DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume **00**, 19xx

A Simple Toolkit for DNA Fragment Assembly

João Meidanis

Abstract. We describe a system for DNA fragment assembly. The system centered around a pre-processing tool, a post-processing tool, and the main assembly module. We outline the algorithms and data structures used and present experimental results. Finally, we discuss possible improvements and extensions to the system.

1. Introduction

A DNA fragment assembly program receives a collection of DNA sequences, called *fragments*, and outputs a collection of (much) longer sequences, called *contigs*, that contain every fragment or its reverse complement as an approximate substring. Ideally, one single contig should be returned, but errors in the input data, lack of coverage, and other factors may prevent complete assembly.

Our goal in this paper is to describe a toolkit for DNA fragment assembly and report on its experimental performance. This toolkit was implemented by the Computational Biology group at the Department of Computer Sciences, University of Campinas, Brazil.

The toolkit implements rather simple ideas. It uses a greedy heuristic, explained in detail later. In fact, one could look at it as an implementation of the bare minimum necessary to have a working assembly system. As a result, it cannot compete with such sophisticated programs as, for instance, Huang's CAP and CAP2 [**Hua92, Hua95**] or Kececioglu and Myers' implementations [**KM95**]. Nevertheless, we feel this report is interesting because it (1) gives an idea on how far a simple approach can get, and (2) points out the main issues that must be worked out next.

While the details of our system are described more clearly in the following sections, we would like to briefly sketch here the historical development of the system. This will hopefully contribute to a better understanding of the several modules and their function.

The initial idea was to implement a greedy strategy, that is, repeatedly choose the pair of fragments that overlap the most and join their contigs (more on that

1991 *Mathematics Subject Classification.* Primary 92-04, 92D20; Secondary 68Q25, 05C85.

This work was supported by Brazilian financing agencies FAPESP (grant 93/0655-1), CNPq (grants 350333/93-9, 680060/94-4, and 680065/94-6), and FAEP-UNICAMP.

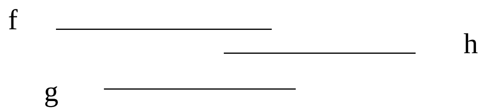


FIGURE 1. Critical overlap situation.

later). We did take into account possible errors in the fragments as well as unknown orientation, so we employed a dynamic programming algorithm to assess overlap strength between each pair of fragments, including reverse complements.

This initial approach resulted in very slow code, mainly because of the dynamic programming routine repeated so many times (N^2 for a collection of N fragments). We then included a pre-processing tool that quickly selects pairs of fragments that are likely to have high overlap. Only these pairs will be submitted to the dynamic programming comparison.

This still took too long. So, we replaced the dynamic programming routine by another approach that looks for long exact matches and applies dynamic programming only between these matches. This kind of approach is very popular in the literature [**Pea90, CZOM95**].

Finally, with this last modification, the system become reasonably fast to be tested in real-size instances. Some of these tests are reported here.

We said earlier that the system uses a greedy heuristic. However, this is not to be confused with the greedy approach encountered in the literature on the Shortest Common Superstring (SCS) problem [**Tur89, BJL$^+$91**], or the greedy approach of some other assembly tools, as the Genetics Computer Group (GCG) [**DHS84**] and Staden's package [**Sta79, Sta80, Sta82, DS91**].

Our selection is always based on the original fragments. Fragments do get merged into contigs, but to select the next overlap we look back at the original fragments. In the SCS greedy algorithm, once a fragment is placed inside a contig, only extremal ends of this fragment are candidates for further joining. In our approach, an "internal" fragment end can be joined as well.

Thus, if fragments f and g both overlap with h as in Figure 1, we could join g and h at a certain point, and then later join f and h, regardless of the relationship between f and g. If one of them turns out to be a false positive, consensus will deteriorate in the part of f and g not shared by h.

We adopted this strategy in order to avoid computing the consensus after each merge, and also to avoid weakening the consensus with the introduction of fragments with high error rates in it. For instance, in Figure 1, if g is a fragment with high error rate, it is better to align f with h alone than with a consensus computed from h and g. However, we did not anticipate the problems that appear when repeats are present.

The rest of this paper is organized as follows. Section 2 describes the main assembly module. Section 3 talks about on the pre- and post-processing tools. Section 4 contains some results from experiments we conducted on data provided for the DIMACS Implementation Challenge. Finally, Section 5 has concluding remarks and future plans.

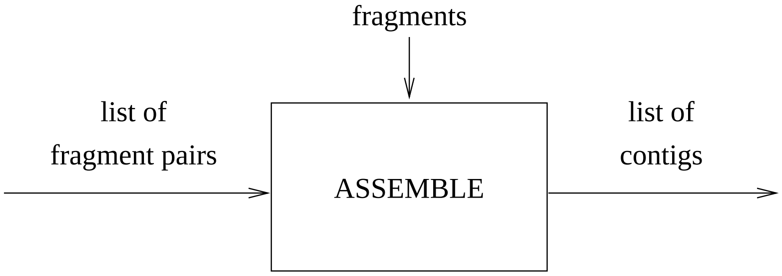


FIGURE 2. The assembly process.

2. Assembly

Figure 2 depicts assembly operation. The program `assemble` receives as input a list of *fragment pairs* which were reported as probably significant for the assembly by the pre-processing phase. More specifically, each pair in the list is composed of two fragments that supposedly have prefix-suffix similarity, or containment similarity. Prefix-suffix similarity means that a prefix of one of the sequences is similar to the suffix of the other. Containment similarity means that one of the sequences is an approximate substring of the other. In addition, two other pieces of information are conveyed along with the pair. These are (1) an estimate of the *number of bases* by which they overlap, and (2) a value that indicates relative orientation of the fragments ($+1$ if they overlap in the orientation they are given, or -1 if they overlap when one of them is reversed and complemented).

