LAV Users' Guide
Version 1.0

Scott Schwartz

May 13, 1993

1 Getting started

This document is an easy introduction to *lav* and related tools. *lav* is an application that provides a way to display and browse through the sequence alignments computed by a variety of programs. The companion programs provide high quality printed output. We assume that the reader already has some of our alignment programs (e.g. *sim*, *blast*, or *pblast*) and wishes to use the *lav* tools to display their output. Currently, *lav* needs to run under the X window system version 11; hardcopy output is generated for *PostScript* printers.

lav and related programs are normally stored together in some directory. For example, `/usr/local/bin`. You'll need to know what directory that is, and add it to your command search path. If you are using the C-Shell, that means doing something like this:

```
setenv PATH /some/directory/bin:$PATH
```

Also, if you have a *PostScript* printer available, you should make it is your default printer, with something like this

```
setenv PRINTER lw0
```

where `lw0` is the name of your printer.

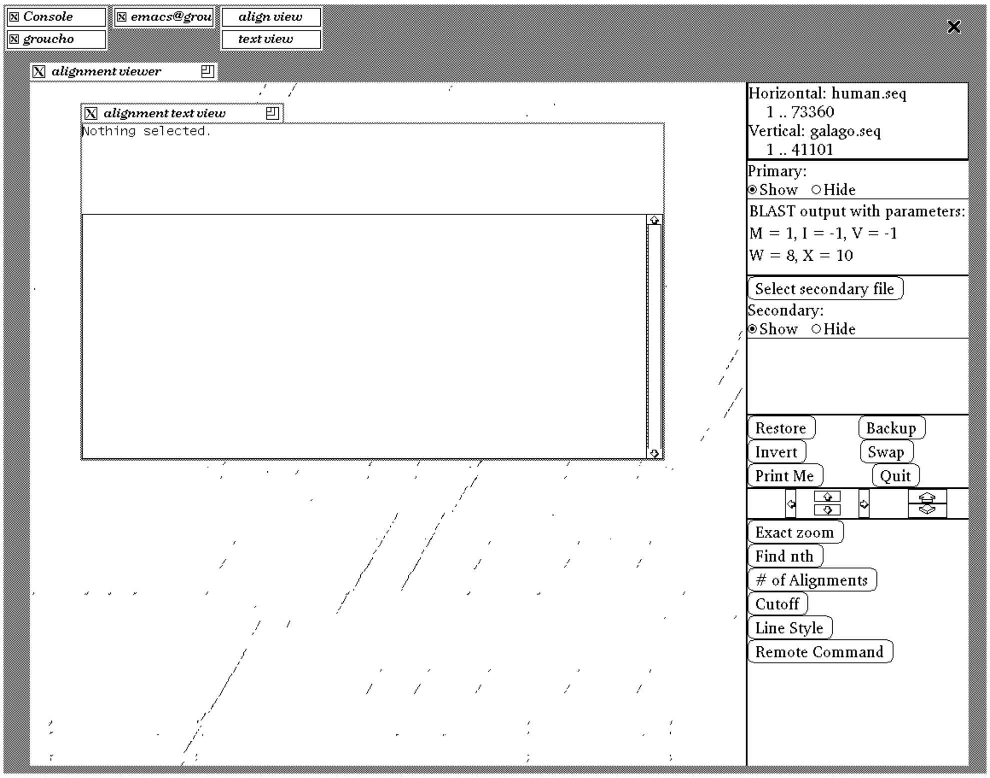
The data that *lav* uses is stored in two places; the alignment files that programs like *sim* output, and the sequence files that the alignment files refer to. For simplicity, we will assume that all the datafiles are in the current directory. If not, set the environment variable *SEQPATH* to point to the directory where the sequences are stored, like this

```
setenv SEQPATH /some/other/directory
```

Then, you should be able to run *lav* on some alignment file by typing

```
lav foo.align
```

