Celera Assembler: Requirements and High-Level Design

Abstract

In a series of ad hoc meetings the Informatics Research Team (IRT) has begun to delimit the requirements and method for the *incremental* assembly of a whole genome shotgun data set. The prevailing concern is that we get the job done: while in principle we are confident that sufficient information is present for assembly to be possible, there is the daunting problem of actually doing so, especially given the time frame available and the further requirement that assembly proceed concurrently with data collection. Our design is based on the idea of finding mate-consistent melds of the uniquely assemblable sub-contigs of the current fragment data set. A key motivation for this design choice was its conceptual rigor and simplicity.

1. Assumptions

The nature of the underlying genome that we will be sampling and the nature of the sequencing data we will be pulling from the Data Acquisition Team (DAT), fundamentally affects our chances of success and the nature of the assembly strategy that needs to be employed. Below we list the assumptions we are making in this regard. Each assumption is followed by a one word assessment of its importance and any relevant notes about the effect of its violation.

1. We assume that the DNA will be coming from *only one donor*. This implies that each SNP will in expectation occur at 1/2 the current coverage.

   CRITICAL: While we could consider strategies like 8x of one donor and 2x of "others", the impact of structural polymorphisms and to a lesser extent SNPs, especially while still at low coverage, will greatly confound the ongoing attempts at assembly.

2. The number of false mate pairings is $<$ 1%.

   CRITICAL: We could tolerate higher rates but would have to redesign/retune for this and we believe it is a critical factor in our chances of success.

3. Structural polymorphisms are quite rare, on the order of hundreds or less between any two genomes.

   CRITICAL: While these can be separated from repeats on the basis of coverage statistics, this is our weakest indicator (see below) implying that the assembly of all such polymorphisms will likely require human inspection and potentially interaction.

4. The sequencing error rate in the "sweet" part of a read is $<$ 2%, and constitutes on average 400-450 bases. Reads are 1000 bases or less in length.

   IMPORTANT-CRITICAL: Problem difficulty increase exponentially with error rate, it should be kept as low as possible.

5. SNPs average 1 in 1000 bases, up to 1 in 100 in hot spots.

IMPORTANT: Rate effects our ability to use micro-heterogeneity to separate repeats, what would be one of our strongest repeat resolvers (see below) if it were not for the presence of SNPs.

6. Peak input of 200,000 reads per day. These may arrive in a batch or in a stream, but a read will never be pulled unless its mate is also available (or it does not have a mate).

   DESIRABLE: Rate affects the hardware required.

7. Only successful reactions and reads are passed to us, with vector successfully removed in all regions where data quality is $<$ 5%.

   DESIRABLE: We'll have our hands full as it is and would appreciate not having to worry about vector removal, please do the best job possible. We anticipate that we will nonetheless have to pay attention to the possibility that some is missed.

Assumption 1 requires further discussion. While we appreciate the commercial and scientific significance of polymorphism, we regretfully insist on the restriction to one donor. On the other hand, once we're confident that we have a solid assembly of one individual's DNA, we can then go on to do as much random sequencing of lots of other individuals in a subsequent round of sequencing in order to gather a potentially unlimited amount of polymorphism information. Thus in some sense, doing 8x of one individual and 2x of others is still quite possible provided: (1) all 8x of the one individual is done *first*, and (2) at 8x the data has come together sufficiently that the additional polymorphisms will no longer confound the assembly. We can make this decision dynamically as assembly will be proceeding in synchrony with the data sequencing: if we get to 7, 8, or 9x and things look good, then we can build some additional clone libraries and go for the genetics.

We close with a number of fundamental assumptions that we're all aware of and have been much discussed, but which may require further examination. First we assume that the reads are as close to uniformly sampled from the genome as possible, i.e., there is no strong bias in the sampling and cloning of fragments for sequencing. For example, the fact that PCR fails on mononucleotide stretches implies that if this is the only strategy we use then we will have sequence gaps at each such site that are also not spanned by a mate pair. While this is not problematic to assembly itself the resulting assembly is less desirable then one where all contigs are linked and ordered. Second we assume that the mix of insert lengths is such that there are a sufficient number of mates of a given length to, with sufficiently high probability, jump every ubiquitous repeat (UR) or cluster of URs. Again the affect on assembly is gaps in coverage. The current decisions with respect to these two items are (1) we will sequence a 4 to 1 ratio of 2Kb inserts and 9-10Kb inserts (for an average effective insert length of 4Kb), and (2) template prep will be a 4 to 1 ratio of PCR- to plasmid-based. There does not appear to be any strong analytic argument or simulation based verification supporting these ratios, although they do of course seem intuitively plausible. The DTT plans to conduct further simulations in the near term to try to provide further support for these decisions.

2 Requirements

The core objective for the DTT is to assemble the shotgun data set with as few unresolved or ambiguous regions as possible. At the end of the project, the deliverable is a set of partially ordered contigs, their consensus sequences, and their relation to all known physical markers. The desire is for the partial order to be as close to a total order as possible (i.e., for there to be as little ambiguity in the ordering as possible) and we further expect our results to permit the correction and refinement of the current physical maps, especially the radiation hybrid maps.