The output of `assemble` is a list of contigs. Each contig is composed of a set of fragments, a multi-sequence trace involving the sequences of the fragments in the contig, and relative orientations of these fragments. The fragment sets of different contigs are disjoint and their union is the whole fragment set given as input.

The program operates as follows. Initially, each fragment is made into a contig. Thus, we have as many contigs as fragments. The list of fragment pairs is then processed, in descending order of estimated overlap. For each pair, we align the fragments using a heuristic, in the reported orientations, and the quality of this alignment is assessed. If the alignment is acceptable, then we check whether the fragments already belong to the same contig or not. If they do, their relative orientation in the contig is compared to their relative orientation in the alignment, and a warning is issued when there is orientation incompatibility (possibly caused by the presence of inverted repeats). However, the process continues since the last alignment is probably weaker. Note also that relative positioning is not checked, just relative orientation. If the fragments belong to different contigs, then these contigs are merged into one bigger contig.

This checking and merging process continues until either the list of pairs is exhausted, in which case the remaining contigs are reported, or until only one contig remains.

2.1. Pair validation.

We will now describe in more detail the verification step, where a supposedly related pair is checked. This step was recently improved to use a technique significantly faster than full dynamic programming to find good

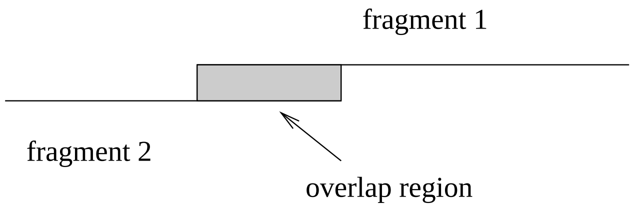


FIGURE 3. Overlap region.

alignments between fragments. Before getting into that, we will describe the general processing of a pair.

First, the second fragment is reversed and complemented if the reported relative orientation is -1. Recall that this relative orientation is given along with the pair by the pre-processor. Then, the best possible prefix-suffix or containment alignment is obtained between the two sequences. Removing columns that contain end gaps we obtain the *overlap region* of such an alignment (see Figure 3). Define the *overlap length* as being the number of columns in the overlap region. We also compute a *score* relative to this overlap region, in a way that which will be explained shortly.

The alignment is considered *acceptable* if

$$\textit{overlap-length} \geq \texttt{MINOVLP}$$

and

$$\frac{\textit{score}}{\textit{overlap-length}} \geq \texttt{MINUSCORE}.$$

The parameters `MINOVLP` and `MINUSCORE` are set by the user. Default values of 12 and 0.85 are provided. The score is computed based on a standard scoring scheme involving matches, mismatches, and gaps in the overlap region as follows.

$$\textit{score} = \sum_{i=1}^{\textit{overlap-length}} s(a_i, b_i),$$

where a_i and b_i are the symbols at position i in the overlap region of sequence 1 and 2, respectively, and the function s is given by

$$s(a,b) = \begin{cases} p(a,b) & \text{if } a \text{ and } b \text{ are characters} \\ -g & \text{if one of } a,\, b \text{ is a gap,} \end{cases}$$

where p and g are specified by the user. The function p is normalized so that

$$\max_{a \in \Sigma} p(a) = 1.$$

Thus, an alignment is accepted if its overlap region has at least a certain minimum length and its *unit score*, or average score per position, is at least a certain given value. In other words, the absolute score is at least a certain fraction of the maximum possible score for regions of the same length. For instance, if the overlap length is 30 and the default values are in effect, the score has to be at least $30 \times 0.85 = 25.5$ for an alignment to be accepted.

When the alignment is accepted and the two fragments belong to distinct contigs, the contigs are merged. This involves the complete reversal of one of the

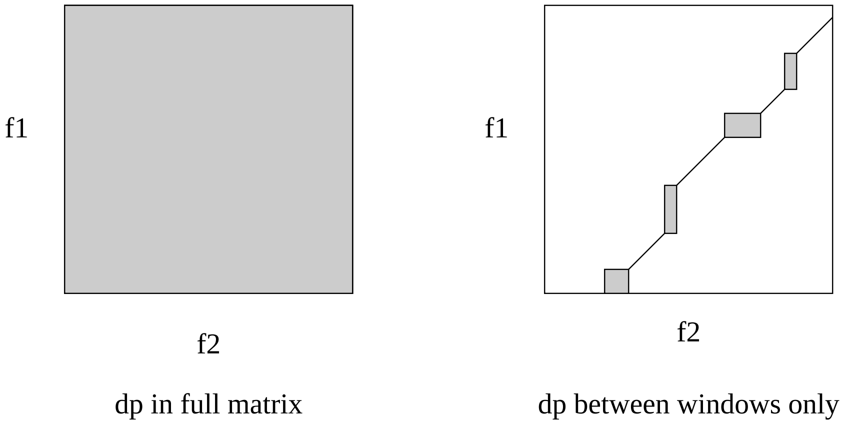


FIGURE 4. Windows approach.

contigs at times. In this case, the smaller contig (in number of fragments) is reversed. Another consequence of contig merging is the *unification* of matching bases in the accepted alignment. This is explained in more detail in Section 3.2.

2.2. Computing the best alignment. Originally, a standard dynamic programming (dp) algorithm was being applied to the two fragments involved. Prior to this procedure, the second fragment was reversed and complemented in case the relative orientation reported was -1. The dp algorithm was configured so that end gaps in either sequence are not penalized. This way, it returns good prefix-suffix or containment alignments.

