PROGRAM NAME: # parseblast.pl

AUTHOR: **Josep F. Abril** ..... jabril@imim.es

LAST UPDATE: **September 3, 2001**

DESCRIPTION: parseblast.pl is a filter that extracts high-scoring segment pairs (HSPs) from the output of any BLAST program (WU-blast, NCBI-blast and NetBlast). It allows saving its output into GFF format.

Genome Informatics Research Lab
Grup de Recerca en Infomàtica Biomèdica
Institut Municipal d'Investigació Mèdica
Universitat Pompeu Fabra

Contents

List of Tables

List of Figures

< Id: parseblast.nw,v 0.1 2001/09/03 16:15:59 jabril Exp jabril >

1 Introduction

1.1 Description

1.2 Input

1.3 Output

1.4 To Do

- This is a first draft of the parseblast.pl. . . . . [*Section* 2.1, *page* 2]

2 Implementation

2.1 Program outline

2a ⟨*parseblast* 2a⟩≡
```
⟨PERL shebang 15a⟩
#
# MODULES
#
⟨Use Modules 2b⟩
#
# VARIABLES
#
⟨Global Vars 2c⟩
#
# MAIN LOOP
#
⟨Main Loop 2d⟩
#
# FUNCTIONS
#
⟨Functions 2e⟩
#
# TEMPORARY
#
⟨whole script 2f⟩
```

2b ⟨*Use Modules* 2b⟩≡

2c ⟨*Global Vars* 2c⟩≡

2d ⟨*Main Loop* 2d⟩≡
```
exit(0);
```

2e ⟨*Functions* 2e⟩≡
```
sub {
} #
```

TO DO

- This is a first draft of the parseblast.pl.