Below we list the requirements that we believe our design must meet in order to achieve the core objective and to provide commercially useful information at the first opportunity. Each requirement is followed by a set of bullets giving the rationale for it. The requirements are roughly in order of greatest to least importance.

1. We must assemble the data concurrent with its production by the DAT. Interim assemblies of the current aggregate data set will be produced on a regular periodic basis from the outset of data production. In tech-terms, we must build an *incremental assembler*.
   - Unacceptable to deliver answer 3-6 months after the data has been collected.
   - Increases commercial and scientific value of data that has been collected.
   - Confirms that clone sampling and sequencing are proceeding as desired.
2. The assembler subsystem will *pull* data from the DB produced by the DAT by polling it on a regular basis. The assembler subsystem is expected to never be too far behind the production of data, at most 48 hours, so older data in the massive DAT data set can be regularly archived if desired.
   - The pull design provides greater flexibility in the event of outages and subsequent restarts. It is consistent with an object-oriented approach.
   - The average processing rate of the assembler will be designed to be twice that of peak flow, but variance in this rate recommends a pull-design.
   - Polling as opposed to message passing should be sufficient given the frequency with which data is transferred.
3. The assembler will utilize all physically mapped information of *sufficient* reliability and all pairs of distant mates (e.g. BAC-ends). The assembler will be able to accommodate updates to this repository of information. The current list of information includes STSs, BAC-ends, and known BAC sequences.
   - Having the information correlated with existing information is essential for scientific and commercial purposes.
   - A by-product will be the ordering of this information, also of scientific and commercial value.
   - Congruence with this information is an essential validation of the progress of our assembly.
4. We will find and mark all ubiquitous repeats (URs) and all specified contaminants.
   - We need to mark URs for our job and it makes no sense for the Data Mining Team to subsequently duplicate this effort.
   - Detecting contaminant sequence such as *E. Coli* is the same computation as that for URs so it makes no sense for the Data Acquisition Team to duplicate this effort.
5. We will produce consensus sequences of each contig sufficient for annotation, and do so for each interim report.
   - This is traditionally part of assembly and the DAT team already has the software and expertise to readily do this.
   - The Anson/Myers round-robin approach that we will use provides excellent alignments with sufficient speed to permit its execution for each interim report.
6. All interim large scale data structure – sequence index, overlap index, and genome snapshot – will be stored on internal raid disks and check-pointed regularly. Interim reports of a genome snapshot will be transmitted to an external data base on a regular basis.
   - We will need all the bandwidth to and from these private disks for our ongoing assembly computation.
   - On-line results may not be worth looking at when we have the occasional

blowout.

- We envision running a number of consistency checks and producing consensus sequences for the interim reports, but cannot afford the time to produce these on an on-line basis.

7. One must be able to delete (a small number of) fragments as well as add them.

- Data tracking errors and other unanticipated problems may cause bad data to enter the system.

8. We will supply a stand-alone tool for both internal and external use that will take any two assemblies of the fragments and compute an alignment or correspondence between them.

- Annotation of an existing assembly must be updated and this provides a clean approach.
- Users will want to understand how an assembly has changed.

The first requirement is clearly essential but unfortunately is also the most demanding and difficult as its immediate implication is that contiging will change over time including things coming apart that were formerly together. Such *reversible* schemes are inherently more complex and difficult to structure and code. An important implication of this requirement to the downstream teams is that they will have to be prepared to deal with interim reports in which the data is put together differently over time. We are designing a level-of-confidence hierarchy so that one need not waste time on the contigs that are very preliminary and speculative, but focus on those that are solidly supported. Nonetheless, the non-linear history of contig formation must be anticipated. Tentatively, the assembly team proposes to deliver a separate program (Item 8 above) that will compute an alignment or mapping between any two assemblies that will facilitate feature mapping. The specification of this program has been given in a separate document (see "Feature Tracking Straw Man").

3 Design

While our initial impression was that we would need a very flexible AI-based strategy in order to accommodate all the requirements above, we have been successful in creating a design which consists of a sequence of well-delineated steps each of which achieves a computation that can be rigorously and functionally described. Thus if a particular step on input x produces $y=f(x)$, then to achieve an incremental assembly all one has to do is arrange for the software to compute $f(x+\Delta)$ for some small update of x in an efficient way. Such an approach is particularly appealing because it conceptually gives one a concisely definable and hence comprehensible connection between the input and output of the assembler.