However, this procedure has time complexity $O(mn)$, where m and n are the lengths of the fragments being compared. In practice, $m \cong n \cong 500$ characters. This time complexity was a problem. Typical assemblies involve 10^4 to 10^5 fragment pairs, even after the filtering done by pre-processing. Thus, it was difficult even to test the system with actual data from large-scale sequencing projects.

Recently, this part of the processing has been replaced by a faster heuristic method. Instead of applying dp on the whole fragments, *common windows* of length at least `WINDOW` are sought. The parameter `WINDOW` is set by the user, and defaults to 12. The best collection of compatible windows is then derived, and dp is applied only on the intervals between windows (see Figure 4).

A common window between two sequences is a common substring. A list of all common windows of length at least `WINDOW` can be efficiently computed using suffix trees [**Apo85, GI93**]. In our implementation, a related method, using the idea of suffix arrays [**MM90**], is employed. The minimum size requirement (parameter `WINDOW`) prevents the list of common windows from getting too large, which would defeat the overall goal of more speed.

After the list of common windows is constructed, a *combination* of them is sought to form the skeleton of a good alignment. This is similar to the approach followed in the well-known FASTA program [**WL83, LP85, PL88, Pea90**]. In our implementation, a method derived from [**JMT92**] is employed.

The main difference between our implementation and the method described in [**JMT92**] is the handling of intervals between windows. Given two common windows $[x^1_{\min}, x^1_{\max}] \times [y^1_{\min}, y^1_{\max}]$ and $[x^2_{\min}, x^2_{\max}] \times [y^2_{\min}, y^2_{\max}]$, the *interval* between them is the region $[x^1_{\max}+1, x^2_{\min}-1] \times [y^1_{\max}+1, y^2_{\min}-1]$. In the cited reference, those are given weight zero, while in the implementation we assign them a weight depending on their size. An $\Delta x \times \Delta y$ interval will be given weight

$$-g|\Delta x - \Delta y| + a \min(\Delta x, \Delta y),$$

where a is the average score for a base match. This weight reflects that there will be at least $|\Delta x - \Delta y|$ gaps in any alignment of the interval, plus the expected score for the aligned part, assuming all character matches equally probable.

In addition, common windows of length l are given weight l. We seek a combination of windows such that the total weight of the windows plus the intervals between them is maximized. This maximum is an approximation of the real score of an alignment based on these windows. From this combination, an actual alignment is obtained by matching the exact windows and applying a dp algorithm in the intervals.

3. Pre- and post-processing

In this section we briefly describe what is done in the pre- and post-processing phases.

3.1. Pre-processing. The pre-processing phase is aimed at finding quickly which pairs of fragments are likely to have a significant overlap.

Each pair of fragments is tested in all possible orientations. It suffices to perform two comparisons, keeping one of the fragments in a fixed orientation and trying the other one both as given and in reverse complement. This is because comparing f and g is equivalent to comparing f^{RC} and g^{RC}. Likewise, comparing f and g^{RC} is equivalent to comparing f^{RC} and g.

The comparison relies, again, on common windows. The idea is that, if two fragments overlap, then their sequences must have many common windows. The window size w is set by the user, and need not be the same used later in aligning the fragments. Its default value is 7.

One possible approach here would be to count the common windows between the two sequences. However, a faster method is used in our implementation, which is to *estimate* the number of common windows by statistical methods.

More precisely, we construct, for each sequence, the *set* of all windows of length w. Then, we compute a short *signature* of the set and compare these signatures instead of comparing the sets.

The signatures are computed as follows. Each word of length w over the alphabet $\{\mathtt{A}, \mathtt{G}, \mathtt{C}, \mathtt{T}\}$ is given a distinct number between 0 and $4^w - 1$. A set of windows of length w becomes then a set of integers from the range $[0, 4^w - 1]$. Instead of representing each set as a bit-vector of size 4^w, which could be too big, we *hash* the numbers from $[0, 4^w - 1]$ into a much smaller interval $[0, q-1]$, where q is a prime number. The hash function h is simply

$$h(x) = x \bmod q.$$

Hence, signatures are bit-vectors of size q. To estimate the number of common windows between sequences f and g we count the number of common bits in these vectors and apply a correction to compensate for the effect of hashing. Counting bits is astonishingly fast because one can use bit operations in entire words. A 2000-fragment set can be compared in this way (all against all) in about 8 minutes in a SPARC 10 or equivalent machine.

We end this section with some more detail on this correction formula, since we feel it is a method that can be used in other applications and readers can benefit from this extra information.

Let $\mathcal{W}_w(f)$ be the set of windows of length w of a sequence f. We are trying to estimate

$$a = |\mathcal{W}_w(f) \cap \mathcal{W}_w(g)|,$$

and we compute

$$c = |h(\mathcal{W}_w(f)) \cap h(\mathcal{W}_w(g))|.$$

This looks just like a classic statistical estimation problem in the following way. Let $A = \mathcal{W}_w(f)$ and $B = \mathcal{W}_w(g)$. Suppose we knew the values of a, $|A|$, and $|B|$. The we can compute the statistical distribution of c for A, B uniformly distributed, assuming h is "random". Standard occupancy models yield this distribution [**JK77, KSC78, Gon81**]. In particular, a formula for $\bar{c}$, the distribution mean, can be derived:

$$\bar{c} = |A|' + |B|' - (|A| + |B| - a)',$$

where x' is an abbreviation of

$$x' = q\left(1 - e^{-x/q}\right),$$

with q being the hash table size and e being the base of natural logarithms. Solving this equation for a, we can obtain an estimator $\hat{a}$ for a given c, $|A|$, $|B|$:

$$\hat{a} = |A| + |B| + q \ln\left(e^{-|A|/q} + e^{-|B|/q} - 1 + \frac{c}{q}\right).$$

More details on this method can be found in [**MJ92, Mei92**]. A related method is described in [**DN82**]. A pair is considered "likely to have an overlap" when $\hat{a} \geq$ `CUTOFF`, where `CUTOFF` is a parameter set by the user.

