Alternate Assembler

Eric L Anson

1. Overview

The alternative assembler is an adaptation of the assembler described in "Algorithms for Whole Genome Shotgun Sequencing" (Anson & Myers, RECOMB 99). The assembler described in the paper returns an ordered set of contigs connecting two adjacent markers on the genome. The alternate assembler will return such an ordered set for all pairs of input markers that are confirmed to be adjacent.

The alternate assembler will sit between the chunk graph builder and either the consensus module or the repeat separator/resolver, so it might be more apropos to call this an alternate chunk graph walker. One difference between this and the chunk graph walker is that the alternate assembler is not designed to be incremental. Another is that the alternate assembler builds between markers, so such markers are expected as part of the input.

2. Memory Usage

The majority memory consuming data structure is the list of footprints (contigs). These are analogous to the extended chunks described in the design documentation for the chunk graph walker. The footprints are connected chunks, together with information on confirmed neighbors (other footprints with confirmed mate edges) and approximate location relative to the markers being connected. Other potentially large data structures are the list of markers and their status and the list of reads with an indication of which footprint(s) they lie in.

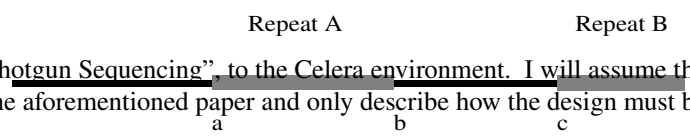
3. Interface

The input for the alternate assembler will include the chunk messages, the link messages, and the distance messages as described in the Prototyping I/O Conventions document. In addition access to the fragment store will be required to obtain the sequences for some fragments. This will allow the alternate assembler to look for missing overlaps. As stated in section 1, partially ordered sets of markers will be required by this assembler. It is not clear to me at this point whether this will dictate a separate input, or whether this list will be inferred from the link messages. If the markers do not form distinct sequences, the program will have to decide on chains of markers to assemble. Lastly I may end up using the branch point messages and/or the repeat item messages. I will determine this of course by the final draft of this document.

The output of the alternate assembler is a set of ordered sets of contigs, each set connecting a sequence of markers. In addition to the order, an approximate distance between the contigs is known. The form this output will take will be dictated by the module that will follow it (either consensus or a repeat separator). However, even though the next module won't use it, I believe it would be useful to have the alternate assembler output a message specifying the sequence of markers connected by each ordered list of contigs.

4. Design

The design issues here are how to adapt the algorithms described in, "Algorithms for Whole Genome



Shotgun Sequencing", to the Celera environment. I will assume that the reader can obtain a copy of the aforementioned paper and only describe how the design must be adapted.

4.1. The Database Query

The assembler described in the RECOMB paper tried to avoid the expense of calculating the entire overlap graph. When overlap information was needed a *database query* was performed. This allows inter-marker assembly to be treated as pseudo local problem that can be completed with much fewer fragment comparisons than building the entire overlap graph. Assembling the entire genome in this manner will require fewer fragment comparisons than building the entire overlap graph by a factor approximately equal to the coverage.

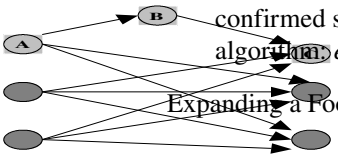
The database query takes a sequence (usually a subsequence of a read) and returns all the reads that overlap with it up to some maximum number. It also returns the leftmost right edge, the rightmost left edge, the leftmost left edge, and the rightmost right edge of any repeats it finds. Using this information and judiciously querying the database with subsequences of reads when required, one can determine all repeat edges within a footprint. One may also attempt to span short repeats when expanding the footprint.

The amazing Celera assembly team has demonstrated that building the entire overlap graph is not too expensive an operation. Instead of the database query operation, therefore, we have access to the overlap graph. A query of the database with a read would be accomplished by looking at the edges of the read in the overlap graph (an inexpensive operation). The two major differences between the operations are that the branch point messages only contain the leftmost left edge and rightmost right edge of a repeat and the furthest point from those edges there is evidence that the read is repetitive. In addition one cannot do a query using only a subsequence of a read. For example, look at figure 1.

If this fragment were being used to expand a footprint to the right, a second query to the database would be made using the unique portion of the fragment between points **b** and **c**. We do this hoping to find a read which spans repeat B. The overlapper, however, will not record branch point **c**, though it will indicate that the right end of the read is repetitive. One can sometimes simulate querying the data with a subsequence of a read (e.g. the portion of the fragment to the left of point **c**) by searching the edges in the overlap graph for overlaps that extend some minimal distance into the subsequence desired. If repeat B were of very high frequency, than the number of overlaps to search through would be huge. The implementation of "superrepeats" by the overlapper should prevent this from happening. Notice that if we looked at the subsequence to the right of point **c** instead, than such a search of the overlap graph would not simulate querying the database with that subsequence. This should not cause us problems here though.
While it would be possible to try to implement the alternate assembler using the overlap graph to replace the database query as outlined above, it makes more sense to tie the assembler in after the unitigs have been formed. This causes us to think not so much about how to implement a database query, but how to implement the actions (expand & take step) which use the database query.

