Supplementary Materials, Part 5, for the Blueprint of the "Alignment Neighborhood Explorer" (ANEX) (tentatively named),

by Kiyoshi Ezawa

(Finished on February 8th, 2019; TOC edited on August 14th, 2020)

Table of Contents

 # Differences from "draft8":
(1) Replaced @lb_sorted_set & @orders_lb with @blocks_w_spec_lb,
and also replaced @rb_sorted_set & @orders_rb with @blocks_w_spec_rb. (DONE on 2019/01/15&16)

(2) Introduced TWO ADDITIONAL categories, '<(pa)' and '>(ch)', into the relations stored in @inter_block_relations.
'<(pa)' means that $bl2 includes $bl1, and that $bl2 is effectively the "parent" of $bl1.
'>(ch)' means that $bl2 is included in $bl1, and that $bl2 is effectively a "child" of $bl1.
(DONE on 2019/01/18.)

Supplementary Methods

SM-5. Examining the effects of alignment changes including simple topological changes (ignored when merely "shift"ing gap-blocks). (Refer to: Sections 4 & 5 of "blueprint1_ANEX.draft5.pdf")

[NOTE1: Here, the moves such as "purge", "merge", etc., refer to the moves from the original (input) alignment to alternative alignments, but NOT to the (possible erroneous) moves from the correct alignment to the input alignment.]

[NOTE2: The overall workflow is the same for (almost) all of the following moves.
It is as follows (in a loop for presenting candidates,) :
(1) present a candidate, which gives the **"base" alignment**;
(2) re-compute the gap-blocks based on the simplest parsimonious indel history;
(3) re-compute the log-probabilities of the "base" alignment for substitutions and for indels;
(4) explore the space of simultaneous "shift"-like moves (with the "base" alignment at the origin);
(5) summarize the results for each "base" alignment
]

(ii) "Purge" (of complementary blocks of the identical size):

```
my @to_be_purged = ();
my @to_be_merged2 = (); # See also (iii-b). # ADDED on Dec 25, 2018.

for (my $bl1=0; $bl1 < $B; $bl1++) { # Modified on 2019/01/27.
# for (my $bl1=0; $bl1 < $B; $bl1++) {

    my $rels_w_bl1 = $inter_block_relations[$bl1];

    for (my $bl2 = $bl1+1; $bl2 < $ub_bl; $bl2++) { # Modified on 2019/01/27.
#    for (my $bl2 = $bl1+1; $bl2 < $B; $bl2++) {

        my $rel = $rels_w_bl1→[$bl2];
        unless ($rel eq 'Cp') { next; }

#       unless ($block_sizes[$bl1] == $block_sizes[$bl2]) { next; } # Skip if the block sizes differ. (OBSOLETE as of Dec 25, 2018.)

        my ($dist1, $dist2) = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations); # This subroutine measures the distance between $bl1 and $bl2, while taking account of the blocks between the two blocks. (See Appendix G I.) #
#       my $dist = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations);
        my $dist = $dist2;  # MODIFIED on Jan 13, 2019.

        if ($block_sizes[$bl1] == $block_sizes[$bl2]) {  # ADDED on Dec 25, 2018.

            if ($dist <= $THRSH_DIST_PURGE) { push @to_be_purged, [$bl1, $bl2];  }

        } else {

            if ($dist <= $THRSH_DIST_MERGE2) { push @to_be_merged2, [$bl1, $bl2, $dist];  }
        }  # ADDED on Dec 25, 2018.
    }
}

foreach my $sbjct_pair (@to_be_purged) {
```

```
    my ($bl1, $bl2) = @{$sbjct_pair};

    my @cp_set_columns0 = copy (@set_columns0);
    my @cp_bds_blocks0 = copy (@bds_blocks0);
    my $ln_prob_new_aln0 = $ln_prob_aln0;

    my @cp_bds_bl_coords = copy (@bds_bl_coords);
    my @cp_org_bl_coords = copy (@org_bl_coords);
    my @cp_inter_block_relations = copy (@inter_block_relations);  # ADDED on Dec 25, 2018.

    my ($lb1, $rb1) = @{$cp_bds_blocks0[$bl1]};
    my ($lb2, $rb2) = @{$cp_bds_blocks0[$bl2]};

    if ($lb1 < $lb2) {
        while ($lb1 < $lb2) {
            {shift $bl2 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) }

            $ln_prob_new_aln0 += $incr_ln_prob;
        }
    } else { # if ($lb2 < $lb1)
        while ($lb2 < $lb1) {
            {shift $bl2 to the right (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) }

            $ln_prob_new_aln0 += $incr_ln_prob;
        }
    }

    {create @new_set_columns0, by removing the columns, $lb1, …, $rb1, from
@cp_set_columns0.}

    {create @new_inter_block_relations, by removing the rows and columns for $bl1 and $bl2
from @cp_inter_block_relations.}

    {create @new_bds_bl_coords, by removing the $bl1 th and $bl2 th elements of
@cp_bds_bl_coords.}

    {create @new_org_bl_coords, by removing the $bl1 th and $bl2 th elements of
@cp_org_bl_coords.}  # ADDED on Dec 25, 2018.

    {create @new_blocks_w_spec_lb0 and @new_blocks_w_spec_rb0, probably from scratch.}
# ADDED on Jan 16, 2019.

    {create other necessary things as well, either from scratch or by using the corresponding ones
before the purge.}

    {compute the indel component of the log-probability,
      either from scratch or by smartly using the result before the purge (refer to:  section 5-1 of
"blueprint1_ANEX.xxxx.pdf")}

    {perform the simultaneous "shift"-like moves of the remaining blocks.}

    => {Output the results}; # ADDED on 2019/01/26.
}
```

(iii-a) "Merge" (same type):

 # Here, we will merge only pairs of neighboring blocks.

```
my @to_be_merged1 = ();

for (my $bl2=1; $bl2 < $ub_bl; $bl2++) { # Modified on 2019/01/27.
# for (my $bl2=1; $bl2 < $B; $bl2++) {

    my $bl1 = $bl2-1;
    my $rel = $inter_block_relations[$bl1]→[$bl2];
    unless ($rel eq '=') { next; }

    my ($dist1, $dist2) = inter_block_distance ($bl1, $bl2, @bds_blocks0,
@inter_block_relations); # (See Appendix G I.) #
    my $dist = $dist2; # MODIFIED on Jan 13, 2019.
#     my $dist = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations);
    if ($dist <= $THRSH_DIST_MERGE1) { push @to_be_merged, [$bl1, $bl2, $dist];  }
}


foreach my $sbjct_pair (@to_be_merged1) {

        my ($bl1, $bl2, $dist) = @{$sbjct_pair};

        my @cp_set_columns0 = copy (@set_columns0);
        my @cp_bds_blocks0 = copy (@bds_blocks0);
        my $ln_prob_new_aln0 = $ln_prob_aln0;

        my @cp_bds_bl_coords = copy (@bds_bl_coords);
        my @cp_org_bl_coords = copy (@org_bl_coords);
        my @cp_inter_block_relations = copy (@inter_block_relations);

        my ($lb1, $rb1) = @{$cp_bds_blocks0[$bl1]};
        my ($lb2, $rb2) = @{$cp_bds_blocks0[$bl2]};

        my ($lb_coord1, $rb_coord1) = @{$bds_bl_coords[$bl1]};
        my ($lb_coord2, $rb_coord2) = @{$bds_bl_coords[$bl2]};
        my ($org1, $org2) = @org_bl_coords[$bl1, $bl2];


        my $shift1 = int ($dist/2);
        my $shift2 = $dist - $shift1;

        my ($left_range, $right_range);

        if ($lb1 < $lb2) {

            $left_range   = $org1 - $lb_coord1 + $shift1;
            $right_range = $rb_coord2 - $org2 + $shift2;

            for (my $i=0; $i < $shift1; $i++) {
```

```
                {shift $bl1 to the right (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) }

                $ln_prob_new_aln0 += $incr_ln_prob;
            }

            for (my $i=0; $i < $shft2; $i++) {
                {shift $bl2 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) }

                $ln_prob_new_aln0 += $incr_ln_prob;
            }

        } else { # if ($lb2 < $lb1)

            $left_range  = $org2 - $lb_coord2 + $shift2;
            $right_range = $rb_coord1 - $org1 + $shift1;

            for (my $i=0; $i < $shift1; $i++) {

                {shift $bl1 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) }

                $ln_prob_new_aln0 += $incr_ln_prob;
            }

            for (my $i=0; $i < $shft2; $i++) {
                {shift $bl2 to the right (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) }

                $ln_prob_new_aln0 += $incr_ln_prob;
            }
        }
```

{As long as #{null columns} = 0, **@new_set_columns0** = @cp_set_columns0 after the "shifts" in the "if {} else {}" blocks above.}

{create **@new_inter_block_relations**, by removing the row and column for $bl2 from @cp_inter_block_relations}

{create **@new_bds_bl_coords**, by removing the $bl2 th element of @cp_bds_bl_coords and by replacing its $bl1 th element with [0, $left_range + $right_range].}

{create **@new_org_bl_coords**, by removing the $bl2 th element of @cp_org_bl_coords and by replacing its $bl1 th element with $left_range.}

{create **@new_blocks_w_spec_lb0** and **@new_blocks_w_spec_rb0**, probably from scratch} # ADDED on Jan 16, 2019.

{create **other necessary things** as well, either from scratch or by using the corresponding ones before the merge}

{compute the indel component of the log-probability,

either from scratch or by smartly using the result before the merge (refer to: **section 5-1** of "**blueprint1_ANEX.xxxx.pdf**")}

{perform the **simultaneous "shift"-like moves** of the *remaining* blocks}

=> {Output the results}; # ADDED on 2019/01/26.
}

(iii-b) "Merge" (complementary types):

 # See (ii) for the preparation of @to_be_merged2.

```
foreach my $sbjct_pair (@to_be_merged2) {

    my ($bl1, $bl2, $dist) = @{$sbjct_pair};

    my @cp_set_columns0 = copy (@set_columns0);
    my @cp_bds_blocks0 = copy (@bds_blocks0);
    my $ln_prob_new_aln0 = $ln_prob_aln0;

    my @cp_bds_bl_coords = copy (@bds_bl_coords);
    my @cp_org_bl_coords = copy (@org_bl_coords);
    my @cp_inter_block_relations = copy (@inter_block_relations);

    my ($lb1, $rb1) = @{$cp_bds_blocks0[$bl1]};
    my ($lb2, $rb2) = @{$cp_bds_blocks0[$bl2]};

    my ($lb_coord1, $rb_coord1) = @{$bds_bl_coords[$bl1]};
    my ($lb_coord2, $rb_coord2) = @{$bds_bl_coords[$bl2]};
    my ($org1, $org2) = @org_bl_coords[$bl1, $bl2];

    my ($size1, $size2) = @block_sizes[$bl1, $bl2];

    my ($size_S, $size_L) = ($size1 < $size2) ? ($size1, $size2) : ($size2, $size1) ;

    my ($left_range, $right_range);

    if ($lb1 < $lb2) {

        $left_range  = $org1 - $lb_coord1;
        $right_range = $rb_coord2 - $org2 + $dist;

        for (my $i=0; $i < $dist + $size_S; $i++) {

            {shift $bl2 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
            , while computing the log-probability increment (= $incr_ln_prob) }

            $ln_prob_new_aln0 += $incr_ln_prob;
        }

    } else { # if ($lb2 < $lb1)

        $left_range  = $org2 - $lb_coord2 + $dist;
        $right_range = $rb_coord1 - $org1;
```

```
            for (my $i=0; $i < $dist + $size_S; $i++) {

                {shift $bl2 to the right (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) }

                $ln_prob_new_aln0 += $incr_ln_prob;
            }
        }

        ($lb1, $rb1) = @{$cp_bds_blocks[$bl1]}; # Re-compute the boundaries of the $bl1 th block.

        my @null_clms = ();
        for ($i=$lb1; $i <= $rb1; $i++) {
            my $cnct_clm = join ('', @{$cp_set_columns0[$i]});
            if ($cnct_clm eq $cnct_null_clm) { push @null_clms, $i; }
        }

        my ($rmvd_bl, $remaining_bl) = ($size1 < $size2) ? ($bl1, $bl2) : ($bl2, $bl1);

    {create @new_set_columns0, by removing the columns listed in @null_clms from
@cp_set_columns0.}

    {create @new_inter_block_relations, by removing the row and column for $rmvd_bl from
@cp_inter_block_relations}

    {create @new_bds_bl_coords, by removing the $rmvd_bl th element of @cp_bds_bl_coords
and by replacing its $remaining_bl th element with [0, $left_range + $right_range].}

    {create @new_org_bl_coords, by removing the $rmvd_bl th element of @cp_org_bl_coords
and by  replacing its $remaining_bl th element with $left_range.}

    {create @new_blocks_w_spec_lb0 and @new_blocks_w_spec_rb0, either from scratch or by
using the old ones} # ADDED on Jan 16, 2019.

    {create other necessary things as well, either from scratch or by using the corresponding ones
before the merge}

    {compute the indel component of the log-probability,
        either from scratch or by smartly using the result before the merge (refer to:  section 5-1 of
"blueprint1_ANEX.xxxx.pdf")}

    {perform the simultaneous "shift"-like moves of the remaining blocks}

    => {Output the results}; # ADDED on 2019/01/26.
}
```

(iv-a) "Split" (into blocks of the same type):

```
        # It would be better to use a window containing blocks fewer than considered by one.

for (my $bl=0; $bl < $ub_bl; $bl++) { # Outer for-loop (over gap-blocks). #
# for (my $bl=0; $bl < $B; $bl++) { # Outer for-loop (over gap-blocks). #

            # Information on the block to be split. #
    my $size = $block_sizes[$bl];
    if ($size < 2) { next; }
```

```
    my ($lb, $rb) = @{$cp_bds_blocks0[$bl]};
    my ($lb_coord, $rb_coord) = @{$bds_bl_coords[$bl]};
    my $org = $org_bl_coords[$bl];

    my @involved_gp_blocks = {retrieved from the information on the original local alignment};
# (IMPORTANT!!) A list of gap-pattern blocks involved in this ($bl th) block.

    my $ct_gp_blks = @involved_gp_blocks;
    for (my $i=0; $i < $ct_gp_blks; $i++) {  # Middle for-loop (over gap-pattern blocks). #

        my $indx_gpb = $involved_gp_blocks[$i];
        my ($lb_gpb, $rb_gpb) = {retrieved from the information on the $indx_gpb th gap-pattern
block. Assume the full-closed convention.};

        my @involved_gp_blocks1 = ($i >0) ? @involved_gp_blocks[0 .. $i-1] : ();
        my @involved_gp_blocks2 = ($i < $ct_gp_blks-1) ? @involved_gp_blocks[$i+1 ..
$ct_gp_blks-1] : ();

        my $rb_rb_left = $rb_gpb;
        if ($i == $ct_gp_blks-1) { $rb_rb_left--; }

        for (my $rb_left = $lb_gpb;  $rb_left <= $rb_rb_left; $rb_left++) { # Inner for-loop (over
columns in the gap-pattern block). #

            my $lb_right = ($rb_left < $rb_gpb) ? $rb_left+1 : {$lb_gpb for the
$involved_gp_blocks[$i+1] th gap-pattern block} ;

                # … This part may be superfluous … #
#           my @left_gpb = ($lb_gpb, $rb_left);
#           my @right_gpb = ($lb_right, $rb_gpb);
#
#           my @new_involved_gp_blocks1 = @involved_gp_blocks1;
#           my @new_involved_gp_blocks2 = @involved_gp_blocks2;
#
#           if ($rb_left < $rb_gpb) {
#               push @new_involved_gp_blocks1, {info on @left_gpb};
#               unshift @new_involved_gp_blocks2, {info on @right_gpb};
#           } else { # if ($rb_left == $rb_gpb)
#               push @new_involved_gp_blocks1, $indx_gpb;
#           }
                # END of "… This part may be superfluous …" #

            my @bds_block_left = ($lb, $rb_left);
            my @bds_block_right = ($lb_right, $rb);

            my @cp_set_columns0 = copy (@set_columns0);

            my @cp_bds_blocks0 = copy (@bds_blocks0);
            my $ln_prob_new_aln0 = $ln_prob_aln0;

            my @cp_bds_bl_coords = copy (@bds_bl_coords);
            my @cp_org_bl_coords = copy (@org_bl_coords);
            my @cp_inter_block_relations = copy (@inter_block_relations);


                # Create @new_bds_blocks0. #
```

```
            my @new_bds_blocks0 = ($bl>0) ? @cp_bds_blocks0[0 .. $bl-1];
            push @new_bds_blocks0, \@bds_block_left, \@bds_block_right;
            if ($bl < $ub_bl -1) { push @new_bds_blocks0, @cp_bds_blocks0[$bl+1 .. $ub_bl -
1]; }
#           if ($bl < $B -1) { push @new_bds_blocks0, @cp_bds_blocks0[$bl+1 .. $B -1]; }

                # Prepare for creating @new_bds_bl_coords & @new_org_bl_coords. #

            my $lb_coord1 = my $lb_coord2 = $lb_coord;
            my $rb_coord1 = my $rb_coord2 = $rb_coord;
            my $org1 = my $org2 = $org;

                # MOVE the "split" parts of the $bl th block, so that they will indeed be split. #

            if (($rb < {right-end of the local alignment}) and ($rb_coord > 0)) {

                { SHIFT the right-part of $bl to the right by one column (maybe using
"shift_bl_and_compt_prob_incr (@@@$$) {...}")
                , while computing the log-probability increment (= $incr_ln_prob)};

                $ln_prob_new_aln0 += $incr_ln_prob;

                $lb_coord2++;
                $org2++;
                $rb_coord2--;

            } else {

                {SHIFT the left-part of $bl to the left by one column (maybe using
"shift_bl_and_compt_prob_incr (@@@$$) {...}")
                , while computing the log-probability increment (= $incr_ln_prob)};

                $ln_prob_new_aln0 += $incr_ln_prob;

                $lb_coord1--;
                $org1--;
                $rb_coord1++;

            }


                # Create @new_bds_bl_coords & @new_org_bl_coords. #

            my @new_bds_bl_coords  = ($bl>0) ? @cp_bds_bl_coords[0 .. $bl-1] : ();
            push @new_bds_bl_coords, [$lb_coord1, $rb_coord1], [$lb_coord2, $rb_coord2];
            if ($bl < $ub_bl -1) { push @new_bds_bl_coords, @cp_bds_bl_coords[$bl+1 .. $ub_bl -
1]; }
#           if ($bl < $B  -1) { push @new_bds_bl_coords, @cp_bds_bl_coords[$bl+1 .. $B  -1]; }

            my @new_org_bl_coords = ($bl>0) ? @cp_org_bl_coords[0 .. $bl-1] : () ;
            push @new_org_bl_coords, $org1, $org2;
            if ($bl < $ub_bl -1) { push @new_org_bl_coords, @cp_org_bl_coords[$bl+1 .. $ub_bl -
1]; }
#           if ($bl < $B -1) { push @new_org_bl_coords, @cp_org_bl_coords[$bl+1 .. $B  -1]; }

                # Create @new_inter_block_relations. #

             my  @cp_rels_w_bl = @{$cp_inter_block_relations[$bl]};
```

```
            my @new_inter_block_relations = ($bl>0) ? @cp_inter_block_relations[0 .. $bl-1] : ();
            push @new_inter_block_relations, $cp_inter_block_relations[$bl], \@cp_rels_w_bl;
            if ($bl < $ub_bl -1) { push @new_inter_block_relations,
@cp_inter_block_relations[$bl+1 .. $ub_bl -1] ; }
#           if ($bl < $B -1) { push @new_inter_block_relations, @cp_inter_block_relations[$bl+1 ..
$B  -1] ; }
            for (my $bl1=0; $bl1 <= $ub_bl; $bl1++) {
#           for (my $bl1=0; $bl1 <= $B ; $bl1++) {
                my $rels_w_bl1 = $new_inter_block_relations[$bl1];
                my $rel = $rels_w_bl1->[$bl];
                my @new_rels_w_bl1 = ($bl>0) ? @{$rels_w_bl1}[0 .. $bl-1] : ();
                push @new_rels_w_bl1, $rel, $rel;
                if ($bl < $ub_bl -1) { push @new_rels_w_bl1, @{$rels_w_bl1}[$bl1+1 .. $ub_bl -
1]; }
#               if ($bl < $B -1) { push @new_rels_w_bl1, @{$rels_w_bl1}[$bl1+1 .. $B -1]; }
                $new_inter_block_relations[$bl1] = \@new_rels_w_bl1;
            }
            $new_inter_block_relations[$bl]->[$bl+1] = '=';
            $new_inter_block_relations[$bl+1]->[$bl] = '=';
```

{create **@new_blocks_w_spec_lb0** and **@new_blocks_w_spec_rb0**, probably from scratch} # ADDED on Jan 16, 2019.