2f ⟨*whole script* 2f⟩≡
```
my $PROGRAM = "parseblast";
my $VERSION = "Version 1.0";
my $Start = time;

use strict;
use Getopt::Long;


##############################################################################
##                        Getting Comand-line Options                       ##
##############################################################################

Getopt::Long::Configure("bundling");

my ($hsp_flg, $gff_flg, $fullgff_flg, $aplot_flg, $nogff_flg, $subject_flg, $sequence_flg
    $full_scores, $comment_flg, $nocmmnt_flg, $split_flg, $help_flg, $err_flg,
```

```
    $expanded_flg, $pairwise_flg, $msf_flg, $aln_flg, $bit_flg, $ids_flg);
my ($prt, $main, $seqflg, $hsp, $fragment, $param, $aln_split, $prt_pos_flg
    ) = (0, 0, 0, 0, 0, 0, 0, 0);

GetOptions( "G|gff"            => \$gff_flg      ,
            "F|fullgff"        => \$fullgff_flg  ,
            "A|aplot"          => \$aplot_flg    ,
            "S|subject"        => \$subject_flg  ,
            "Q|sequence"       => \$sequence_flg ,
            "X|extended"       => \$expanded_flg ,
            "P|pairwise"       => \$pairwise_flg ,
            "M|msf"            => \$msf_flg      ,
            "N|aln"            => \$aln_flg      ,
            "W|show-coords"    => \$prt_pos_flg  ,
            "b|bit-score"      => \$bit_flg      ,
            "i|identity-score" => \$ids_flg      ,
            "s|full-scores"    => \$full_scores  ,
            "c|comments"       => \$comment_flg  ,
            "n|no-comments"    => \$nocmmnt_flg  ,
            "v|verbose"        => \$err_flg      ,
            "h|help|\?"        => \$help_flg      );

($help_flg) && &prt_help;

{   # first choose disables any other command-line option.
    $aln_flg
        && ($expanded_flg=$msf_flg=$pairwise_flg=$hsp_flg=$comment_flg=0,
            $nogff_flg=1, last);
    $msf_flg
        && ($expanded_flg=$aln_flg=$pairwise_flg=$hsp_flg=$comment_flg=0,
            $nogff_flg=1, last);
    $pairwise_flg
        && ($expanded_flg=$aln_flg=$msf_flg=$hsp_flg=$comment_flg=0,
            $nogff_flg=1, last);
    $gff_flg
        && ($expanded_flg=$aln_flg=$msf_flg=$pairwise_flg=$fullgff_flg=$aplot_flg=$hsp_f
            $nogff_flg=0, last);
    $fullgff_flg
        && ($expanded_flg=$aln_flg=$msf_flg=$pairwise_flg=$gff_flg=$aplot_flg=$hsp_flg=0,
            $nogff_flg=0, last);
    $aplot_flg
        && ($expanded_flg=$aln_flg=$msf_flg=$pairwise_flg=$gff_flg=$fullgff_flg=$hsp_flg=
            $nogff_flg=0, last);
    $expanded_flg
        && ($aln_flg=$msf_flg=$pairwise_flg=$hsp_flg=0,
            $nogff_flg=1, last);
    $hsp_flg = 1; $nogff_flg = 1;
};

{   # first choose disables any other command-line option.
    $bit_flg && ($ids_flg=0, last);
    $ids_flg && ($bit_flg=0, last);
};

$nocmmnt_flg && ($comment_flg = 0);


################################################################################
##                           Global Variables                                 ##
################################################################################
```

```
my ($prog_params, $program, $version, $seqname);
my ($scQ, $scS);   # reSCale_Query/Subject : 1 to re-scalate lengths.
my ($query_name, $db_name, $score, $descr,
    $ori, $end, $seq, $txt, $tt, $pt, $ht);
my (@seqlist, %prgseq, %dbase, %query, %cnt,
    %desc, %sco, %hsp_start, %hsp_end, %hsp_seq);
my ($qm, $sm, $x, $y, $ml, $a, $b, $aq, $as, $sql, $lq, $ls, $sq);
my $foe = 0;
my $index = 0;
my $chars_per_line = 50; # chars of sequences to show per line (in alignment modes)


##############################################################################
##                            Global Subroutines                            ##
##############################################################################

sub prt_progress {
    $err_flg && do {
        print STDERR ".";
        (($_[0] % 50) == 0) && print STDERR "[".&fill_left($_[0],6,"0")."]\n";
    };
}
sub prt_foeprg {
    $err_flg && ((($_[0] % 50) != 0) && print STDERR "[".&fill_left($_[0],6,"0")."]\n" );
}

sub fill_right { $_[0].($_[2] x ($_[1] - length($_[0]))) }
sub fill_left  { ($_[2] x ($_[1] - length($_[0]))).$_[0] }

sub max { my ($z) = shift @_; my $l; foreach $l (@_) { $z = $l if $l > $z ; }; $z; }

#
# Timing.

sub get_exec_time {
    $err_flg && do {
        my $End = $_[0];
        my ($c,$s,$m,$h,$r);
        $r = $End - $Start;
        $s = $r % 60;
        $r = ($r - $s) / 60;
        $m = $r % 60;
        $r = ($r - $m) / 60;
        $h = $r % 24;
        ($s,$m,$h) = (&fill_left($s,2,"0"),&fill_left($m,2,"0"),&fill_left($h,2,"0"));
print STDERR «EOF;
############################################################
## \"$PROGRAM\"  Execution Time:  $h:$m:$s
############################################################
EOF
    };
} # END_SUB: get_exec_time

#
# Print help to STDERR.

sub prt_help {
    open(HELP, "| more") ;
    print HELP «EndOfHelp;
```

```
PROGRAM:
        $PROGRAM
        $VERSION

USAGE:  parseblast.pl [options] <results.from.blast>

COMMAND-LINE OPTIONS:

    "$PROGRAM" prints output in "HSP" format by default (see below).
