BIOINFORMATICS

Vol. 19 no. 1 2003
Pages 22–29

A divide-and-conquer approach to fragment assembly

Hasan H. Otu* and Khalid Sayood

University of Nebraska-Lincoln, Department of Electrical Engineering, 209N WSEC, Lincoln, NE 68503, USA

Received on February 4, 2002; revised on June 10, 2002; July 6, 2002; accepted on July 9, 2002

ABSTRACT

Motivation: One of the major problems in DNA sequencing is assembling the fragments obtained by shotgun sequencing. Most existing fragment assembly techniques follow the overlap—layout—consensus approach. This framework requires extensive computation in each phase and becomes inefficient with increasing number of fragments.

Results: We propose a new algorithm which solves the overlap, layout, and consensus phases simultaneously. The fragments are clustered with respect to their Average Mutual Information (AMI) profiles using the k-means algorithm. This removes the unnecessary burden of considering the collection of fragments as a whole. Instead, the orientation and overlap detection are solved efficiently, within the clusters. The algorithm has successfully reconstructed both artificial and real data.

Availability: Available on request from the authors.

Contact: otu@eecomm.unl.edu

INTRODUCTION

Shotgun sequencing, the most popular method used in DNA sequencing, involves obtaining *fragments* (subsequences of a few hundred base pairs) from both strands of the *target* DNA sequence. This sampling process is essentially random and uses multiple copies of the target. The goal in *fragment assembly* is to reconstruct the target from the collection of fragments.

There are a few supplementary and alternative approaches to shotgun DNA sequencing. Among the most common ones are *direct sequencing*, *dual end sequencing* and *sequencing by hybridization*. However, most of these methods are simply alternative ways to produce fragments and there is still a need to assemble the fragments.

The earliest work on fragment assembly was very application specific and the problem was not formally defined. These approaches simply tried to 'meld' the fragments together. A formal definition of the *Sequence Reconstruction Problem* (SRP) was the first attempt at formulating fragment assembly (Peltola *et al.*, 1984). The objective in SRP is to find the shortest consensus sequence S, which contains each fragment or its reverse complement within a small edit distance. The SRP approach provides a common framework for fragment assembly by dividing the problem into three phases: *overlap*, *layout*, and *consensus*.

Staden *et al.* (Staden, 1982; Dean and Staden, 1991) approach the problem in terms of contigs. Fragments are processed one at a time and compared to the existing layout. This results in one of three cases: a new contig is formed, an old contig is extended, or two contigs are connected. After each iteration the consensus sequence is recomputed. Almost all fragment assembly algorithms resemble these two basic approaches.

In EULER (Pevzner *et al.*, 2001), the fragment assembly problem is reduced to a variation of the classical Eulerian path problem. Repeats are handled by defining an 'Eulerian Superpath Problem' in which an Eulerian path that contains a collection of paths in an existing Eulerian graph is investigated. The collection of paths consists of 'fragments' and 'untangled repeat' regions. An error correction scheme is applied to fragments before performing assembly. This preprocessing of fragments, in which more than 95% of errors are corrected, not only enables the application of Eulerian path to fragment assembly but also drastically improves the performances of existing assemblers.

Variations in the overlap—consensus—layout framework include approaches using graph theory (Kececioglu and Myers, 1995), statistics (Myers, 1995), pattern matching (Kim and Segre, 1999), suffix trees and suffix arrays (Chen and Skiena, 1997), simulated-annealing (Churchill *et al.*, 1993; Burks *et al.*, 1994) and simulated-tempering (Sandelin, 2001) methods.

Huang *et al.* (Huang, 1992; Huang and Madan, 1999) introduced the CAP family in which he applies the technique of Chang and Lawler (1990) for fast detection of overlapping fragments. Pairs of fragments whose alignment score is below a fixed threshold are ignored. The assembler PHRAP (Green, 1994) clips the low quality regions (generated by another program PHRED) to find

*To whom correspondence should be addressed.

overlaps and constructs contig layouts, using consistent pairwise matches. This clipping capability is also included in the final version of the CAP program. The idea of having a fast method for detection of overlaps is also used in Meidanis (1993) which uses the Karp–Rabin string matching algorithm (Karp and Rabin, 1987). However in both techniques all fragments are used for comparison with the improvement of using a faster algorithm for finding overlapping fragments. These approaches along with others (Churchill *et al.*, 1993; Burks *et al.*, 1994; Sandelin, 2001) introduce a new class in the overlap and/or layout phase of the assembly process. This can be characterized by the use of stochastic search algorithms instead of some other directed method.