{create **other necessary things** as well, either from scratch or by using the corresponding ones before the merge}

{compute the indel component of the log-probability,
either from scratch or by smartly using the result before the merge (refer to: **section 5-1** of "**blueprint1_ANEX.xxxx.pdf**")}

{perform the **simultaneous "shift"-like moves** of the *remaining* blocks}

=> {Output the results}; # ADDED on 2019/01/26.

```
        }   # END of the inner for-loop (over columns in the gap-pattern block). #

    }  # END of the middle for-loop (over gap-pattern blocks). #

} # END of the outer for-loop (over gap-blocks). #
```

(iv-b) "Split" (into blocks of complementary types):

```
        # It would be better to use a window containing blocks fewer than considered by one.

        # NOTE: Possible candidates may be short-listed using the result of pre-scanning for "ex-
nihilo" candidates (in (v) below). #

my $if_pure_split = 1; # ADDED on Jan 8, 2018.

for (my $bl=0; $bl < $ub_bl; $bl++) { # Outer for-loop (over gap-blocks). #
#for (my $bl=0; $bl < $B; $bl++) { # Outer for-loop (over gap-blocks). #

            # Information on the block to be "split". #
    my $size = $block_sizes[$bl];
    my ($lb, $rb) = @{$cp_bds_blocks0[$bl]};
    my ($lb_coord, $rb_coord) = @{$bds_bl_coords[$bl]};
```

```
    my $org = $org_bl_coords[$bl];

    my ($br, $u_or_d) = @{$block_info[$bl]}[$indx_br, $indx_u_or_d];
    my $pa  = $node2pa->{$br}; # ADDED on Dec 30, 2018.
    my $eq_br = $eq2br->{$br}; # ADDED on Dec 30, 2018.

    my $rels_w_bl = $inter_block_relations[$bl];

    my @indices_aff_classes = @{$affected_classes[$bl]}; # ADDED on Dec 31, 2018.


    # Assume that we have @blocks_w_spec_lb and @blocks_w_spec_rb. # MODIFIED on Jan 15, 2019. (1a) #

                # OBSOLETE as of Jan 15, 2019. (1a) #
    # Assume that we have @lb_sorted_set, @orders_lb, @rb_sorted_set, and @orders_rb.
#
#     my $order_lb = $orders_lb[$bl];
#     my $order_rb = $orders_rb[$bl];
#    my %clm2involved_l;
                # END of "OBSOLETE as of Jan 15, 2019. (1a)" #

#    my $left_margin = $lb;
    my $le_left_margin = 0; # ADDED on Jan 7, 2018.

    for (my $bl2 = 0; $bl2 < $ub_bl; $bl2++) { # 1st middle for-loop (over blocks on the left of $bl).
        if ($bl2 == $bl) { next; }     # MODIFIED on Jan 15, 2019. (1b)
                # OBSOLETE as of Jan 15, 2019. (1b) #
#    for (my $k = $order_lb-1; $k >= 0; $k--) { # 1st middle for-loop (over blocks on the left of $bl).
#        my $bl2 = $lb_sorted_set[$k];
                # END of "OBSOLETE as of Jan 15, 2019. (1b)" #

        my $size2 = $block_sizes[$bl2];
        my ($lb2, $rb2) = @{$cp_bds_blocks0[$bl2]};
        if ($lb <= $rb2) { next; }

        my $rel = $rels_w_bl->[$bl2];
        if (($rel eq '=') or ($rel eq 'Cp')) {
            # $left_margin -= $lb2+1;
            $le_left_margin = $rb2+2;
            last;
        } elsif ($rel eq 'S') { # $bl and $bl2 are "siblings". # ADDED on Dec 30, 2018.
            # $left_margin -= $lb2;
            $le_left_margin = $rb2+1;
            last;
                # MODIFIED on Jan 18, 2019. (1) #
        } elsif ($rel eq '>(ch)') {
                $le_left_margin = $rb2+1;
                last;

#        } elsif ($rel eq '>') { # ADDED on Dec 30, 2018.
#            my ($br2, $u_or_d2) = @{$block_info[$bl2]}[$indx_br, $indx_u_or_d];
#            my $pa2 = $node2pa->{$br2};
#            my $eq_br2 = $eq2br->{$br2};
#            if (($br2 == $pa) or ($br == $pa2)
#                or ((defined $eq_br) and ($eq_br == $pa2))
#                or ((defined $eq_br2) and ($eq_br2 == $pa)) ) { # $bl2 is a "child" of $bl.
#                $left_margin -= $lb2;
```

```
#                   $le_left_margin = $rb2+1;
#                   last;
#               }
                    # END of "MODIFIED on Jan 18, 2019. (1)" #

            } elsif (($rel eq '<') or ($rel eq '<(pa)')) { # $bl2 vertically includes $bl. # Added '<(pa)' on Jan 18, 2019.
                # $left_margin -= $size2;
                for (my $c= $lb2; $c <= $rb2; $c++) {
                    my $involved = $clm2involved_l{$c};
                    unless (defined $involved) { $involved = $clm2involved_l{$c} = []; }
                    push @{$involved}, $bl2;
                }
            } elsif (($rel eq 'ONN') or ($rel eq 'ONCS')) { # $bl2 and $bl overlap but do not nest.
                # $left_margin -= $size2;
                for (my $c= $lb2; $c <= $rb2; $c++) {
                    my $involved = $clm2involved_l{$c};
                    unless (defined $involved) { $involved = $clm2involved_l{$c} = []; }
                    push @{$involved}, $bl2;
                }
            }

    }  # End of the 1st middle for-loop (over blocks on the left of $bl).


            # ADDED on Jan 7, 2018. #
        my @set_left_flanking_clms = ();
        for (my $c = $le_left_margin; $c < $lb; $c++) {
            unless (defined $clm2involved_l{$c}) { push @set_left_flanking_clms, $c; }
        }
        my $left_margin = @set_left_flanking_clms;
            # END of "ADDED on Jan 7, 2018." #

        my %clm2involved_r;
       # my $right_margin = $right_end_laln - $rb;
        my $re_right_margin = $right_end_laln; # ADDED on Jan 7, 2018.



        for (my $bl2 = 0; $bl2 < $ub_bl; $bl2++) { # 2nd middle for-loop (over blocks on the right of $bl).
            if ($bl2 == $bl) { next; } # MODIFIED on Jan 15, 2019. (2) #

                    # OBSOLETE as of Jan 15, 2019. (2)" #
     #    for (my $k = $order_rb+1; $k < $B $ub_bl; $k++) { # 2nd middle for-loop (over blocks on the right of $bl).
#           my $bl2 = $rb_sorted_set[$k];
                    # END of "OBSOLETE as of Jan 15, 2019. (2)" #

            my $size2 = $block_sizes[$bl2];
            my ($lb2, $rb2) = @{$cp_bds_blocks0[$bl2]};
            if ($lb2 <= $rb) { next; }

            my $rel = $rels_w_bl→[$bl2];
            if (($rel eq '=') or ($rel eq 'Cp')) {
                # $right_margin -= ($right_end_laln - $rb2)+1;
                $re_right_margin = $lb2 – 2;
                last;
```

```
        } elsif ($rel eq 'S') { # $bl and $bl2 are "siblings". # ADDED on Dec 30, 2018.
            # $right_margin -= $right_end_laln - $rb2;
            $re_right_margin = $lb2 -1;
            last;

                # MODIFIED on Jan w18, 2019. (2) #
        } elsif ($rel eq '>(ch)') {
                $re_right_margin = $lb2 -1;
                last;

#       } elsif ($rel eq '>') { # ADDED on Dec 30, 2018.
#           my ($br2, $u_or_d2) = @{$block_info[$bl2]}[$indx_br, $indx_u_or_d];
#           my $pa2 = $node2pa->{$br2};
#           my $eq_br2 = $eq2br->{$br2};
#           if (($br2 == $pa) or ($br == $pa2)
#               or ((defined $eq_br) and ($eq_br == $pa2))
#               or ((defined $eq_br2) and ($eq_br2 == $pa)) ) { # $bl2 is a "child" of $bl.
#               $right_margin -= $right_end_laln - $rb2;
#               $re_right_margin = $lb2 -1;
#               last;
#            }
                # END of "MODIFIED on Jan w18, 2019. (2)" #

        } elsif (($rel eq '<') or ($rel eq '<(pa)')) { # $bl2 vertically includes $bl. # Added '<(pa)' on
Jan 18, 2019.
            # $right_margin -= $size2;
            for (my $c= $lb2; $c <= $rb2; $c++) {
                my $involved = $clm2involved_r{$c};
                unless (defined $involved) { $involved = $clm2involved_r{$c} = []; }
                push @{$involved}, $bl2;
            }
        } elsif (($rel eq 'ONN') or ($rel eq 'ONCS')) {# $bl2 and $bl overlap but do not nest.
            # $right_margin -= $size2;
            for (my $c= $lb2; $c <= $rb2; $c++) {
                my $involved = $clm2involved_r{$c};
                unless (defined $involved) { $involved = $clm2involved_r{$c} = []; }
                push @{$involved}, $bl2;
            }
        }

    }  # End of the 2nd middle for-loop (over blocks on the right of $bl).


        # ADDED on Jan 7, 2018. #
    my @set_right_flanking_clms = ();
    for (my $c = $rb+1; $c <= $re_right_margin; $c++) {
        unless (defined $clm2involved_r{$c}) { push @set_right_flanking_clms, $c; }
    }
    my $right_margin = scalar (@set_right_flanking_clms);
        # END of "ADDED on Jan 7, 2018". #

    my $if_on_the_left = ($left_margin >= $right_margin) 1 : 0;
    my $margin = ($left_margin >= $right_margin) $left_margin : $right_margin;

    my $ub_size = ($margin < $THRSH_SIZE_SPLIT2) ? $margin : $THRSH_SIZE_SPLIT2;

    my $clm2involved = ($if_on_the_left) ? \%clm2involved_l : \%clm2involved_r ;
```

```
    # my $to_be_split = ($if_on_the_left) ?  $lb-1 : $rb+1; # REMOVED on Jan 8, 2018.

                    # ADDED on Dec 30, 2018. #

          # Determine the rank of the complement block to be created. #

     my $lb_cmpl =  my $rb_cmpl = ($if_on_the_left) ? $lb-1 : $rb+2;
     my ($indices_seqs_affected_by_cmpl, $dummy, $common) =  diff (@indices_all_seqs,
@indices_seqs_affected_by_bl);
     my $ct_seqs_affected_by_cmpl = @{$indices_seqs_affected_by_cmpl};
          => {According to these pieces of information,  assign the right rank, $bl_cmpl, to the
complement};

     my @new_set_columns0 = copy (@set_columns0);

     my @new_bds_blocks0 = copy (@bds_blocks0);
          => {Insert [$lb_cmpl, $rb_cmpl] between the $bl_cmpl -1 and $bl_cmpl th elements of the
current @new_bds_blocks0};

     my $ln_prob_new_aln0 = $ln_prob_aln0;

     my @new_bds_bl_coords = copy (@bds_bl_coords);
     my @bds_bl_coords_cmpl = ($lb_coord, $rb_coord + $size) ;
          => {Insert \@bds_bl_coords_cmpl between the $bl_cmpl -1 and $bl_cmpl th elements of the
current @new_bds_bl_coords}; # Once the range is given, it will remain unchanged even while the
complementary pair expands!! (ADDED on Jan 1, 2019.)

     my @new_org_bl_coords = copy (@org_bl_coords);
     my $org_bl_coords_cmpl = ($if_on_the_left) ? $org -1 : $org +$size + 1;
          => {Insert $org_bl_coords_cmpl between the $bl_cmpl-1 and $bl_cmpl th elements of the
current @new_org_bl_coords};

     my @new_inter_block_relations = copy (@inter_block_relations);
     my @rels_w_cmpl = ();
     for (my $bl3=0; $bl3 < $B; $bl3++) {
          $rels_w_cmpl[$bl3]  = {determine the relation between $bl3 and $bl_cmpl, as in Appendix
F of "suppl3_blueprint1_ANEX.xxxx.odt"};
     }
          => {Insert \@rels_w_cmpl between the $bl_cmpl -1 and $bl_cmpl th elements of
@new_inter_block_relations};
    for (my $bl3=0; $bl3<=$B; $bl3++) {
          my $rel = ($bl3 == $bl_cmpl) ? undef : $new_inter_block_relations[$bl_cmpl]->[$bl3];
           => {Insert $rel between $bl_cmpl -1 and $bl_cmpl th elements of
@{$new_inter_block_relations[$bl3]};
     }

                    # MODIFIED on Jan 15, 2019. (3) #

    => {Create @new_blocks_w_spec_lb and @new_blocks_w_spec_rb,
          according to @new_bds_blocks0.};

                    # OBSOLETE as of Jan 15, 2019. (3) #
#    {Create @new_lb_sorted_set and @new_rb_sorted_set
#          by increasing the ranks by one if they are greater than $bl_cmpl, and
#          by inserting the new $bl_cmpl into the proper places in (the copies of)
#          @lb_sorted_set and @rb_sorted_set.};
#     => {Record the new orders into @new_orders_lb and @new_orders_rb.};
```

```
                # END of "OBSOLETE as of Jan 15, 2019. (3)" #
                # END of "MODIFIED on Jan 15, 2019. (3)" #

                # END of "ADDED on Dec 30, 2018". #

    my $new_bl = ($bl < $bl_cmpl) ? $bl : $bl+1; # ADDED on Dec 31, 2018.


    for (my $size_cmpl=1; $size_cmpl <= $ub_size; $size_cmpl++) { # 3rd middle for-loop (over the sizes of the new complementary block). #

                # MODIFIED on Jan 7, 2018. #

        my $to_be_split = ($if_on_the_left) ? (pop @set_left_flanking_clms) : (shift @set_right_flanking_clms);

#       while (defined $clm2involved->{$to_be_split}) {
#           ($if_on_the_left) ?  ($to_be_split--) : ($to_be_split++) ;
#       }
                # END of "MODIFIED on Jan 7, 2018". #

        { split the $to_be_split th column (in the original local alignment) at the branch $br,
          and move the $u_or_d side  to the (($if_on_the_left) ? right  : left),
           and "merge" it with the $bl th block,
           and also "merge" its complement with the complement of the $bl th block.};  # See Appendix H J.

                # ADDED on Jan 1, 2019. #

                # Modify the coordinate ranges. #
        if ($if_on_the_left) {
                # The $new_bl th block does NOT change its range.
                # The $bl_cmpl th block moves its coordinate origin by one to the left.
            $new_org_bl_coords[$bl_cmpl]--;
        } else {
                # The $new_bl th block does NOT change its range.
                # The $bl_cmpl th block moves its  coordinate origin by one to the right.
            $new_org_bl_coords[$bl_cmpl]++;
        }

                # MODIFIED on Jan 15, 2019. (5) #
        => {Update @new_blocks_w_spec_lb0 and @new_blocks_w_spec_rb0.};

                # OBSOLETE as of Jan 15, 2019. (5) #
#       => {Re-sort the left-bounds and the right-bounds to update @new_lb_sorted_set and @new_rb_sorted_set, respectively};
#       => {Update @new_orders_lb and @new_orders_rb};
                # END of "OBSOLETE as of Jan 15, 2019. (5)" #
                # END of "MODIFIED on Jan 15, 2019. (5)" #

        {Modify other important data sets accordingly};

         =>  {Perform the simultaneous "shift"-like moves of the resulting blocks}

                # END of "ADDED on Jan 1, 2019." #

         => {Output the results}; # ADDED on 2019/01/26.
```

```
                # RESTARTED on Dec 30, 2018. #

#       ($if_on_the_left) ?  ($to_be_split--) : ($to_be_split++) ; # REMOVED on Jan 7, 2018.

    } # End of the 3rd middle for-loop (over the sizes of the new complementary block). #

} # END of the outer for-loop (over gap-blocks). #
```

(v) "Ex-nihilo" (creating a pair of complementary blocks) (optional):

```
        # It would be better to use a window containing blocks fewer than considered by TWO.

Use the result of pre-scanning the input alignment. (See, e.g., "suppl2_blueprint1_ANEX.xxxx.doc"
and  "suppl2_addendum.xxxx.doc".)

        # Consider Concrete Algorithm (for determining possible candidate regions) Later. #

        # RESTARTED on Jan 1, 2019. #

# Here we will assume that a set of candidate regions are given.

my @null_column = ();
for (1 .. $ct_seqs) { push @null_column, $GAP; }

foreach my $candidate (@set_cands_ex_nihilo) {

        my ($br, $leftmost, $rightmost) = @{$candidate};
        my $size_x_nhl = $rightmost - $leftmost + 1;

        my $nodes_lower_side =fetch_ext_offsprings ($br, %node2ch);
        my @seqs_lower_side = ();
        foreach my $node (@{$nodes_lower_side}) { push @seqs_lower_side,
$node2indx_seq{$node}; }
#       my @std_seqs_lower_side = sort {$a <=> $b} @seqs_lower_side;

        my @cp_set_columns0 = copy (@set_columns0); # Use the full-sequence representation,
instead of the representation using classes.

        my @new_set_columns0 = ($leftmost>0) ? @cp_set_columns0[0 .. $leftmost-1] : ();

        for (my $c = $leftmost; $c <= $rightmost; $c++) {

            # Place the gap-block on the upper-side to the left,
            # and that on the lower-side to the right.
            my @clm_left = copy (@null_column) ;
            my @clm_right = @{$cp_set_columns0[$c]};

            foreach my $seq (@seqs_lower_side) {
                my $tmp = $clm_left[$seq];
                $clm_left[$seq] = $clm_right[$seq];
                $clm_right[$seq] = $tmp;
            }

            $new_set_columns0[$c] = \@cp_clm_left;
            $new_set_columns0[$c+$size_x_nhl] = \@cp_clm_right;
        }
```

```
if ($rightmost < $#cp_set_columns0) {
    push @new_set_columns0, @cp_set_columns0[$rightmost + 1 .. $#cp_set_columns0];
}

foreach my $pos (all positions for @set_columns0) {
    if (( $leftmost <= $pos) and ($pos <= $rightmost)) {
        if ($pos is on the upper-side of $br) {  $pos += $size_x_nhl;  }
    } elsif ($rightmost < $pos) {
        $pos += $size_x_nhl;
    }
}

my @bds_left = ($leftmost, $rightmost);
my @bds_right = ($leftmost + $size_x_nhl, $rightmost + $size_x_nhl);
```

{Determine the ranks of the newly created blocks, **$bl_left** and **$bl_right**, according to the sets of sequences affected by the respective blocks(, as well as to their horizontal positions).};

=> {Create **@new_bds_blocks0**, by inserting **\@bds_left** between the ($bl_left-1) th and $bl_left th elements of @cp_bds_blocks0, and by inserting **\@bds_right** between the ($bl_right -1) th and $bl_right th elements of the resulting **@cp_bds_blocks0**, if $bl_left < $bl_right};

{ If $bl_left > $bl_right, swap the order of the inserted block boundaries};

{Create **@new_inter_block_relations**, by examining the relationships between the $bl_left th block and the old ones, as well as between the $bl_right th block and the old ones,

then, by setting:

$new_inter_block_relations[$bl_left]→[$bl_right] =

$new_inter_block_relations[$bl_right]→[$bl_left] = 'Cp'.};

{Create **@new_blocks_w_spec_lb0** and **@new_blocks_w_spec_rb0**, probably from scratch} # ADDED on Jan 16, 2019.