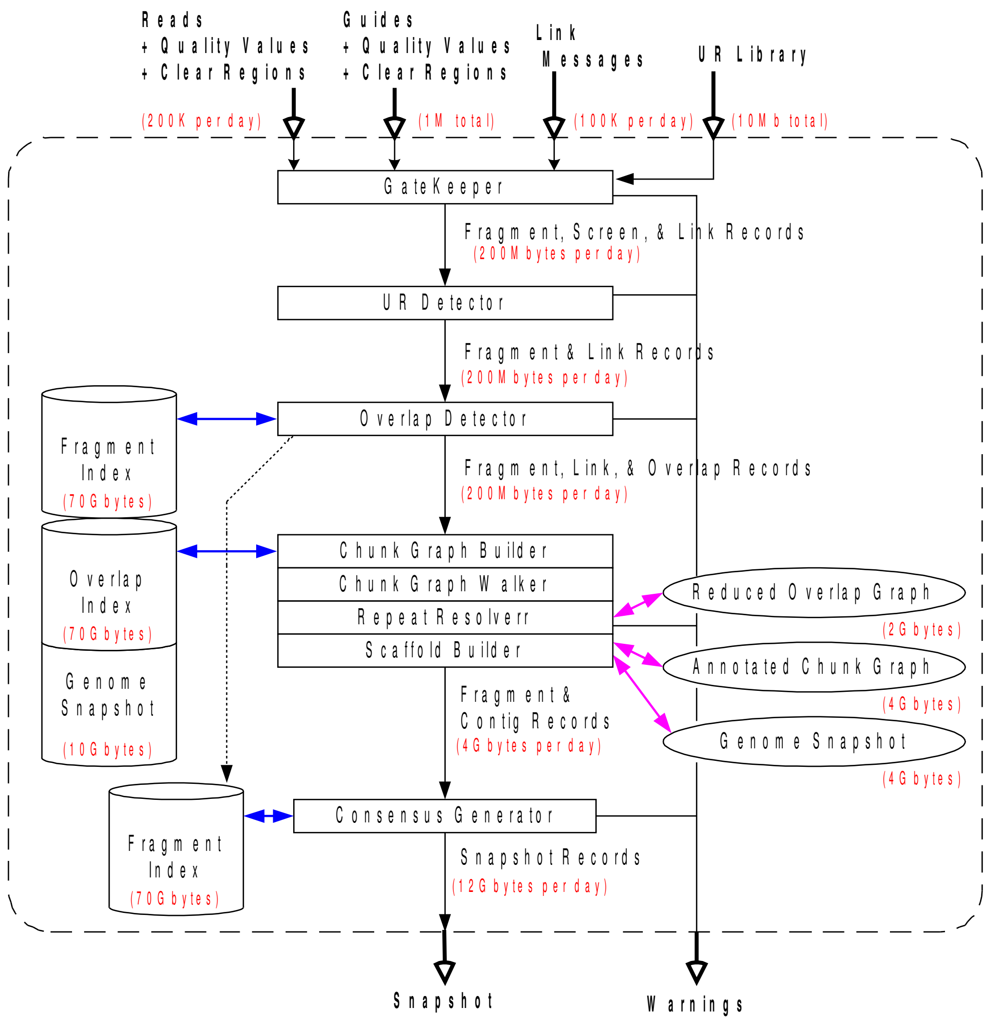


Figure 1: Celera Assembler Data Flow

An interesting advertising/publicity possibility is to develop an interface which shows “genome assembly” taking place in real-time as it progresses. We anticipate building a technical version of such a display for our own diagnostic purposes and it would be a straightforward exercise to arrange for a simple animation for visitors, clients, and public presentations.

Figure 1 sketches a preliminary data flow model showing the items and processes transforming them. Input to the system consists of *reads*, *guides,* and *links*. A read models the results of a sequencing reaction and consists of a sequence, the quality values for the base calls in the read’s sequence, and an initial *clear region* that is assumed to be of acceptable quality and free of vector. A guide is any known piece of DNA sequence obtained from outside the main Celera sequencing pipeline, and we currently anticipate four major sources of such information: (1) end-sequenced BACS, (2) mapped STS's, (3) finished BACs, and (4) unfinished BAC contigs. Each guide is modeled identically to a read. A *fragment* is either a read or a guide. The length of a guide will always be assumed to be on the order of the length of a read. Thus large known segments of DNA, such as finished BACs, must be input as a series of perfectly interleaved 500bp segments. Critical to our ability to assembly a set of fragments, is the further input of a collection of link relationships between fragments. In general a link message specifies that two fragments should occur within a given distance range from each other in a valid assembly. So link messages include the mate relationships between 2K and 10K reads from the Celera pipeline, the distance relationship between end sequenced BACs, and the possible ordering relationships between STS's. Link messages can also serve as user constraints on the solution: a user picks a pair of fragments and a distance relationship between them thus asking the assembler to incorporate such information into its results.

Initially all fragments are time stamped when they enter the system. All ubiquitous repeats and any DNA contaminants in a fragment are found and recorded. The fragment data and any tags are forwarded to a separate overlap subsystem that stores the record in the fragment index on a disk and then determines all overlaps between the fragment and all proceeding fragments (that are not predominantly contaminant DNA). The fragment index disk system is attached to the overlap subsystem because this system places the greatest demand to said disk system, all other accesses to fragments from the assembly sub-component are transmitted via the connecting network. Similarly, overlaps found are not stored on a disk associated with the overlap subsystem but passed to the assembly subsystem for storage on an overlap index, again because this subsystem places the greatest demand on the given disk system.

The assembly subsystem consists of a pipeline of four functional units – unitig builder, unitig walker, repeat resolver, and scaffold builder – whose access pattern to the shared-memory structures of the assembler will require a careful synchronization design. By pruning the overlap graph of transitively inferrable edges (to be defined later), a reduced overlap graph can be maintained in memory that is sufficient for construction of the chunk graph and snapshot representations.