The algorithm presented in this paper falls into the same category as these approaches. However, unlike the techniques mentioned above, the stochastic search process defined in this work significantly reduces the size of the problem by comparing only subsets of the entire set of fragments. For the coarser classification, we use the ideas from Staden's and Peltola's work. Hence we can place the current work as an algorithm that links these two basic approaches. These arguments are explained more precisely in the sequel.

METHODS AND ALGORITHMS

In this paper, we introduce a method which solves the orientation, overlap, and layout phases simultaneously. Most existing algorithms put a significant amount of computational burden in the overlap phase where each fragment is compared with all the remaining fragments and their reverse complements. This is unnecessary as the number of 'similar' fragments is on the order of the coverage which is much smaller than the total number of fragments. We significantly reduce the computational burden by clustering the fragments before exploring overlaps. We use the AMI profile to measure the degree of 'closeness' between fragments and employ the k-means algorithm to generate the clusters. This turns out to be very powerful as fragments coming from the same regions of the target sequence have similar AMI profiles. Moreover, AMI profiles are robust to errors and remain unchanged when calculated for the reverse complement of fragments. Therefore, we solve the orientation and overlap problems within the clusters which already contain fragments coming from the same region of the target.

Average mutual information

For a sequence S, the average mutual information function is defined as (Shannon, 1948):

$$I(r) = \sum_{i,j} p_{ij}(r) \log_2 \left(\frac{p_{ij}(r)}{p_i p_j} \right)$$

where p_i is the probability of occurrence of the symbol i and $p_{ij}(r)$ is the joint probability of observing symbol i and j separated by a distance r. $I(r)$ is the amount of information symbol i carries about symbol j at a distance r. For an independent identically distributed (iid) sequence, $I(r) = 0$ for all $r > 0$. AMI profiles of DNA sequences have been used to recognize the coding regions (Grosse *et al.*, 2000) and to study statistical correlation of nucleotides (Luo *et al.*, 1998). Recently AMI profiles of genomic sequences have been used for analysis of evolutionary history (Bauer, 2001).

In the proposed technique, for each fragment f_i we calculate the vector $\mathbf{A_i}$, where $\mathbf{A_i}(r) = I(r)$, $r = 1, \ldots, R$, estimating the required probabilities from f_i. At this point we have R-dimensional vectors, namely the AMI profiles associated with each fragment. These vectors are clustered using the k-means algorithm.

Clustering

Clustering can be defined as grouping the elements of a set subject to some measure of similarity. In particular, k-clustering partitions the given set into k non-empty sets. The k-means algorithm (MacQueen, 1967) is a special case of the more general k-clustering algorithms. We use this algorithm along the lines of Vector Quantization, a popular method in data compression (Sayood, 2000). Given N vectors $\{A_i\}_{i=1}^{N}$, the algorithm obtains J clusters by trying to minimize the total distortion $D_N = \sum_{j=1}^{J} \sum_{i=1}^{|C_j|} d(B_j, A_{ij})$ where A_{ij} is the i^{th} vector in the j^{th} cluster, C_j, $|C_j|$ is the cardinality of the cluster C_j, B_j is the representative vector in cluster C_j, and $d(A, B)$ is the Euclidean distance between the vectors A and B. The algorithm iteratively finds the representative vectors and halts when the improvement in the total distortion is below a preset threshold.

Processing the clusters

The J clusters obtained using the clustering algorithm described above are further processed to address three issues: clustering errors, multiple alignment and consensus.

Clustering error A clustering error occurs if fragments in a given cluster belong to non-overlapping regions of the target sequence. When this happens, we need to divide the cluster to obtain subsets of overlapping fragments, C_j^i. We do this through set partitioning where $f \in C_j^i$ if and only if there exists a $g \in C_j^i$, $g \neq f$, such that $s(f, g) > \lambda(f, g)$ or $s(\bar{f}, g) > \lambda(\bar{f}, g)$, where $s(f, g)$ is the score of the optimal semiglobal alignment (we do not charge for gaps in the extremities of either sequence) between the fragments f and g. $\lambda(f, g)$ is a threshold function which indicates the minimum score of the optimal semiglobal alignment between f and g showing significant similarity.

```
(∗)  A  C  T  G  A  C  _  _
(○)  _  _  C  G  A  C  T  A

     ∗∗∗○∗○∗○∗○○○

     ∗  ∗  ∗  ∗  ∗  ∗  _  _
     _  _  ○  ○  ○  ○  ○  ○
```

Fig. 1. Actual alignment, corresponding interlacing, and the alignment implied by the interlacing of two sequences.

```
(∗)  A  C  T  G  _  A  C  _  _
(○)  _  _  C  G  _  A  C  T  A
(◇)  _  C  C  G  A  A  C  _  _

     A  C  T  G  A  C  _  _
     _  _  C  G  A  C  T  A

     ∗∗∗○∗○∗○∗○○○

     _  C  G  _  A  C  T  A
     C  C  G  A  A  C  _  _

     ◇○◇○◇○◇○◇○◇○○

     ∗∗◇∗○◇∗○◇◇∗○◇∗○◇○○

     ∗  ∗  ∗  ∗  _  ∗  ∗  _  _
     _  _  ○  ○  _  ○  ○  ○  ○
     _  ◇  ◇  ◇  ◇  ◇  ◇  _  _
```

Fig. 2. Multiple alignment of three sequences combining the pairwise interlacings and generating the multiple alignment implied by the combined interlacing.