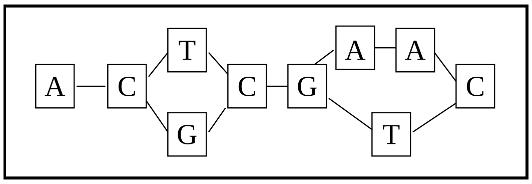
The method is very fast, but it has two drawbacks. First, since we are computing the *set* of common windows for a fragment, repetitions of the same window will be lost. Second, only errorless windows are counted. If a window has any character other than `A`, `G`, `C`, or `T`, the window is discarded. Thus, this method can perform poorly when there are repeats or a high error rate.

3.2. Post-processing. One of the pieces of information stored along with each contig during assembly is a multi-sequence trace. These traces are internally represented as linked lists of bases, an idea already used by Hein [**Hei90**]. As contigs are merged, base unification occurs. Contigs are always merged based on an alignment between two fragments. Each column of the alignment that contains identical bases causes these bases to be unified in the trace structure. An example is given in Figure 5, where two singleton contigs (contigs with only one fragment) are merged based on the alignment shown.

The post-processing phase receives a contig and delivers a multiple alignment of the fragments and a consensus sequence.

Alignment `ACTCGAAC`

`ACGCGT-C`



Unified structure

FIGURE 5. Unification.

We employ a greedy multiple alignment strategy described in detail elsewhere to construct the alignment given the trace [**MS95**]. From it, the consensus is obtained by a simple voting scheme. The consensus character in a column is the character with highest frequency in that column. If more than one character has maximum frequency in a column, one of these is chosen according to a fixed priority scheme.

4. Results

We ran our system with data provided by the organization of the 1995 DIMACS Implementation Challenge. This data is available at the site

`http://dimacs.rutgers.edu/SpecialYears/1994_1995/challenge.html`

For all runs, the parameters used were the following.

- Pre-processing:
  Window size $w = 7$ (default)
  Hash table size $q = 2003$ (default)
  Cutoff value `CUTOFF` $= 30$
- Assembly:
  Minimum overlap length `MINOVLP` $= 12$ (default)
  Minimum unit score `MINUSCORE` $= 0.7$
  Alignments scored using

$$s(a,b) = \begin{cases} 1 & \text{if } a = b, \text{ and } a,\ b \text{ not gaps} \\ -1 & \text{if } a \neq b, \text{ and } a,\ b \text{ not gaps} \\ -2 & \text{if one of } a,\ b \text{ is a gap} \end{cases}$$

  Window size `WINDOW` $= 5$ (default)
- Post-processing:
  No parameters needed

In Section 4.1 we describe the data sets in some detail. Each data set consists of two files, one containing the fragments and the other, the consensus sequence that is to be expected from the fragments. In some cases, we edited the consensus for reasons explained below.

Set	Total Sequence Length	Number of Fragments	Total Number of Bases	Mean Coverage	Mean Fragment Length
247-ed	34476	828	393469	11.4	475
b1496	41295	1811	421426	10.2	233
b2126	44566	1504	348396	7.8	232
l247	44438	1704	418136	9.4	245
l518	41412	1910	434390	10.4	227
cov3	73308	440	221156	3.0	503
cov5	73308	734	368460	5.0	502
cov7	73308	1027	514896	7.0	501

TABLE 1. The input fragment sets. In this table, "Mean Coverage" is the total number of bases divided by the total sequence length, and "Mean Fragment Length" is the total number of bases divided by the number of fragments. In the human DNA sets cov3, cov5, and cov7, the parameters refer to the set prior to error introduction.

We outline the general characteristics of each data set, paying attention to the ones that have more impact on the assembly. In particular, we analyze in some detail the repeat structure of each set.

Next, in Section 4.2, we report on the performance obtained by our system on each data set. This includes the number and size of contigs, and a map positioning each contig in the target consensus.

4.1. Data sets. The data sets can be grouped in three categories, according to their origin.

The first category contains two data sets, originated at the laboratory of Prof. Leroy Hood [**SKH93**]. These sets correspond to a real sequencing project carried out in this lab and described elsewhere [**SKH93**].

The two sets share the same consensus sequence and differ only with respect to the fragment set. One set contains "raw" data, and the other contains "edited" data with less errors. In our tests we used only the edited data, referred to as 247-ed from now on.

Table 1 lists some general information on all data sets. The particular set 247-ed has the shortest consensus (34 kbp), high coverage, and long fragments.

The fragments in this set contain, apart from the usual `A`, `G`, `C`, `T` bases, a special symbol `N` to indicate an ambiguous base call. Less than 1% of all bases are `N`. The actual error rate is somewhat larger, due to mismatches and indels that are not detectable before assembly.

The repeat structure of the consensus 247-ed is shown in Figure 6. We detected a single repeat, occurring twice in the sequence, at the 98% similarity level. All repeat structures in this paper were determined with the help of the `Miropeats` tool, available from the Genome Sequencing Center, Washington University, St. Louis, USA. This tool can detect exact as well as approximate repeats.