{Create or modify other important data sets, including **@new_bds_bl_coords** and **@new_org_bl_coords**, accordingly};

{Especially, create **@new_bds_bl_coords** and **@new_org_bl_coords** by inserting the boundaries for the $new_bl th and $bl_cmpl th blocks and by taking account of modifications of the boundaries for the other blocks.}; # ADDED on Jan 2, 2019.

=> {Perform the **simultaneous "shift"-like moves** of the resulting blocks}

=> {Output the results}; # ADDED on 2019/01/26.

}

END of "RESTARTED on Jan 1, 2019."

(vi-a) "(incomplete) Vertical-merge" (sibling blocks):

… Actually, it is already incorporated in the simultaneous "shift"-like moves…
… How to recognize them may be important, though …
=> Consider the method later!!

(vi-b) "(incomplete) Vertical-merge" (complementary-sibling blocks (i.e., sibling sequence-blocks)**):**

RESTARTED on Jan 2, 2019.

```
my @to_be_vmerged2 = ();
```

```
for (my $bl1=0; $bl1 < $ub_bl; $bl1++) { # Modified on 2019/01/27.
# for (my $bl1=0; $bl1 < $B; $bl1++) {

    my $rels_w_bl1 = $inter_block_relations[$bl1];

    for (my $bl2 = $bl1+1; $bl2 < $ub_bl; $bl2++) { # Modified on 2019/01/27.
#     for (my $bl2 = $bl1+1; $bl2 < $B; $bl2++) {
        my $rel = $rels_w_bl1→[$bl2];
        unless ($rel eq 'ONCS') { next; }

        my ($dist1, $dist2) = inter_block_distance ($bl1, $bl2, @bds_blocks0,
@inter_block_relations); # This subroutine measures the distance between $bl1 and $bl2, while
taking account of the blocks between the two blocks. (See Appendix G I.) #
        my $dist = $dist2; # MODIFIED on Jan 13, 2019.
#       my $dist = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations);

        if ($dist <= $THRSH_DIST_VMERGE2) { push @to_be_vmerged2, [$bl1, $bl2, $dist];  }

    }
}

foreach my $sbjct_pair (@to_be_vmerged2) {

        my ($bl1, $bl2, $dist) = @{$sbjct_pair};

        my @cp_set_columns0 = copy (@set_columns0);
        my @cp_bds_blocks0 = copy (@bds_blocks0);
        my $ln_prob_new_aln0 = $ln_prob_aln0;

        my @cp_bds_bl_coords = copy (@bds_bl_coords);
        my @cp_org_bl_coords = copy (@org_bl_coords);
        my @cp_inter_block_relations = copy (@inter_block_relations);

        my ($lb1, $rb1) = @{$cp_bds_blocks0[$bl1]};
        my ($lb2, $rb2) = @{$cp_bds_blocks0[$bl2]};

        my ($lb_coord1, $rb_coord1) = @{$bds_bl_coords[$bl1]};
        my ($lb_coord2, $rb_coord2) = @{$bds_bl_coords[$bl2]};
        my ($org1, $org2) = @org_bl_coords[$bl1, $bl2];

        my ($size1, $size2) = @block_sizes[$bl1, $bl2];

        my ($size_S, $size_L) = ($size1 < $size2) ? ($size1, $size2) : ($size2, $size1) ;
        my $diff_size = $size_L - $size_S;

        my ($new_lb12, $new_rb12);
        my ($new_lb1, $new_rb1);
        my ($new_lb2, $new_rb2);
        my ($left_range12, $right_range12);
        my ($left_range1, $right_range1);
        my ($left_range2, $right_range2);

        if ($lb1 < $lb2) {

            my $left_range  = $org1 - $lb_coord1;
            my $right_range = $rb_coord2 - $org2 + $dist;

            for (my $i=0; $i < $dist; $i++) {
```

```
                {shift $bl2 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) };

                $ln_prob_new_aln0 += $incr_ln_prob;
            }
            for (my $j=0; $j < $size_S; $j++) {
                {shift the sequence block complement to $bl2 to the left (maybe using
"shift_cmpl_bl_and_compt_prob_incr (@@@$$) {...}")
                , while computing the log-probability increment (= $incr_ln_prob) }; # Align the left-
ends of the complement $bl1 and the complement $bl2.
                {Remove the resulting null-column};

                $ln_prob_new_aln0 += $incr_ln_prob;
            }
            => {Decrease by $size_S the positions greater than $rb1};

            if ($size1 < $size2) {
                $left_range12 = $left_range;
                $right_range12 = $right_range + $diff_size;
                $new_lb12 = $lb1;
                $new_rb12 = $rb1;
                $left_range2 = $left_range + $size_S;
                $right_range2 = $right_range;
                $new_lb2 = $new_rb12 +1;
                $new_rb2 = $cp_bds_blocks0[$bl2]→[1]; # After the above moves. #
            } elsif ($size1 > $size2) {
                $left_range12 = $left_range;
                $right_range12 = $right_range + $diff_size;
                $new_lb12 = $lb1;
                $new_rb12 = {Maybe from the output of "shift_cmpl_bl_and_compt_prob_incr
(@@@$$) {...}"}; #  … or ($lb1 + $size_S)  or ($rb1 - $diff_size);
                $left_range1 = $left_range + $size_S;
                $right_range1 = $right_range;
                $new_lb1 = $new_rb12 + 1;
                $new_rb1 = $rb1;
            } else {
                $left_range12 = $left_range;
                $right_range12 = $right_range;
                $new_lb12 = $lb1;
                $new_rb12 = $rb1;
            }

        } else { # if ($lb2 < $lb1)

            my $left_range   = $org2 - $lb_coord2 + $dist;
            my $right_range = $rb_coord1 - $org1;

            for (my $i=0; $i < $dist; $i++) {

                {shift $bl2 to the right (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) };

                $ln_prob_new_aln0 += $incr_ln_prob;
            }
            for (my $j=0; $j < $size_S; $j++) {
```

```
                {shift the sequence block complement to $bl2 to the right (maybe using
"shift_cmpl_bl_and_compt_prob_incr (@@@$$) {...}")
                , while computing the log-probability increment (= $incr_ln_prob) }; # Align the
right-ends of the complement $bl1 and the complement $bl2.
                {Remove the resulting null-column};

                $ln_prob_new_aln0 += $incr_ln_prob;
            }
            => {Decrease by $size_S the positions greater than or equal to $lb1};

            if ($size1 < $size2) {
                $left_range12 = $left_range + $diff_size;
                $right_range12 = $right_range;
                $new_lb12 = $lb1;
                $new_rb12 = $rb1;
                $left_range2 = $left_range;
                $right_range2 = $right_range + $size_S;
                $new_lb2 = $cp_bds_blocks0[$bl2]→[0];
                $new_rb2 = $new_lb12 -1;

            } elsif ($size1 > $size2) {
                $left_range12 = $left_range + $diff_size;
                $right_range12 = $right_range;
                $new_lb12 = {Maybe from the output of "shift_cmpl_bl_and_compt_prob_incr
(@@@$$) {...}"}; #  … or ($lb1 + $size_diff)  or ($rb1 - $size_S);
                $new_rb12 = $rb1;
                $left_range1 = $left_range;
                $right_range1 = $right_range + $size_S;
                $new_lb1 = $lb1;
                $new_rb1 = $new_lb12 -1;

            } else {
                $left_range12 = $left_range;
                $right_range12 = $right_range;
                $new_lb12 = $lb1;
                $new_rb12 = $rb1;
            }

        }

                # Determine the vertical state of the "merged" block ($bl12). #

        my ($br1, $u_or_d1) = @{$block_info[$bl1]}[$indx_br, $indx_u_or_d];
        my ($br2, $u_or_d2) = @{$block_info[$bl2]}[$indx_br, $indx_u_or_d];
        my ($br12, $u_or_d12) ;

        if ($u_or_d1 eq 'L') {
            if ($u_or_d2  eq 'L') {
                # This canNOT happen!! #
                {FAIL};
            } else { # if ($u_or_d2 eq 'U')
                if ($br1 == $node2pa→{$br2}) {
                    foreach my $ch (@{$node2ch→{$br1}}) {
                        if ($ch == $br2) { next; }
                        $br12 == $ch;
                        last;
                    }
                    $u_or_d12  = 'L';
```

```
            }
        }

    } else { # if ($u_or_d1 eq 'U')
        if ($u_or_d2  eq 'L') {
            if ($br2 == $node2pa->{$br1}) {
                foreach my $ch (@{$node2ch->{$br2}}) {
                    if ($ch == $br1) { next; }
                    $br12 = $ch;
                    last;
                }
                $u_or_d12 = 'L';
            }

        } else { # if ($u_or_d2 eq 'U')
            # $br1 and $br2 must be siblings. #
            my $pa1 = $node2pa->{$br1};
            if ($node2pa->{$br2} != $pa1) {
                my $eq_br1 = $eq_br{$br1};
                my $eq_br2 = $eq_br{$br2};
                if ((defined $eq_br1) and ($eq_br1 == $node2pa->{$br2})) {
                    foreach my $ch (@{$node2ch->{$eq_br1}}) {
                        if ($ch == $br2) { next; }
                        $br12 = $ch;
                        last;
                    }
                    $u_or_d12 = 'L'; # MODIFIED on 2019/01/26.
                    # $u_or_d12 = 'U';

                } elsif ((defined $eq_br2) and ($eq_br2 == $node2pa->{$br1})) {
                    foreach my $ch (@{$node2ch->{$eq_br2}}) {
                        if ($ch == $br1) { next; }
                        $br12 = $ch;
                        last;
                    }
                    $u_or_d12 = 'L'; # MODIFIED on 2019/01/26.
                    # $u_or_d12 = 'U';

                } else {
                    {FAIL};
                }

            } elsif ($pa1 == $top_node) {
                if (@{$node2ch->{$top_node}} == 3) {
                    foreach my $ch (@{$node2ch->{$top_node}}) {
                        if ($ch == $br1) { next; }
                        if ($ch == $br2) { next; }
                        $br12 = $ch;
                        last;
                    }
                    $u_or_d12 = 'L';
                }
            } else {
                $br12 = $pa1;
                $u_or_d12 = 'U';
            }
        }
    }
```

```
        unless (defined $br12) { # ADDED on 2019/01/27.
                {FAIL};
        }

#       ($lb1, $rb1) = @{$cp_bds_blocks[$bl1]}; # Re-compute the boundaries of the $bl1 th block.
#
#       my @null_clms = ();
#       for ($i=$lb1; $i <= $rb1; $i++) {
#           my $cnct_clm = join ('', @{$cp_set_columns0[$i]});
#           if ($cnct_clm eq $cnct_null_clm) { push @null_clms, $i; }
#       }

        my ($rmvd_bl, $remaining_bl) = ($size1 < $size2) ? ($bl1, $bl2) : ($bl2, $bl1);

        my ($left_range, $right_range) = ($remaining_bl == $bl1) ? ($left_range1, $right_range1) :
($left_range2, $right_range2);
        my @bds_remaining = ($remaining_bl == $bl1) ? ($new_lb1, $new_rb1) : ($new_lb2,
$new_rb2);

        => {Determine the rank of the "complementary-merged" block, $bl12, using $br12 and
$u_or_d12, as well as ($lb12, $rb12).};

    {create @new_set_columns0, immediately from @cp_set_columns0.}

    {create @new_bds_blocks0, by removing the $rmvd_bl th element of @cp_bds_bl_coords and
by replacing its $remaining_bl th element with \@bds_remaining, and inserting a new $bl12 th
element of [$new_lb12, $new_rb12].}
        {NOTE: if ($size1 == $size2), the $remaining_bl th element is also removed.}

    {create @new_inter_block_relations, by removing the row and column for $rmvd_bl (and
also by removing those for $remaining_bl if ($size1 == $size2)) from @cp_inter_block_relations,
        and inserting the row and column designated for new relations with $bl12.}

    {create @new_bds_bl_coords, by removing the $rmvd_bl th element of @cp_bds_bl_coords
and by replacing its $remaining_bl th element with [0, $left_range + $right_range], and inserting a
new $bl12 th element of [0, $left_range12, $right_range12].}
        {NOTE: if ($size1 == $size2), the $remaining_bl th element is also removed.}

    {create @new_org_bl_coords, by removing the $rmvd_bl th element of @cp_org_bl_coords
and by  replacing its $remaining_bl th element with $left_range, and inserting a new $bl12 th
element of $left_range12.}
        {NOTE: if ($size1 == $size2), the $remaining_bl th element is also removed.}

    {create @new_blocks_w_spec_lb0 and @new_blocks_w_spec_rb0, probably from scratch} #
ADDED on Jan 16, 2019.

    {create other necessary things as well, either from scratch or by using the corresponding ones
before the merge}

    {compute the indel component of the log-probability,
        either from scratch or by smartly using the result before the merge (refer to:  section 5-1 of
"blueprint1_ANEX.xxxx.pdf")}

    {perform the simultaneous "shift"-like moves of the resulting blocks}

    => {Output the results}; # ADDED on 2019/01/26.
}
```

END of "RESTARTED on Jan 2, 2019.

(vii-a) "Vertical-split" (into sibling blocks):

It would be better to use a window containing blocks fewer than considered by one.

Possible candidates may be short-listed using the result of pre-scanning for "ex-nihilo" candidates (in (v) above).

RESTARTED on Jan 3, 2019.

Here, **let's assume** that we already have **@set_to_be_vsplit1**, which lists the blocks to be ve rtically split.

```
foreach my $bl_old (@set_to_be_vsplit1) {
# for (my $bl = 0; $bl < $ub_bl; $bl++) {

    my $size_old = $block_sizes[$bl_old];
    my ($lb_old, $rb_old) = @{$bds_blocks0[$bl]_old};
    my ($lb_coord_old, $rb_coord_old) = @{$bds_bl_coords[$bl_old]};
    my $org_old  = $org_bl_coords[$bl];

    my ($br_old, $u_or_d_old) = @{$block_info[$bl_old]}[$indx_br, $indx_u_or_d];

    my @cp_set_columns0 = copy (@set_columns0);
    my @cp_bds_blocks0 = copy (@bds_blocks0);
    my $ln_prob_new_aln0 = $ln_prob_aln0;

    my @cp_bds_bl_coords = copy (@bds_bl_coords);
    my @cp_org_bl_coords = copy (@org_bl_coords);
    my @cp_inter_block_relations = copy (@inter_block_relations);


        # Determine the separating branches and the 'U'/'L' statuses of the new blocks. #

    my ($br1, $u_or_d1);
    my ($br2, $u_or_d2);

    if ($u_or_d_old eq 'L') {

        my $chidren = $node2ch->{$br_old};
        if ((defined $children) and (@{$children} == 2)) {
            ($br1, $br2) = @{$children};
            $u_or_d1 = $u_or_d2 = 'L';
        }

    } else {  # if ($u_or_d_old eq 'U')
        my $pa = $node2pa->{$br_old};
        my $sibs = $node2ch->{$pa};
        if ($pa == $top_node) {
            my $eq_br = $eq_br->{$br_old};
            if (defined $eq_br) {
                my $children_eq = $node2ch->{$eq_br};
                if ((defined $children_eq) and (@{$children_eq} == 2)) {
                    ($br1, $br2) = @{$children};
                    $u_or_d1 = $u_or_d2 = 'L';
                }
```

```
                } elsif (@{$sibs} == 3) {
                    foreach my $sib (@{$sibs}) {
                        if ($sib == $br_old) { next; }
                        if (defined $br1) {
                            $br2 = $sib;
                        } else {
                            $br1 = $sib;
                        }
                    }
                    $u_or_d1 = $u_or_d2 = 'L';
                }
            } elsif (@{$sibs} == 2) { # ($pa != $top_node)
                $br1 = $pa;
                $u_or_d1 = 'U';
                foreach my $sib (@{$sibs}) {
                    if ($sib != $br_old) { $br2 = $sib; }
                }
                $u_or_d2 = 'L';
            }

        } # END of "if ($u_or_d_old eq 'L') {....} else {...}".


        unless ((defined $br1) and (defined $br2)) { # Added on 2019/01/27.
            {FAIL};
            next;
        }

            # Determine the positions of the boundaries, as well as the coordinate frames, of the new blocks. #

        my ($lb1, $rb1) = my ($lb2, $rb2) = ($lb_old, $rb_old);
        my ($lb_coord1, $rb_coord1) = my ($lb_coord2, $rb_coord2) = ($lb_coord_old, $rb_coord_old);
        my $org1 = my $org2 = $org_old;


        => {Determine the ranks (after removing $bl_old) of the new blocks, $bl1 & $bl2, using ($br1, $u_or_d1) & ($br2, $u_or_d2), as well as ($lb1, $rb1) & ($lb2, $rb2).};

        # Create essential data sets for representing the new set of gap-blocks in the local alignment. #

    {create @new_set_columns0, immediately from @cp_set_columns0, by vertically splitting the old block ($bl_old) into new blocks ($bl1 & $bl2).}

    {create @new_inter_block_relations, by removing the row and column for the old block ($bl_old) from @cp_inter_block_relations,
        and inserting the rows and columns designated for the new blocks ($bl1 & $bl2).}

     {create @new_bds_blocks0, by removing the $bl_old th element of @cp_bds_bl_coords and by inserting new $bl1 th & $bl2 th elements of [$lb1, $rb1]  & [$lb2, $rb2].}

        # Move $bl2 by one column, to finish the preparation of the new block-set. #

        my $if_success = 0;
        if ( ($org2 < $rb_coord2) and ($rb2 < $right_end_laln) ) {
```

{shift $bl2 to the right (maybe using "**shift_bl_and_compt_prob_incr (@@@$$) {...}**")
, while computing the log-probability increment (= $incr_ln_prob) }; # This should accompany **a necessary change of [$lb2, $rb2]**.