  It takes input from <STDIN> or single/multiple files, and writes
  its output to <STDOUT>, so user can redirect to a file but
  he also could use the program as a filter within a pipe.
    "-N", "-M", "-P", "-G", "-F", "-A" and "-X" options (also the long
  name versions for each one) are mutually exclusive, and their
  precedence order is shown above.

  GFF OPTIONS:

    -G, -gff            : prints output in GFF format.
    -F, -fullgff        : prints output in GFF "alignment" format.
    -A, -aplot          : prints output in APLOT "GFF" format.
    -S, -subject        : projecting GFF output by SUBJECT (default by QUERY).
    -Q, -sequence       : append query and subject sequences to GFF record.
    -b, -bit-score      : set <score> field to Bits (default Alignment Score).
    -i, -identity-score : set <score> field to Identities (default Alignment).
    -s, -full-scores    : include all scores for each HSP in each GFF record.

  ALIGNMENT OPTIONS:

    -P, -pairwise       : prints pairwise alignment for each HSP in TBL format.
    -M, -msf            : prints pairwise alignment for each HSP in MSF format.
    -N, -aln            : prints pairwise alignment for each HSP in ALN format.
    -W, -show-coords    : adds start/end positions to alignment output.

  GENERAL OPTIONS:

    -X, -expanded       : expanded output (producing multiline output records).
    -c, -comments       : include parameters from blast program as comments.
    -n, -no-comments    : do not print "#" lines (raw output without comments).
    -v, -verbose        : warnings sent to <STDERR>.
    -h, -help           : shows this help pages.

OUTPUT FORMATS:

    "S_" stands for "Subject_Sequence" and "Q_" for "Query_Sequence". <Program>
  name is taken from input blast file. <Strands> are calculated from <start> and
  <end> positions on original blast file. <Frame> is obtained from the blast
  file if is present else is set to ".". <SCORE> is set to Alignment Score by
  default, you can change it with "-b" and "-i".
    If "-S" or "-subject" options are given, then QUERY fields are referred to
  SUBJECT and SUBJECT fields are relative to QUERY (this only available for GFF
  output records).
    Dots ("...") mean that record description continues in the following line,
  but such record is printed as a single line by "$PROGRAM".

[HSP]  <- (This is the DEFAULT OUTPUT FORMAT)
 <Program> <DataBase> : ...
   ... <IdentityMatches> <Min_Length> <IdentityScore> ...
   ... <AlignmentScore> <BitScore> <E_Value> <P_Sum> : ...
   ... <Q_Name> <Q_Start> <Q_End> <Q_Strand> <Q_Frame> : ...
```

```
    ... <S_Name> <S_Start> <S_End> <S_Strand> <S_Frame> : <S_FullDescription>

[GFF]
 <Q_Name> <Program> hsp <Q_Start> <Q_End> <SCORE> <Q_Strand> <Q_Frame> <S_Name>

[FULL GFF]  <- (GFF showing alignment data)
 <Q_Name> <Program> hsp <Q_Start> <Q_End> <SCORE> <Q_Strand> <Q_Frame> ...
   ... Target "<S_Name>" <S_Start> <S_End> ...
   ... E_value <E_Value> Strand <S_Strand> Frame <S_Frame>

[APLOT]  <- (GFF format enhanced for APLOT program)
 <Q_Name>:<S_Name> <Program> hsp <Q_Start>:<S_Start> <Q_End>:<S_End> <SCORE> ...
   ... <Q_Strand>:<S_Strand> <Q_Frame>:<S_Frame> <BitScore>:<HSP_Number> ...
   ... \# E_value <E_Value>

[EXPANDED]
 MATCH(<HSP_Number>): <Q_Name> x <S_Name>
 SCORE(<HSP_Number>): <AlignmentScore>
 BITSC(<HSP_Number>): <BitScore>
 EXPEC(<HSP_Number>): <E_Value> Psum(<P_Sum>)
 IDENT(<HSP_Number>): <IdentityMatches>/<Min_Length> : <IdentityScore> \%
 T_GAP(<HSP_Number>): <TotalGaps(BothSeqs)>
 FRAME(<HSP_Number>): <Q_Frame>/<S_Frame>
 STRND(<HSP_Number>): <Q_Strand>/<S_Strand>
 MXLEN(<HSP_Number>): <Max_Length>
 QUERY(<HSP_Number>): length <Q_Length> : gaps <Q_TotalGaps> : ...
   ... <Q_Start> <Q_End> : <Q_Strand> : <Q_Frame> : <Q_FullSequence>
 SBJCT(<HSP_Number>): length <S_Length> : gaps <S_TotalGaps> : ...
   ... <S_Start> <S_End> : <S_Strand> : <S_Frame> : <S_FullSequence>

BUGS:    Report any problem to: abril\@imim.es

AUTHOR:  $PROGRAM is under GNU-GPL (C) 2000 - Josep F. Abril

EndOfHelp
    close(HELP);
exit(1);
# print STDOUT "HSP:$hsp_flg GFF:$gff_flg COM:$comment_flg SPL:$split_flg HLP:$help_flg\n
}

#
# Get new lines while empty line is not found and append to last line.

sub get_lines {
    my $spc = $_[0];
    my $tmp;
    # local($tmp);
    while (<>) {
        last if /^\s*$/;
        # print STDERR "$_";
        chop;
        s/^\s*/$spc/;
        $tmp .= $_;
    };
    $tmp;
}

#
# Getting scores from scoring vector extracted from HSP record.
```

```
sub get_scores {
    my $t = $_[0];
    my ($sc, $bt, $ex, $pv, $id);
    my ($qfr, $sfr) = ('.', '.');
    (($t =~ /Score[^\s]*\s+=\s+\b(\d+)\b\s+\([^,]*,/) || ($t =~ /Score\s+=\s+[^,]*\s+\((
        ($sc = $1);
    ($t =~ /Score[^\s]*\s+=.*[\s\(][+-]?(\d+\.?\d*|\.\d+))\b \bbits[^,]*,/o) &&
        ($bt = $1);
    ($t =~ /Expect[^\s]*\s+=\s+([+-]?([Ee][+-]?\d+|(\d+\.?\d*|\.\d+)([Ee][+-]?\d+)?))\s*,
        ($ex = $1);
    ($ex =~ /^[Ee]/o) && ($ex = "1".$ex);
    ($t =~ /Sum[^\s]*\s+\bP[^\s]*\s+=\s+([+-]?([Ee][+-]?\d+|(\d+\.?\d*|\.\d+)([Ee][+-]?\d