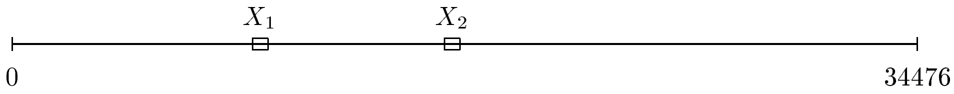


FIGURE 6. Map of repeats for 247-ed. There is only one repeat with length greater than 100 bases. Its two occurrences are marked X_1 and X_2 in the figure. They are 98% similar.

Set	Before	After
b1496	41295	42145
b2126	44566	44570
l247	44438	45828
l518	41412	41797

TABLE 2. Sizes of GTC consensus sequences before and after editing.

The second category contains four data sets, all originated in real sequencing projects at the Genome Therapeutics Corporation (GTC), and made available to the Challenge by H. Safer.

These sets are identified by the codes b1496, b2126, l247, and l518. From Table 1 we see that they correspond to consensus sequences in the 40-45 kbp range, and contain very many fragments. Coverage varies but in all cases is enough to guarantee a good assembly. The fragments tend to be shorter that in other categories.

One interesting characteristic of these GTC sets is the extremely low error rate. Most fragments are an exact copy of some contiguous region in the consensus. Thus, these sets are a good test on the effect of other factors in the assembly.

Another interesting point is that many of the fragments come in dual-end sequencing pairs. This means that a single insert was sequenced from both ends, generating two fragments that have to appear in opposite orientations and not very far apart in the final layout. Our system did not take advantage of this fact, though.

Finally, a word must be said about the consensus sequences of these GTC sets. As we did with all sets, we used the local alignment tool `sim2` [**CZOM95**] to position all given fragments in the given consensus sequence. For the GTC sets, in certain columns of the multiple alignments produced by `sim2`, all fragments covering the region would agree on some base, but the consensus would disagree. We interpreted these discrepancies as indications that the consensus was wrong and edited it to agree with the fragments. The consensus was only modified when (1) at least two fragments covered the base, and (2) all fragments agreed. In some sets this occurred in just a few places, but in others it affected as many as 1000 bases or more. For completeness, we tabulate in Table 2 the consensus sizes before and after editing. The overwhelming majority of corrections were insertions and substitutions. We used the edited consensus sequences in our maps.

The repeat structures of the GTC sets are shown in Figures 7 to 10. Notice that b1496 does not have repeats. The captions in the figures explain in more detail these repeat maps.

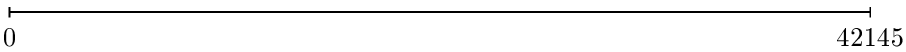


FIGURE 7. Map of repeats for b1496. There are no exact or approximate repeats longer than 40 characters.

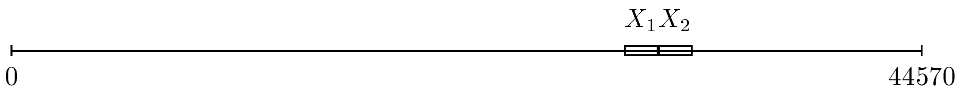


FIGURE 8. Map of repeats for b2126. There is only one repeat with length greater than 100 bases. Its two occurrences are marked X_1 and X_2 in the figure, and they are identical.

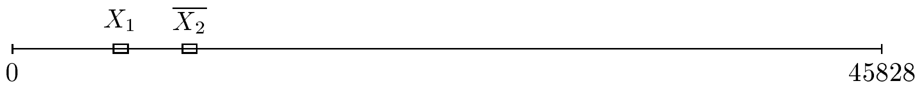


FIGURE 9. Map of repeats for l247. There is only one repeat longer than 100 bases, and it is an inverted repeat. The similarity between X_1 and X_2 is 91%.

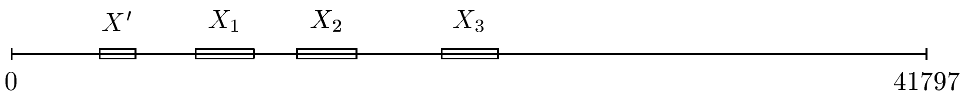


FIGURE 10. Map of repeats for l518. The region labeled X' is an (approximate) suffix of X_1, at the level of 96% similarity. The three X_i exhibit pairwise similarity rates in the range 65-70%.

The third category is distinct from the previous ones in that it as not originated from a real sequencing project. Rather, a already known sequence, the *human β-like globin gene cluster*, was artificially chopped off into fragments to generate the data sets. This 73 kbp sequence, identified as HUMHBB, is stored in GenBank under accession number U01317, among others. All sets in this category have HUMHBB as their consensus.

The repeat structure of HUMHBB is shown in Figure 11.

Fragments were obtained using the `genfrag` package [**EB93, EB94**]. For all sets, the target fragment length was set to 500 bases in average, with a maximum deviation of 100 bases. Initially, three sets were generated with predefined mean coverage of 3, 5, and 7, identified as cov3, cov5, and cov7, respectively. Then, for each cov set, four derived sets were obtained by introducing errors in the fragments in a way that mimics errors encountered in real projects [**CH92**]. Error rates of 0.5, 1.0, 1.5, and 2.0 times a certain standard profile were used for each cov set, with the standard profile representing about 2.6% of differences (mismatches and indels). Thus, we ended up with a total of 12 sets in this category.

Coverage 3 is quite low, so the fragments in this set group in 24 contigs rather than a single one. The cov5 set gives 4 contigs, and cov7 gives 1 contig.

