Fragment Correction

In Fragment Correction we use information from fragment overlaps to make corrections in fragments. We then can take all existing overlaps and re-evaluate them with the corrected fragments. We expect that such overlaps will have a much lower error rate, hence we can use a lower error-rate threshold for unitig construction. Such unitigs should be longer and less repetitive.

How The Current Version Works

1. The original overlaps are duplicated (so that each overlap appears as both $A \rightarrow B$ and $B \rightarrow A$) and sorted.

   [This step is done by programs *get-olaps* and *rev-olaps*.]

2. A given range of fragments *Lo*…*Hi* is extracted from the fragment store and saved in memory. All overlaps involving these fragments are read and sorted with respect to the other fragment involved. The other fragments are then streamed from the fragment store, one at a time, and all of their overlaps with fragments saved in memory are processed.

   For each overlap the alignment is calculated. Let *A* be the fragment saved in memory and *B* be the fragment being streamed from the fragment store. The alignment is then used to cast votes about each base in *A* involved in the alignment:

   - In an exact-match region of length ≥ *k*, each base at least *x* bases in from the end of the exact-match region receives a CONFIRM vote.
   - Any substitution or indel that is surrounded by *v* or more exact-match bases casts a vote for that error. Only a single base insertion is allowed at any position in *A*.
   - The values of *k*, *x* and *v* are all parameters to the program.

   E.g., in the following alignment, assuming that *k*=7, *x*=1 and *v*=9, the last line indicates the votes that would be cast, where `C` means confirm, `I` means insert, `D` means delete, `S` means substitute, and a blank indicates no vote:

```
acgttacgtaccag-taattagcattacgattagcatattac-t
acgttacgtacctggtaattag-att-cgactagcatattacgt
            ^ ^       ^   ^   ^           ^

 CCCCCCCCCC S         D       S CCCCCCCCC I
```

3. After all overlaps with the fragments saved in memory have be processed, the votes are “tabulated” as follows:

   - A position with 2 or more confirm votes is left unchanged.
   - A position with 0 confirm votes and a strict majority of 2 or more votes for a specific correction has that correction recorded in the output file.
   - A position with exactly 1 confirm vote but also a strict majority of 6 or more votes for a specific correction has that correction recorded in the output file.
   - All other positions are left unchanged.

4. The corrections are written to a binary file called *frag.cor* . An indication is also written about the overlap degree on each end of the fragment. If the degree is less

than a specified threshold, an indicator bit is set. This is because a fragment with very few overlaps cannot accumulate enough votes to make any corrections using the above rules.

[Steps 2–4 are done by program *correct-frags*.]

5. After corrections have been recorded for all the fragments, overlaps are recalculated based on the corrected fragments. This process mirrors the one used to determine the fragment corrections.

   First a range of fragments *Lo*…*Hi* is read from the fragment store and saved in memory. Then the fragment corrections for these fragments are read and applied to them.

   Next, all overlaps involving these fragments are read and sorted with respect to the other fragment involved. The other fragments are streamed from the fragment store, one at a time, with each corrected prior to aligning with fragments in memory. For each overlap, the alignment is calculated and a revised OVL message generated if the resulting error is suffciently low (or if the low-degree bit described above was set).

6. As an option, we also allow a `.CGB` file to be input. From it we read the FOM messages (which indicate fragment overlaps off the ends of unitigs) and we create a file listing any of those whose overlap was processed and had too high an error rate. If we delete these overlaps from the original set of overlaps we hope to preserve all unitigs in the `.CGB` file, and possibly extend them.

7. We also build a map of fragments to unitigs. We use this to filter overlaps. We keep an overlap iff its fragments are in the same unitig or are in two unitigs that are connected by a low-corrected-error overlap. Thus, we eliminate overlaps between unitigs that do not have a “good” overlap.

   [Steps 5–7 are done by program *correct-olaps*.]