$ln_prob_new_aln0 += $incr_ln_prob;
$if_success = 1;
$org2++;

} elsif (($lb_coord2 < $org2) and ($left_end_laln < $lb2)) {

{shift $bl2 to the left (maybe using "**shift_bl_and_compt_prob_incr (@@@$$) {...}**")
, while computing the log-probability increment (= $incr_ln_prob) }; # This should accompany **a necessary change of [$lb2, $rb2]**.

$ln_prob_new_aln0 += $incr_ln_prob;
$if_success = 1;
$org2--;
}

unless ($if_success) { # ADDED on Jan 16, 2019.
{FAIL};
next;
}

Create auxiliary data sets for the new set of gap-blocks (in the local alignment).

{create **@new_bds_bl_coords**, by removing the $bl_old th element of @cp_bds_bl_coords and by inserting new $bl1 th & $bl2 th elements of [$lb_coord1, $rb_coord1] & [$lb_coord2, $rb_coord2].}

{create **@new_org_bl_coords**, by removing the $bl_old th element of @cp_org_bl_coords and by inserting new $bl1 th & $bl2 th elements of $org1 & $org2.}

{create **@new_blocks_w_spec_lb0** and **@new_blocks_w_spec_rb0**, probably from scratch} # ADDED on Jan 16, 2019.

{create **other necessary things** as well, either from scratch or by using the corresponding ones before the split}

{compute the indel component of the log-probability,
either from scratch or by smartly using the result before the merge (refer to: **section 5-1** of "**blueprint1_ANEX.xxxx.pdf**")}

{perform the **simultaneous "shift"-like moves** of the resulting blocks}
(NOTE: ***Prohibit*** the perfect alignment of $bl1 and $bl2, by imposing some conditions on the exploration of the new coordinate space.) # ADDED on Jan 5, 2019.

=> {Output the results}; # ADDED on 2019/01/26.
}

END of "RESTARTED on Jan 3, 2019."

(vii-b) "Vertical-split" (into complementary-sibling blocks (i.e., sibling sequence-blocks)**):**

```
        # It would be better to use a window containing blocks fewer than considered by one.

        # Possible candidates may be short-listed using the result of pre-scanning for "ex-nihilo" candidates (in (v) above). #


                # RESTARTED on Jan 5, 2019. #

# Here, let's assume that we already have @set_to_be_vsplit2, which lists the blocks whose complement to be vertically split. #

foreach my $bl_old (@set_to_be_vsplit2) {
# for (my $bl = 0; $bl < $ub_bl; $bl++) {

        my $size_old = $block_sizes[$bl_old];
        my ($lb_old, $rb_old) = @{$bds_blocks0[$bl]_old};
        my ($lb_coord_old, $rb_coord_old) = @{$bds_bl_coords[$bl_old]};
        my $org_old  = $org_bl_coords[$bl];

        my ($br_old, $u_or_d_old) = @{$block_info[$bl_old]}[$indx_br, $indx_u_or_d];

        my @cp_set_columns0 = copy (@set_columns0);
        my @cp_bds_blocks0 = copy (@bds_blocks0);
        my $ln_prob_new_aln0 = $ln_prob_aln0;

        my @cp_bds_bl_coords = copy (@bds_bl_coords);
        my @cp_org_bl_coords = copy (@org_bl_coords);
        my @cp_inter_block_relations = copy (@inter_block_relations);


                # Determine the separating branches and the 'U'/'L' statuses of the new blocks. #

        my ($br1, $u_or_d1);
        my ($br2, $u_or_d2);


        if ($u_or_d_old eq 'U') { # Sequence-block is on the lower-side of $br_old.

            my $chidren = $node2ch→{$br_old};
            if ((defined $children) and (@{$children} == 2)) {
                ($br1, $br2) = @{$children};
                $u_or_d1 = $u_or_d2 = 'U';
            }

        } else {  # if ($u_or_d_old eq 'L')  # Sequence-block is on the upper-side of $br_old.

            my $pa = $node2pa→{$br_old};
            my $sibs = $node2ch→{$pa};
            if ($pa == $top_node) {
                my $eq_br = $br2eq→{$br_old};
                if (defined $eq_br) {
                    my $children_eq = $node2ch→{$eq_br};
                    if ((defined $children_eq) and (@{$children_eq} == 2)) {
                        ($br1, $br2) = @{$children};
                        $u_or_d1 = $u_or_d2 = 'U';
                    }
                } elsif (@{$sibs} == 3) {
                    foreach my $sib (@{$sibs}) {
```

```
                        if ($sib == $br_old) { next; }
                        if (defined $br1) {
                            $br2 = $sib;
                        } else {
                            $br1 = $sib;
                        }
                    }
                    $u_or_d1 = $u_or_d2 = 'U';
                }
            } elsif (@{$sibs} == 2) { # ($pa != $top_node)

                        # MODIFIED on 2019/02/08. #
                my $eq_pa = $br2eq->{$pa};
                if (defined $eq_pa) {
                    $br1 = $eq_pa;
                    $u_or_d1 = 'U';
                } else {
                    $br1 = $pa;
                    $u_or_d1 = 'L';
                }
#               $br1 = $pa;
#               $u_or_d1 = 'L';
                        # End of "MODIFIED on 2019/02/08." #

                foreach my $sib (@{$sibs}) {
                    if ($sib != $br_old) { $br2 = $sib; }
                }
                $u_or_d2 = 'U';
            }

        } # END of "if ($u_or_d_old eq 'U') {....} else {...}".

        unless ((defined $br1) and (defined $br2)) { # Added on 2019/01/27.
                {FAIL};
                next;
        }

                # Determine the positions of the boundaries, as well as the coordinate frames, of the new blocks. #

        my ($lb1, $rb1) = my ($lb2, $rb2) = ($lb_old, $rb_old);
        my ($lb_coord1, $rb_coord1) = my ($lb_coord2, $rb_coord2) = ($lb_coord_old, $rb_coord_old);
        my $org1 = my $org2 = $org_old;


        => {Determine the ranks (after removing $bl_old) of the new blocks, $bl1 & $bl2, using ($br1, $u_or_d1) & ($br2, $u_or_d2), as well as ($lb1, $rb1) & ($lb2, $rb2).};

        # Create @new_set_columns0, immediately from @cp_set_columns0, by vertically splitting the old sequence-block ($bl_old) into new sequence-blocks ($bl1 & $bl2), as follows. #

    my @new_set_columns0 = ($lb_old > 0) ? @cp_set_columns0[0 .. $lb_old-1] : ();
    my @new_clms1 = my @new_clms2 = ();

    my @affected_by_seqblk1 = (sequences (or classes) on the upper/lower-side of $br1 if ($u_or_d1 eq 'L'/'U'));
    my $cnct_null_clm = join (':', @null_column);
```

```
#      my $ct_null_clms1 = my $ct_null_clms2 = 0;
       my @null_clms1  = my @null_clms2 = ( );

      for (my $c = $lb_old; $c <= $rb_old; $c++) {

                    # Split the column into two parts, one constituting block 1 and the other constituting
block 2. #
            my @old_clm = @{$cp_set_columns0[$c]};
            my $delta_ln_prob_old = ln_prob (@old_clm);
            $ln_prob_new_aln0 -= $delta_ln_prob_old;

            my @new_clm1 = copy (@null_column);
            my @new_clm2 = my @old_clm;

            foreach my $indx (@affected_by_seqblk1) {
                  my $tmp = $new_clm2[$indx];
                  $new_clm2[$indx] = $new_clm1[$indx];
                  $new_clm1[$indx] = $tmp;
            }

            my $cnct_new_clm1 = join (':', @new_clm1);
            my $cnct_new_clm2 = join (':', @new_clm2);

            if ($cnct_new_clm1 eq $cnct_null_clm) {
#                 $ct_null_clms1++;
                  push @null_clms1, $c;
            } else {
                 push @new_clms1, \@new_clm1;
                 my $delta_ln_prob1 = ln_prob (@new_clm1);
                 $ln_prob_new_aln0 += $delta_ln_prob1;
            }

            if ($cnct_new_clm1 eq $cnct_null_clm) {
#                 $ct_null_clms2++;
                  push @null_clms2, $c;
            } else {
                  push @new_clms2, \@new_clm2;
                  my $delta_ln_prob2 = ln_prob (@new_clm2);
                  $ln_prob_new_aln0 += $delta_ln_prob2;
            }
      }
      my $ct_null_clms1 = scalar (@null_clms1);
      my $ct_null_clms2 = scalar (@null_clms2);
      my $ct_null_clms = $ct_null_clms1 + $ct_null_clms2;

      push @new_set_columns0, @new_clms1, @new_clms2; # Place block 1 on the left, and
block 2 on the right. #

      if ($rb_old < $#cp_set_columns0) {
            push @new_set_columns0, @cp_set_columns0[$rb_old + 1 .. $#cp_set_columns0];
      }

      my $ct_new_clms1 = scalar (@new_clms1);
      my $ct_new_clms2 = scalar (@new_clms2);

            # Create essential data sets for representing the new set of gap-blocks in the local
alignment. #
```

{create **@new_inter_block_relations**, by removing the row and column for the old block ($bl_old) from @cp_inter_block_relations,
and inserting the rows and columns designated for the new blocks ($bl1 & $bl2).}

{create **@new_bds_blocks0**, by removing the $bl_old th element of @cp_bds_bl_coords and by inserting new $bl1 th & $bl2 th elements of [$lb1, $rb1] & [$lb2, $rb2].}

=> {Change [$lb1, $rb1] & [$lb2, $rb2] to
[$lb_new = $lb_old, $rb1_new = $lb_new + $ct_new_clms1 -1]
& [$lb2_new = $rb1_new + 1, $rb2_new = $lb2_new + $ct_new_clms2 -1],
AND Increase all other positions in **@new_bds_blocks0**
(a) by ($rb2_new - $rb2_old) $if ($pos > $rb_old),
(b) by (- #{columns (< $pos) in @null_clms1}) if ($lb_old <= $pos <= $rb_old) and if the gap-block belongs to @new_clms1,
(c) by ($rb1_new - $rb1_old - #{columns (< $pos) in @null_clms2}) if ($lb_old <= $pos <= $rb_old) and if the gap-block belongs to @new_clm2.
}

Create auxiliary data sets for the new set of gap-blocks (in the local alignment).

{create **@new_bds_bl_coords**, by removing the $bl_old th element of @cp_bds_bl_coords and by inserting new $bl1 th & $bl2 th elements of [$lb_coord1, $rb_coord1] & [$lb_coord2, $rb_coord2].}

{create **@new_org_bl_coords**, by removing the $bl_old th element of @cp_org_bl_coords and by inserting new $bl1 th & $bl2 th elements of $org1 & $org2.}

{create **@new_blocks_w_spec_lb0** and **@new_blocks_w_spec_rb0**, probably from scratch} # ADDED on Jan 16, 2019.

{create **other necessary things** as well, either from scratch or by using the corresponding ones before the split}

{compute the indel component of the log-probability,
either from scratch or by smartly using the result before the merge (refer to: **section 5-1** of "**blueprint1_ANEX.xxxx.pdf**")}

{perform the **simultaneous "shift"-like moves** of the resulting blocks}

=> {Output the results}; # ADDED on 2019/01/26.
}

SM-6. Examining the effects of alignment changes including topological changes involving both "split" and "merger" of gap-blocks.

(iii-vi-a) (Horizontal) Merge + Split (same type):

 # Here, we will merge & split only pairs of neighboring blocks.

```
my @to_be_merged_split1 = ();

for (my $bl2=1; $bl2 < $ub_bl; $bl2++) { # Modified on 2019/01/26.
# for (my $bl2=1; $bl2 < $B; $bl2++) {

    my $bl1 = $bl2-1;
```

```
    my $rel = $inter_block_relations[$bl1]→[$bl2];
    unless ($rel eq '=') { next; }

    my ($dist1, $dist2) = inter_block_distance ($bl1, $bl2, @bds_blocks0,
@inter_block_relations); # (See Appendix G I.) #
    my $dist = $dist2; # MODIFIED on Jan 13, 2019.
 #    my $dist  = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations); # (See
Appendix G.) #

    if ($dist <= $THRSH_DIST_MERGE_SPLIT1) { push @to_be_merged_split1, [$bl1, $bl2,
$dist];  }  # It would be appropriate to set  $THRSH_DIST_MERGE_SPLIT1 =
$THRSH_DIST_MERGE1 .
}


foreach my $sbjct_pair (@to_be_merged_split1) { # Outer foreach-loop (over the pair of blocks). #

        my ($bl1, $bl2, $dist) = @{$sbjct_pair};

        my @cp_set_columns0 = copy (@set_columns0);
        my @cp_bds_blocks0 = copy (@bds_blocks0);
        my $ln_prob_new_aln0 = $ln_prob_aln0;

        my @cp_bds_bl_coords = copy (@bds_bl_coords);
        my @cp_org_bl_coords = copy (@org_bl_coords);
        my @cp_inter_block_relations = copy (@inter_block_relations);

        my ($lb1, $rb1) = @{$cp_bds_blocks0[$bl1]};
        my ($lb2, $rb2) = @{$cp_bds_blocks0[$bl2]};

        my ($lb_coord1, $rb_coord1) = @{$bds_bl_coords[$bl1]};
        my ($lb_coord2, $rb_coord2) = @{$bds_bl_coords[$bl2]};
        my ($org1, $org2) = @org_bl_coords[$bl1, $bl2];


        my $shift1 = int ($dist/2);
        my $shift2 = $dist - $shift1;

        my ($left_range, $right_range);

        if ($lb1 < $lb2) {

#           $left_range   = $org1 - $lb_coord1 + $shift1;
#           $right_range = $rb_coord2 - $org2 + $shift2;
            $org1 += $shift1;
            $org2 -= $shift2;

            for (my $i=0; $i < $shift1; $i++) {

                {shift $bl1 to the right (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) }

                $ln_prob_new_aln0 += $incr_ln_prob;
            }

            for (my $i=0; $i < $shft2; $i++) {
```

{shift $bl2 to the left (maybe using "**shift_bl_and_compt_prob_incr (@@@$$) {...}**")
, while computing the log-probability increment (= $incr_ln_prob) }

$ln_prob_new_aln0 += $incr_ln_prob;
}

} else { # if ($lb2 < $lb1)

$left_range = $org2 - $lb_coord2 + $shift2;
$right_range = $rb_coord1 - $org1 + $shift1;
$org1 -= $shift1;
$org2 += $shift2;

for (my $i=0; $i < $shift1; $i++) {

{shift $bl1 to the left (maybe using "**shift_bl_and_compt_prob_incr (@@@$$) {...}**")
, while computing the log-probability increment (= $incr_ln_prob) }

$ln_prob_new_aln0 += $incr_ln_prob;
}

for (my $i=0; $i < $shft2; $i++) {
{shift $bl2 to the right (maybe using "**shift_bl_and_compt_prob_incr (@@@$$) {...}**")
, while computing the log-probability increment (= $incr_ln_prob) }

$ln_prob_new_aln0 += $incr_ln_prob;
}
}

Up to here, the processes are essentially identical to those for **(iii-a) "Merge" (same type).**

{As long as #{null columns} = 0, **@new_set_columns00** = @cp_set_columns0 after the "shifts" in the "if {} else {}" blocks above.}
{**@new_bds_blocks00** = @cp_bds_blocks after the "shifts" in the "if {} else {}" blocks above.}

my ($size1, $size2) = @block_sizes[$bl1, $bl2];
my $sum_size = $size1 + $size2;

{create **@new_inter_block_relations** = @cp_inter_block_relations .} # This will NOT change while merge & split (same type) are examined !! #

{create **@new_bds_bl_coords0** = @cp_bds_bl_coords .}

{create **@new_org_bl_coords0**, by replacing the $bl1 th and $bl2 th elements of @cp_org_bl_coords with the new $org1 and new $org2, respectively.}

{create **@new_blocks_w_spec_lb0** and **@new_blocks_w_spec_rb0**, probably from scratch} # ADDED on Jan 16, 2019.