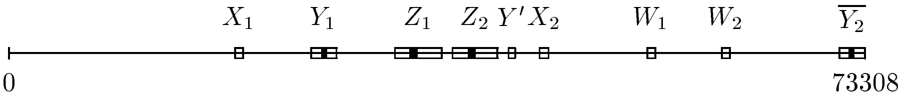


FIGURE 11. Map of repeats for HUMHBB. Regions X_1 and X_2 share a 70% similarity. Regions Z_1 and Z_2 are 87% similar. Moreover, X_1 and X_2 are similar, at levels ranging from 68-73%, to a block occurring inside both Z_1 and Z_2, marked as a black box in the figure. The two occurrences of this block are almost identical (>99% similarity). The similarity between W_1 and W_2 is 89%. All these are direct repeats. There is an inverted repeat, represented here by Y_1 and Y_2, which share an 87% similarity. The region Y' is similar to blocks inside Y_1 and Y_2, with similarities around 80% among Y' and the two blocks, and between the blocks.

Contig	Fragments	Length (bp)	Similar to	Singleton
0	26	1976	vector	
1	54	2805	vector	
2	1	433	?	6813seq
3	744	27801	consensus	
4	1	466	vector	6424seq
5	1	206	vector	5060seq
6	1	325	?	5750seq
7	1	308	?	5513seq

TABLE 3. General information about contigs generated by the toolkit on data set 247-ed.

4.2. Results. We now report on the performance of our toolkit on the several data sets previously described.

With the exception of b1496, we present a map for each data set depicting the best alignment between contigs and the consensus. The set b1496 has no map since the assembly was almost perfect. The maps also contain repeat structures for comparison. All maps were constructed with the help of `sim` and `sim2` (`sim` is a local similarity tool implemented by X. Huang).

The section closes with a report on the computational time and space used, as well as other technical information.

The set 247-ed resulted in 8 contigs. Table 3 gives the sizes and the sequence each contig fits in. This last information was included because many contigs resulted similar to the cloning vector pWE15A used in the laboratory experiments. In other words, this is foreign DNA that contaminated about 10% of the fragments. The vector sequence is available from GenBank under accession number Z12112.

Three contigs containing just one fragment each did not show similarity with either vector or consensus. Their sequence identifiers appear in Table 3.

The largest contig, containing 744 fragments and covering 27.8 kbp is similar to the consensus. Figure 12 shows a map of the similarities. Two large regions of very good similarity flank a middle region containing a large gap, a small good region, and a region with poor similarity with the consensus. The gap of about 7 kbp explains why the contig is shorter that the consensus.

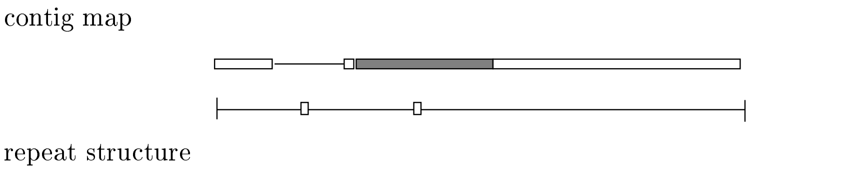


FIGURE 12. Alignment map between contig 3 and consensus of 247-ed. Upper line: contig 3, with regions of similarity shown as boxes, and gaps shown as horizontal lines. Good similarity regions shown as white boxes. Grey boxes indicate poor similarity. Lower line: consensus, with repeats marked as white boxes.

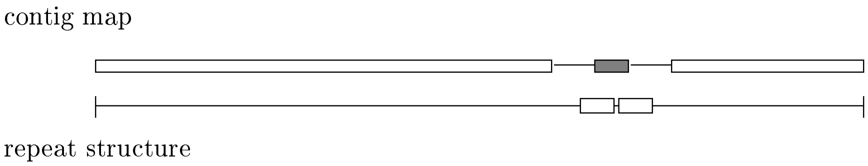


FIGURE 13. Alignment map for b2126. Upper line: the single contig, with regions of similarity shown as boxes, and gaps shown as horizontal lines. Good similarity regions shown as white boxes. Grey boxes indicate poor similarity. Lower line: consensus, with repeats marked as white boxes.

The gap occurs exactly were one copy of the repeat is. What probably happened here is that all fragments that come from the gap region were instead placed near the other copy of the repeat, causing consensus deterioration there.

The set b1496 was assembled almost perfectly by the toolkit. Two contigs were obtained, one containing just one 35 bp fragment and the other with the remaining 1810 fragments. The isolated fragment was too short to be detected in the pre-processing. The other contig matched the consensus. There were only 4 mismatches in a 42 kbp alignment. This was the best result obtained by our system. This is also the only set without long repeats.

The set b2126 produced only one contig, involving all fragments. A map showing how it fits in the consensus is given in Figure 13. As in the case of 247-ed, the repeats confused the assembler and resulted in gaps and poor similarity nearby. However, since the repeats in this case are closer, the affected region is much shorter, although the repeats themselves are longer (1.5 kbp versus 0.5 kbp in 247-ed).

The set l247 produced one contig. Figure 14 shows its relationship to the consensus. This sequence has an inverted repeat, which caused a similar problem as the directed repeats in 247-ed and b2126.

The last set in the GTC category, l518, is an interesting example. Four contigs were generated, two of which are singleton contigs with short fragments. The other two contigs, shown in Figure 15, cover about 87% of the consensus. Again the repeats caused damage, although some repeat regions were assembled correctly.

Finally, we report on the HUMHBB data. We show here just one set for each coverage class, the one with error rate 1.0 times the profile. Sets with higher error rates tended to reproduce the maps we are about to present, only with more contigs and more bad regions. We did not run the toolkit in the 0.5 error class.