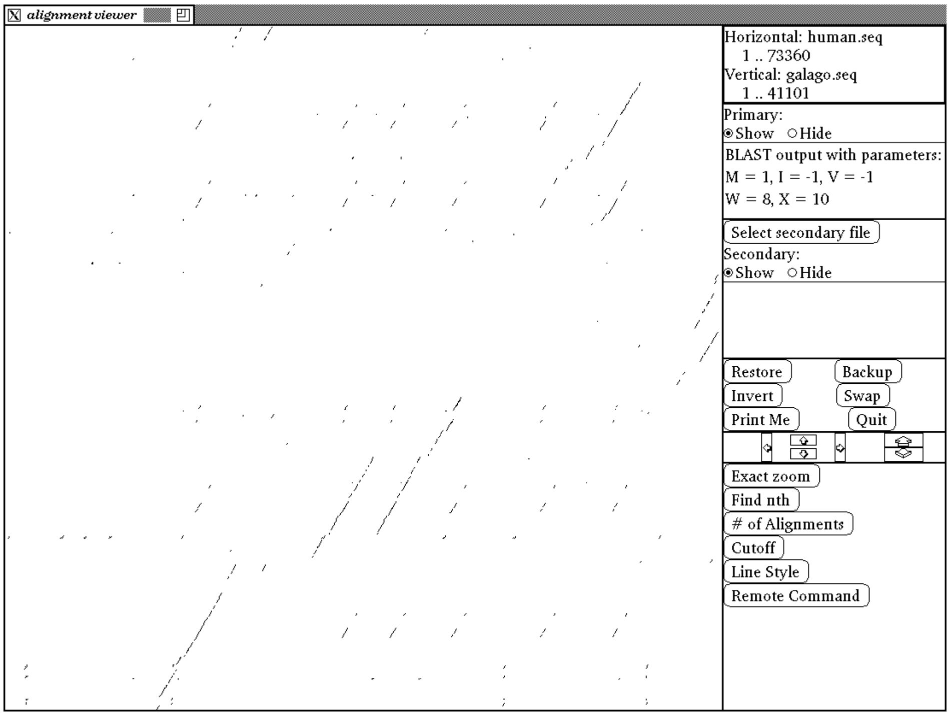
This tells *lav* to load the file `foo.align`. When *lav* starts running, it opens two top-level windows on the X display, which your window manager may ask you to place. One of these contains the *dotplot view* and a number of control panels. The other contains the *text view* and is initially empty except for the message "Nothing selected". On my workstation, running with some sample data, it looks like this:



The workstation screen

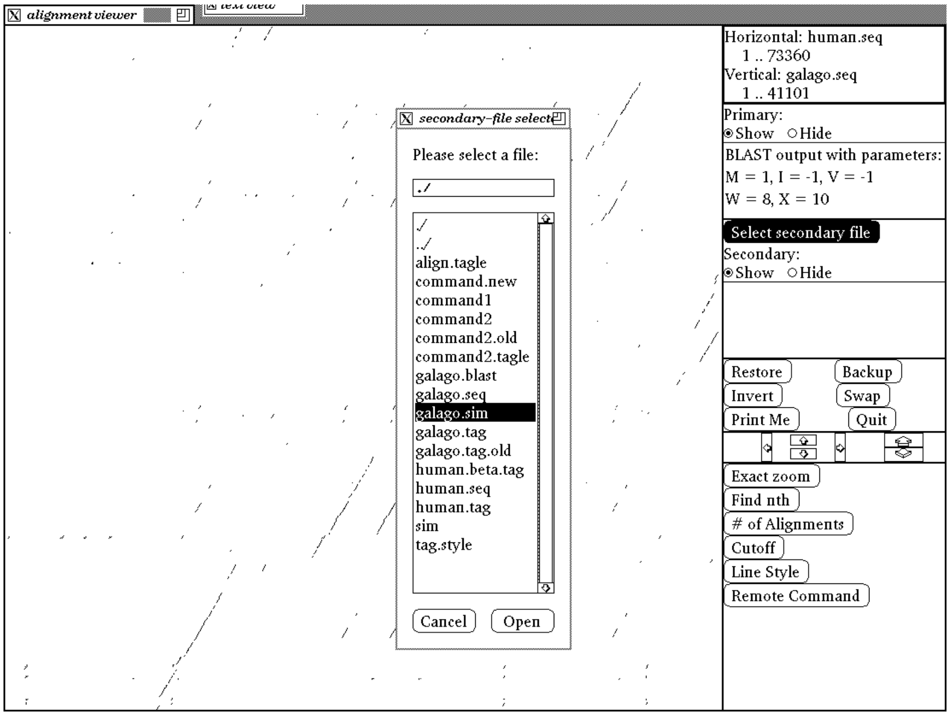
2 The DotView window

lav is highly interactive. You can start using it without worrying about the meaning of all the items on the control panel. Let's look at the large window containing the diagonal lines—the *dotplot view*—first. You probably want to drag the text view off to the side, or iconify it, for now. The dotplot view presents a representation of the alignment data analagous to a conventional dotplot, with the origin in the lower left corner.



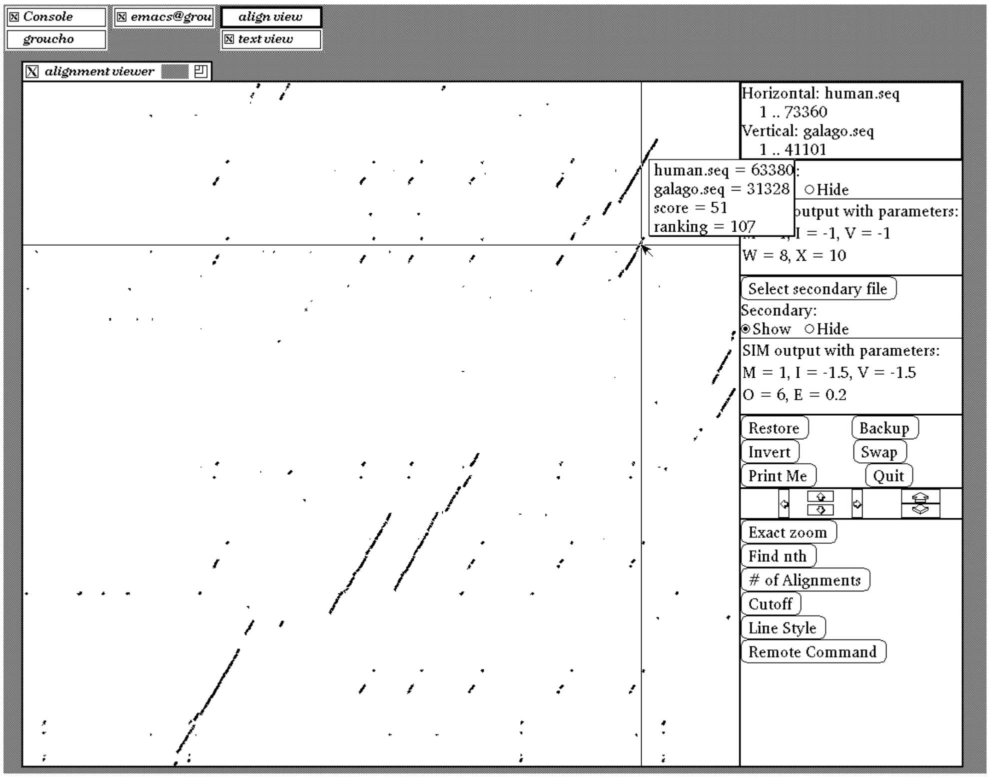
The dotplot view window

Along the right hand side of the window there are a number of smaller panes. The top one, with the labels *Horizontal* and *Vertical*, shows the names of the currently loaded primary sequences, and the portion currently visible in the dotplot view pane. The next two panes below that contain switches to *show* or *hide* the primary and secondary alignments, and a field that displays a small amount of explanatory text from the data files. When *lav* starts, it loads at least one file, named on the command line, which becomes the primary data. Actually, *lav* has an optional second parameter, which names the secondary data, but since you don't always know ahead of time that you want to view two sets of alignments, you can load a secondary data file using the *Select secondary file* button, which will pop up a directory browser for you.



