Fragment Placing

Knut Reinert

1. Overview

At the end of the chunk graph walker we will have placed a number of surrogates chunk instances that are used to merge into contigs. Initially the fragments list of these split chunk instances is empty. They originated from a split operation invoked either by stone throwing or gap walking. The consensus sequence of the contig will reflect this by using the consensus sequence of the surrogate for its consensus sequence but having a quality value of 0 in the region only covered by the surrogate.

Llllldlslllllllllllllslllslllfsldddlllllll000000000000000llllllsllleldcldlllllllllllllllllslllslllle000000000000000

The red, split CIs have to be assigned fragments from their parent CI. Unfortunately a parent CI may have several children competing for its fragments. Hence we have to apply a procedure that partitions the fragments and assigns them to either one of its children or discards them if they are apparently of bad quality. At the end of this procedure all fragments of the parent CI should be assigned or discarded and the parent dies.

2. Method

We basically cycle through the list of parent chunk instances in several stages. In each stage we assign a part of its fragments to its children.

Stage 1: for all chunk instances that have exactly one surrogate (they might stem from walks or from non-singleton stones) we put the fragments into the respective multialignment. So all remaining chunk instances have more than one child.

Stage 2: for all remaining chunk instances we first place fragments according to mate link information. That means we check whether the mate fragment is in an appropriate position in a scaffold. This excludes mate links to other split, still unscaffolded chunk instances. Whenever a mate link places the fragment without doubt into exactly one of its children, we place it there. So now some of the fragments have been placed into its children, where in their covering part a new consensus with quality values has been computed.

Lllllllllllldldlldlllllllllll00000llllll00sdlllllldllflllldlllllllllflllflllllllll0000000000000000000000llflflllflllllllll

Stage 3: in this round we cycle again through the list in order to place the remaining fragments. They either can be discarded as crappy or placed into one of their children chunk instances. The decision were they are placed is based on

a) the quality of their overlap with a non-zero quality contig consensus stretch.
b) the quality of their overlap with a non-zero quality chunk instance consensus stretch.

Since we trust the fragments placed by mate links and unique flanking unitigs more than surrogate sequences of repetitive chunk instances we prefer to place fragments based on overlaps of type a)
We say that a fragment f has a quality q_i in chunk instance c_i if it has a sufficiently large quality overlap with the consensus of the corresponding contig at that position, where quality overlap means the portion of the overlap that has non-zero quality. If no such overlap exists we set the quality to 1.0
We propose two strategies in descending order of preference:

1) If a fragment has at least one quality q_i< 1.0 then we place it in the best such location provided q_i < T1, say, where T1 is some threshold.
2) We wait until – through placement of other fragments – the fragment has a quality value in each of its children. Then we place the fragment in the contig of the child with the best quality. If the best quality is greater than some second threshold T2 then discard the fragment as crappy.

3. Needed methods

- **Merge fragment into child chunk instance resp. contig.** Replaces part of the old surrogate consensus/quality with a new combined consensus/quality.
- **Compute quality overlap with a consensus sequence that only partially has non-zero values.** We simply compute the alignment with the corresponding band which should be pretty tight. Then we analyze only the part of the alignment that has a non-zero quality.