On a periodic basis the current snapshot is output to an external agent after computing consensus sequences for each contig. This later computation requires rapid access to all of the fragment sequence and so has its own shadow copy of the fragment index forwarded to it over time (200Mbytes per day).

Tentative sizes of disks, in memory structures, and flow rates across network connections are given in Figure 1. In terms of generic 600MHz processors, we need 1 processor and 4Gbytes of memory to achieve the UR and contaminant detection subsystem. 4 processors and 4Gbytes of memory should suffice for overlap detection, and 4 processors with 10Gbytes of memory are estimated to be needed to perform assembly given the overlaps. 3 RAIDed disk systems will be needed for the various indices. The memory and processor estimates are fairly crude at this time. The hardware will be capable of performing the last day of computing in 12 hours. Thus, fault-recovery will consist of regularly check-pointing the snapshot and recomputing from the earliest checkpoint prior to failure. An extra chassis and RAID controller will be in-house so that down-time never exceeds more than a handful of hours in any stretch.

3.1 Input Specification: Reads, Guides, Screens, and Joins.

In this preliminary document we simply attempt to carefully list the information that is required in each of the primary input and output datatypes. We have tried as hard as possible to avoid any suggestion of actual implementation but to some extent such structuring issues are impossible to completely evade. It is our expectation that ultimately the specified information will be translated to and/or from whatever external formats are designed by the data support team.

A general convention for our specification is that entities have unique ID numbers in the `Accession`-field. These IDs are unique 64-bit integers assigned by the Data Management System, but to convey what type of object an ID refers to, we will type such references as `<X>_ID` where `<X>` is the entity name.

3.1.1 Fragments: Read, Guides, and Links.

The primary input to the assembler is a fragment message, either for a read or a guide. Guides are further categorized as either being (a) BAC-ends, (b) pseudo-reads from unfinished BACs, (c) pseudo reads from finished BACs, and (d) STSs. The difference between the messages for reads versus guides are as follows.

1. Every guide has an associated locale whereas this field is undefined for reads. The interpretation of the locale is different for each kind of guide, but is always expected to be a 64-bit UID produced by the Celera database. For the BAC-based guides it is a UID assigned to the particular BAC from which the guide came. The assembler does not care about the nature of this UID other than that a distinct integer be given to each BAC. If over time, end reads, then contigs, and finally finished sequence for a BAC become available, the same BAC number should be given to the associated guides for that BAC. For

an STS guide, a distinct locale number should be assigned to each bin formed when a sufficiently high LOD score is used to order STSs. We suggest 6.

2. Every read has a quality vector and a clear range whereas guides need not. In such a case the quality vector field is NULL and the clear range is the entire fragment.
3. Each contig of an unfinished BAC and each finished BAC is assumed to have been partitioned into a set of neatly overlapping pseudo-reads that are given as guides to the assembler. For these pseudo-reads, the assembler will expect to be given the interval of the underlying BAC from which the pseudo-read was excised in the locpos interval of the record. This field is defined only for unfinished and finished BACs. In the case of the several unordered contigs for a given unfinished BAC, simply assign a disjoint interval to each contig and then give the position of each pseudo-read of a contig with respect to the assigned interval.

The connections between mates and guides are established separately via link messages to be described below. A fragment message requests either that a fragment be inserted or deleted. If it is to be deleted then the only other information required in the message is the unique accession number of the fragment. If it is to be inserted then one needs to also specify if the message is for a read or a particular type of guide, the locale and locale position information (if a guide), the source of the data, the time of entry, the sequence, the quality value vector, and the clear range for the fragment. It is assumed that the sequence/quality vector of every fragment is less than 1000 bases in length.

```
FragMesg: record
            action:    scalar (AS_ADD,AS_DELETE)
            accession: Fragment_ID
            variant of action:
             AS_ADD: record
                        type: scalar (AS_READ,AS_EXTR,AS_TRNR,
                                      AS_EBAC,AS_FBAC,AS_UBAC,AS_STS)
                        locale:     union(BAC_ID,STS_Bin_ID)
                        locpos:     SeqInterval
                        source:     "description of data source"
                        entry_time: time_t
                        sequence:   string(char)
                        quality:    string(bytes)
                        clear_rng:  SeqInterval
                      end
            end
          end

SeqInterval: record
               bgn, end: int
             end
```

Note carefully that we adopt the convention that intervals are specified as a pair of positions within a sequence and positions are the points between symbols of the sequence. The leftmost position is numbered 0 so that for example, (0,4) specifies the first 4 symbols of a sequence, (2,2) specifies the position between the second and third symbols, and so on.

