of parameters in common. They all use the `-tag` option to load two tag files. The file `tag.style` is a "style sheet" containing definitions of features we often wish to represent, but does not draw anything itself. An example of a similar tag file was given above. The file `circles` file contains calls to routines which actually plot circular markers in the plot. It looks like this:

```
plotdata
        19355 5780 300 0 0 Circle
        13700 500 1000 0 0 Circle
        17855 7280 18855 6280 0 0 Line
        18355 6280 18855 6280 0 0 Line
        18855 6780 18855 6280 0 0 Line
```

The four invocations also use the `-htag` option, which implicitely prefixes the `horizontal` operator to the subsequent tag files. The human.tag file contains

```
1097 1198 [[(HS4) /Times-Roman]] HS
4537 4776 [[(HS3) /Times-Roman]] HS
8486 8866 [[(HS2) /Times-Roman]] HS
13100 14000 [[(HS1) /Times-Roman]] HS
```

Similarly, the `-vtag` option orders the subsequent tags to be placed on the vertical axis of a plot.

Each of the runs also includes the line `-a2d -clip 12001 22000 1 10000`. In this series of options, the `-a2d` flag signals that subesquent options are intended to be transmitted to the *align2dot* program. The `-clip x0 x1 y0 y1` option requests that the source alignment be clipped to the given rectangle. Analagously, the `-dot` flag signals that subsequent options will be passed to *dotplot*. The `-nc, -nb, -nv` and `-nh` flags suppress various items that are normally included in a plot: clipping to the plot boundaries, a banner line across the top of the page, and vertical or horizontal tic marks along the edge.

The `-ppi` option manually selects the size of the plot. Normally a plot will be scaled to fill most of the page, but with several to be rendered at once, a smaller size must be manually set. The argument is the number of points per inch, where a point is one unit in the dotplot's coordinate system. Thus, each of these small plots is 10000 base pairs wide, so we set the plot to be slightly smaller than one inch.

The `-ht` and `-vt` flags manually set the horizontal and vertical tic spacing. Normally the program automatically picks what it considers to be a pleasing default. The `-portrait` flag similarly overrides the program's judgement.

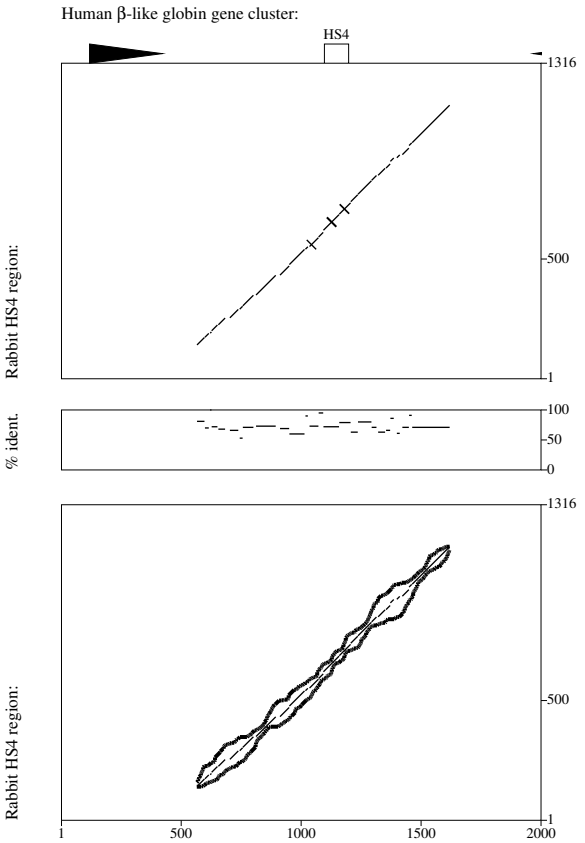
The first three invocations of *laps* are with the `-noshowpage` option, which supresses the normal page break at the end of a plot. The final plot then generates a page consisting of itself and the previous three.

The position of each subplot on the page is specified with the `-x` or `-y` option. The request `-x1.2` places a plot 1.2 inches to the right of the basic position on the page.

Finally, the bounding box for each subplot is explicitely given.

Annotations across multiple plots.

Sometimes one wishes to draw graphical annotations which span span multiple plots. For example, in the following figure, we draw a number of markers which correlate features of interest in two dotplots. [[Except that I can't find the correct figure10.]]



The script that generated this plot is as follows:

```
#!/bin/sh
DNAsites h0-3.rhs4.sim > /tmp/sites
percents -g h0-3.rhs4.sim > /tmp/perc
{
 laps h0-3.rhs4.sim /tmp/sites -w '1 10' -tag tag.style \
        -htag human.tag -vtag rhs4.tag -dot -nb -ppi400 -nh -y4.6 \
```

```
        -vt500 -portrait -noshowpage
 pmps -Y 250 -w 1 -align /tmp/perc -vtag percent.tag -dot -nb -ppi400 \
        -vt50 -nh -portrait -noshowpage -y3.65
 laps h0-3.rhs4.sim h0-3.rhs4.robust -tag tag.style \
        -vtag rhs4.tag -dot -vt500 -ht500 -nb -ppi400 -portrait
} > annotated.ps
```

Here we generate two alignment plots above and below a third kind of plot (percent match). The alignments use a tag file that draws lines from one coordinate system (the bottom plot) to another (the top plot).

1 Internals

Extending the set of operators provided by *dotplot* requires a degree of understanding of its internal organization. To that end, here is an outline of the major components of *dotplot*.

1. options processing 2. main 3. horiz, vert, plot modes 4. generating labels 5. scaling