The CGW to DP_Compare interface

Knut Reinert

1. Overview

This document describes the current state of the functions that are used to check for overlaps between unitigs respectively contigs and exhibits a wish list of simpler interfaces to the DP_Align family of functions that are an integral part of the assembler and should be used because of their high efficiency. These functions are collectively declared in the file `ChunkOverlap_CGW.c`. Computed overlaps are stored in a hash table in so called “canonical” form which is intended to be hidden from the user of the calling function.

Hence all functions manipulate some data before calling the function `DP_Compare`. After returning they interpret the result as intended by the calling function.

We will first describe the interface of the `DP_Compare` function in section 2. In section 3 we describe how one works (They basically work the same but have different interfaces).

2. The DP_Compare interface

The following paragraph is taken from the header file AS_ALN_dpaligner.h and describes the input parameter to `DP_Compare`.

Given fragments a and b, find the best overlap between them subject to the following parameters / thresholds. The overlap must start on one of the diagonals of the d.p. matrix in the interval [beg,end]. For example if one gives the interval [-10,20], then the overlap either has less than the first 20bp of a unaligned or less than the first 10bp of b unaligned. If the boolean variable opposite is nonzero then the fragments are to be aligned in the opposite orientation. One is assuming an error rate of erate in the sequences, and is guaranteed to find only alignments for which the number of differences d in each prefix of length n is such that Sum_k=d^n (n choose k) erate^k (1-erate)^(n-k) < thresh. One should note carefully, that alignments not satisfying this property may be found, the point is that ones that don't may be missed. In addition, the alignment must involve at least minlen symbols of the prefix-sequence in the overlap. The option what specifies what kind of comparison is to be performed as follows:

AS_FIND_OVERLAP:
  Just find a good alignment to the boundary of the d.p. matrix. From
  this extrapolate a rough overlap relationship without an alignment
  and return the result (if there is one).
AS_FIND_ALIGN:
  For this option, further go to the trouble of computing the alignment
  and store it in the overlap message.
AS_FIND_ALIGN_NO_TRACE
  A hack by SAK...don't compute delta encoding.
AS_FIND_QVALIGN:
  NOT YET IMPLEMENTED. Will ultimately use quality values to compute
  the best possible alignment.

by the routine, and must be copied if it is to be retained beyond the given call.

In addition the following things are to be noted about this interface. `DP_Compare` is not input any orientation of the queried overlap, but it returns an orientation.
By definition DP_Compare only returns nonnegative ahg. Hence it internally switches the order of the fragments if the internally computed ahang is
- < 0, or
- = 0 and bhg >= 0

In this case it returns as orientation of the overlap
- AS_OUTTIE if called with opposite = TRUE
- AS_NORMAL if called with opposite = FALSE

If the ahang is positiv, the order of the fragments is preserved and DP_Compare returns as orientation
- AS_INNIE if called with opposite = TRUE
- AS_NORMAL if called with oppositce = FALSE

Hence any customer of DP_Compare must pay attention to the order of fragments and the orientation returned to interpret the results.

3. The flow of logic for ComputeUOMQualityOverlap

The function `ComputeUOMQualityOverlap` computes the overlap quality for a UOM mesg and is used in the quality filter step of the assembler. That means it computes the overlap between the two chunks and statistically evaluates the alignment using the quality values.

The function works as follows:
the overlap message is transferred to a struct `ChunkOverlapCheckT` that is stored in a hashtable for later lookup and which holds the following fields:
- **ChunkOverlapSpecT (orientation, cidA, cidB)** holds the ids of both chunks and the orientation of the overlap. The orientation should be with respect to the first id.
- **minOverlap, maxOverlap** the minimum and maximum overlap the two chunks can have. It is determined as ((alen – ahg) + (blen – bhg))/2
- **cgbMinOverlap, cgbMaxOverlap** are the same values as above as they are passed on by the unitigger. The unitigger uses a slightly different routine for computing the same entity.
- **errorRate** holds the errorRate with which the overlap was computed.
- **quality** holds the quality of the overlap. Initially it is filled by the unitigger as the percentage mismatches in the alignment
- **overlap** holds the actual length of the overlap
- **ahg, bhg** is used to store the ahg and bhg returned by DP_Compare.
- **min_offset, max_offset** ???? (historical? Saul?)
- **computed** indicates whether the values in the struct are based on a `DP_Compare` call or not
- **fromCGB** indicates whether the overlap was computed by CGB
- **hasBayesianQuality** indicates whether the statistical quality was already computed for that overlap.
- **AcontainsB**
- **BcontainsA**
- **Suspicious** indicates whether DP_Compare did not compute the overlap we expected (due to big beg, end values)

The ChunkOverlapSpecT is used to to lookup an overlap in the hashtable. So we transform the queried overlap into canonical form which does the following
- it sets the orientation ANTINORMAL to NORMAL and switches the chunk ids
- for INNIEs and OUTTIEs it puts the chunk into order such that the smallest ID comes first

All functions record whether the Ids got switched and switches them back before returning the result.