        ($pv = $1) : ($pv = "."); # $pv not defined then $pv="."
    ($t =~ /Identities[^\s]*\s+=\s+(\d+)\/(\d+)\s+/o) && ($id = $1);
    ( $scQ && !$scS) && do {   # BLASTX (translated nucleotides vs protein)
        ($t =~ /Frame[^\s]*\s+=\s+(\+|\-)(\d)\b/o) && ($qfr = $2);
    };
    (!$scQ &&  $scS) && do {   # TBLASTN (protein vs translated nucleotides)
        ($t =~ /Frame[^\s]*\s+=\s+(\+|\-)(\d)\b/o) && ($sfr = $2);
    };
    ( $scQ &&  $scS) && do {   # TBLASTX (translated nucleotides vs translated nucleotide
        ($t =~ /Frame[^\s]*\s+=\s+(\+|\-)(\d)\s*\/\s*(\+|\-)(\d)\b/o) && ($qfr = $2, $sfr
    };
    $sc, $bt, $ex, $pv, $id, $qfr, $sfr;
}

sub chk_strand {
    my ($first, $last) = @_ ;
    my $st = "+";
    ($first >= $last) && (($first, $last) = ($last, $first), $st = "-");
    $first, $last, $st;
}

#
# Formatting output as plain, HSPs, GFF, APLOT, ALN.
my ($n, $nm, $wnm, $tq, $ts,       # couNter, NaMe, TagQuery, TagSubject
    $sc, $bt, $ex, $pv, $id,       # SCore, BiTscore, EvalueX, IDentityscore
    $scores,                       # showing all scores
    $frq, $frs,                    # QueryFRame, SubjectFRame
    $stq, $sts,                    # QuerySTrand, SubjectSTrand
    $gsc, $prg,                    # GroupSCore, PRoGram
    $hsq, $hss, $heq, $hes,        # HspStartQuery, HspStartSubject, HspEndQuery, HspEndS
    $lnq, $lns, $lnmin, $lnmax,    # LeNgthQuery, LeNgthSubject, LeNgthMINqueyxsubject, L
    $lnmx, $gpq, $gps, $gpt        # LeNgthMaXhspseq, GaPQuery, GaPSubject, GaPTotal
    );

sub prt_hsp {
print STDOUT «"EndOfHSPs";
$prg $dbase{$nm} : $id $lnmin $gsc $sc $bt $ex $pv : $query{$nm} $hsq $heq $stq $frq : $w
EndOfHSPs
last PRINT;
}
sub prt_ext {
print STDOUT «"EndOfPlain";
MATCH($n): $query{$nm} x $wnm
SCORE($n): $sc\nBITSC($n): $bt\nEXPEC($n): $ex Psum($pv)
IDENT($n): $id/$lnmin : $gsc \%
T_GAP($n): $gpt\nFRAME($n): $frq/$frs\nSTRND($n): $stq/$sts\nMXLEN($n): $lnmx
QUERY($n): length $lnq : gaps $gpq : $hsq $heq : $stq : $frq : $hsp_seq{$tq}
SBJCT($n): length $lns : gaps $gps : $hss $hes : $sts : $frs : $hsp_seq{$ts}
```

```
\#\#
EndOfPlain
last PRINT;
}
sub prt_Q_gff {
print STDOUT «"EndOfGFF";
$query{$nm}\t$prg\thsp\t$hsq\t$heq\t$gsc\t$stq\t$frq\t$wnm\t\# E_value $ex$scores$sq
EndOfGFF
last PRINT;
}
sub prt_S_gff {
print STDOUT «"EndOfGFF";
$wnm\t$prg\thsp\t$hss\t$hes\t$gsc\t$sts\t$frs\t$query{$nm}\t\# E_value $ex$scores$sq
EndOfGFF
last PRINT;
}
sub prt_Q_fullgff {
print STDOUT «"EndOfFullGFF";
$query{$nm}\t$prg\thsp\t$hsq\t$heq\t$gsc\t$stq\t$frq\tTarget \"$wnm\"\t$hss\t$hes ;\tStra
EndOfFullGFF
last PRINT;
}
sub prt_S_fullgff {
print STDOUT «"EndOfFullGFF";
$wnm\t$prg\thsp\t$hss\t$hes\t$gsc\t$sts\t$frs\tTarget \"$query{$nm}\"\t$hsq\t$heq ;\tStra
EndOfFullGFF
last PRINT;
}
sub prt_Q_aplot {
print STDOUT «"EndOfAPLOT";
$query{$nm}:$wnm\t$prg\thsp\t$hsq:$hss\t$heq:$hes\t$gsc\t$stq:$sts\t$frq:$frs\t$bt:$n\t\#
EndOfAPLOT
last PRINT;
}
sub prt_S_aplot {
print STDOUT «"EndOfAPLOT";
$wnm:$query{$nm}\t$prg\thsp\t$hss:$hsq\t$hes:$heq\t$gsc\t$sts:$stq\t$frs:$frq\t$bt:$n\t\#
EndOfAPLOT
last PRINT;
}
sub prt_pairwise {
    $prt_pos_flg && do {
        $ml = &max(length($hsq),length($heq),length($hss),length($hes));
        ($x,$y) = (" ".&fill_left($hsq,$ml," ")." ".&fill_left($heq,$ml," ")." $stq $frq
                   " ".&fill_left($hss,$ml," ")." ".&fill_left($hes,$ml," ")." $sts $frs
    };
    print "#\n" if $aln_split;
    $aln_split = 1;
print STDOUT «"EndOfALIGN";
$a$x$hsp_seq{$tq}
$b$y$hsp_seq{$ts}
EndOfALIGN
last PRINT;
}
sub prt_cmn_aln {
print STDOUT «"EndOfALN" if !$nocmmnt_flg;
################################################################################
##
##  $query{$nm} x $wnm #$n
##
```

```
##    $prg $dbase{$nm}
##    Identity: $gsc  Score: $sc  Bits: $bt  E_value: $ex  P_value: $pv
##    DESCR: $desc{$nm}
##
EndOfALN
}
sub prt_msf {
    my ($i,$jq,$js,$vs,$ve,$chw);
    ($aq,$as,$sql) = ($hsp_seq{$tq},$hsp_seq{$ts},length($hsp_seq{$tq}));
    $aq =~ s/\.//og;
    $as =~ s/\.//og;
    ($lq,$ls) = (length($aq),length($as));
    &prt_cmn_aln;
print STDOUT «"EndOfMSF";
\n
$query{$nm}\_x_$wnm\_\#$n.msf  MSF: $sql  Type: P  May 4th, 2000  Check: 0  ..\n
Name: $a  Len: $lq  Check: 0  Weight: 1.00
Name: $b  Len: $ls  Check: 0  Weight: 1.00
\n//
EndOfMSF
    for ($i=0; $i<=($sql-1); $i+=$chars_per_line) {
        ($jq, $js) = (substr($hsp_seq{$tq},$i,50), substr($hsp_seq{$ts},$i,50));
        print STDOUT "\n";
        $prt_pos_flg && do {
            $chw = length($jq);
            ($vs, $ve) = ($i+1, $i+$chw);
            $chw = $chw-length($vs.$ve);
            print STDOUT " "x($ml+1).$vs." "x($chw>1 ? $chw : 1).$ve."\n";
        };
print STDOUT «"EndOfALIGN";
$a $jq
$b $js
EndOfALIGN
    };
    print STDOUT "\n\n";
    last PRINT;
}
sub prt_aln {
    my ($i,$jq,$js,$vs,$ve,$chw);
    ($aq,$as,$sql) = ($hsp_seq{$tq},$hsp_seq{$ts},length($hsp_seq{$tq}));
    $aq =~ s/\./-/og;
    $as =~ s/\./-/og;
    &prt_cmn_aln;
    for ($i=0; $i<=($sql-1); $i+=$chars_per_line) {
        ($jq, $js) = (substr($aq,$i,50), substr($as,$i,50));