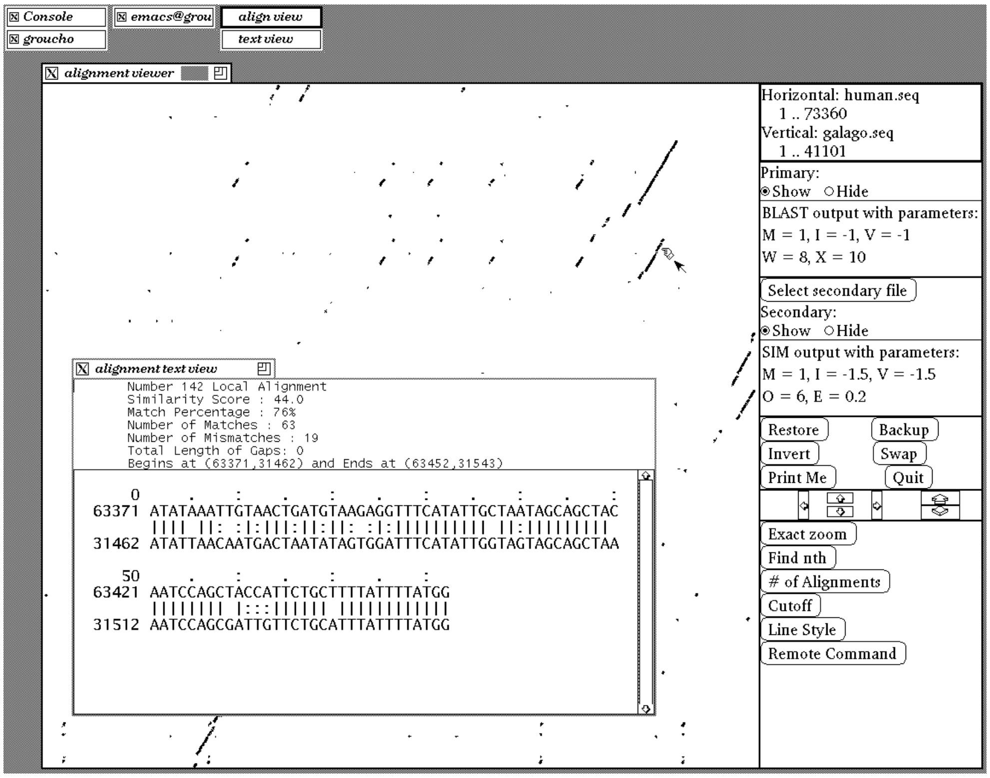
Selecting a secondary file

Now that there is some data loaded, we can use the mouse to examine it more closely. In the dotplot view, the mouse buttons are bound to three functions. The *left* is bound to *zoom*, the *middle* is bound to *select* and the *right* to *info*. When you point the cursor on an alignment and click the right mouse button, a small notifier will pop up and describe what you are pointing at.



Information on a point

When you click middle, the indicated alignment becomes current and its text appears in the text view window. The alignment text view window is initially very uninteresting; it simply says `Nothing selected` in its upper pane. But after you select an alignment in the dotplot view, by clicking on a line with the middle mouse button, the text view will fill itself in with the text of the alignment (in the lower pane) and a description of the alignment (in the upper pane). The portion of the alignment under the mouse in the dotplot view will have a hand-like pointer pointing at it, and will be be highlighted in the text view.