After the relevant fragments have been added to the system one may then add (or delete) pairwise distance constraints or *links* between them. A link message always contains the type of link being added or deleted and the two fragments involved. If a link is being added then one also needs to specify the time of entry, a reference to the distance record specifying the distance range between the fragments, and a scalar indicating whether the fragments are in the same, opposite orientation, or unknown orientation. Note that mates and BAC guides are always in the opposite orientation with respect to each other. The distance constraint always refers to the distance between the 5' end of the two fragments, regardless of orientation. Links are divided into six categories according to the source of the link. AS_MATE links are for mated pairs of 2K and 10K, 50K, and transposon library end reads from the Celera sequencing pipeline. Mated pairs of transposon library reads must have the AS_OUTTIE orientation. Mated pairs of Celera and external reads (AS_READ and AS_EXTR) must have the AS_INNIE orientation. AS_BAC_GUIDE links are between end-sequenced BACs and AS_STS_GUIDE links are between paired STSs. The `AS_REREAD` link, permits one to specify that two reads were sequenced from the same end of an insert. These are rereads that were resequenced for some reason, e.g. the PCR-prep encountered a mononucleotide repeat and stuttered, and thus was resequenced with a plasmid prep. In this case neither the distance or orientation fields convey any information. The last two link types model user input constraints and may be between any pair of fragments in the system. The MAY_JOINs represent single links that may be incorporated if there is not conflicting information, and the MUST_JOINs represent infinitely weighted links that will be followed if at all possible. Finally, note fully that there should be exactly one distance record for each insert library, even if the library was designed to have insert sizes equal to that of another library. The reason for this is that the assembler will be empirically determining a distribution of observed mate distances and these distributions will be different, even for libraries designed to have the same mean distance

```
LinkMesg: record
            action:       scalar (AS_ADD,AS_DELETE)
            type:         scalar (AS_MATE,AS_BAC_GUIDE,AS_STS_GUIDE,
                                  AS_REREAD,AS_MAY_JOIN,AS_MUST_JOIN)
            frag1, frag2: Fragment_ID
            variant of action:
              AS_ADD: record
                        entry_time:  time_t
                        distance:    Distance_ID
                        orientation: scalar (AS_NORMAL,AS_INNIE,AS_OUTTIE,
                                             AS_ANTI,AS_UNKNOWN)
                      end
            end
          end
```

The distance between guides and mates are specified in distance records that are passed to the assembly system as records requesting that a given distance entity be added or deleted. The message record specifies the action, the ID of the distance entity, and (in the case of insertion) the normal distribution from which the distances were sampled. The field `mean` gives the mean of the distribtuion and `stddev` gives the standard deviation. Thus, for example, 99% of all links referring to a particular distance message will be of length in `[mean-3stddev,mean+3stddev]`.

```
DistanceMesg: record
                action:        scalar (AS_ADD,AS_DELETE)
                accession:     Distance_ID
                mean, stddev:  float
              end
```

3.1.2 UR and Contaminant Library:

The UR and contaminant library is assumed to be an accruing input that has some initial set of entries at startup of the assembly process and for which entries are only added over time. The library consists of a collection of items to be screened against. Each item has a scalar type declaring it to be either a UR or a contaminant, a unique accession number, a small enumerative int identifying the type of repeat (e.g. Alu, Line, etc.), a description of its source, the sequence of the item, a scalar indicating that one should screen only proper overlaps or any matching substring, and the stringency of the matches to be considered significant. There is no length restriction on the sequence (e.g. one may give the entire sequence of *E. Coli*) and the variation parameter must not exceed 10%. The relevance field is bit vector used indicate how each screen item or fragments matching the screen item should be processed. It may be used differently for contaminants and repeats. Currently, if a screen item's relevance value binary-ands with AS_OVL_HEED_RPT (1), the overlapper will not look for potential overlaps based on matching intervals.

If the value binary-ands with AS_URT_IS_SIMPLE (8), this identifies the screen item as a simple repeat (or heterochromatin) and allows the screener to consolidate matching intervals better.

```
ScreenItemMesg: record
                  type:       scalar (AS_UBIQREP,AS_CONTAMINANT)
                  acession:   ScreenItem_ID
                  repeat_id:  [0…Num_of_Repeat_Types-1]
                  relevance:  bit-vector
                  source:     "description of data source"
                  sequence:   string(char)
                  variation:  real in [0,.1]
                  min_length: int
                end
```

For each type of repeat, one is further passed a record containing the repeats enumerative int, common name, and length in base pairs

```
RepeatItemMesg: record
                  repeat_id:  [0…Num_of_Repeat_Types-1]
                  which:      string(char)
                  length:     int
                end
```

3.2 Output Specification: Genome Snapshots.

The principal output of the Celera assembler is a snapshot generated on a regular periodic basis, tentatively once per week. Please see the glossary of the “Feature Tracking Straw Man” document for a description of many of the terms (such as snapshot) used in this section.

On a conceptual level a snapshot consists of a set of fragment records giving a few bits of information about the fragment *not already known to the DMS*, a unitig graph of all unitigs (a.k.a. chunks) and the essential edges between them, a scaffold graph of contigs and all the mate-induced edges between them, and the assembler’s best guess at the correct scaffolding of the contigs., modelled as a set of vertex-disjoint paths through the scaffold graph. A unitig is a sub-assembly of fragments that join in a unique way with respect to the current data set and either represent a correctly assembled set of fragments covering a unique stretch of the genome, or represent an over-collapsed set of fragments strictly interior to a repeat within the genome.