        print STDOUT "\n";
        $prt_pos_flg && do {
            $chw = length($jq);
            ($vs, $ve) = ($i+1, $i+$chw);
            $chw = $chw-length($vs.$ve);
            print STDOUT " "x($ml+1).$vs." "x($chw>1 ? $chw : 1).$ve."\n";
        };
print STDOUT «"EndOfALIGN";
$a $jq
$b $js
EndOfALIGN
    };
    print STDOUT "\n\n";
    last PRINT;
}
```

```
sub prt_out {
    $prt = 0;
    $err_flg && print STDERR ("#"x58)."\n## WRITING OUTPUT TO STDOUT ".("#"x30)."\n".("#"
    while (@seqlist) {
        $nm = shift(@seqlist);
        ($wnm = $nm) =~ s/\_\d+$//o;
        (!$hsp_flg && $comment_flg) && (print STDOUT "#\n# $prgseq{$nm} :: DB $dbase{$nm}
SCR: $desc{$nm}\n#\n");
        ($cnt{$nm}>0) && do {
            for ($n = 1; $n <= $cnt{$nm}; $n++) {
                &prt_progress(++$ht);
                $tq = $nm."query".$n;
                $ts = $nm."sbjct".$n;
                ($sc, $bt, $ex, $pv, $id, $frq, $frs) = &get_scores($sco{$nm.$n});
                ($hsq, $heq, $stq) = &chk_strand($hsp_start{$tq}, $hsp_end{$tq});
                ($hss, $hes, $sts) = &chk_strand($hsp_start{$ts}, $hsp_end{$ts});
                $lnq = $heq - $hsq + 1 ;
                $scQ && ($lnq = $lnq / 3) ;
                $lns = $hes - $hss + 1 ;
                $scS && ($lns = $lns / 3) ;
                $lnmin = ($lnq>$lns) ? $lns : $lnq;
                $lnmax = ($lnq<$lns) ? $lns : $lnq;
                $lnmx = length($hsp_seq{$tq});
                {
                    my $hh = $hsp_seq{$tq};
                    $gpq = ($hh =~ s/-/ /og) || 0;
                    $hh = $hsp_seq{$ts};
                    $gps = ($hh =~ s/-/ /og) || 0;
                };
                $gpt = $gpq + $gps;
                {
                    ($ids_flg || $expanded_flg || $hsp_flg) && # score is Identities di-
vided by minlength
                        (($gsc) = eval(($id/$lnmin)*100) =~ /^(\d+(\.\d{0,3})?)/o, last);
                    $bit_flg && ($gsc = $bt, last);
                    $gsc = $sc;
                };
                ($prg) = $prgseq{$nm} =~ /^([^\s]+)\s/o;
                # GFF format
              PRINT: {
                    &prt_hsp       if $hsp_flg; # default output
                    #
                    do {
                        $sq = $scores = "";
                        $scores = " ;\tP_sum $pv ;\tAln_Score $sc ;\tBit_Score $bt ;\tIdn_S
                            sprintf("%6.2f",($id/$lnmin) * 100)." ($id/$lnmin) ;\tGaps ".
                            sprintf("%4.2f",($gpt/$lnmx) * 100).
                            " (Q:$gpq|S:$gps) ;\tLengths Q:$lnq|S:$lns|T:$lnmx"
                            if $full_scores;
                        $subject_flg && do {
                            $sq = " #-S: $hsp_seq{$ts} #-Q: $hsp_seq{$tq}" if $sequence_flg
                            &prt_S_gff     if $gff_flg;
                            &prt_S_fullgff if $fullgff_flg;
                            &prt_S_aplot   if $aplot_flg;
                        }; # $subject_flg
                        $sq = " #-Q: $hsp_seq{$tq} #-S: $hsp_seq{$ts}" if $sequence_flg;
                        &prt_Q_gff     if $gff_flg;
                        &prt_Q_fullgff if $fullgff_flg;
                        &prt_Q_aplot   if $aplot_flg;
```

```
                } unless $nogff_flg;
                #
                &prt_ext       if $expanded_flg;
                #
                # ($qm, $sm, $x, $y) = ("$query{$nm}\_\#$n", "$nm\_\#$n", " ", " ");
                ($qm, $sm, $x, $y) = ("$query{$nm}", "$wnm", " ", " ");
                $ml = &max(length($qm),length($sm));
                ($a,$b) = (&fill_right($qm,$ml," "),&fill_right($sm,$ml," "));
                &prt_pairwise  if $pairwise_flg;
                &prt_msf       if $msf_flg;
                &prt_aln       if $aln_flg;
            } # PRINT
          } # for $cnt{$nm}
       } # do if $cnt{$nm}>0
    } # foreach
    &prt_foeprg($ht);
}


##################################################
## Main Loop
##################################################

$err_flg && print STDERR ("#"x58)."\n## PARSING STDIN FROM BLAST ".("#"x30)."\n".("#"x58)
while (<>) {
    # s/\r\n$/\n/; # if your input records finish with "\r\n" (like EMBL).
    next if /^\s*$/;  # /^\s*$/ is similar to AWK /^[ \t]*$/
    &prt_progress(++$pt);
    my $tmpinput = $_;
    chop;
    # print STDOUT "$. : $_ \n"; # "$." is record number && "$_" is whole record
  CHECK: {
      /^\s*T?BLAST[PNX]?/o        && do { # Starts with "T?BLAST[PNX]?" ?
          # print STDERR "$_\n";
          $prt && &prt_out;
          ($program, $version) = split;
          # typeQ/typeS: 0 for proteins - 1 for nucleic acids.
          ($program =~ /^BLASTP$/o ) && ( $scQ = 0, $scS = 0); # Amino Acids vs Amino Aci
          ($program =~ /^BLASTN$/o ) && ( $scQ = 0, $scS = 0); # Nucleotides vs Nucleotid
          ($program =~ /^BLASTX$/o ) && ( $scQ = 1, $scS = 0); # Nucleotides vs Amino Aci
          ($program =~ /^TBLASTN$/o) && ( $scQ = 0, $scS = 1); # Amino Acids vs Nucleotid