{create Initialize **other necessary things** as well, either from scratch or by using the corresponding ones before the merge}

```
                # Initialize the new pair of blocks. #

        my ($new_size1, $new_size2) = ($sum_size, 0);

    if ($lb1 < $lb2) {
        $new_bds_blocks00[$bl1]→[1] = $rb2;
        $new_bds_blocks00[$bl2]→[0] = $rb2+1;
        $new_bds_bl_coords0[$bl1]→[1] -= $size2;

    } else { # if ($lb2 < $lb1)
        $new_bds_blocks00[$bl1]→[0] = $lb2;
        $new_bds_blocks00[$bl2]→[1] = $lb2-1;
        $new_bds_bl_coords0[$bl1]→[1] -= $size2;
        $new_org_bl_coords0[$bl1] -= $size2;
    }

                # END of "RESTARTED on Jan 5, 2019." #

                # Move the boundary between the blocks in the pair. #

                # RESTARTED on Jan 6, 2019. #

    my @pos_to_be_used = ();
    for ($left_end_laln .. $right_end_laln) { push @pos_to_be_used, 1; }
    my $new_rel_w_bl1 = $new_inter_block_relations[$bl1];
    for (my $bl3 = 0; $bl3 < $ub_bl; $bl3++) {
        if (($bl3 == $bl1) or ($bl3 == $bl2)) { next; }
        my $rel = $new_rel_w_bl1→[$bl3];
        if (($rel eq '<') or ($rel eq '<(pa)') or (($rel eq '=') and ($bl3 < $bl1))
                # or ($rel eq 'ONN') or ($rel eq 'ONCS')
            ) { # Added '<(pa)' on Jan 18, 2019.
             my ($lb3, $rb3) = @{$new_bds_blocks00[$bl3]};
             for (my $c = $lb3; $c <= $rb3; $c++) { $pos_to_be_used[$c] = 0; }
        }
    }

    my $pos_bd;
    if ($lb1 < $lb2) {
        $pos_bd = $rb2 + 1;
        while (($pos_bd <= $right_end_laln) and ($pos_to_be_used[$pos_bd] == 0)) { $pos_bd+
+; }
        if ($pos_bd > $right_end_laln) {
                {FAIL};
        }
    } else {
        $pos_bd = $lb2 – 1;
        while (($pos_bd >= $left_end_laln) and ($pos_to_be_used[$pos_bd] == 0)) { $pos_bd--; }
        if ($pos_bd < $left_end_laln) {
                {FAIL};
        }
    }

    my @new_bds_blocks0 = copy (@new_bds_blocks00);
    my @new_set_columns0 = copy (@new_set_columns00);
```

```
    my @affected_by_bl1 = {retrieve from somewhere};

    for (my $ss=1; $ss < $sum_size; $ss++) { # Middle for-loop (over the sizes of the new blocks).
#

        $new_size1--;
        $new_size2++;

#       my @new_bds_blocks0 = copy (@new_bds_blocks00);

        my $pos_bd_prev = $pos_bd;

        if ($lb1 < $lb2) { # Move the boundary to the left "by one site". #
#           $pos_bd --;
#           while ($pos_to_be_used[$pos_bd] == 0) { $pos_bd--; }
            $pos_bd = $new_bds_blocks0[$bl1]→[1];
            my $new_lb2  = $pos_bd_prev;
            my $new_rb1 = $pos_bd -1;
            while ($pos_to_be_used[$new_rb1] == 0) { $new_rb1--; }

            $new_bds_blocks0[$bl2]→[0] = $new_lb2;
            $new_bds_blocks0[$bl1]→[1] = $new_rb1;

            $new_bds_bl_coords0[$bl1]→[1]++;

        } else { #  if ($lb2 < $lb1) # Move the boundary to the right "by one site".
#           $pos_bd++;
#           while ($pos_to_be_used[$pos_bd] == 0) { $pos_bd++; }
            $pos_bd = $new_bds_blocks0[$bl1]→[0];
            my $new_rb2 = $pos_bd_prev;
            my $new_lb1 = $pos_bd + 1;
            while ($pos_to_be_used[$new_lb1] == 0) { $new_lb1++; }

            $new_bds_blocks0[$bl2]→[1] = $new_rb2;
            $new_bds_blocks0[$bl1]→[0] = $new_lb1;

            $new_bds_bl_coords0[$bl1]→[1]++;
            $new_org_bl_coords0[$bl1]++;
        }

        {Use @new_inter_block_relations as it is.}

        {Modify @new_blocks_w_spec_lb0 and @new_blocks_w_spec_rb0, accordingly} #
ADDED on Jan 16, 2019.

        {Modify other necessary things accordingly.}

        my $clm_bd_curr = $new_set_columns0[$pos_bd];
        my $clm_bd_prev = $new_set_columns0[$pos_bd_prev];

        my $ln_prob_clm_curr_bf = ln_prob (@{$clm_bd_curr});
        my $ln_prob_clm_prev_bf = ln_prob (@{$clm_bd_prev});

                # Modify the alignment by swapping, between the new and old boundaries, the sites
in sequences (or classes) affected by $bl1. #
        foreach my $indx (@affected_by_bl1) {
            my $tmp = $clm_bd_curr→[$indx];
            $clm_bd_curr→[$indx] = $clm_bd_prev→[$indx];
```

```
            $clm_bd_prev→[$indx] = $tmp;
        }

        my $ln_prob_clm_curr_af = ln_prob (@{$clm_bd_curr});
        my $ln_prob_clm_prev_af = ln_prob (@{$clm_bd_prev});

        $ln_prob_new_aln0 += $ln_prob_clm_curr_af + $ln_prob_clm_prev_af -
$ln_prob_clm_curr_bf - $ln_prob_clm_prev_bf;


        if (($new_size1 == $size1) or ($new_size1 == $size2)) { # Skip if the new pair becomes
equivalent to the old pair. #
            next;
        }

        {compute the indel component of the log-probability,
        either from scratch or by smartly using the result before the merge (refer to:  section 5-1 of
"blueprint1_ANEX.xxxx.pdf")}

        {perform the simultaneous "shift"-like moves of the resulting blocks}

        => {Output the results}; # ADDED on 2019/01/26.

    }  # End of the middle for-loop (over the sizes of the new blocks). #

} # END of the outer foreach-loop (over the pair of blocks). #
```

(iii-vi-b) (Horizontal) Merge (or purge) + Split (or ex-nihilo) (complementary types) (including incomplete merge (or purge))**:**

```
my @to_be_merged_split2 = ();

for (my $bl1=0; $bl1 < $ub_bl; $bl1++) { # Modified on 2019/01/26.
# for (my $bl1=0; $bl1 < $B; $bl1++) {
    my $rels_w_bl1 = $inter_block_relations[$bl1];

    for (my $bl2 = $bl1+1; $bl2 < $ub_bl; $bl2++) { # Modified on 2019/01/27.
#     for (my $bl2 = $bl1+1; $bl2 < $B; $bl2++) {
        my $rel = $rels_w_bl1→[$bl2];
        unless ($rel eq 'Cp') { next; }

        my ($dist1, $dist2) = inter_block_distance ($bl1, $bl2, @bds_blocks0,
@inter_block_relations); # This subroutine measures the distance between $bl1 and $bl2, while
taking account of the blocks between the two blocks. (See Appendix G I.) #
        my $dist = $dist2; # MODIFIED on Jan 13, 2019.
#       my $dist  = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations);

        if ($dist <= $THRSH_DIST_MERGE_SPLIT2) { push @to_be_merged_split2, [$bl1, $bl2,
$dist];  }  # It would be fine to set $THRSH_DIST_MERGE_SPLIT2 =
$THRSH_DIST_MERGE2. If you prefer, however, you could set a smaller value. #
    }
}


my $if_pure_split = 0; # ADDED on Jan 8, 2018.
```

```
foreach my $sbjct_pair (@to_be_merged_split2) {  # Outer foreach-loop (over subject pairs). #

        my ($bl1, $bl2, $dist) = @{$sbjct_pair};

        my @new_set_columns0 = copy (@set_columns0);
        my @new_bds_blocks0 = copy (@bds_blocks0);
        my $ln_prob_new_aln0 = $ln_prob_aln0;

        my @new_bds_bl_coords = copy (@bds_bl_coords);
        my @new_org_bl_coords = copy (@org_bl_coords);
        my @new_inter_block_relations = copy (@inter_block_relations);

        {Copy other necessary things as well.}


        my ($lb01, $rb01) = @{$new_bds_blocks0[$bl1]};
        my ($lb02, $rb02) = @{$new_bds_blocks0[$bl2]};

#       my ($lb_coord1, $rb_coord1) = @{$bds_bl_coords[$bl1]};
#       my ($lb_coord2, $rb_coord2) = @{$bds_bl_coords[$bl2]};
#       my ($org1, $org2) = @org_bl_coords[$bl1, $bl2];

        my ($size01, $size02) = @block_sizes[$bl1, $bl2];

        my ($size0_S, $size0_L) = ($size01 < $size02) ? ($size01, $size02) : ($size02, $size01) ;

#       my ($left_range, $right_range);

        if ($lb01 < $lb02) {

#           $left_range   = $org1 - $lb_coord1;
#           $right_range = $rb_coord2 - $org2 + $dist;

            for (my $i=0; $i < $dist; $i++) { # 1st middle for-loop (over shifts). #

                {shift $bl2 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) } # Modify
@new_set_columns0 & @new_bds_blocks0, as well as other things, accordingly.

                $ln_prob_new_aln0 += $incr_ln_prob;

            } # End of the 1st middle for-loop (over shifts). #

            $new_org_bl_coords[$bl2] -= $dist;

        } else { # if ($lb02 < $lb01)

#           $left_range   = $org2 - $lb_coord2 + $dist;
#           $right_range = $rb_coord1 - $org1;

            for (my $i=0; $i < $dist; $i++) { # 2nd  middle for-loop (over shifts). #

                {shift $bl2 to the right (maybe using "shift_bl_and_compt_prob_incr (@@@$$)
{...}")
                , while computing the log-probability increment (= $incr_ln_prob) } # Modify
@new_set_columns0 & @new_bds_blocks0, as well as other things, accordingly.
```

```
                $ln_prob_new_aln0 += $incr_ln_prob;

            } # End of the 2nd  middle for-loop (over shifts). #

            $new_org_bl_coords[$bl2] += $dist;

        }

                # Up to here, the processes are mostly equivalent to those for (iii-b) "Merge" (complementary types), but with a few differences.
                # (For example, the "shifts" are performed here until $bl1 and $bl2 become immediately adjacent, but NOT until the shorter block is completely "merge"d with the longer one.)

                # Copy the necessary data sets once again.
                # (They will be used in the 2nd half of the following processes.) #

        my @new_set_columns0_2 = copy (@new_set_columns0);
        my @new_bds_blocks0_2 = copy (@new_bds_blocks0);
        my $ln_prob_new_aln0_2 = $ln_prob_new_aln0;

        my @new_bds_bl_coords2 = copy (@new_bds_bl_coords);
        my @new_org_bl_coords2 = copy (@new_org_bl_coords);
        {Copy other necessary things as well.}

        my @affected_by_bl1 = {retrieved from somewhere};
        # my @affected_by_bl2 = {retrieved from somewhere};

                # First, merge the two blocks column by column. #

        my ($size1, $size2) = ($size01, $size02);
        my @sites_in_bl1 = (a list of sites belonging to $bl1 in @new_bl_coords0);
        my @sites_in_bl2 = (a list of sites belonging to $bl2 in @new_bl_coords0);

        for (my $delta 1= 1; $delta1 < $size01; $delta1++) { # 3rd for-loop (over merged columns). #

            $size1--;
            $size2--;

                # END of "RESTARTED on Jan 6, 2019". #

                # RESTARTED on Jan 7, 2019. #

            my ($sbjct_1, $sbjct_2);
            if ($lb01 < $lb02) {
                # Merge the leftmost column of $bl1 and that of $bl2.

                $sbjct_1 = shift @sites_in_bl1;
                $sbjct_2 = shift @sites_in_bl2;

                for (my $i=0; $i < @sites_in_bl2; $i++) { $sites_in_bl2[$i]--; }

                $new_bds_blocks0[$bl1]→[0] = $sites_in_bl1[0];
                $new_bds_blocks0[$bl2]→[0] = $sites_in_bl2[0];
                $new_bds_blocks0[$bl2]→[1]--;
                foreach my $pos (other positions) {
                    if ($pos == $sbjct_2) {
                        $pos = $sbjct_1;
```

```
                } elsif ($pos > $sbjct_2) {
                    $pos--;
                }
            }

            $new_org_bl_coords[$bl1]++;

        } else { # if ($lb02 < $lb01)
            # Merge the rightmost column of $bl2 and that of $bl1.

            $sbjct_1 = pop @sites_in_bl1;
            $sbjct_2 = pop @sites_in_bl2;

            for (my $i=0; $i < @sites_in_bl1; $i++) { $sites_in_bl1[$i]--; }

            $new_bds_blocks0[$bl1]→[0]--;
            $new_bds_blocks0[$bl1]→[1] = $sites_in_bl1[$#sites_in_bl1];
            $new_bds_blocks0[$bl2]→[1] = $sites_in_bl2[$#sites_in_bl2];
            foreach my $pos (other positions) {
                if ($pos == $sbjct_2) {
                    $pos = $sbjct_1;
                } elsif ($pos > $sbjct_2) {
                    $pos--;
                }
            }

            $new_org_bl_coords[$bl1]--;
        }

            # Swap the sequences (or classes) affected by $bl1. #

        my $sbjct_clm_1 = $new_set_columns0[$sbjct_1];
        my $sbjct_clm_2 = $new_set_columns0[$sbjct_2];

        my $ln_prob_sbjct1_bf = ln_prob (@{$sbjct_clm_1});
        my $ln_prob_sbjct2_bf = ln_prob (@{$sbjct_clm_2});

        foreach my $k (@affected_by_bl1) {
            my $tmp = $sbjct_clm_1→[$k];
            $sbjct_clm_1→[$k] = $sbjct_clm_2→[$k];
            $sbjct_clm_2→[$k] = $tmp;
        }

        my $ln_prob_sbjct1_af = ln_prob (@{$sbjct_clm_1});
#       my $ln_prob_sbjct2_af = ln_prob (@{$sbjct_clm_2});

        $ln_prob_new_aln0 += $ln_prob_sbjct1_af - $ln_prob_sbjct1_bf - $ln_prob_sbjct2_bf ;
#       $ln_prob_new_aln0 += $ln_prob_sbjct1_af + $ln_prob_sbjct2_af - $ln_prob_sbjct1_bf -
$ln_prob_sbjct2_bf ;


            # Update @new_set_columns0.
            # (Always remove $sbjct_clm_2 !!)

        my @newnew_set_columns0 = ($sbjct_2>0) ? @new_set_columns0[0 .. $sbjct_2 -1] :
();
        if ($sbjct_2 < $#new_set_columns0) { push @newnew_set_columns0,
@new_set_columns0[$sbjct_2+1 .. $#new_set_columns0]; }
```

@new_set_columns0 = @newnew_set_columns0;

{Create **@new_blocks_w_spec_lb0** and **@new_blocks_w_spec_rb0**, probably from scratch} # ADDED on Jan 16, 2019.

{Update **other necessary things** as well.}

{Compute the indel component of the log-probability,
either from scratch or by smartly using the result before the merge (refer to: **section 5-1** of "**blueprint1_ANEX.xxxx.pdf**")}

{Perform the **simultaneous "shift"-like moves** of the *resulting* blocks}

=> {Output the results}; # ADDED on 2019/01/27.

} # End of the 3rd for-loop (over merged columns). #

 # Reset some important data sets. #

@new_set_columns0 = copy (@new_set_columns0_2);
@new_bds_blocks0 = copy (@new_bds_blocks0_2);
@new_bds_bl_coords = copy (@new_bds_bl_coords2);
@new_org_bl_coords = copy (@new_org_bl_coords2);

 # Second, split the flanking columns, one by one,
 # and merge them to the subject blocks.

{Create **@set_left_flanking_clms** and **@set_right_flanking_clms**,
in a way similar to the same processes in **(iv-b) "Split" (into blocks of complementary types)};

my $ct_l_flank = @set_left_flanking_clms;
my $ct_r_flank = @set_right_flanking_clms;
my **$margin** = ($ct_l_frank > $ct_r_frank) ? $ct_l_frank : $ct_r_frank;
my **$if_on_the_left** = ($ct_l_flank > $ct_r_flank) ? 1 : 0;

my $ct_to_be_split = ($THRSH_SIZE_SPLIT2 > $size0_S) ?
$THRSH_SIZE_SPLIT2 - $size0_S : 0; # **$THRSH_SIZE_SPLIT2** came from **(iv-b) "Split" (into blocks of complementary types).**

Another choice of the above condition would be:
my $ct_to_be_split = $THRSH_SIZE_MERGE_SPLIT2 (fixed).

my **$ub_delta2** = ($margin < $ct_to_be_split) ? $margin : $ct_to_be_split ;

($size1, $size2) = ($size01, $size02);

my **($bl_cmpl, $bl_sbj)** = (($if_on_the_left and ($lb01 < $lb02)) or (!($if_on_the_left) and ($lb02 < $lb01)) ? ($bl1, $bl2) : ($bl2, $bl1) ;

my ($br_sbj, $u_or_d_sbj) = @{$block_info[$bl_sbj]}[$indx_br, $indx_u_or_d];

for (my $delta2 = 1; $delta2 <= $ub_delta2; $delta2++) { # 4th for-loop (over columns to be split).

```
            $size1++;
            $size2++;

            my $to_be_split = ($if_on_the_left) ? (pop @set_left_flanking_clms) : (shift
@set_right_flanking_clms) ;

            { split the $to_be_split th column (in the original local alignment) at the branch
$br_sbj,
                and move the $u_or_d_sbj side  to the (($if_on_the_left) ? right  : left),
                and "merge" it with the $bl_sbj th block,
                 and also "merge" its complement with the $bl_cmpl th block.};  # See Appendix H
J, with @new_set_columns0_2 playing the role of @set_columns0.

                    # Modify the coordinate ranges. #
                if ($if_on_the_left) {
                    # The $new_bl th block does NOT change its range.
                    # The $bl_cmpl th block moves its coordinate origin by one to the left.
                  $new_org_bl_coords0[$bl_cmpl]--;
                } else {
                    # The $new_bl th block does NOT change its range.
                    # The $bl_cmpl th block moves its  coordinate origin by one to the right.
                  $new_org_bl_coords[$bl_cmpl]++;
                }

                # MODIFIED on Jan 15, 2019. (6) #

                => {Create @new_blocks_w_spec_lb0 and @new_blocks_w_spec_rb0, either
from scratch or by using the old ones};  # Further MODIFIED on Jan 16, 2019.
#               => {Update @new_blocks_w_spec_lb and @new_blocks_w_spec_rb, similarly to
@new_set_columns0 and in coordination with @new_bds_blocks0};  # OBSOLETE as of Jan 16,
2019.

                # OBSOLETE as of Jan 15, 2019. (6) #
#               => {Re-sort the left-bounds and the right-bounds
#                       to update @new_lb_sorted_set and @new_rb_sorted_set, respectively};

#               => {Update @new_orders_lb and @new_orders_rb};
                # END of "OBSOLETE as of Jan 15, 2019. (6)" #
                # END of "MODIFIED on Jan 15, 2019. (6)" #

                {Modify other important data sets accordingly};

                 =>  {Perform the simultaneous "shift"-like moves of the resulting blocks}

                # END of "RESTARTED on Jan 7, 2019". #

                => {Output the results}; # ADDED on 2019/01/26.

        } # End of the 4th for-loop (over columns to be split).

#       ($lb1, $rb1) = @{$cp_bds_blocks[$bl1]}; # Re-compute the boundaries of the $bl1 th block.
#
#       my @null_clms = ();
#       for ($i=$lb1; $i <= $rb1; $i++) {
#           my $cnct_clm = join ('', @{$cp_set_columns0[$i]});
#           if ($cnct_clm eq $cnct_null_clm) { push @null_clms, $i; }
#       }
```

```
#
#         my ($rmvd_bl, $remaining_bl) = ($size1 < $size2) ? ($bl1, $bl2) : ($bl2, $bl1);
#
#     {create @new_set_columns0, by removing the columns listed in @null_clms from
@cp_set_columns0.}
#
#     {create @new_inter_block_relations, by removing the row and column for $rmvd_bl from
@cp_inter_block_relations}
#
#     {create @new_bds_bl_coords, by removing the $rmvd_bl th element of @cp_bds_bl_coords
and by replacing its $remaining_bl th element with [0, $left_range + $right_range].}
#
#     {create @new_org_bl_coords, by removing the $rmvd_bl th element of @cp_org_bl_coords
and by  replacing its $remaining_bl th element with $left_range.}
#
#     {create other necessary things as well, either from scratch or by using the corresponding
ones before the merge}
#
#     {compute the indel component of the log-probability,
#       either from scratch or by smartly using the result before the merge (refer to:  section 5-1 of
"blueprint1_ANEX.xxxx.pdf")}
#
#     {perform the simultaneous "shift"-like moves of the resulting blocks}

}  # END of the outer foreach-loop (over subject pairs). #
```

