# 1 Introduction

This directory contains source code for the Blastz alignment program, which is used by PipMaker and for the whole-genome human-mouse alignments available at the PipMaker web site. The other code used by PipMaker, e.g., for producing PDF files containing percent identity plots, is not currently in a form that can be installed easily at other sites. I intend that it will eventually be made public, but have no prediction for when that will happen. Blastz output can be browsed with the LAJ interactive alignment viewer (also available at http://bio.cse.psu.edu/) or the output can be parsed by programs that you write.

I make three requests to users of this source code. First, if the code contributes in some significant way to a manuscript that you publish, please acknowledge that fact in the paper and cite the PipMaker paper in Genome Research (April 2000). Second, please help me prepare useful documentation, as follows. The current documentation is brief by design; I am wary of writing too much and thereby discouraging users from reading any of it. If you have any questions that you can't answer in a few minutes, or if you have suggestions for either the code or this documentation, please write me at webb@bio.cse.psu.edu so I can learn what users really want. Third, if you using this code for comparative evaluation of another program, please check with me that you have Blastz configured in a reasonable way for the data you are using.

We use Blastz in two modes, PipMaker (where the typical job is to compare orthologous BAC-sized sequences) and for whole-genome alignments of human and mouse (where the typical job is to compare two 1-Mb sequences that aren't orthologous). Optimal configuration for Blastz differs between these two uses. See below.

# 2 Blastz Command-Line Options

Typing just "blastz" (or whatever you name it) should get a "usage" usage message like the following.

```
m(80M) bytes of space for trace-back information
c(0) 0: quiet; 1: print census
v(0) 0: quiet; 1: verbose progress reports to stderr
r(0) 0: old style report; 1: new style
B(2) 0: single strand; >0: both strands
C(0) 0: no chaining; 1: just output chain; 2: chain and extend;
        3: just output HSPs
E(30) gap-extension penalty
G(0) diagonal chaining penalty.
H(0) prechain: use hsp,chain,retry strategy
K(3000) threshold for MSPs
L(K) threshold for gapped alignments
M(50) mask threshold for seq1, if a bp is hit this many times
O(400) gap-open penalty
P(1) 0: entropy not used; 1: entropy used; >1 entropy with feedback
Q load the scoring matrix from a file
R(0) antidiagonal chaining penalty
T(0) 0: use W for word size;  1: use 12of19;  2: use 11of18
W(8) word size
Y(O+300E) X-drop parameter for gapped extension
```

They are used in commands such as

```
blastz sequence1 sequence2 B=0 C=2 K=2000 > blastz.out
```

which searches only one direction (strand), requires matches to be in the same order (and orientation, because B=0) in both sequences, and lowers the score threshold (relative to the substitution scores listed at the top of Blastz's output) from 3000 to 2000. These three command-line options account for over 95% of my normal usage.

# 3 Blastz Output

The first line, `#:lav`, is used to tell if you're looking at the right kind of file. Then comes a "`d {`" stanza that records the substitution scores and the main command-line option values, followed by a "`s {`" stanza giving the names of the sequence files and the numbers of basepairs, and a "`h {`" stanza giving the FastA headers. Other stanza may be present.

An actual alignment looks like this:

```
a {
  s 41249
  b 27347 5196
  e 27406 5259
  l 27347 5196 27365 5214 58
  l 27368 5215 27385 5232 72
  l 27386 5239 27406 5259 67
}
```

This gives the score, beginning positions in the two sequences, ending positions, then a list of the successive gap-free pieces, ending with percent identity.

# 4 Configuring Blastz

The version we use for whole-genome alignment is tuned for aligning a 1-Mb interval of the human genome with all of the mouse sequence on a CPU having about 500 Mb of RAM. In this case it is desirable to use the T=1 ("12of19") option for finding word-hits. This employs the non-consecutive seed suggested by Ming Li, allowing one transition-type substitution. We programmed this in a way that gains a little more speed with a lot more RAM, which was appropriate in this case because we were still well within the 500 Mb limit. Incidentally, the design where one CPU sees all of the mouse sequence was appropriate for effectively using dynamic masking (i.e., masking a human segment after it is found to match 50 places in the mouse genome; see the `M` parameter.)

An earlier version of Blastz used a linear-space algorithm for dynamic programming. We've gone back to a simpler system that can use more memory. If Blastz complains that it ran out of trace-back memory, you can adjust the `m` parameter. If it runs out of trace-back memory, then Blastz splits a long alignment into several shorter ones.

For best performance under some circumstances, you can consider use of the C-compiler flag `-DUSE_OBSTACK` (see the Makefile). It helps with performance in general, and is necessary to work around a bug in some versions of gnu malloc on large memory Linux systems (e.g. the version of redhat 6.2, with 1GB). Obstack.c is in glibc, but it also comes with many gnu programs (like gcc and emacs), so it's likely to already be on your computer if you are running something other than Linux.