          ($program =~ /^TBLASTX$/o) && ( $scQ = 1, $scS = 1); # Nucleotides translated v
cleotides translated
          $prog_params = "#\n# $program $version\n#";
          $query_name = $db_name = ”;
          $main = 1;
          $seqflg = $hsp = $fragment = $param = 0;
          last CHECK;
      }; # /^\s*T?BLAST[PNX]?/
      /^>/o                    && do { # Starts with ">" ?: sequences.
          # print STDERR "$_\n";
          ($seqname, $descr) = split(/\s+/, $_, 2);
          $seqname =~ s/^\s*>//o;
          $seqname =~ s/\s|:|\|/_/og;
          $seqname .= "_".($index++);
          $prgseq{$seqname} = "$program ($version)";
          $query{$seqname} = $query_name;
          ($db_name =~ /([^\/]+)$/o) && ($dbase{$seqname} = $1);
          push(@seqlist,$seqname);
          $desc{$seqname} = join(’ ’, $descr, &get_lines(’ ’));
```

```
#         $cnt{$seqname} = 0;
          $seqflg = 1;
          $main = $hsp = $fragment = 0;
          last CHECK;
      }; # /^\s*>/
      (/^\s*Score/o && $seqflg) && do { # Starts with "Score" ?: HSPs.
          # print STDERR "$_\n";
          ( $score = $_ ) =~ s/^\s*//o;
          $cnt{$seqname}++;
          $sco{$seqname.$cnt{$seqname}} = join(", $score, &get_lines(', '));
          # print STDOUT $sco{$seqname.$cnt{$seqname}}."\n";
          $hsp = 1;
          $fragment = 0;
          last CHECK;
      }; # (/^\s*Score/ && ($seqflg))
      (/^Query:/o && $hsp)    && do { # Starts with "Query" ?: Fragments.
          # print STDERR "$_\n";
          $fragment = 1;
          last CHECK;
      }; # (/^\s*Query/ && ($hsp))
      (/^\s*(?:Database|Parameters):/o && !$main) && do { # Parameters Section.
          # print STDERR "$_\n";
          $prt = $param = 1;
          $seqflg = $hsp = $fragment = 0;
          last CHECK;
      }; # (/^\s*(?:Database|Parameters)/ && ($hsp))
  } # CHECK Block
    # print STDOUT "$. : MAIN=$main SEQFLG=$seqflg HSP=$hsp FRAGMENT=$fragment => SEQNAME
  LOAD: {
      $fragment && do { # We are within a fragment.
          $txt = ";
          $tt = $cnt{$seqname};
          ($txt,$ori,$seq,$end) = split;
          if ($txt =~ /^Query:/o) {
              # print STDERR "$_\n";
              ($hsp_start{$seqname."query".$tt}) || ($hsp_start{$seqname."query".$tt} = $
              $hsp_end{$seqname."query".$tt} = $end;
              $hsp_seq{$seqname."query".$tt} .= $seq;
          } # if ($txt =~ /Query/)
          elsif ($txt =~ /^Sbjct:/o) {
              # print STDERR "$_\n";
              ($hsp_start{$seqname."sbjct".$tt}) || ($hsp_start{$seqname."sbjct".$tt} = $
              $hsp_end{$seqname."sbjct".$tt} = $end;
              $hsp_seq{$seqname."sbjct".$tt} .= $seq;
              $fragment = 0;
          } # elsif ($txt =~ /Sbjct/)
          else { last LOAD; };
      }; # ($fragment)
      $main && do { # We are within the blast file header.
          /^\s*Query= +(.*)\s*$/o && do {
              # print STDERR "$_\n";
              ($query_name = $1) =~ s/\s|:|\|/_/g ;
          };
          /^\s*Database: +(.*)\s*$/o && do {
              # print STDERR "$_\n";
              $db_name = $1;
              while (<>) {
                  last if /^(?:.*\bletter.*|\s*)$/o;
                  # print STDERR "$_\n";
                  chop;
```

```
                    s/^\s*//o;
                    s/\s*$//o;
                    $db_name .= $_;
                }; # while getline
            }; # /^\s*Database: +(.*)\s*$/
            last LOAD;
        }; # ($main)
        $param && do { # We are within the blast file trailer.
            /^\s*Query= +(.*)\s*$/o && do {
                # print STDERR "$_\n";
                &prt_foeprg($pt);
                $prt && &prt_out;
                $err_flg && print STDERR ("#"x58)."\n## PARSING STDIN FROM BLAST (New BLAST
sults) ".("#"x9)."\n".("#"x58)."\n";
                @seqlist = ();
                ($query_name = $1) =~ s/\s|:|\|/_/g ;
                $main = 1;
                $seqflg = $hsp = $fragment = $param = 0;
                last LOAD;
            };
            if (/^\s*[^\[\<\-]/o) {
                # print STDERR "$_\n";
                chop;
                s/^/\n\# /o;
                $prog_params = join(", $prog_params, $_);
            } else { $param = 0; }
            last LOAD;
        }; # ($param)
    } # LOAD Block
    close(ARGV) if (eof);
} # while
&prt_foeprg($pt);

# $prt && &prt_out;
&prt_out;

&get_exec_time(time);

exit(0);

#####################################################
## TESTING how to compile into a binary file.... ##
#
# C libraries at: /usr/lib/perl5/5.00503/i386-linux/CORE/
#
#   perl -MO=C ../parseblast.pl > parseblast.c
#   gcc parseblast.c -E -I /usr/lib/perl5/5.00503/i386-linux/CORE/ -o ./parseblast.i
#   gcc parseblast.i -o parseblast
#
## STILL NOT WORKING...
#####################################################
#           "a|align-score "   => \$aln_flg     ,
#           "s|split-output"   => \$split_flg   ,
#   -a, -align-score    : set <score> field to Alignment Score.
#   -s, -split-output : output each sequence match in a separate file in the current dire
```