(iii-vii-a) Horizontal merge + (incomplete) Vertical split (into sibling gap-blocks):
(Refer to: Appendix in "simultaneous_moves_of_multiple_blocks_METH.odp", and Figure A2 in "figures_simultaneous_moves_of_multiple_blocks_METH.odp")

```
                # RESTARTED on Jan 8, 2019. #

my @to_be_merged_vsplit1 = ();

for (my $bl1=0; $bl1 < $ub_bl; $bl1++) { # Preliminary outer for-loop (over $bl1). # Modified on
2019/01/26.
# for (my $bl1=0; $bl1 < $B; $bl1++) { # Preliminary outer for-loop (over $bl1).

    my ($br1, $u_or_d1) = @{$block_info[$bl1]}[$indx_br, $indx_u_or_d];
    my $pa1 = $node2pa->{$br1};
    my $eq_br1 = $eq_br->{$br1};
    my $children1 = $node2ch->{$br1};
    my $sibs1 = $node2ch->{$pa1};

    my $rels_w_bl1 = $inter_block_relations[$bl1];

    for (my $bl2 = $bl1+1; $bl2 < $ub_bl; $bl2++) { # Preliminary inner for-loop (over $bl2). #
Modified on 2019/01/26.
#   for (my $bl2 = $bl1+1; $bl2 < $B; $bl2++) { # Preliminary inner for-loop (over $bl2).


        my $rel = $rels_w_bl1->[$bl2];

                # MODIFIED on Jan 18, 2019. (6) #
        unless ($rel eq '>(ch)') { next; } # Skip unless $bl2 is an "effective child" of $bl1.
#       unless ($rel eq '>') { next; } # Skip unless $bl2 is vertically included in $bl1.
```

```
        # END of "MODIFIED on Jan 18, 2019. (6)" #

my ($br2, $u_or_d2) = @{$block_info[$bl1]}[$indx_br, $indx_u_or_d];
my $pa2 = $node2pa→{$br2};
my $children2 = $node2ch→{$br2};
my $eq_br2 = $eq_br→{$br2};
my $sibs2 = $node2ch→{$pa2};

my ($new_br1, $new_u_or_d1);

if ($u_or_d1 eq 'L') {
    if (($pa2 == $br1) and ($u_or_d2 eq 'L') and (@{$children1} == 2)){
        foreach my $ch (@{$children1}) {
            if ($ch == $br2) { next; }
            $new_br1 = $ch;
            last;
        }
        $new_u_or_d1 = 'L';
    }
} else { # if ($u_or_d1 eq 'U')
    if (defined $eq_br1) {
        if (($pa2 == $eq_br1) and ($u_or_d2 eq 'L') and (@{$sibs2} ==2)) {
            foreach my $sib (@{$sibs2}) {
                if ($sib == $br2) { next; }
                $new_br1 = $sib;
                last;
            }
            $new_u_or_d1 = 'L';
        }
    } else {
        if (($pa1 == $br2) and ($u_or_d2 eq 'U') and (@{$children2}==2)) {
            foreach my $ch (@{$children2}) {
                if ($ch == $br1) { next; }
                $new_br1 = $ch;
                last;
            }
            $new_u_or_d1 = 'L';

        } elsif (($pa1 == $pa2) and ($u_or_d2 eq 'L')) {
            if ($pa1 == $top_node) {
                if (@{$sibs1} == 3) {
                    foreach my $sib (@{$sibs1}) {
                        if (($sib == $br1) or ($sib == $br2)) { next; }
                        $new_br1 == $sib;
                        last;
                    }
                    $new_u_or_d1 = 'L';
                }
            } elsif (@{$sibs1} == 2) {
                $new_br1 = $pa1;
                $new_u_or_d1 = 'U';
            }
        } elsif ((defined $eq_br2) and ($pa1 == $eq_br2) and ($u_or_d2 eq 'L')) {
            if (@{$sibs1} == 2) {
                foreach my $sib (@{$sibs1}) {
                    if ($sib == $br1) { next; }
                    $new_br1 = $sib;
                    last;
```

```
                    }
                    $new_u_or_d1 = 'L';
                }

            } # End of "if (($pa1 == $br2) and ($u_or_d2 eq 'U') and (@{$children2}==2)) {…} elsif (($pa1 == $pa2) and ($u_or_d2 eq 'L')) { …} elsif ((defined $eq_br2) and ($pa1 == $eq_br2) and ($u_or_d2 eq 'L')) {...}"

        } # End of " if (defined $eq_br1) {…} else {...}"

    } # End of "if ($u_or_d1 eq 'L') {…} else {...}"

    unless (defined $new_br1) { next; }

    my($dist1, $dist2) = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations); # This subroutine measures the distance between $bl1 and $bl2, while taking account of the blocks between the two blocks. (See Appendix G I.) #
    my $dist = $dist2; # MODIFIED on Jan 13, 2019.
#   my $dist  = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations);

    if ($dist <= $THRSH_DIST_MERGE_VSPLIT1) { push @to_be_merged_vsplit1, [$bl1, $bl2, $dist, $new_br1, $new_u_or_d1];  }

  } # End of the preliminary inner for-loop (over $bl2).
} # End of the preliminary outer for-loop (over $bl1).


foreach my $sbjct_pair (@to_be_merged_vsplit1) {  # Outer foreach-loop (over subject pairs). #

    my ($bl1, $bl2, $dist, $new_br1, $new_u_or_d1) = @{$sbjct_pair};

    my @new_set_columns0 = copy (@set_columns0);
    my @new_bds_blocks0 = copy (@bds_blocks0);
    my $ln_prob_new_aln0 = $ln_prob_aln0;

    my @new_bds_bl_coords = copy (@bds_bl_coords);
    my @new_org_bl_coords = copy (@org_bl_coords);
    my @new_inter_block_relations = copy (@inter_block_relations);

    {Copy other necessary things as well.}


    my ($lb01, $rb01) = @{$new_bds_blocks0[$bl1]};
    my ($lb02, $rb02) = @{$new_bds_blocks0[$bl2]};

    my ($size01, $size02) = @block_sizes[$bl1, $bl2];

        # Merge $bl1 and $bl2, so that they will be immediately adjacent. #
    if ($dist > 0) {
        # $bl1 and $bl2 are horizontally separated.

      if ($lb01 < $lb02) {

        for (my $d = 0; $d < $dist; $d++) { # 1st middle for-loop (for shifts of $bl2).

            {shift $bl2 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$$) {...}")
```

```
                    , while computing the log-probability increment (= $incr_ln_prob) } # Modify
@new_set_columns0 & @new_bds_blocks0, as well as other things, accordingly.

                    $ln_prob_new_aln0 += $incr_ln_prob;
                    $new_org_bl_coords[$bl2]--;

                } # End of the 1st middle for-loop (for shifts of $bl2).

            } else { # if ($lb02 < $lb01)

                for (my $d = 0; $d < $dist; $d++) { # 2nd  middle for-loop (for shifts of $bl2).

                    {shift $bl2 to the right (maybe using "shift_bl_and_compt_prob_incr (@@@$
$) {...}")
                    , while computing the log-probability increment (= $incr_ln_prob) } # Modify
@new_set_columns0 & @new_bds_blocks0, as well as other things, accordingly.

                    $ln_prob_new_aln0 += $incr_ln_prob;
                    $new_org_bl_coords[$bl2]++;

                } # End of the 2nd  middle for-loop (for shifts of $bl2).
            }

        } elsif ($dist < 0) {
                # $bl2 horizontally includes $bl1.

                while ($new_bds_blocks[$bl2]→[0] < $new_bds_blocks[$bl1]→[0] ) { # 3rd
middle while-loop (for shifts of $bl1).

                    {shift $bl1 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$
$) {...}")
                    , while computing the log-probability increment (= $incr_ln_prob) } # Modify
@new_set_columns0 & @new_bds_blocks0, as well as other things, accordingly.

                    $ln_prob_new_aln0 += $incr_ln_prob;
                    $new_org_bl_coords[$bl1]--;
                } # End of the 3rd  middle while-loop (for shifts of $bl1).

        }

                # Re-define the boundary between the subject blocks,
                # from "horizontal" (along the sequences) to "vertical" (or phylogenetic).

        my ($lb2, $rb2) = @{$new_bds_blocks0[$bl2]};
        my ($new_lb1, $new_rb1) = @{$new_bds_blocks0[$bl1]};
        my ($lb_coord_new_bl1, $rb_coord_new_bl1) = @{$new_bds_bl_coords[$bl1]};
        my $org_coord_new_bl1 = $new_org_bl_coords[$bl1];

        my ($new_lb2, $new_rb2) = ($lb2 < $new_lb1) ? ($lb2, $new_rb1) : ($new_lb1, $rb2);

        =>  {Re-specify: @{$new_bds_blocks0[$bl2]} = ($new_lb2, $new_rb2). };
                # (The $bl2 th elements of @new_bds_bl_coords and @new_org_bl_coords
remain unchanged.) #

        => {Remove the $bl1 th elements of @new_bds_blocks0, @new_bds_bl_coords,
@new_org_bl_coords};
        => {Remove the $bl1 th row and column of @inter_block_relations};
        => {Assign the rank, $new_bl1,  to the new block resulting from the "vertical" split.};
```

(=> {Modify the **ranks** of the remaining blocks accordingly.};)

=> {Insert **[$new_lb1, $new_rb1]** between the ($new_bl1-1) th and $new_bl1 th elements of **@new_bds_blocks0**};

=> {Insert **[$lb_coord_new_bl1, $rb_coord_new_bl1]** between the ($new_bl1-1) th and $new_bl1 th elements of **@new_bds_bl_coords**};

=> {Insert **$org_coord_new_bl1** between the ($new_bl1-1) th and $new_bl1 th elements of **@new_org_bl_coords**};

=> {Create **@rels_w_new_bl1** for the relations between $new_bl1 th block and other blocks};
=> {Insert **\@rels_w_new_bl1** between the ($new_bl1 -1) th and $new_bl1 th elements of **@inter_block_relations**,
and insert $rels_w_new_bl1[$bl3] between the ($new_bl1 -1) th and $new_bl1 th elements of @{$inter_block_relations[$bl3]},
and specify: **$inter_block_relations[$bl2]→[$new_bl1] = $inter_block_relations[$new_bl1]→[$bl2] = 'S'**};

=> {Create **@new_blocks_w_spec_lb0** and **@new_blocks_w_spec_rb0**, probably from scratch} # ADDED on Jan 16, 2019.

=> {Re-create, or modify, **other necessary things**, accordingly};

END of "RESTARTED on Jan 8, 2019".

If possible, shift $bl2; otherwise, shift $new_bl1,
so that the two blocks horizontally overlap but are NOT nested.

RESTARTED on Jan 9, 2019.

if ($new_lb1 == $lb2) { # Left ends of $new_bl1 and $bl2 align.

{Examine the left margin of $new_bl1}; # In a way similar to creating **@set_left_flanking_clms** in **(iv-b) "Split" (into blocks of complementary types)}, or in (iii-vi-b) (Horizontal) Merge (or purge) + Split (or ex-nihilo) (complementary types)** (including incomplete merge (or purge))**.**

if ($new_bl1 has a non-zero left margin) {

{shift **$new_bl1** to the **left** by one column (maybe using "**shift_bl_and_compt_prob_incr (@@@$$) {...}**")
, while computing the log-probability increment (= $incr_ln_prob) } # Modify **@new_set_columns0 & @new_bds_blocks0,** and **@blocks_w_spec_lb0 & @blocks_w_spec_rb0** (added on Jan 16, 2019), as well as **other things**, accordingly.

$ln_prob_new_aln0 += $incr_ln_prob;
$new_org_bl_coords[$new_bl1]--;

} else {
{Examine the right margin of $bl2}; # In a way similar to creating **@set_right_flanking_clms** in **(iv-b) "Split" (into blocks of complementary types)}, or in (iii-vi-**

b) (Horizontal) Merge (or purge) + Split (or ex-nihilo) (complementary types) (including incomplete merge (or purge))**.**
if ($bl2 has a non-zero right margin) {

{shift **$bl2** to the **right** by one column (maybe using "**shift_bl_and_compt_prob_incr (@@@$$) {...}**")
, while computing the log-probability increment (= $incr_ln_prob) } # Modify **@new_set_columns0 & @new_bds_blocks0,** and **@blocks_w_spec_lb0 & @blocks_w_spec_rb0** (added on Jan 16, 2019), as well as **other things**, accordingly.

$ln_prob_new_aln0 += $incr_ln_prob;
$new_org_bl_coords[$bl2]++;
} else {
{FAIL};
}
}

} else { # if ($new_rb1 == $rb2) # Right-ends of $new_bl1 and $bl2 align.

{Examine the right margin of $new_bl1}; # In a way similar to creating **@set_right_flanking_clms** in **(iv-b) "Split" (into blocks of complementary types)}, or in (iii-vi-b) (Horizontal) Merge (or purge) + Split (or ex-nihilo) (complementary types)** (including incomplete merge (or purge))**.**
if ($new_bl1 has a non-zero right margin) {

{shift **$new_bl1** to the **right** by one column (maybe using "**shift_bl_and_compt_prob_incr (@@@$$) {...}**")
, while computing the log-probability increment (= $incr_ln_prob) } # Modify **@new_set_columns0 & @new_bds_blocks0,** and **@blocks_w_spec_lb0 & @blocks_w_spec_rb0** (added on Jan 16, 2019), as well as **other things**, accordingly.

$ln_prob_new_aln0 += $incr_ln_prob;
$new_org_bl_coords[$new_bl1]++;

} else {
{Examine the left margin of $bl2}; # In a way similar to creating **@set_left_flanking_clms** in **(iv-b) "Split" (into blocks of complementary types)}, or in (iii-vi-b) (Horizontal) Merge (or purge) + Split (or ex-nihilo) (complementary types)** (including incomplete merge (or purge))**.**
if ($bl2 has a non-zero right margin) {

{shift **$bl2** to the **left** by one column (maybe using "**shift_bl_and_compt_prob_incr (@@@$$) {...}**")
, while computing the log-probability increment (= $incr_ln_prob) } # Modify **@new_set_columns0 & @new_bds_blocks0,** and **@blocks_w_spec_lb0 & @blocks_w_spec_rb0** (added on Jan 16, 2019), as well as **other things**, accordingly.

$ln_prob_new_aln0 += $incr_ln_prob;
$new_org_bl_coords[$bl2]--;
} else {
{FAIL};
}
}
}

{Compute the indel component of the log-probability,

either from scratch or by smartly using the result before the merge (refer to: **section 5-1** of "**blueprint1_ANEX.xxxx.pdf**")} **# Omit the computations for the configurations in which $bl2 horizontally includes $new_bl1.**

{Perform the **simultaneous "shift"-like moves** of the *resulting* blocks}; **# Omit the computations (actually, simply mark as 'n/a' in the specified elements of the array for degeneracies) for the configurations in which $new_bl1 is horizontally included in $bl2.**

=> {Output the results}; # ADDED on 2019/01/26.