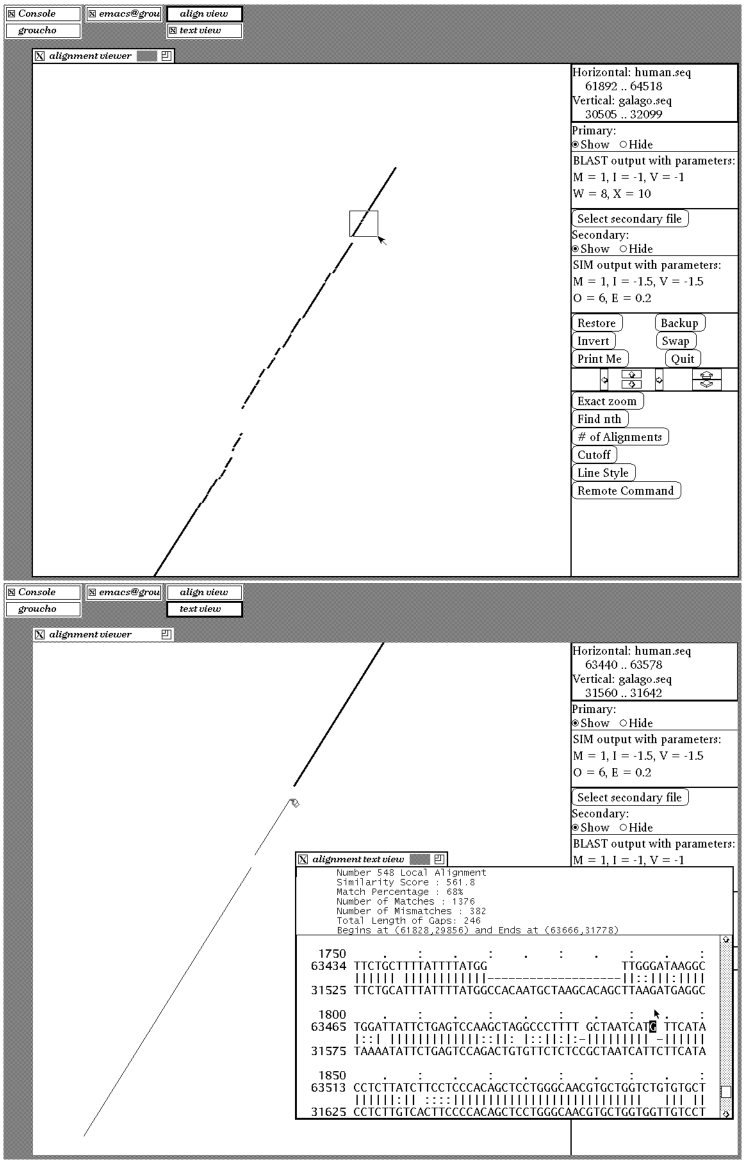


Selecting an alingment

The text view has different mouse bindings than the dotplot view. We digress for a moment to mention them. Here, the *left* mouse button is bound to the *select* function. Clicking on an element of the upper aligned sequence selects the nucleotide in both windows. The *middle* and *right* ones are bound to scrolling functions. Holding the middle button lets the mouse act as a "hand"—moving the mouse up and down makes the hand drag the page. The right one acts as a joystick—pressing it and moving the mouse up or down makes the page scroll rapidly down or up. Additionally, there is a scrollbar to the right, which behaves in the traditional manner—clicking in the up arrow scrolls up, clicking in the down arrow scrolls down.

Finally, you can zoom in on a region in the dotplot view. Sweeping out a rectangle in the dotplot view while holding down the left mouse button causes the delimited area to expand to fill the window when the button is released.

When you are done with the text view, just drag the window off to the side, or iconify it.



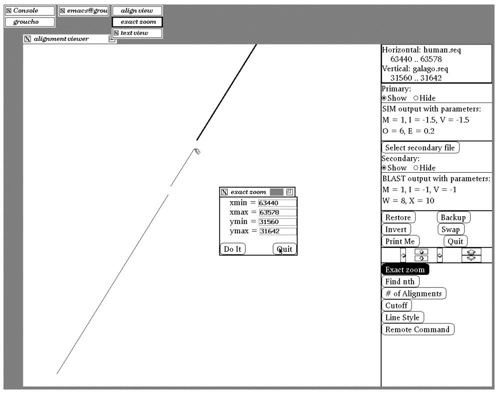
A sequence of zoom operations, then a selection.

Once you've zoomed in, you may want to pan around. On the control panel, there are four arrows pointing up, down, left, and right. These pan the view in the corresponding directions. To the right of those arrows are arrows pointing "in" and "out", which adjust the zoom accordingly.