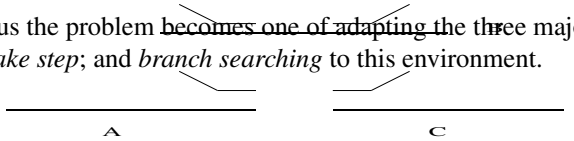
4.2. Unitigs vs. Footprints

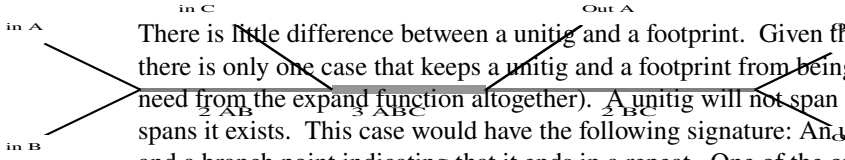
The overlap graph contains more information than is needed to do the assembly. The goal of the unitigger (chunk graph builder) is to trim the extraneous information and to group the fragments into unitigs (chunks), which are contigs which cannot (sensibly) be assembled any other way. Each footprint as described in the paper will contain whole unitigs (i.e. No unitig will be split and placed in separate footprints). This, together with the fact that the overlap graph is much larger than the unitig graph, leads us to believe the alternate assembler should take advantage of the unitig builder. This has the added benefit of allowing take step not to rely purely on overlapping pairs of mate edges for



confirmed steps. Thus the problem becomes one of adapting the three major components of the algorithm: *expand*; *take step*; and *branch searching* to this environment.

Expanding a Footprint

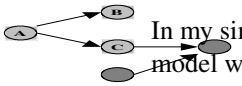




There is little difference between a unitig and a footprint. Given the simulation described in the paper there is only one case that keeps a unitig and a footprint from being equal (and thus eliminating the need from the expand function altogether). A unitig will not span a short repeat, even if a read that spans it exists. This case would have the following signature: An unitig would have multiple out edges and a branch point indicating that it ends in a repeat. One of the out edges would point to an unitig with only one in edge. This unitig should have branch points indicating that its ends are unique. It would have a single out edge pointing to an unitig with multiple in edges. This unitig should have a branch point indicating that it ends in a repeat. Figure 2 illustrates this. If this case occurs, expand should join the three unitigs into a single footprint. If the world worked the same way as the simulator described in the paper, then this is the only case expand would have to worry about. Unfortunately there are a number of cases the simulator didn't address. They are:

1. Staggered entry and exit points in repetitive footprints
2. Chimeric reads
3. Missing edges between reads
4. Polymorphisms

1. Staggered Entry and Exit Points



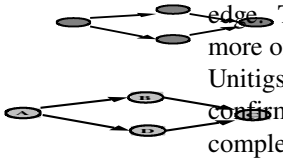
In my simulator I assumed all occurrences of the same type of repeat were identical. In reality a better model would be to make each occurrence a slightly mutated **subsequence** of some master sequence. So in the version of my assembler which is already built, repetitive footprints have multiple branches on each end. A more accurate model (that I must build here) will have in and out branches scattered along its length. Mostly this will only effect the forming of repetitive footprints and the searching of branches (described later). It is mentioned here since it is possible to have a situation where it looks like you're spanning a short repeat, but are in fact spanning a section between a entrance branch and an exit branch. This is illustrated in figure 3. Notice in this figure the section labeled ABC would appear as a short repeat. A fragment which stretched from section AB to section Out A would appear to span this repeat. Spanning using this fragment would be incorrect, however, as this would force the path B to exit the repeat along Out A instead of Out B. Thus it is important to make sure the footprint spanning the repeat is not repetitive.

2. Chimeric reads

This should be a rare case, but one whose signature may be distinct enough to catch. A chimeric read will result in a unitig with two out edges (and the analogous case with two in edges). Each of these will be the lone edge into another unitig. One of these (the chimera) will probably be a singleton. Both will have branch points indicating they end in a repeat, though the originating unitig will not, nor will it be repetitive. The chimeric unitig will also have left and right branch points at the same location. It should have one out edge going into a unitig with two in edges (the analogous case). If a chimeric unitig is discovered it should perhaps be marked as such and the other two unitigs should be joined into a single footprint. Figure 4 illustrates this case.