} # END of the outer foreach-loop (over subject pairs). #

(iii-vii-b) Horizontal merge + (incomplete) Vertical split (into sibling sequence-blocks):

```
my @to_be_merged_vsplit2 = ();

for (my $bl1=0; $bl1 < $ub_bl; $bl1++) { # Preliminary outer for-loop (over $bl1). # Modified on 2019/01/26.
for (my $bl1=0; $bl1 < $B; $bl1++) { # Preliminary outer for-loop (over $bl1).

    my ($br1, $u_or_d1) = @{$block_info[$bl1]}[$indx_br, $indx_u_or_d];
    my $pa1 = $node2pa->{$br1};
    my $eq_br1 = $eq_br->{$br1};
    my $children1 = $node2ch->{$br1};
    my $sibs1 = $node2ch->{$pa1};

    my $rels_w_bl1 = $inter_block_relations[$bl1];

    for (my $bl2 = $bl1+1; $bl2 < $ub_bl; $bl2++) { # Preliminary inner for-loop (over $bl2). # Modified on 2019/01/26.
#   for (my $bl2 = $bl1+1; $bl2 < $B; $bl2++) { # Preliminary inner for-loop (over $bl2).

        my $rel = $rels_w_bl1->[$bl2];

            # MODIFIED on Jan 18, 2019. (5) #
        unless ($rel eq '>(ch)') { next; } # Skip unless $bl2 is an "effective child" of $bl1.
#       unless ($rel eq '>') { next; } # Skip unless $bl2 is vertically included in $bl1.
            # END of "MODIFIED on Jan 18, 2019. (5)" #

        my ($br2, $u_or_d2) = @{$block_info[$bl1]}[$indx_br, $indx_u_or_d];
        my $pa2 = $node2pa->{$br2};
        my $children2 = $node2ch->{$br2};
        my $eq_br2 = $eq_br->{$br2};
        my $sibs2 = $node2ch->{$pa2};

        my ($new_br2, $new_u_or_d2);

        if ($u_or_d1 eq 'L') { # The sequence-block of $bl1 is on the "upper-side" of $br1.

            if (($pa2 == $br1) and ($u_or_d2 eq 'L') and (@{$children1} == 2)){ # The sequence-block of $bl2 is on the "upper-side" of $br2, which is a child of $br1.

                foreach my $ch (@{$children1}) {
                    if ($ch == $br2) { next; }
                    $new_br2 = $ch;
```

```
                last;
            }
            $new_u_or_d2 = 'U';
        }

    } else { # if ($u_or_d1 eq 'U')

        if (defined $eq_br1) { # The sequence-block of $bl1 is on the "upper-side" of $eq_br1.

            if (($pa2 == $eq_br1) and ($u_or_d2 eq 'L') and (@{$sibs2} ==2)) {# The
sequence-block of $bl2 is on the "upper-side" of $br2, which is a child of $eq_br1.

                foreach my $sib (@{$sibs2}) {
                    if ($sib == $br2) { next; }
                    $new_br2 = $sib;
                    last;
                }
                $new_u_or_d2 = 'U';
            }

        } else { # The sequence-block of $bl1 is on the "lower-side" of $br1.

            if (($pa1 == $br2) and ($u_or_d2 eq 'U') and (@{$children2}==2)) { # The
sequence-block of $bl2 is on the "lower-side" of $br2, which is the parent of $br1.

                foreach my $ch (@{$children2}) {
                    if ($ch == $br1) { next; }
                    $new_br2 = $ch;
                    last;
                }
                $new_u_or_d2 = 'U';

            } elsif (($pa1 == $pa2) and ($u_or_d2 eq 'L')) { # The sequence-block of $bl2 is on
the "upper-side" of $br2, which is a sibling of $br1.

                if ($pa1 == $top_node) { # $br1 and $br2 are children of the top-node.
                    if (@{$sibs1} == 3) {
                        foreach my $sib (@{$sibs1}) {
                            if (($sib == $br1) or ($sib == $br2)) { next; }
                            $new_br2 == $sib;
                            last;
                        }
                        $new_u_or_d2 = 'U';
                    }

                } elsif (@{$sibs1} == 2) { # $br1 and $br2 are children of a non-top node.
                    $new_br2 = $pa1;
                    $new_u_or_d2 = 'L';
                }

            } elsif ((defined $eq_br2) and ($pa1 == $eq_br2) and ($u_or_d2 eq 'L')) { # The
sequence-block of $bl2 is on the 'lower-side' of $eq_br2, which is the parent of $br1.

                if (@{$sibs1} == 2) {
                    foreach my $sib (@{$sibs1}) {
                        if ($sib == $br1) { next; }
                        $new_br2 = $sib;
                        last;
```

```
                    }
                    $new_u_or_d2 = 'U';
                }

            } # End of "if (($pa1 == $br2) and ($u_or_d2 eq 'U') and (@{$children2}==2)) {…} elsif (($pa1 == $pa2) and ($u_or_d2 eq 'L')) {…} elsif ((defined $eq_br2) and ($pa1 == $eq_br2) and ($u_or_d2 eq 'L')) {...}"

        }  # End of "if (defined $eq_br1) {…} else {… }"

    } # End of "if ($u_or_d1 eq 'L') {…} else {…}"

    unless (defined $new_br2) { next; }


    my ($dist1, $dist2) = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations); # This subroutine measures the distance between $bl1 and $bl2, while taking account of the blocks between the two blocks. (See Appendix G I.) #
    my $dist = $dist2; # MODIFIED on Jan 13, 2019.
#   my $dist = inter_block_distance ($bl1, $bl2, @bds_blocks0, @inter_block_relations);

    if ($dist <= $THRSH_DIST_MERGE_VSPLIT2) { push @to_be_merged_vsplit2, [$bl1, $bl2, $dist, $new_br2, $new_u_or_d2];  }

  } # End of the preliminary inner for-loop (over $bl2).
} # End of the preliminary outer for-loop (over $bl1).


        # END of "RESTARTED on Jan 9, 2019". #


        # RESTARTED on Jan 10, 2019. #
my $if_pure_split = 1;

foreach my $sbjct_pair (@to_be_merged_vsplit2) {  # Outer foreach-loop (over subject pairs). #

    my ($bl1, $bl2, $dist, $new_br2, $new_u_or_d2) = @{$sbjct_pair};

    my @new_set_columns0 = copy (@set_columns0);
    my @new_bds_blocks0 = copy (@bds_blocks0);
    my $ln_prob_new_aln0 = $ln_prob_aln0;

    my @new_bds_bl_coords = copy (@bds_bl_coords);
    my @new_org_bl_coords = copy (@org_bl_coords);
    my @new_inter_block_relations = copy (@inter_block_relations);

    {Copy other necessary things as well.}


    my ($lb01, $rb01) = @{$new_bds_blocks0[$bl1]};
    my ($lb02, $rb02) = @{$new_bds_blocks0[$bl2]};

    my ($size01, $size02) = @block_sizes[$bl1, $bl2];

        # Merge $bl1 and $bl2, so that they will be immediately adjacent. #

    if ($dist > 0) {
        # $bl1 and $bl2 are horizontally separated.
```

```
            if ($lb01 < $lb02) {

                for (my $d = 0; $d < $dist; $d++) { # 1st middle for-loop (for shifts of $bl2).

                    {shift $bl2 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$
$) {...}")
                     , while computing the log-probability increment (= $incr_ln_prob) } # Modify
@new_set_columns0 & @new_bds_blocks0, as well as other things, accordingly.

                    $ln_prob_new_aln0 += $incr_ln_prob;
                    $new_org_bl_coords[$bl2]--;

                }  # End of the 1st middle for-loop (for shifts of $bl2).

            } else { # if ($lb02 < $lb01)

                for (my $d = 0; $d < $dist; $d++) { # 2nd  middle for-loop (for shifts of $bl2).

                    {shift $bl2 to the right (maybe using "shift_bl_and_compt_prob_incr (@@@$
$) {...}")
                     , while computing the log-probability increment (= $incr_ln_prob) } # Modify
@new_set_columns0 & @new_bds_blocks0, as well as other things, accordingly.

                    $ln_prob_new_aln0 += $incr_ln_prob;
                    $new_org_bl_coords[$bl2]++;

                }  # End of the 2nd  middle for-loop (for shifts of $bl2).
            }

        } elsif ($dist < 0) {
                # $bl2 horizontally includes $bl1.

                while ($new_bds_blocks[$bl2]→[0] < $new_bds_blocks[$bl1]→[0] ) { # 3rd
middle while-loop (for shifts of $bl1).

                    {shift $bl1 to the left (maybe using "shift_bl_and_compt_prob_incr (@@@$
$) {...}")
                     , while computing the log-probability increment (= $incr_ln_prob) } # Modify
@new_set_columns0 & @new_bds_blocks0, as well as other things, accordingly.

                    $ln_prob_new_aln0 += $incr_ln_prob;
                    $new_org_bl_coords[$bl1]--;
                }  # End of the 3rd  middle while-loop (for shifts of $bl1).

        }

                # Up to here, the processes are identical to those in (iii-vii-a) Horizontal merge +
(incomplete) Vertical split (into sibling gap-blocks). #

        my $rels_w_bl2 = $inter_block_relations[$bl2];

                # Re-define the boundary between the subject blocks,
                # from "horizontal" (along the sequences) to "vertical" (or phylogenetic).

        my ($lb1, $rb1) = @{$new_bds_blocks0[$bl1]};
        my ($lb2, $rb2) = @{$new_bds_blocks0[$bl2]};
        my ($lb_coord_bl1, $rb_coord_bl1) = @{$new_bds_bl_coords[$bl1]};
```

```
        my ($lb_coord_new_bl2, $rb_coord_new_bl2) = @{$new_bds_bl_coords[$bl2]};
        my ($org_coord_new_bl1, $org_coord_new_bl2) = @{$new_org_bl_coords}[$bl1, $bl2];

                # NOTE: The range of $bl2 will be carried over to $new_bl2 without any change,
                # because $bl2 and $new_bl2 simply "pass through" the old $bl1 and the new $bl1,
respectively.

#       my ($new_lb1, $new_rb1) = ($lb2 < $lb1) ? ($lb2, $rb1) : ($lb1, $rb2);
        my ($lb_coord_new_bl1, $rb_coord_new_bl1) = ($lb_coord_bl1, $rb_coord_bl1 - $size2);
        if ($lb2 < $lb1) { $org_coord_new_bl1 -= $size2; }

                # NOTE: The range of $bl1 will decrease by $size2, because $bl1 will "inch along"
the $bl2, but will "pass through" $new_bl2.

#       => {Re-specify: @{$new_bds_blocks0[$bl1]} = ($new_lb1, $new_rb1). };
        => {Re-specify @{$new_bds_bl_coords[$bl1]} = ($lb_coord_new_bl1,
$rb_coord_new_bl1). };
        => {Re-specify $new_org_bl_coords[$bl1] = $org_coord_new_bl1.};


        => {Remove the $bl2 th elements of @new_bds_blocks0, @new_bds_bl_coords,
@new_org_bl_coords};
        => {Remove the $bl2 th row and column of @inter_block_relations};
        => {Assign the rank, $new_bl2,  to the new block resulting from the "vertical" split.};

        (=> {Modify the ranks of the remaining blocks accordingly.};)


    # The following processes are somewhat similar to those in (iv-b) "Split" (into blocks of
complementary types). #

        my %clm2involved;

        for (my $k = 0; $k < $B; $k++) {  # 2nd middle for-loop (over blocks $bl3).
            my $bl3 = $k;
            if ($bl3 == $bl) { next; } # MODIFIED on Jan 15, 2018. (7)
            # my $bl3 = $lb_sorted_set[$k]; # OBSOLETE as of Jan 15, 2018. (7)
            if ($bl3 == $bl2) { next; }
            my ($lb3, $rb3) = @{$cp_bds_blocks0[$bl3]};
            if ($rb3 < $lb2) { next; }
            if ($rb2 < $lb3) { next; }

            my $rel = $rels_w_bl2→[$bl3];
            if (($rel eq '=') or ($rel eq 'Cp')) {
#           $right_margin -= ($right_end_laln - $rb2)+1;
#           $re_right_margin = $lb2 – 2;
#           last;

                if ($bl3 < $bl2) {
                  for (my $c= $lb3; $c <= $rb3; $c++) {
                  # for (my $c= $lb2; $c <= $rb2; $c++) {
                    my $involved = $clm2involved{$c};
                    unless (defined $involved) { $involved = $clm2involved{$c} = []; }
                    push @{$involved}, $bl3;
                    # push @{$involved}, $bl2;
                  }
                }
            } elsif ($rel eq 'S') { # $bl and $bl2 are "siblings". # ADDED on Dec 30, 2018.
```

```
#                   $re_right_margin = $lb2 -1;
#                   last;
               } elsif ($rel eq '>') { # ADDED on Dec 30, 2018.
#                   my ($br2, $u_or_d2) = @{$block_info[$bl2]}[$indx_br, $indx_u_or_d];
#                   my $pa2 = $node2pa->{$br2};
#                   my $eq_br2 = $eq2br->{$br2};
#                   if (($br2 == $pa) or ($br == $pa2)
#                   or ((defined $eq_br) and ($eq_br == $pa2))
#                   or ((defined $eq_br2) and ($eq_br2 == $pa)) ) { # $bl2 is a "child" of $bl.
#                       $re_right_margin = $lb2 -1;
#                       last;
#                    }
                                # NOTE added (2019/01/18): If this block is resurrected, the condition should
be changed to ($rel eq '>(ch)'), and the additional conditions should be omitted.

               } elsif (($rel eq '<') or ($rel eq '<(pa)')) { # $bl2 vertically includes $bl. # Added '<(pa)'
on Jan 18, 2019.
                   for (my $c= $lb3; $c <= $rb3; $c++) {
                   # for (my $c= $lb2; $c <= $rb2; $c++) {
                       my $involved = $clm2involved{$c};
                       unless (defined $involved) { $involved = $clm2involved{$c} = []; }
                       push @{$involved}, $bl3;
                       # push @{$involved}, $bl2;
                   }
               } elsif (($rel eq 'ONN') or ($rel eq 'ONCS')) {# $bl2 and $bl overlap but do not nest.
                   for (my $c= $lb3; $c <= $rb3; $c++) {
                   # for (my $c= $lb2; $c <= $rb2; $c++) {
                       my $involved = $clm2involved{$c};
                       unless (defined $involved) { $involved = $clm2involved{$c} = []; }
                       push @{$involved}, $bl3;
                       # push @{$involved}, $bl2;
                   }
               }

           }  # End of the 2nd middle for-loop (over blocks $bl3).


           my @set_to_be_split = ();
           for (my $c = $lb2; $c <= $rb2; $c++) {
               unless (defined $clm2involved{$c}) { push @set_to_be_split, $c; }
           }

           unless (@set_to_be_split == $size2) {
                   {FAIL};
           }


           my $if_on_the_left = ($lb2 < $lb1) ? 1 : 0;
           my $lb_new_bl2 =  my $rb_new_bl2 = ($if_on_the_left) ? $lb1-1 : $rb1 +2;
```

=> {Insert **[$lb_new_bl2, $rb_new_bl2]** between the ($new_bl2 -1) th and $new_bl2 th elements of **@new_bds_blocks0**};

=> {Insert **[$lb_coord_new_bl2, $rb_coord_new_bl2]** between the ($new_bl2-1) th and $new_bl2 th elements of **@new_bds_bl_coords**};

=> {Insert **$org_coord_new_bl2** between the ($new_bl2-1) th and $new_bl2 th elements of **@new_org_bl_coords**};

End of "The following processes are somewhat similar to those in **(iv-b) "Split" (into blocks of complementary types)."**

=> {Create **@rels_w_new_bl2** for the relations between $new_bl2 th block and other blocks};
=> {Insert **\@rels_w_new_bl2** between the ($new_bl2 -1) th and $new_bl2 th elements of **@inter_block_relations**,
and insert $rels_w_new_bl2[$bl3] between the ($new_bl2 -1) th and $new_bl2 th elements of @{$inter_block_relations[$bl3]},
and specify: **$inter_block_relations[$bl1]→[$new_bl2] = $inter_block_relations[$new_bl2]→[$bl1] = 'ONCS'**};

=> {Re-create, or modify, **other necessary things**, accordingly}; # MOVED to below the for-loop (on Jan 16, 2019).

Execute the vertical split.

my @indices_affected_by_bl1 = {indices of the classes or sequences affected by $bl1};

for (my $size_new_bl2=1; $size_new_bl2 <= $size2; $size_new_bl2++) { # 3rd middle for-loop (over the sizes of the $new_bl2). #

my **$to_be_split** = ($if_on_the_left) ? (pop @set_to_be_split) : (shift @set_to_be_split);

{ **split the $to_be_split th column (in the original local alignment) at the branch $br,**
and move the $u_or_d side to the (($if_on_the_left) ? right : left),
and "merge" it with the **$bl1** (= $bl_sbj) th block,
and also "merge" its complement with the **$new_bl2** (= $bl_cmpl) th block.}; # **@new_bds_blocks0** is also modified accordingly. # See **Appendix H J**.

OBSOLETE as of Jan 16, 2019.
=> {Re-sort the left-bounds and the right-bounds to update @new_lb_sorted_set and @new_rb_sorted_set, respectively};
=> {Update @new_orders_lb and @new_orders_rb};
#
{Modify other important data sets accordingly};
END of "OBSOLETE as of Jan 16, 2019".

} # End of the 3rd middle for-loop (over the sizes of the new complementary block). #

=> {Create **@new_blocks_w_spec_lb0** and **@new_blocks_w_spec_rb0**, probably from scratch} # ADDED on Jan 16, 2019.

=> {Re-create, or modify, **other necessary things**, accordingly}; # MOVED from above the for-loop (on Jan 16, 2019).

{Compute the indel component of the log-probability,
either from scratch or by smartly using the result before the merge (refer to: **section 5-1** of "**blueprint1_ANEX.xxxx.pdf**")} **# Omit the computations for the configurations in which $bl2 horizontally includes $new_bl1.**

{Perform the **simultaneous "shift"-like moves** of the *resulting* blocks}; **# Omit the computations (actually, simply mark as 'n/a' in the specified elements of the array for degeneracies) for the configurations in which $new_bl1 is horizontally included in $bl2.**

```
        => {Output the results}; # ADDED on 2019/01/26.

}  # END of the outer foreach-loop (over subject pairs). #

                # END of "RESTARTED on Jan 10, 2019". #
```