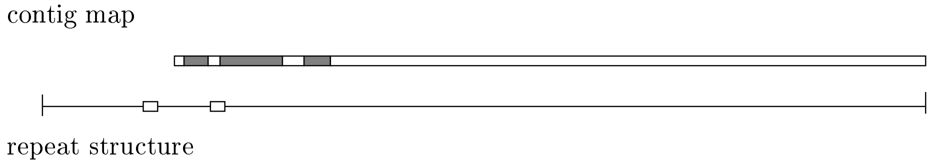


Figure 14. Alignment map for l247. Upper line: the single contig, with regions of similarity shown as boxes, and gaps shown as horizontal lines. Good similarity regions shown as white boxes. Grey boxes indicate poor similarity. Lower line: consensus, with repeats marked as white boxes.

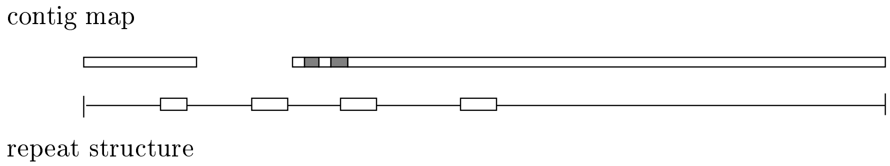


Figure 15. Alignment map for l518. Upper line: the two large contigs, with regions of similarity shown as boxes, and gaps shown as horizontal lines. Good similarity regions shown as white boxes. Grey boxes indicate poor similarity. Lower line: consensus, with repeats marked as white boxes.

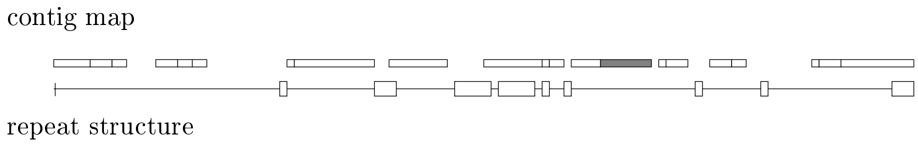


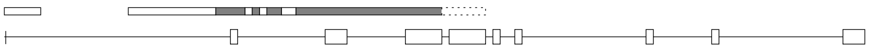
Figure 16. Alignment map for cov3. Upper line: the several contigs, with regions of similarity shown as boxes, and gaps shown as horizontal lines. Good similarity regions shown as white boxes. Grey boxes indicate poor similarity. Lower line: consensus, with repeats marked as white boxes.

The cov3 data originated 24 contigs, distributed fairly well on the consensus, but leaving many regions uncovered. Most contigs showed good similarity, with a few bad regions (see map in Figure 16). Recall that the data in this set represents 24 natural contigs.

The cov5 set produced two contigs. The map in Figure 17 shows that one of them fits well near the beginning of the consensus, where there are no repeats, and the other contains many parts of bad similarity. Less than half of the consensus is covered, a very poor result.

The cov7 set is even worse. Only one fourth of the consensus is well covered. A small bad region and a part that didn't fit the consensus anywhere complete the single contig obtained (see Figure 18).

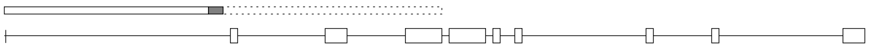
The machines used were SPARC-servers 1000 with 128 to 192 Mb of RAM memory. The code is written partly in C and partly in C++ and was compiled contig map



repeat structure

FIGURE 17. Alignment map for cov5. Upper line: the two contigs, with regions of similarity shown as boxes, and gaps shown as horizontal lines. Good similarity regions shown as white boxes. Grey boxes indicate poor similarity. Lower line: consensus, with repeats marked as white boxes. The dotted box indicates a part of the contig that is not similar to the consensus anywhere.

contig map



repeat structure

FIGURE 18. Alignment map for cov7. Upper line: the two contigs, with regions of similarity shown as boxes, and gaps shown as horizontal lines. Good similarity regions shown as white boxes. Grey boxes indicate poor similarity. Lower line: consensus, with repeats marked as white boxes. The dotted box indicates a part of the contig that is not similar to the consensus anywhere.

Set	Pre-proc. (sec.)	Assembly (sec.)	Post-proc.(∗) (sec.)
247-ed	29	2871	326
b1496	135	905	27
b2126	92	784	21
l247	119	958	27
l518	150	1084	35
cov3-err1.0	10	2011	9
cov5-err1.0	24	5117	99
cov7-err1.0	47	2312	193

TABLE 4. Processing times. (∗) The post-processing times refer to the largest contig only.

with the Gnu C Compiler. Operating system is SunOS 4.3 (also known as Solaris 2.3). The times were measured using the Unix `time` command.

The amount of space used by the system is roughly proportional to the total number of bases, reaching up to 50 Mb for the largest HUMHBB set.

Computation times are presented in Table 4. Each figure in this table is an average among 10 runs in the same machine. Times for HUMHBB set cov5-err1.0 are significantly higher than the rest of the table. This set also caused problems in its err2.0 version, where the system crashed and did not produce any output.

5. Conclusions

We have described the main features of a simple toolkit to help fragment assembly in large-scale shotgun DNA sequencing projects. Experimental results for real data are shown.

The results show that the toolkit performs well in the absence of repeats, despite its lack of sophistication. Even in situations were repeats are present, some regions containing repeats were assembled correctly. In any case, since there is no immediate way of deciding if a collection of fragments has repeats or not, this points out the single major issue that has to be tackled in this system.

The system is reasonably fast and supports data sizes compatible with current needs of sequencing labs. There are no intrinsic limitations in this respect apart from the ones imposed by underlying hardware.

Planned additions to the system include new algorithms to deal with repeats. We also plan to integrate it in a wider system that supports scientific applications based on workflow [**MVW95, MVW96**].

Acknowledgments

We thank Debby Joseph and Fred Blattner, who introduced us to fragment assembly, and our students A. A. M. Almeida, A. Moriya, D. Jerozolimski, and A. R. Cruvinel for help with the implementation. We also acknowledge the comments and suggestions of the anonymous referees, which led to several improvements in the paper.