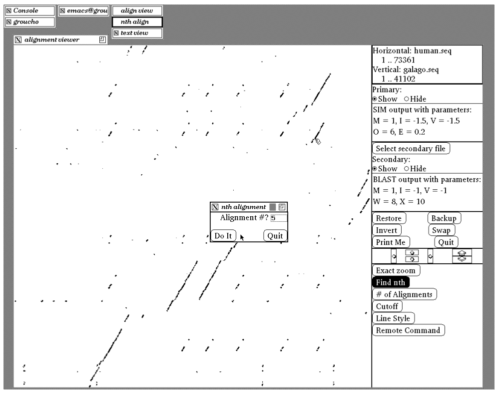
Sometimes, you want to undo a zoom or pan operation. In that case, the button labeled *Backup* backs up as many steps as you like. Also, the button labeled *Restore* clears the stack of pan and zoom operations, leaving you with the initial view. You can't backup past a restore. Around the *Backup* and *Restore* buttons there are a few others. The one labeled *Invert* changes the direction of the sequence displayed on the vertical axis so the diagonal lines in the dotplot view extend from upper left to lower right. This is sometimes useful if you want the dotplot display to look like ones that often appear in print. The button labeled *Swap* exchanges the primary and secondary data sets, if there is more than one data set loaded. The two data are normally rendered with different linestyles; this lets you easily swap them. Also, since only the secondary data can be loaded on the fly (using the *Select secondary file* button), swapping primary and secondary provides a way to replace the primary data if you want to. The button labeled *Print Me* sends a *PostScript* rendition of the dotplot view to your default printer. (To do this, it calls upon a helper program called `dotplot`, which needs to be in your command search path.) Finally, the one labeled *Quit* immediately exits from *lav*.

3 Other controls

The bottom of the control panel has a number of controls for more complex operations.

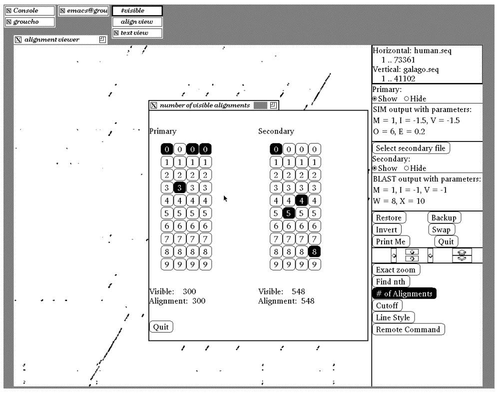


Exact Zoom

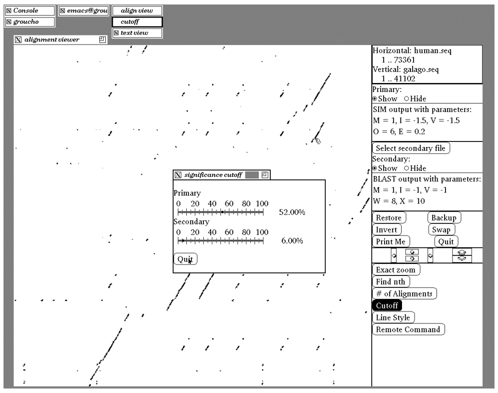


Nth Alignment

The *Exact zoom* operation pops up a dialog box with four fields. By entering the coordinates of interest, one can precicely zoom in on a region. The *Find nth* dialog sets the selection to the beginning of the nth alignment in the primary data set.

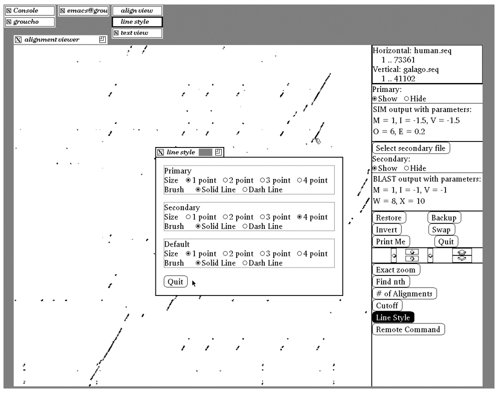


# of Alignments

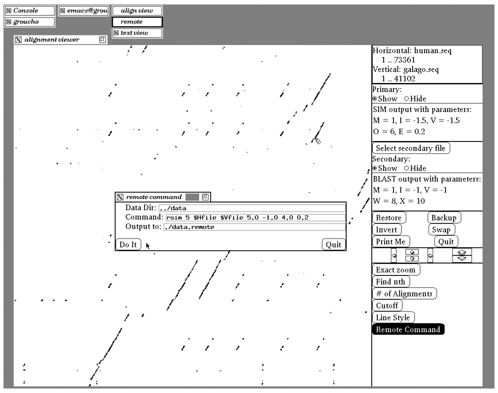


Cutoff

The *# of Alignments* button pops up a dialog box that displays the number of alignments in each data set, and allows one to limit the number displayed to the first n of them. The *Cutoff* dialog is used to set the threshold score below which low scoring alignments won't be displayed. The name of this button may be different sometimes, depending on the criteria defined in the data file. If no other name is defined, "Cutoff" is used.