A empty appendix section

A.1 empty appendix subsection

B Common code blocks

B.1 PERL scripts

15a ⟨*PERL shebang* 15a⟩≡
```
#!/usr/bin/perl -w
# This is perl, version 5.005_03 built for i386-linux
⟨GNU License 17d⟩
⟨Version Control Id Tag 17c⟩
#
use strict;
```

15b ⟨*Global Constants - Boolean* 15b⟩≡
```
my ($T,$F) = (1,0); # for 'T'rue and 'F'alse
```

We also set here the date when the script is running and who is the user running it.

15c ⟨*Global Vars - User and Date* 15c⟩≡
```
my $DATE = localtime;
my $USER = $ENV{USER};
```

B.1.1 Timing our scripts

The '`Benchmark`' module encapsulates a number of routines to help to figure out how long it takes to execute a piece of code and the whole script.

15d ⟨*Use Modules - Benchmark* 15d⟩≡
```
use Benchmark;
  ⟨Timer ON 15e⟩
```

See '`man Benchmark`' for further info about this package. We set an array to keep record of timing for each section.

15e ⟨*Timer ON* 15e⟩≡
```
my @Timer = (new Benchmark);
```

15f ⟨*Common PERL subs - Benchmark* 15f⟩≡
```
sub timing() {
    push @Timer, (new Benchmark);
    # partial time
    $_[0] ||
        (return timestr(timediff($Timer[$#Timer],$Timer[($#Timer - 1)])));
    # total time
    return timestr(timediff($Timer[$#Timer],$Timer[0]));
} # timing
```

B.1.2 Printing complex Data Structures

With '`Data::Dumper`' we are able to pretty print complex data structures for debugging them.

15g ⟨*Use Modules - Dumper* 15g⟩≡
```
use Data::Dumper;
local $Data::Dumper::Purity = 0;
local $Data::Dumper::Deepcopy = 1;
```

B.1.3 Common functions

15h ⟨*Skip comments and empty records* 15h⟩≡
```
next if /^\#/o;
next if /^\s*$/o;
chomp;
```

16a ⟨*Common PERL subs - Min Max* 16a⟩≡
```
#
sub max() {
    my $z = shift @_;
    foreach my $l (@_) { $z = $l if $l > $z };
    return $z;
} # max
sub min() {
    my $z = shift @_;
    foreach my $l (@_) { $z = $l if $l < $z };
    return $z;
} # min
```

16b ⟨*Common PERL subs - Text fill* 16b⟩≡
```
#
sub fill_right() { $_[0].($_[2] x ($_[1] - length($_[0]))) }
sub fill_left()  { ($_[2] x ($_[1] - length($_[0]))).$_[0] }
sub fill_mid()   {
    my $l = length($_[0]);
    my $k = int(($_[1] - $l)/2);
    ($_[2] x $k).$_[0].($_[2] x ($_[1] - ($l+$k)));
} # fill_mid
```

These functions are used to report to STDERR a single char for each record processed (useful for reporting parsed records).

16c ⟨*Common PERL subs - Counter* 16c⟩≡
```
#
sub counter { # $_[0]~current_pos++ $_[1]~char
    print STDERR "$_[1]";
    (($_[0] % 50) == 0) && (print STDERR "[".&fill_left($_[0],6,"0")."]\n");
} # counter
#
sub counter_end { # $_[0]~current_pos   $_[1]~char
    (($_[0] % 50) != 0) && (print STDERR "[".&fill_left($_[0],6,"0")."]\n");
} # counter_end
```

16d ⟨*Global Vars - Counter* 16d⟩≡
```
my ($n,$c); # counter and char (for &counter function)
```

B.1.4 Common functions for reporting program processes

Function '`report`' requires that a hash variable '`%MessageList`' has been set, such hash contains the strings for each report message we will need. The first parameter for '`report`' is a key for that hash, in order to retrieve the message string, the other parameters passed are processed by the `sprintf` function on that string.

16e ⟨*Common PERL subs - STDERR* 16e⟩≡
```
sub report() { print STDERR sprintf($MessageList{ shift @_ },@_) }
```

The same happens to '`warn`' function which also requires a hash variable '`%ErrorList`' containing the error messages.