3.2.1 Augmented Fragment Record:

At the information modeling level, the first item to be output for a snapshot is a list of augmented fragment records, one for every fragment known to the assembler. Each fragment record contains:

1) the unique ID of the fragment,
2) a list of all matching regions found by the UR and contaminant screener, along with the matching item and the part of it that matched,
3) whether or not the fragment has been categorized as being chimeric, and
4) the clear range of the fragment (this may be changed by the assembler).

```
AugFragment: record
               accession:       Fragment_ID
               screened:        list of ScreenMatch
               in_chunk:        Chunk_ID
               chimeric:        boolean
               clear_rng:       SeqInterval
             end
```

```
ScreenMatch: record
               where:        SeqInterval in fragment
               what:         ScreenItem_ID
               repeat_id:    [0…Num_of_Repeat_Types-1]
               relevance:    bit-vector
               portion_of:   SeqInterval in screen item
               direction:    scalar (AS_FORWARD, AS_REVERSE)
             end
```

After the list of fragment records, comes what we expect to be a very short list of distribution records that give information on the distribution of mate lengths observed for those pairs both of which are in the same contig (and thus whose distance is known precisely). For each distance message referenced by a AS_MATE or AS_BAC_GUIDE link message, a record describing the distribution of observed mate-link distances is given. The record first indicates which distance ID it refines, followed by the minimum and maximum observed distances, followed by a histogram covering that range in `num_buckets` and a count for each bucket.

```
MateDistanceMesg: record
                    refines:      Distance_ID
                    min, max:     int
                    num_buckets:  int
                    histogram:    list of int
                  end
```

3.2.2 The Unitig Graph:

The next component of a snapshot that is output is a series of unitig records representing the vertices of the unitig graph and a series of unitig-link records representing the edges of this graph. Every fragment occurs in one and only one unitig. In an effort to keep an interim report to the smallest possible size, only the essential information about each unitig is transmitted, one can conceptually capture more convenient records of such things as an actual multi-alignment for a unitig in a postprocess. It is important to realize that this is necessary: towards the end of the project an interim report will be as large as 12 Gbytes and require 20 minutes to transmit on a Super SCSI at full throttle. A unitig record contains the following information:

1) A unique accession number assigned by the assembler.
2) The assembler's current usage of the unitig: `AS_UNIQUE` if the unitig is used in only one contig, `AS_CHIMER` if the unitig represents a single fragment that has been deemed chimeric, `AS_NOTREZ` if the unitig represents an overcollapsed repeat at least some of whose fragments could not be separated and placed in a unique contig, and `AS_SEP` if the unitig represents an overcollapsed repeat all of whose fragments have been assigned to unique contigs (a separable repeat).
3) The two extreme branch-point positions on this unitig (if any have been found).. The ends of a unitig are arbitrarily designated as the 'a' and 'b' ends. Branch points are signed, if positive the half line away from the center of the unitig is repetitive and the other half is unique, the reverse is true if the branchpoint is negative. A zero value indicates no branch point.
4) A consensus for the unitig and the layout of its fragments with respect to that consensus (see the following description of contigs) as a list of MultiPos records. Note that this list of fragments gives all the fragments in the unitig as well as the type of each fragment. The scalar value AS_UNITIG is never set in a unitig's list (but can be in a contigs list). The consensus sequence is gapped and the associated string of quality scores is of the same length as the gapped sequence, i.e. there is a quality score for every column of the multi-alignment, including those whose consensus call is '-'.

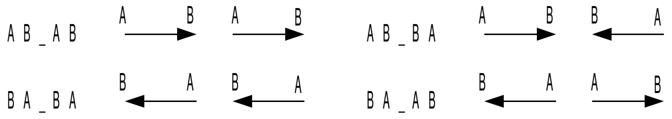
Every fragment given to the assembler is placed in one and only one unitig. On the otherhand this will not be true of contigs, as in some cases a unitig represents a repeat all of whose fragments could be resolved into specific instance of the repeat in various contigs.

```
Unitig: record
          accession:      Unitig_ID
          coverage_stat:  float
          status:         scalar (AS_UNIQUE, AS_CHIMER, AS_NOTREZ, AS_SEP)
          ~~occurences:     list of Contig_ID~~
          a_branch_point,
          b_branch_point: int
          ~~a_list:         list of UnitigOverlap~~
          ~~b_list:         list of UnitigOverlap~~
          consensus:      string(char)
          quality:        string(bytes)
          ~~reads,guides,~~
          f_list:         list of MultiPos
        end


MultiPos: record
              type:     scalar (AS_READ,AS_EXTR,AS_TRNR,AS_EBAC,AS_FBAC,
                                AS_UBAC,AS_STS,AS_UNITIG)
              ident:    union(Fragment_ID,Unitig_ID)
              position: SeqInterval
              delta     list of short
            end
```

A unitig link record repesents an edge in the graph that models all overlap *and mate* information that consistently positions two unitigs with respect to each other. It contains the following information:

1) The two unitigs connected by the edge/link.
2) The orientation between the two unitigs as follows:



3) Whether the two unitigs are (a) not connected by an overlap (AS_NO_OVERLAP), (b) connected by an overlap that does not involve a micro-satellite (AS_OVERLAP), or (c) connected by an overlap that does involves a micro-satellite repeat.
4) One link and an overlap is extremely reliable except when the the end of one chunk is one of the reads in the pair and there are no confirming overlaps from other fragments in that chunk. In this special case the is_possible_chimera boolean is set.
5) The set of links modelled by the edge includes a guide link (AS_BAC_GUIDE or AS_STS_GUIDE).
6) The mean and standard deviation of the estimated distance between the two unitigs. This is negative the overlap amount if the two unitigs overlap.
7) A scalar indicating how the edge is deemed to be related to the assembly: AS_IN_ASSEMBLY if it is consistent with the contigs and final scaffold chosen, AS_POLYMORHPISM if it is believed to be valid but indicative of a structural polymorphism, and AS_BAD if the links and overlaps within it are all believed to be false.
8) Finally, one is give a list of the pairs of fragments mates of all the links modelled by the edge.

```
UnitigLink: record
              unitig1:            Unitig_ID
              unitig2:            Unitig_ID
              orientation:        scalar (AB_AB, BA_BA, AB_BA, BA_AB)
              overlap_type:       scalar (AS_NO_OVERLAP,AS_OVERLAP
                                          AS_TANDEM_OVERLAP)
              is_possible_chimera: boolean
              includes_guides:    boolean
              mean_distance:      float
              std_deviation:      float
              status:             scalar (AS_IN_ASSEMBLY,
                                          AS_POLYMOPHISM,AS_BAD)
              jump_list:          list of Mate_Pairs
           end

Mate_Pairs: record  in1, in2:  Fragment_ID  end
```

3.2.3 The Scaffold Graph:

The next component of a snapshot is a series of records modelling the vertices and edges of the scaffold graph. Each vertex models a contig and is encoded in a Contig record. A contig record contains the following information:

1) A unique accession number assigned by the assembler.
2) A consensus for the unitig and the layout of its fragments with respect to that consensus (see the following description of contigs) as a list of MultiPos records. Note that this list of fragments gives all the fragments in the unitig as well as the type of each fragment. The scalar value AS_UNITIG is never set in a unitig's list (but can be in a contigs list). The consensus sequence is gapped and the associated string of quality scores is of the same length as the gapped sequence, i.e. there is a quality score for every column of the multi-alignment, including those whose consensus call is '-'.
3) The MultiPos records indicate how a fragment or unitig consensus sequence (AS_UNITIG) is to be aligned to the gapped consensus sequence. Positions in the consensus are locations between characters starting at 0 on the left. The Position interval in the MultiPos record gives the beginning and end of the piece, where the beginning coordinate is greater than the ending coordinate iff the piece should be incorporated in the reverse orientation. The delta is a series of positive integers indicating the positions within the piece at which to insert a dash.
4) Note carefully, that the idea of surrogates is still present in this new encoding scheme but present in a different way. In the case that an overcompressed unitig could not be fully resolved, then its consensus is included as a piece in the multi-alignment of the contig, and is identifiable as such because its MultiPos record types it AS_UNITIG.
5) Finally the position of every unitig in the Contig is given in a list of ElementPos records that are basically abbreviated MultiPos records in that a delta is not required and the type of the element is known to be AS_UNITIG.

```
Contig: record
          accession:       Contig_ID
          ~~last_mod:        time_t~~
          consensus:       string(char)
          quality:         string(bytes)
          ~~reads,guides,~~
          ~~surrogates~~
          piece_list:      list of MultiPos
          ~~good_link_count,~~
          ~~bad_link_count:  int~~
          unitigs:         list of ElementPos
          ~~predecessors,~~
          ~~successors:      list of ScaffLink_ID~~
        end

ElementPos: record
              ident:    Unitig_ID
              position: SeqInterval
        end
```

The edges of the graph are encoded as a series of ContigLink messages. These edges are identical in information content to the edges of the unitig graph. The only differences is in the types of the objects being related.

```
ContigLink: record
              contig1:              Contig_ID
              contig2:              Contig_ID
              orientation:          scalar (AB_AB, BA_BA, AB_BA, BA_AB)
              overlap_type:         scalar (AS_NO_OVERLAP,AS_OVERLAP
                                            AS_TANDEM_OVERLAP)
              is_possible_chimera: boolean
              includes_guides:      boolean
              mean_distance:        float
              std_deviation:        float
              status:               scalar (AS_IN_ASSEMBLY,
                                            AS_POLYMOPHISM,AS_BAD)
              jump_list:            list of Mate_Pairs
           end
```

3.2.4 Best Assembly/Scaffold:

The final component of a snapshot is a series of what the assembler thinks is the correct scaffold for the current data set. Each scaffold record encodes a path in the scaffold graph that is vertex/contig disjoint from every other scaffold record. A scaffold record simply gives a list of the consecutive pairs of contigs comprising the scaffold with an estimate of the distance and standard deviation of that distance for each pair. This later information is based on all the edges supporting the particular pairing and is therefore generally more accurate than the placement information of any of the scaffold edges supporting that pairing.

```
Scaffold: record
            ~~accession:   Scaffold_ID~~
            contigs:   list of ContigPairs
            ~~good_guides,~~
            ~~bad_guides:  int~~
          end

ContigPairs: record
               contig1, contig2: Contig_ID
               mean, stddev:     float
             end
```