3. Missing Edges

The overlapper is not perfect and will miss approximately 1% of good overlaps. Also it is conceivable that a unfortunate read will have just a hair too many sequencing errors and miss an edge. These missing edges may just cause unconnected reads or unitigs that can't be dealt with. On the other hand, missing edges can cause multiple unitigs. The form these take will depend on which edges are missing, but the basic motif is a unitig with two out edges. Each unitig it points to (they will probably be small) has a single in and out edge. The out edges of these unitigs point to the same unitig. There can be variations of this with one or more of the middle edges missing. There would be no branch points connected with these splits and joins. Unitigs found to be split due to a missing edge should be joined into a single footprint. This case could be confirmed by doing a comparison of fragments from the middle unitigs. This case occurs in a more complex form when dealing with a repetitive section of the genome. This will be discussed later.



4. Polymorphisms

It is not clear to me how we want to handle polymorphisms. As I understand it, polymorphisms are differences between two copies of a chromosome. SNPs should cause us little to no problems but larger polymorphisms will. Is our goal to put polymorphic sections in a single contig so that the multi-alignment shows half the rows to say one value and the other half to say another? Do we want them in separate contigs? This question refers to heterogeneous regions of the same chromosome. If a read comes from a piece of DNA that has been uniquely mutated, then the goal will clearly be to exclude that read from the assembly. Obviously I need clarification on this section before I decide what, if any, action Expand should do when it encounters polymorphisms.

All of the above examples give simple cases. Before I'm ready to implement Expand I will need to look in further depth at complicating factors. For example a chimera that overlaps a repeat will have a different signature. The assembler depends on expand to be a very safe operation (i.e. with the exception of repetitive footprints, separated portions of the genome are not put together in the same footprint.) While absolute safety may be impossible, I must be sure all joining of unitigs done by Expand has a high probability of being correct.

4.3. Take a Step

The procedure for taking a step benefits greatly from building on top of the unitig graph. In the original algorithm I needed to rely on mate pairs which overlapped to define a confirmed step. Now the distance between two reads lying in the same unitig is known. So we can define a *confirmed step* from a footprint to be any unitig which contains two *nonconflicting* mate pairs in the footprint. By nonconflicting we mean that the relative location of the reads on each footprint is not in disagreement with the expected distance between mate edges.

4.4. Repetitive Footprints

In the paper I assumed all occurrences of the same repeat were identical. This allowed me to create repetitive footprints, which are a collapsing of all occurrences of a low frequency repeat. These footprints have multiple branches at each end (one for each repeat occurrence). In reality repeats are mutated subsequences of some master sequences. To model the subsequences I need to allow for staggered entry and exit points from a repeat (figure 3). Thus my repetitive footprint would have in and out branches at places other than the ends. It would not be too hard to adjust the program for such cases. The biggest change is that currently I build an entire repetitive footprint as soon as I encounter it. (By expanding until a repeat edge is reached.) This would no longer be feasible since upon reaching a repeat edge I would not be able to tell whether one of the branches was a continuation of the repeat. When building repetitive footprints in an adjust program one would expand until an out branch (relative

to the direction being expanded) is reached. A procedure would have to be included for joining together footprints found to be part of the same master repeat, but the functioning of the algorithm should not be adversely effected.

The major difference is the 'repeat tangles.' In a perfect world each segment of figure 3 (in A, in B, AB, in C, ABC, etc.) would be a single unitig in the unitig graph. Unfortunately the various mutations in different occurrences of a repeat lead to repetitive sections being broken up into a tangle of unitigs which are constantly joining and splitting. One solution would be to try to collapse this tangle into the simpler structure of figure 3. Ways of doing this are being thought about.

Each unitig has a discriminator number that indicates whether it is unique. While this doesn't let us safely classify all unitigs as unique or not, we can find a large subset of the unitigs which can be safely called unique. So another way around the repeat tangle is to search the tangle for these 'discriminator unique' unitigs. These are the out branches of the repetitive area and the algorithm may carry on from here as before.

5. Limitations

The two major limitations have been stated earlier, but I will list them again here:

2. The algorithm is **not** incremental. It was designed under the assumption that all the reads have been sequenced and are available. There is no provision for rearranging the assembly to accommodate new information.

3. The algorithm only assembles stretches of the genome between markers. Thus to assemble the whole genome, markers which span the genome must be supplied to the algorithm.

6. Status

The alternate assembler is currently in the design phase. Although a version using simulated data implementing the design as described in the RECOMB 99 paper is in working condition.

AUTHORS

Eric Anson:

Created February 22, 1999