From now on we deal with a canonical overlap in the function. We fill the above ChunkOverlapCheckT with the values we have from the UOM (or an edge) and call The function `ComputeCanonicalOverlapWithTrace` (which has an equivalent in `ComputeCanonicalOverlap` <- they should be merged !).. Other functions first look up the hashtable before they compute an overlap.

In this function we lookup the sequence and quality strings of the sequence as well as their (ungapped) length.

Then we prepare the query to DP_Compare which involves the computation of the [beg,end] Interval as well as the flag opposite (indicating that the second sequence should be reverse complemented). These values are computed as follows (together with the behaviour we expect from DP_Compare):

```
case BA_BA:// ANTINORMAL
      // there can be no ANTINORMAL canonical overlap
    assert(0);
    break;
case AB_AB: // NORMAL
      // The normal canonical overlaps that can occur are (first frag
is A, second frag is B)
     1)   =============>
          +++++=============>

     DP_Compare should return the orientation NORMAL, and NOT flip the
fragments

     2)   ==================>
          ++++========>

     DP_Compare should return the orientation NORMAL, and NOT flip the
fragments


     3)   -----=====>
          ================>

     DP_Compare should return the orientation NORMAL, and DO flip the
fragments

     we do not complement the second fragment
     and set beg,end to the interval around the marked  region
     (+++ means positiv, --- negativ)
     we need the intended to check whether DP_Compare computed
     the overlap we intended to find.

    *beg = lengthA - canOlap->maxOverlap;
    *end = lengthA - canOlap->minOverlap;
    *intended = lengthA - canOlap->overlap;
```

```
    *opposite = FALSE;
    break;

  case AB_BA: // INNIE
    /* We need to reverse chunk B */
    The innie overlaps that can occur are (first frag is A, second frag
is B)
     1)   ============>
          +++++<===========

     DP_Compare should return the orientation INNIE, and NOT flip the
fragments

     2)   ================>
          +++++<======

     DP_Compare should return the orientation INNIE, and NOT flip the
fragments

     3)   ------=======>
          <===============

     DP_Compare should return the orientation OUTTIE , and DO flip the
fragments

     we DO complement the second fragment
     and set beg,end to the interval around the marked  region
     (+++ means positiv, --- negativ)
    *beg = lengthA - canOlap->maxOverlap;
    *end = lengthA - canOlap->minOverlap;
    *intended = lengthA - canOlap->overlap;
    *opposite = TRUE;
    break;

  case BA_AB: // OUTTIE
     The outtie overlaps that can occur are (first frag is A, second
frag is B)
      since DP_Compare cannot handle ANTINORMAL, that means we cannot
complement the first seuquence we compute a different
     beg, end which means that means we really ask DP_Compare for an
INNIE

     1)   <==============---
               ===========>

     DP_Compare should return the orientation OUTTIE, and flip the
fragments

     2)   <================
               ======>+++++

     DP_Compare should return the orientation INNIE, and NOT flip the
fragments

     3)       <=======----
          ===============>

     DP_Compare should return the orientation OUTTIE , and flip the
fragments
```

```
 we DO complement the second fragment
 and set beg,end to the interval around the marked  region
 (+++ means positiv, --- negativ)

*beg = -(lengthB - canOlap->minOverlap);
*end = -(lengthB - canOlap->maxOverlap);
*intended = -(lengthB - canOlap->overlap);
*opposite = TRUE;
break;
```

After `DP_Compare` has computed the the ahg, bhg and the orientation we have to translate back the results into an overlap. This is done in `adapt_overlap`. The overlap is always computed as ((lengthA – ahg)+(lengthB – bhg)) /2. Then we check whether DP_Compare computed indeed the overlap we intended it to compute, which means that we check the conditions of the above case distinction.

- For NORMAL overlaps it should always return orientation NORMAL.
- For INNIE overlaps it should not simultaneously return the orientation OUTTIE and flip the fragment Ids
- For OUTTIE overlaps it should either flip the fragment Ids or return orientation INNIE.
- In addition, we do not want to shift the overlap such that (allowing some slop) the sign of the intended ahg is changed. The intended ahg is computed by using the overlap field in the ChunkOverlapCheckT (which in turn is set to the edge->distance.mean or the field best_overlap_length from a UOM mesg). If one of this conditions is violated we mark the overlap as suspicious and set its length to 0.

We now return to `ComputeUOMQualityOverlap` where we return FALSE if we did not find an overlap and TRUE otherwise. Before returning we switch the back the order of the unitigs if they have not been canonical in the beginning.

4. Wish list for additional functions in the DP_Align family

In general a suitable interface would a function with input

- two sequences no matter what orientation
- the overlap orientation with respect to the first sequence
- a range like [beg,end] for the narrowing the search

The function would return the ahg, the bhg AND the length of the alignment.
This function would greatly reduce the overhead as it is now and should be done.

Any other wishes?