References

[Apo85] A. Apostolico. The myriad virtues of subword trees. In A. Apostilico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO ASI Series F: Computer and System Sciences*, pages 85–96. Springer-Verlag, 1985.

[BJL+91] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 328–336, 1991.

[CH92] W.-Q. Chen and T. Hunkapiller. Sequencing accuracy of large DNA sequencing projects. *DNA Sequence*, 2:335–342, 1992.

[CZOM95] K.-M. Chao, J. Zhang, J. Ostell, and W. Miller. A local alignment tool for very long DNA sequences. *Computer Applications in the Biosciences (CABIOS)*, 11(2):147–153, 1995.

[DHS84] J. Devereux, P. Haeberli, and D. Smithies. A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Research*, 12:387–395, 1984.

[DN82] J. Dumas and J. Ninio. Efficient algorithms for folding and comparing nucleic acid sequences. *Nucleic Acids Research*, 10(1):197–206, 1982.

[DS91] S. Dean and R. Staden. A sequence assembly and editing program for efficient management of large projects. *Nucleic Acids Research*, 19(14):3907–3911, 1991.

[EB93] M.L. Engle and C. Burks. Artificially generated data sets for testing DNA fragment assembly algorithms. *Genomics*, 16:286–288, 1993.

[EB94] M.L. Engle and C Burks. GenFrag 2.1: New features for more robust fragment assembly benchmarks. *Computer Applications in the Biosciences (CABIOS)*, 10:567–568, 1994.

[GI93] R. Grossi and G. F. Italiano. Suffix trees and their applications in string algorithms. In R. Baeza-Yates and N. Ziviani, editors, *Proc. 1st South American Workshop on String Processing*, pages 57–76, September 1993.

[Gon81] Gaston H. Gonnet. Expected length of the longest probe sequence in hash code searching. *JACM*, 28(2):289–304, April 1981.

[Hei90] Jotun Hein. Unified approach to alignment and phylogenies. In Russell F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, pages 626–645. Academic Press, 1990.

[Hua92] X. Huang. A contig assembly program based on sensitive detection of fragment overlaps. *Genomics*, 14:18–25, 1992.

[Hua95] X. Huang. An improved sequence assembly program. In M. Vingron, editor, *4th DIMACS Implementation Challenge - Fragment Assembly*, 1995.

[JK77] N. L. Johnson and S. Kotz. *Urn Models and Their Application.* John Wiley & Sons, New York, 1977.

[JMT92] D. Joseph, J. Meidanis, and P. Tiwari. Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In *Proc. of the Third Scandinavian Workshop on Algorithm Theory*, volume 621 of *Lecture Notes in Computer Science*, pages 326–337, 1992.

[KM95] J. D. Kececioglu and E. W. Myers. Combinatorial algorithms for DNA fragment assembly. *Algorithmica*, 13:7–51, 1995.

[KSC78] Valentin F. Kolchin, Boris A. Sevast'yanov, and Vladimir P. Chistyakov. *Random Allocations.* Scripta Series in Mathematics. V. H. Winston & Sons, Washington, D.C., 1978.

[LP85] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity search. *Science*, 227:1435–1441, 1985.

[Mei92] J. Meidanis. *Algorithms for Problems in Computational Genetics.* PhD thesis, University of Wisconsin-Madison, 1992.

[MJ92] J. Meidanis and D. Joseph. Hashing and back prediction methods for computing sequence similarity for fragment assembly. Submitted to the SIAM Symposium on Discrete Algorithms, 1992.

[MM90] U. Manber and G. Myers. Suffix arrays: A new method for on–line string searches. In *Proceedings of the First Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 319–327. SIAM, 1990.

[MS95] J. Meidanis and J. C. Setubal. Multiple alignment of biological sequences with gap flexibility. In R. Baeza-Yates, E. Goles, and P. V. Poblete, editors, *Proc. of the II Latin American Theoretical Informatics*, volume 911 of *LNCS*, pages 411–426, Valparaíso, Chile, April 1995. Springer-Verlag.

[MVW95] C. B. Medeiros, G. Vossen, and M. Weske. WASA: A workflow-based architecture to support scientific database applications. Technical Report 02/95-1, University of Münster, 1995.

[MVW96] J. Meidanis, G. Vossen, and M. Weske. Using workflow management in DNA sequencing. In K. Aberer and A. Helal, editors, *Proc. First IFCIS Conference on Cooperative Information Systems*, 1996.

[Pea90] William R. Pearson. Rapid and sensitive sequence comparison with FASTP and FASTA. In Russell F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, pages 63–98. Academic Press, 1990.

[PL88] William R. Pearson and David J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the U. S. A.*, 85:2444–2448, 1988.

[SKH93] D. Seto, B.F. Koop, and L. Hood. An experimentally derived data set constructed for testing large-scale DNA sequence assembly algorithms. *Genomics*, 15:673–676, 1993.

[Sta79] R. Staden. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Research*, 6:2601–2610, 1979.

[Sta80] R. Staden. A new computer method for the storage and manipulation of DNA gel reading data. *Nucleic Acids Research*, 8(16):3673–3694, 1980.

[Sta82] R. Staden. Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequencing. *Nucleic Acids Research*, 10:4731–4751, 1982.

[Tur89] Jonathan S. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation*, 83:1–20, 1989.

[WL83] W. Wilbur and D. Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences of the U. S. A.*, 80:726–730, 1983.

Institute of Computing, University of Campinas, P. O. Box 6176, Zip Code 13083-970 Campinas SP, Brazil

E-mail address: meidanis@dcc.unicamp.br