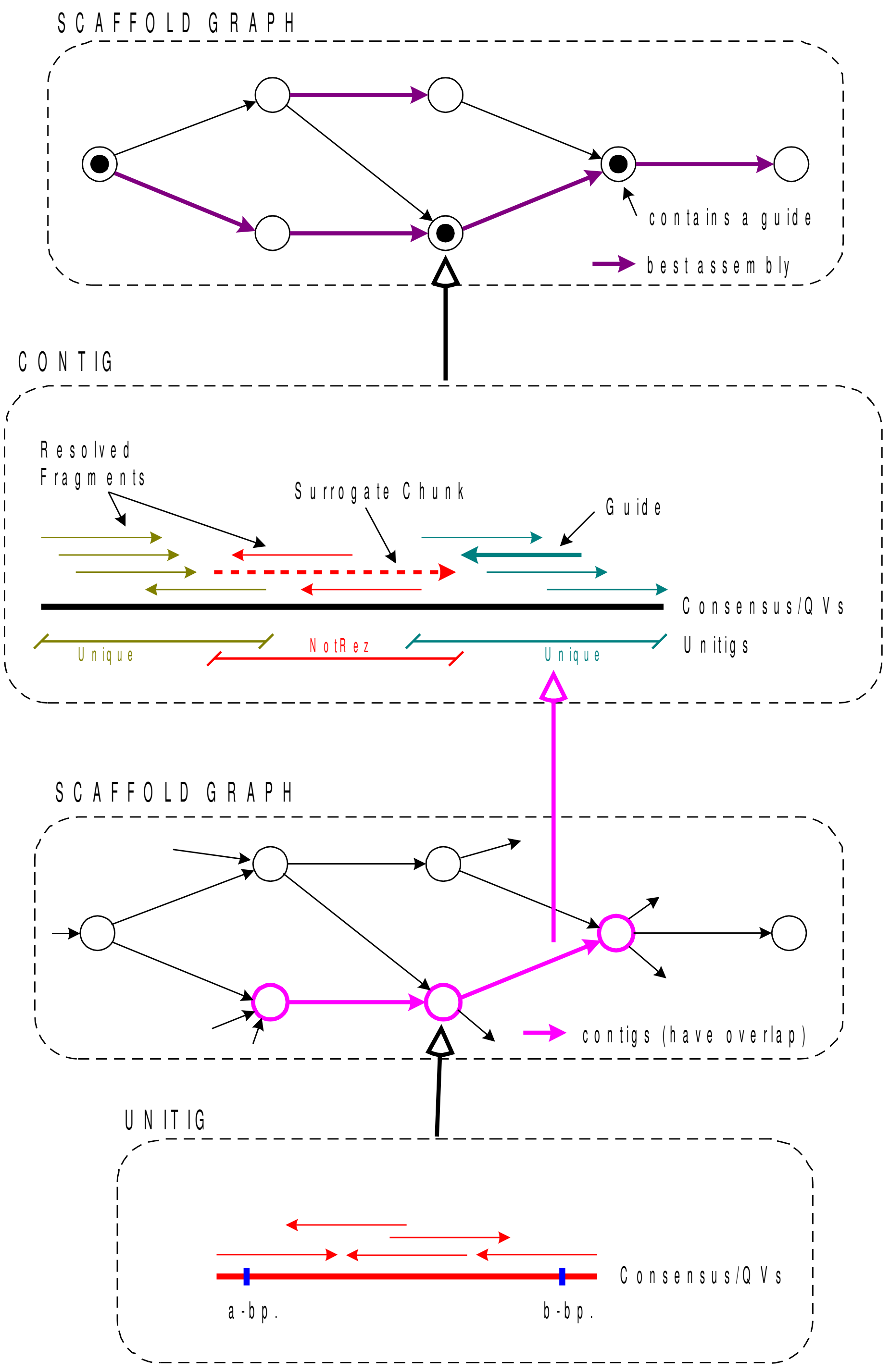


Figure 2: Genome Snapshot

4. Testing

Categories of repeats and other nasties to consider in any approach:

- Repeats < length of fragment
- Repeats > length of fragment
- Nested repeats
- Tandem repeats
- µSat repeats
- Inversion polymorphism
- Deletion polymorphism
- Insertion polymorphism
- Transposition polymorphism
- Chimeric Read
- Chimeric clone
- False guide neighbors
- SNPs

AUTHORS

Gene Myers

```
Created October 14, ’98

Revised: Mar. 5, ’99 by Ian Dew
         Added relevance field to ScreenItemMesg and ScreenMatch.

Revised: Mar. 29, ’99 by Ian Dew
         Elaborated on relevance field.

Revised: April 3, '99 by Gene Myers
         Augmented fragment input to include guide information.

Revised: Apr 9, 1999 by Art. Delcher
         Minor changes to be compatible with new Genome Snapshot
         section in ProtoSpec.rtf.

Revised: April 19, '99 by Gene Myers
         New upgrades to system Input focusing on guides and join
         constraints

Revised: May 17, '99 by Gene Myers
         Finalization of assembler output structures.
```