Line Style



Remote Command

The *Line Style* dialog allows the user to set the width and style used to render lines in the dotplot view. Three things may be set: the default style, used when there is only one set of data visible, either because only one was loaded or because one of the two has been hidden; and the styles for primary and secondary when they are both onscreen. By default, the primary data is drawn in thin solid lines, and the secondary data is drawn in thick solid lines. Finally, the *Remote command* dialog allows the user to specify an external command which can process the currently loaded data. The environment contains several variables, such as `Hfile` and `Vfile` containing the names of the sequences currently loaded—currently the filenames contanining sequence data, but ultimately some more generic representation—and `H` and `V` containing the extent of the horizontal and vertical files, in the form *start end*. Given this, the user can refer to shell variables on the command line, as in

```
rsim $Hfile $H $Vfile $V
```

which may expand to something like

```
rsim mouse.seq 1 1234 human.seq 9 5678
```

The *Remote command* dialog also lets you specify a file to store the results in, and a default directory to run in.

4 Hardcopy

lav has several companion programs which can be used from the command line to get hardcopy output, in either dotplot or textual form.

The tool used to get the textual version is called *lat*. *lat* accepts the name of an alignment file on the command line, optionally followed by a number indicating which alignment to emit. If no number is given, *lat* will read numbers from standard input. A typical example would be

```
% lat foo.align 105 | lpr
```

or

```
% lat foo.align | lpr
1
2
3
^D
```

The output may look something like this:

```
Number 105 Local Alignment
Similarity Score : 19.0
Match Percentage : 79%
Number of Matches : 26
Number of Mismatches : 7
Total Length of Gaps: 0
Begins at (105318,3733) and Ends at (105350,3765)
```

```
      0     .    :    .    :    .    :
 105318  ATGGCCCAGCTGCGCCAGGGAAAAGAATAGAAG
         ||:||||||||||||||:|: ||| || |:|||
   3733  ATAGCCCAGCTGCGCCAAGATAAATAAAAAAAG
```

lat has four optional arguments which control the output format. Option `-s` requests that the output contain a header with the names and lengths of the sequences used. The `-d` option emits a header containing the description of the alignment data from data file. The `-g` option emits a header containing the “generation field” in the data file. This field contains a list of the programs which were used to generate the output. Finally the `-c` option followed by an integer selects the number of columns that will be used in the output. The default is 50 columns, not including the address down the left hand side. For example,

```
% lat -s -g -d -c 100 foo.align 5 | lpr
```

For graphical hardcopy there is *laps*, which generates a *PostScript* rendition of the alignments. The parameters to *laps* are several optional arguments, followed by an alignment file, followed by optional tag files. Ignoring the optional parameters, one can generate a graphical plot with

```
% laps foo.align tag1 tag2 tag3 | lpr
```

Tag files are used to label the top and left borders or the dotplot view, and are built using a library of *PostScript* routines. First, a line with either `horizontal` or `vertical` indicates where these tags will be drawn. Second, 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 110967 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
```

The optional arguments to *laps* control details about how the page is laid out. They are:

```
-ip       draw an inverted dot plot
-if       draw inverted features
-in       draw inverted numbers
-inv      draw everything inverted
-nc       don't clip to plot boundaries
```

Here, as in the interactive too, 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.

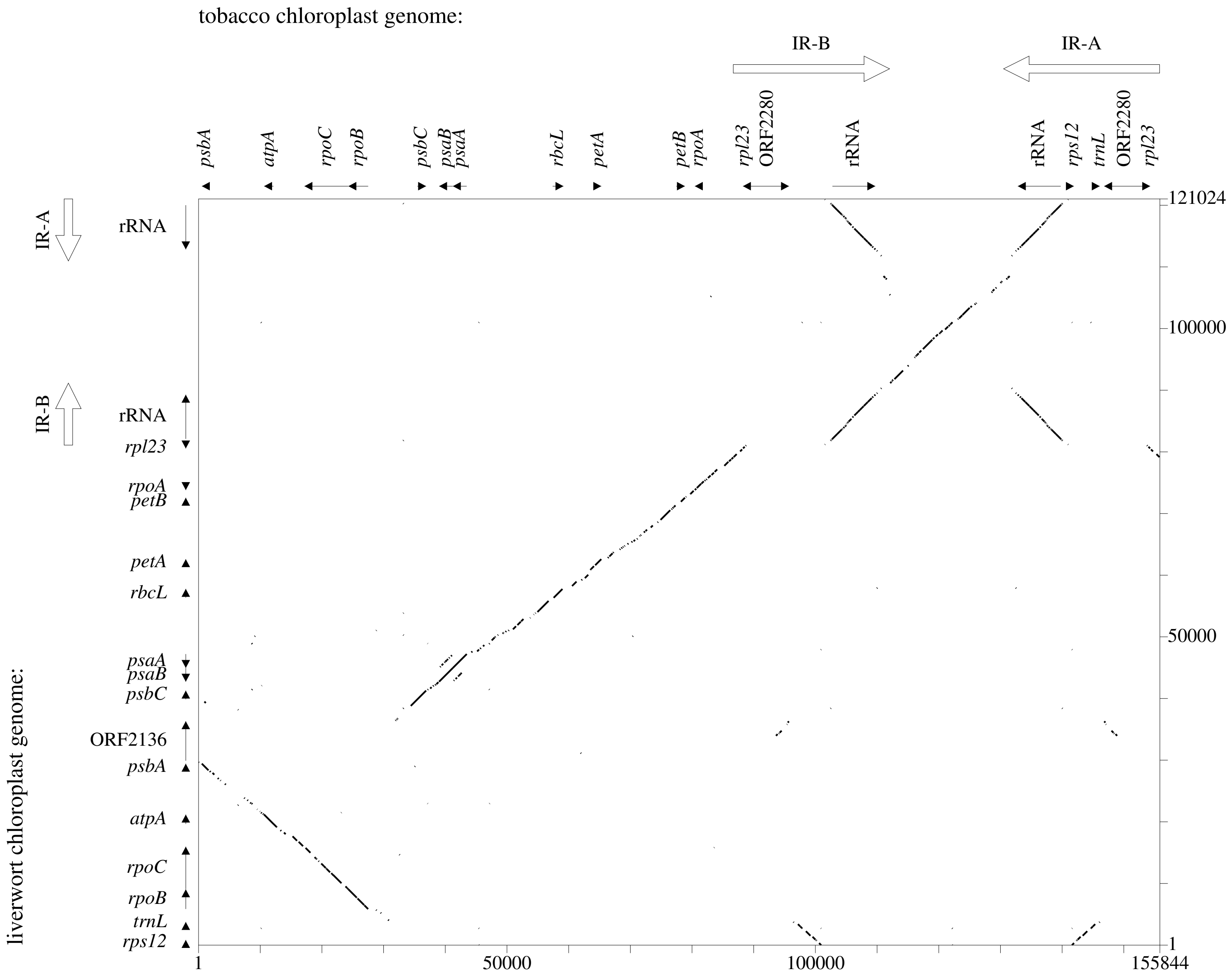
All of these options are available in *PostScript*, so you can embed them in tag files if necessary, under the names `/inverted-dotplot`, `/inverted-features`, `/inverted-numbers`, `/clip-features`.

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.

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

```
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. 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:


tobacco chloroplast genome:
IR-B
IR-A
psbA
atpA
rpoC
rpoB
psbC
psaB
psaA
rbcL
petA
petB
rpoA
rpl23
ORF2280
rRNA
rRNA
rps12
trnL
ORF2280
rpl23
121024
100000
50000
1
1
50000
100000
155844
liverwort chloroplast genome:
IR-A
rRNA
IR-B
rRNA
rpl23
rpoA
petB
petA
rbcL
psaA
psaB
psbC
ORF2136
psbA
atpA
rpoC
rpoB
trnL
rps12