SM-7. (IMPORTANT!!) Transforming set of gap-blocks when Dollo-parsimony does not give any parsimonious indel history. (Refer to: Appendix in "simultaneous_moves_of_multiple_blocks_METH.odp", and Figure A1 in "figures_simultaneous_moves_of_multiple_blocks_METH.odp")

```
                # RESTARTED on Jan 12, 2019. #

# Modify @bds_blocks, @inter_block_relations, @info_blocks, and other necessary things.

my $bl1 = 0;
while ($bl1 < @bds_blocks) {  # Outermost while-loop (over $bl). #

        my ($lb1, $rb1) = @{$bds_blocks[$bl1]};

        my ($br1, $u_or_d1) = @{$block_info[$bl1]}[$indx_br, $indx_u_or_d];
        my $pa1 = $node2pa->{$br1};
        my $eq_br1 = $eq_br->{$br1};
        my $children1 = $node2ch->{$br1};
        my $sibs1 = $node2ch->{$pa1};

        my $rels_w_bl1 = $inter_block_relations[$bl1];

        my ($cand_splitting_left, $cand_splitting_right);
        for (my $bl2 = $bl1+1; $bl2 < @bds_blocks; $bl2++) { # Middle while-loop (over $bl2).

            my $rel = $rels_w_bl1->[$bl2];

                # MODIFIED on Jan 18, 2019. (3) #
            unless ($rel eq '>(ch)') { next; }

#           unless ($rel eq '>') { next; }
#
#           my ($br2, $u_or_d2) = @{$block_info[$bl2]}[$indx_br, $indx_u_or_d];
#           my $pa2 = $node2pa->{$br2};
#           my $eq_br2 = $eq_br->{$br2};
#           my $children2 = $node2ch->{$br2};
#
                # Examine whether $bl2 is a "child" of $bl1. #
#           my $if_ch = 0;
#           if ($u_or_d1 eq 'L') {
#               if (($br1 == $pa2) and ($u_or_d2 eq 'L') and (@{$children1}==2)) {
#                   $if_ch = 1;
#               }
#
#           } else { # if ($u_or_d1 eq 'U')
#               if (defined $eq_br1) {
#                   if (($eq_br1 == $pa2) and ($u_or_d2 eq 'L') and (@{$children1}==2)) {
```

```
#                       $if_ch = 1;
#                   }
#               } elsif ($pa1 == $top_node) {
#                   if ((@{$sibs1} == 3) and ($pa2 == $top_node) and ($u_or_d2 eq 'L')) {
#                       $if_ch = 1;
#                   }
#               } elsif (@{$sibs1} == 2) {
#                   if ($pa1 == $br2) {
#                       if ($u_or_d2 eq 'U') { $if_ch = 1; }
#
#                   } elsif ((defined $eq_br2) and ($pa1 == $eq_br2)) {
#                       if ($u_or_d2 eq 'L') { $if_ch = 1; }
#                   } elsif ($pa1 == $pa2) {
#                       if ($u_or_d2 eq 'L') { $if_ch = 1; }
#                   }
#               }
#           }
#
#           unless ($if_ch) { next; }
                # END of "MODIFIED on Jan 18, 2019. (3)" #

            my ($lb2, $rb2) = @{$bds_blocks[$bl2]};
#           my $rels_w_bl2 = $inter_block_relations[$bl2];

            if ($rb1 < $lb2) {
                if (defined $cand_splitting_right) { next; }

                if ($rb1 + 1 == $lb2) {
                    $cand_splitting_right = $bl2;

                } else {
                    my ($lb_med, $rb_med) = ($rb1 + 1, $lb2-1);
                    my $mediating = $blocks_w_spec_lb[$lb_med];  # @{$blocks_w_spec_lb[$k]}
lists the blocks whose left-bounds are $k.
                    while (@{$mediating}>0) {
                        my $padding;
                        foreach my $bl3 (@{$mediating}) { # 1st inner foreach-loop (over mediating
blocks with $lb_med). #
                            $rel13 = $rel_w_bl1->[$bl3];
#                           $rel23 = $rel_w_bl2->[$bl3];
                            if ( ($rel13 eq '<') or ($rel13 eq '<(pa)') or
#                               ($rel23 eq 'ONN) or ($rel23 eq 'ONCS') ) {
                                ($rel13 eq 'ONN) or ($rel3 eq 'ONCS') ) { # Added '<(pa)' on Jan 18,
2019.
                                $padding = $bl3;
                                last;
                            }

                        } # 1st inner foreach-loop (over mediating blocks with $lb_med). #

        j               unless (defined $padding) { last; } # ADDED on Jan 18, 2019.

                        # Update $lb_med. #
                        $lb_med = $bds_blocks[$padding]->[1] + 1;
                        if ($rb_med < $lb_med) { last; }

                        $mediating = $blocks_w_spec_lb[$lb_med];
```

```
                    } # End of the while-loop. #

                    if ($rb_med < $lb_med) { $cand_splitting_right = $bl2; }
                }

            } elsif ($rb2 < $lb1) {
                if (defined $cand_splitting_left) { next; }

                if ($rb2 + 1 == $lb1) {
                    $cand_splitting_left = $bl2;

                } else {
                    my ($lb_med, $rb_med) = ($rb2 + 1, $lb1-1);
                    my $mediating = $blocks_w_spec_lb[$lb_med];  # @{$blocks_w_spec_lb[$k]}
lists the blocks whose left-bounds are $k.
                    while (@{$mediating}>0) {
                        my $padding;
                        foreach my $bl3 (@{$mediating}) { # 1st inner foreach-loop (over mediating
blocks with $lb_med). #
                            $rel13 = $rel_w_bl1->[$bl3];
#                           $rel23 = $rel_w_bl2->[$bl3];
                            if ( ($rel13 eq '<') or ($rel13 eq '<(pa)') or
#                               ($rel23 eq 'ONN) or ($rel23 eq 'ONCS') ) {
                                ($rel13 eq 'ONN) or ($rel13 eq 'ONCS') ) { # Added '<(pa)' on Jan
18, 2019.
                                $padding = $bl3;
                                last;
                            }

                        } # 1st inner foreach-loop (over mediating blocks with $lb_med). #

                        unless (defined $padding) { last; } # ADDED on Jan 18, 2019.

                        # Update $lb_med. #
                        $lb_med = $bds_blocks[$padding]->[1] + 1;
                        if ($rb_med < $lb_med) { last; }

                        $mediating = $blocks_w_spec_lb[$lb_med];

                    } # End of the while-loop. #

                    if ($rb_med < $lb_med) { $cand_splitting_right = $bl2; }

                }

            } else {
                next;
            }

            if ((defined $cand_splitting_left) and (defined $cand_splitting_right)) { last; }

        } # End of the middle while-loop (over $bl2).


        unless ((defined $cand_splitting_left) and (defined $cand_splitting_right)) {
            $bl1++;
            next;
        }
```

```
        unless ($inter_block_relations→[$cand_splitting_left][$cand_splitting_right] eq 'S') { # The two candidate blocks must be siblings.
            $bl1++;
            next;
        }

                # Now that we found that $bl1 can indeed be split,
                # actually split the $bl1.

        my ($bl_left, $bl_right) = ($cand_splitting_left, $cand_splitting_right);

        my ($lb_left, $rb_left) = @{$bds_blocks[$bl_left]};
        my ($lb_right, $rb_right) = @{$bds_blocks[$bl_right]};

         while ($rb1 + 1 < $lb_right) {# MODIFIED on Jan 18, 2019.
#        if  ($rb1 + 1 < $lb_right) {
            my $if_rlv = 0;  # ADDED on Jan 18, 2019.
            my $rel_w_right = $inter_block_relations[$bl_right];# ADDED on Jan 18, 2019.
            foreach my $bl5 (@{$blocks_w_spec_rb[$lb_right-1]}) {# ADDED on Jan 18, 2019.
                    # ADDED on Jan 18, 2019.
                my ($lb5, $rb5) = @{$bds_blocks[$bl5]};
                my $rel = $rel_w_right→[$bl5];

                if (($rel eq '<') or ($rel eq '<(pa)')) {
                        $lb_right = $bds_blocks[$bl_right]→[0] = $lb5;
                        $if_rlv = 1;
                } elsif (($rel eq 'ONN') or ($rel eq 'ONCS')) {# ADDED on Jan 18, 2019.

               { Swap the intervals, [$lb5, $lb_right-1]  and [$lb_right, $rb_right],
                 of the local alignment (i.e., @set_columns). }; # Modified on Jan 18, 2019.
#               { Swap the intervals, [$rb1+1, $lb_right-1]  and [$lb_right, $rb_right],
#                 of the local alignment (i.e., @set_columns). };

               => { Update @bds_blocks accordingly.}; # Use @blocks_w_spec_lb and @blocks_w_spec_rb. (ADDED on Jan 16, 2019)

               => { Swap the intervals, [$lb5, $lb_right-1]  and [$lb_right, $rb_right],
                 of @blocks_w_spec_lb and @blocks_w_spec_rb. }; # ADDED on Jan 16, 2019. # Modified on Jan 18, 2019.
#               => { Swap the intervals, [$rb1+1, $lb_right-1]  and [$lb_right, $rb_right],
#                 of @blocks_w_spec_lb and @blocks_w_spec_rb. }; # ADDED on Jan 16, 2019. #

                   ($lb_right, $rb_right) = @{$bds_blocks[$bl_right]};
                    $if_rlv = 1;

                } # End of "if () {} elsif () {}" (ADDED on Jan 18, 2019)

                if ($if_rlv) { last; }  # ADDED on Jan 18, 2019.

            } # END of foreach over $bl5. (ADDED on Jan 18, 2019)

            unless ($if_rlv) { # ADDED on Jan 18, 2019.
                {FAIL};
            }
        } # END of "while ($rb1 + 1 < $lb_right) {...}". # MODIFIED on Jan 18, 2019.

        while ($rb_left + 1 < $lb1) {# MODIFIED on Jan 18, 2019.
```

```
#       if ($rb_left + 1 < $lb1) {

            my $if_rlv = 0;  # ADDED on Jan 18, 2019.
            my $rel_w_left = $inter_block_relations[$bl_left];# ADDED on Jan 18, 2019.
            foreach my $bl5 (@{$blocks_w_spec_lb[$rb_left + 1]}) {# ADDED on Jan 18, 2019.
                    # ADDED on Jan 18, 2019.
                my ($lb5, $rb5) = @{$bds_blocks[$bl5]};
                my $rel = $rel_w_left->[$bl5];

                if (($rel eq '<') or ($rel eq '<(pa)')) {
                        $rb_left = $bds_blocks[$bl_left]->[1] = $rb5;
                        $if_rlv = 1;
                } elsif (($rel eq 'ONN') or ($rel eq 'ONCS')) {# ADDED on Jan 18, 2019.


             { Swap the intervals, [$lb_left, $rb_left] and [$rb_left + 1, $rb5],
                of the local alignment (i.e., @set_columns). }; # Modified on Jan 18, 2019.
#            { Swap the intervals, [$lb_left, $rb_left] and [$rb_left + 1, $lb1 -1],
#               of the local alignment (i.e., @set_columns). };

            => { Update @bds_blocks accordingly.};  # Use @blocks_w_spec_lb and
@blocks_w_spec_rb. (ADDED on Jan 16, 2019)

            => { Swap the intervals, [$lb_left, $rb_left] and [$rb_left + 1, $rb5],
                of @blocks_w_spec_lb and @blocks_w_spec_rb. }; # ADDED on Jan 16, 2019. #
Modified on Jan 18, 2019.
#           => { Swap the intervals, [$lb_left, $rb_left] and [$rb_left + 1, $lb1 -1],
#               of @blocks_w_spec_lb and @blocks_w_spec_rb. }; # ADDED on Jan 16, 2019.

                ($lb_left, $rb_left) = @{$bds_blocks[$bl_left]};
                 $if_rlv = 1;

                } # End of "if () {} elsif () {}" (ADDED on Jan 18, 2019)

                if ($if_rlv) { last; }  # ADDED on Jan 18, 2019.

            } # END of foreach over $bl5. (ADDED on Jan 18, 2019)

            unless ($if_rlv) { # ADDED on Jan 18, 2019.
                {FAIL};
            }

        } # END of "while ($rb_left + 1 < $lb1) {...}" # MODIFIED on Jan 18, 2019.


                # Modify @bds_blocks. #

        $bds_blocls[$bl_left]->[1] = $rb1;
        $bds_blocks[$bl_right]->[0] = $lb1;

        => {Remove the $bl1 th element of @bds_blocks.};

        => {Update @blocks_w_spec_lb and @blocks_w_spec_rb, by removing $bl1 and moving
$bl_left and $bl_right accordingly.} # ADDED on Jan 16, 2019.

                # Modify @bds_bl_coords and @org_bl_coords . #

        => {Remove the $bl1 th element of @bds_bl_coords.};
```

=> {Remove the $bl1 th element of **@org_bl_coords**.};

NOTE: The ranges of $bl_left and $bl_right need NOT be changed,
because they simply "passed through" $bl1.

Modify **@inter_block_relations**.

=> {Remove the $bl1 th row and column of **@inter_block_relations**.};

=> {Modify **other necessary things**.};

 # Keep $bl1 unchanged!! # (This is because the new $bl1 th block in the next session was the ($bl1 +1) th block in this session.)

} # END of the outermost while-loop (over $bl). #

END of "RESTARTED on Jan 12, 2019".

Appendixes G-H I-J (re-labelled on Jan 18, 2019)

APPENDIX G I (re-labelled on Jan 18, 2019): Computing the distance between two blocks.

```
                # RESTARTED on Jan 13, 2019. #

my ($dist1, $dist2) = inter_block_distance ($bl1, $bl2, @bds_blocks, @inter_block_relations);

        # $dist1 is the number of "shift"s that takes $bl1 to the immediate neighbor of $bl2.
        # $dist2 is the number of "shift"s that takes $bl2 to the immediate neighbor of $bl1.
        # $dist1 = $dist2 = -1 if $bl1 and $bl2 horizontally overlap each other.

sub inter_block_distance ($$\@\@) {

        my ($b1, $b2, $bds_blocks, $inter_block_relations) = @_;

        my ($lb1, $rb1) = @{$bds_blocks→[$bl1]};
        my ($lb2, $rb2) = @{$bds_blocks→[$bl2]};

                # The two blocks are immediate neighbors. #
        if ($rb1 + 1 = $lb2) { return (0, 0); }
        if ($rb2 + 1 = $lb1) { return (0, 0); }

                # The two blocks overlap. #
        if  ((lb2 <= $rb1) and ($lb1 <= $rb2)) { return (-1, -1); }

                # Determine the interval in between $bl1 and $bl2. #
        my ($lb_med, $rb_med) = ($lb1 < $lb2) ? ($rb1+1, $lb2-1) : ($rb2+1, $lb1-1);


                # Enumerate the columns (or sites) that could be "passed through" by $bl1 or $bl2. #

        my $rels_w_bl1 = $inter_block_relations→[$bl1];
        my $rels_w_bl2 = $inter_block_relations→[$bl2];

        my %clm2involved1;
        my %clm2involved2;

        for (my $bl3=0; $bl3 < $B; $bl3++) { # 1st main for-loop (over $bl3).

            if (($bl3 == $bl1) or ($bl3 == $bl2)) { next; }

            my ($lb3, $rb3) = @{$bds_blocks→[$bl3]};
            unless (($lb3 <= $rb_med) and ($lb_med <= $rb3)) { next; }

                # Compute the boundaries of the intersection. (Added on 2017/01/27.) #
            my $lb_insc = ($lb_med > $lb3) ? $lb_med : $lb3;
            my $rb_insc = ($rb_med < $rb3) ? $rb_med : $rb3;

            my $rel13 = $rels_w_bl1→[$bl3];
            my $rel23 = $rels_w_bl2→[$bl3];

             if (($rel13 eq '<') or ($rel13 eq '<(pa)') or (($rel13 eq '=') and ($bl3 < $bl1))
                or ($rel13 eq 'ONN') or ($rel13 eq 'ONCS')) { # Added '<(pa)' on Jan 18, 2019.

                for (my $c = $lb_insc; $c <= $rb_insc; $c++) { # Modified on 2019/01/27.
#               for (my $c = $lb3; $c <= $rb3; $c++) {
                    my $involved1 = $clm2involved1{$c};
```

```
                unless (defined $involved1) { $involved1 = $clm2involved1{$c} = []; }
                push @{$involved1}, $bl3;
            }
        }

        if (($rel23 eq '<') or ($rel23 eq '<(pa)') or (($rel23 eq '=') and ($bl3 < $bl2))
            or ($rel23 eq 'ONN') or ($rel23 eq 'ONCS')) { # Added '<(pa)' on Jan 18, 2019.

            for (my $c = $lb_insc; $c <= $rb_insc; $c++) { # Modified on 2019/01/27.
#           for (my $c = $lb3; $c <= $rb3; $c++) {
                my $involved2 = $clm2involved12{$c};
                unless (defined $involved2) { $involved2 = $clm2involved2{$c} = []; }
                push @{$involved2}, $bl3;
            }
        }

    } # End of the 1st main for-loop (over $bl3).


        # Compute $dist1 and $dist2. #

    my $dist1 = $dist2 = 0;

    for (my $c = $lb_med; $c <= $rb_med; $c++) { # 2nd main for-loop (over mediating columns).

        unless (defined $clm2involved1{$c}) { $dist1++; }
        unless (defined $clm2involved2{$c}) { $dist2++; }

    }  # End of the 2nd main for-loop (over mediating columns).

    return ($dist1, $dist2);

}  # END of "sub inter_block_distance ($$\@\@) {...}"


        # END of "RESTARTED on Jan 13, 2019". #
```

APPENDIX H J (re-labelled on Jan 18, 2019)**: Splitting column and "merg"ing two resulting columns with pair of neighboring blocks.**

```
#       { split the $to_be_split th column (in the original local alignment) at the branch $br,
#         and move the $u_or_d side  to the (($if_on_the_left) ? $right  : $left),
#          and "merge" it with the $bl th block,
#         and also "merge" its complement with the complement of the $bl th block. };
```

(1) split the $to_be_split th column (in the original local alignment) at the branch $br,

```
my @column_af = copy (@{$set_columns0[$to_be_split]});
my $ct_classes = scalar (@copy_columns);

my $ln_prob_clm_bf =  {compute the log-probability of @column_af};
my @column_cmpl = ();
for (1 .. $ct_classes) { push @column_cmpl, $GAP; }

foreach my $indx_ac (@indices_affected_classes) {
```

```
        my $tmp = $column_af[$indx_ac];
        $column_af[$indx_ac] = $column_cmpl[$indx_ac];
        $column_cmpl[$indx_ac] = $tmp;
}

my $ln_prob_clm_af = {compute the log-probability of @column_af};
my $ln_prob_clm_cmpl = {compute the log-probability of @column_cmpl};

my $delta_ln_prob = $ln_prob_clm_af + $ln_prob_clm_cmpl - $ln_prob_clm_bf;
```

(2) Move the $u_or_d side to the (($if_on_the_left) ? $right : $left), and "merge" it with the $bl th block, and also "merge" its complement with the complement of the $bl th block.

```
my $to_be_removed = ($if_on_the_left) ? $to_be_split : $to_be_split + ($size_cmpl - $size_cmpl0 -1);  # Added the "- $size_cmpl0" on Jan 8, 2018.
  => {Remove the $to_be_removed th column from the current @new_set_columns0.};

my $bds_new_bl = $new_bds_blocks0[$new_bl];
my $bds_bl_cmpl = $new_bds_blocks0[$bl_cmpl];


if ($if_on_the_left) { # The split columns will be merged to the left of the $new_bl th block. #

        $bds_new_bl->[1]++;
        if (!($if_pure_split) or ($size_cmpl>1)) { $bds_bl_cmpl→[0]--; } # Expanded the condition on Jan 8, 2018.
        my ($new_lb, $new_rb) = @{$bds_new_bl};
        my ($cmpl_lb, $cmpl_rb) = @{$bds_bl_cmpl};

        foreach my $pos (all positions (including the boundaries in @new_bds_blocks0, excluding the $new_bl and $bl_cmpl th element) ) {
            if ($pos == $to_be_removed) {
                if ($pos is included in @column_af) {
#               if ($pos is included in $new_bl ) {
                    $pos = $new_lb;
                } elsif ($pos is included in  @column_cmpl) {
#               } elsif ($pos is included in $bl_cmpl ) {
                    $pos = $cmpl_lb;
                }
            } elsif (($to_be_removed < $pos) and ($pos <= $cmpl_lb)) {
                $pos--;
            } elsif ($new_lb <= $pos) {
                $pos++;
            }
        }

            # ADDED on Jan 1, 2019. (1) #
        => {Insert \@column_cmpl between the ($cmpl_lb -1) th and $cmpl_lb th elements of the current @new_set_columns0};
        => {Insert \@column_af between the ($new_lb -1) th and $new_lb th elements of the current @new_set_columns0};

            # END of "ADDED on Jan 1, 2019. (1)" #


} else { # The split columns will be merged to the right of the $new_bl th block. #
```

```
        $bds_new_bl→[1]++;
        if (!($if_pure_split) or ($size_cmpl>1)) { # Expanded the condition on Jan 8, 2018.
            $bds_bl_cmpl→[0]++;
             $bds_bl_cmpl→[1] += 2;
        }

        my ($new_lb, $new_rb) = @{$bds_new_bl};
        my ($cmpl_lb, $cmpl_rb) = @{$bds_bl_cmpl};

        foreach my $pos (all positions (including the boundaries in @new_bds_blocks0, excluding
the $new_bl and $bl_cmpl th element) ) {
            if ($pos == $to_be_removed) {
                if ($pos is included in @column_af) {
#               if ($pos is included in $new_bl ) {
                    $pos = $new_rb;
                } elsif ($pos is included in @column_cmpl) {
#               } elsif ($pos is included in $bl_cmpl) {
                    $pos = $cmpl_rb;
                }
            } elsif ($to_be_removed < $pos) {
                $pos++;
            } elsif (( $cmpl_rb-2< $pos) and ($pos < $to_be_removed)) {
                $pos += 2;
            } elsif (($new_rb <= $pos) and ($pos <= $cmpl_rb-2)) {
                $pos++;
            }
        }

                # ADDED on Jan 1, 2019. (2) #
        => {Insert \@column_af between the ($new_rb -1) th and $new_rb th elements of the
current @new_set_columns0};
        => {Insert \@column_cmpl between the ($cmpl_rb -1) th and $cmpl_rb th elements of the
current @new_set_columns0};

                # END of "ADDED on Jan 1, 2019. (2)" #

}
```