16f ⟨*Common PERL subs - STDERR* 16e⟩+≡
```
sub warn() { print STDERR sprintf($ErrorList{ shift @_ }, @_) }
```

B.2 BASH scripts

17a ⟨*BASH shebang* 17a⟩≡
```
#!/usr/bin/bash
# GNU bash, version 2.03.6(1)-release (i386-redhat-linux-gnu)
⟨Version Control Id Tag 17c⟩
#
SECONDS=0 # Reset Timing
# Which script are we running...
L="####################"
{ echo "$L$L$L$L";
  echo "### RUNNING [$0]";
  echo "### Current date:‘date‘";
  echo "###"; } 1>&2;
```

17b ⟨*BASH script end* 17b⟩≡
```
{ echo "###"; echo "### Execution time for [$0] : $SECONDS secs";
  echo "$L$L$L$L";
  echo ""; } 1>&2;
#
exit 0
```

B.3 Version control tags

This document is under Revision Control System (RCS). The version you are currently reading is the following:

17c ⟨*Version Control Id Tag* 17c⟩≡
```
# $Id: parseblast.nw,v 0.1 2001/09/03 16:15:59 jabril Exp jabril $
```

B.4 GNU General Public License

17d ⟨*GNU License* 17d⟩≡
```
# #-----------------------------------------#
# #                          parseblast                                 #
# #-----------------------------------------#
#
#                Extracting HSPs from blast output.
#
#     Copyright (C) 2001 - Josep Francesc ABRIL FERRANDO
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
#
# #-----------------------------------------#
```

C Extracting code blocks from this document

From this file we can obtain both the code and the documentation. The following instructions are needed:

C.1 Extracts Script code chunks from the NOWEB file

Remember when tangling that '-L' option allows you to include program line-numbering relative to original NOWEB file. Then the first line of the executable files is a comment, not a shebang, and must be removed to make scripts runnable.

18a ⟨*tangling* 18a⟩≡
```
# showing line numbering comments in program
notangle -L -R"parseblast" $WORK/$nwfile.nw | \
    perl -ne '$.>1 && print' | cpif $BIN/parseblast.pl ;
# making them runnable
chmod a+x $BIN/parseblast.pl ;
```

18b ⟨*tangling* 18a⟩+≡
```
# program without line numbering comments
notangle -t4 -R"parseblast" $WORK/$nwfile.nw | \
    cpif $BIN/parseblast.pl ;
# reformating program with perltidy
notangle -R"parseblast" $WORK/$nwfile.nw | \
    perltidy - | cpif $BIN/parseblast.pl ;
# pretty-printing program with perltidy
notangle -R"parseblast" $WORK/$nwfile.nw | \
    perltidy -html - | cpif $DOCS/html/parseblast.html ;
#
```

C.2 Extracting different Config Files

18c ⟨*tangling* 18a⟩+≡
```
notangle -R"root" $WORK/$nwfile.nw | \
    cpif $DATA/root_config ;
```

C.3 Extracting documentation and LaTeX'ing it

18d ⟨*tangling* 18a⟩+≡
```
notangle -Rweaving  $WORK/$nwfile.nw | cpif $WORK/nw2tex ;
notangle -RLaTeXing $WORK/$nwfile.nw | cpif $WORK/ltx ;
chmod a+x $WORK/nw2tex $WORK/ltx;
```

18e ⟨*tangling complementary LaTeX files* 18e⟩≡
```
notangle -R"HIDE: LaTeX new definitions" $WORK/$nwfile.nw | cpif $DOCS/defs.tex ;
notangle -R"HIDE: TODO" $WORK/$nwfile.nw | cpif $DOCS/todo.tex ;
```

18f ⟨*weaving* 18f⟩≡
```
⟨BASH shebang 17a⟩
# weaving and LaTeXing
⟨BASH Environment Variables 19b⟩
⟨tangling complementary LaTeX files 18e⟩
noweave -v -t4 -delay -x -filter 'elide "HIDE: *"' \
        $WORK/$nwfile.nw | cpif $DOCS/$nwfile.tex ;
# noweave -t4 -delay -index $WORK/$nwfile.nw > $DOCS/$nwfile.tex
pushd $DOCS/ ;
#
latex $nwfile.tex ;
dvips $nwfile.dvi -o $nwfile.ps -t a4 ;
#
```

```
popd;
⟨BASH script end 17b⟩
```

19a ⟨*LaTeXing* 19a⟩≡

```
⟨BASH shebang 17a⟩
# only LaTeXing
⟨BASH Environment Variables 19b⟩
pushd $DOCS/ ;
#
echo "### RUNNING LaTeX on $nwfile.tex" 1>&2 ;
latex $nwfile.tex ;
latex $nwfile.tex ;
latex $nwfile.tex ;
dvips $nwfile.dvi -o $nwfile.ps -t a4 ;
#
# pdflatex $nwfile.tex ;
echo "### CONVERTING PS to PDF: $nwfile" 1>&2 ;
ps2pdf $nwfile.ps $nwfile.pdf ;
#
popd ;
⟨BASH script end 17b⟩
```

C.4 Defining working shell variables for the current project

19b ⟨*BASH Environment Variables* 19b⟩≡

```
#
# Setting Global Variables
WORK="/home/ug/jabril/development/softjabril/parseblast" ;
BIN="$WORK/bin" ;
PARAM="$BIN/param" ;
DOCS="$WORK/docs" ;
DATA="$WORK/data" ;
nwfile="parseblast" ;
export WORK BIN PARAM DOCS DATA nwfile ;
#
```

19c ⟨*tangling* 18a⟩+≡

```
#
# BASH Environment Variables
notangle -R'BASH Environment Variables' $WORK/$nwfile.nw | \
         cpif $WORK/.bash_VARS ;
source $WORK/.bash_VARS ;
#
```