For simulation purposes we use $\lambda(f,g) = c\min(|f|,|g|)$ assuming the scoring scheme is normalized so that the score of a match is 1. The constant c is input by the user. The value for c used in simulations was between 0.05 and 0.1. If for a given fragment $f \in C_j$ there exists no $g \in C_j$ such that $s(f,g) > \lambda(f,g)$ or $s(\bar{f},g) > \lambda(\bar{f},g)$, then f is placed in a subset of its own.

Multiple alignment We now need to find a consensus sequence for each subgroup from the cluster C_j. This requires forming a multiple alignment of the fragments in the subgroup. The multiple alignment strategy we employ is based on the observation that any alignment between two sequences can be represented by an interlacing of the two sequences. This is illustrated in Figure 1.

A multiple alignment is a natural generalization of this approach. However, in order to combine two interlacings we need one sequence to be common in both of the interlacings. This is shown in Figure 2.

Note that there is a one-to-one correspondence between any such combined interlacing and a multiple alignment.

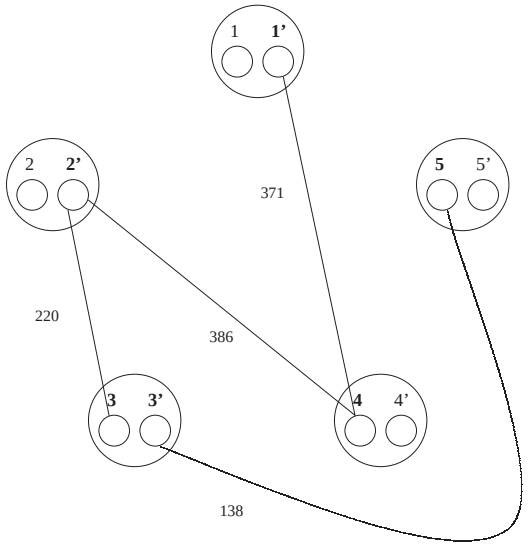

Fig. 3. The maximum spanning tree to be used in the multiple alignment.

Therefore, if we want to construct a multiple alignment of n sequences, we need a maximum spanning tree over the overlap graph for the sequences. This is similar to the approach taken in Kececioglu and Myers (1995). However, in Kececioglu and Myers (1995) a subset of the constraints in a collection of pairwise alignments are satisfied in forming the multiple alignment. Here, we are able to satisfy all such constraints. Moreover, the proposed approach is a new look at the multiple alignment problem in terms of sequence interlacing.

Let T be the number of fragments in the subgroup. As we already have the scores of the optimal semiglobal alignment of each pair in the subgroup, this induces a graph with $2T$ nodes and $2T(T-1)$ edges. The maximum spanning tree for the induced graph can be found using Prim's algorithm (Gross and Yellen, 1999) in $O(T + T\log T)$ time. Note that although the induced graph contains $2T$ nodes (as both f and $\bar{f}$ are represented by a node), we need a maximum spanning tree of T nodes where each node is either f_i or $\bar{f}_i$. In order to obtain the multiple alignment all we need is the relative orientation of the fragments as we add a branch to the maximum spanning tree (the optimal alignment between f_i and $\bar{f}_j$ implies the optimal alignment between $\bar{f}_i$ and f_j with the same score). A sample overlap graph is illustrated in Figure 3, where only the branches of the maximum spanning tree are drawn.

Prim's algorithm finds the maximum spanning tree by iteratively adding the edges with the highest weights in a greedy fashion. Therefore, in forming the multiple alignment we merge the best scoring pairs following the

maximum spanning tree. In the proposed approach this corresponds to combining two interlacings that have a sequence in common.

Consider the interlacing in Figure 1. Without loss of generality, one needs to assign a priority order to the sequences. In this case $*$ has a higher priority than $\circ$. This implies the following rule: In the interlacing, if a $\circ$ appears to the immediate right of an $*$ then it is placed under this $*$ in the alignment. On the other hand, if an $*$ appears to the immediate right of a $\circ$ that $*$ results in the start of a new column in the alignment. If there are two consecutive $*$s, the one on the left is aligned with a 'space' (_), whereas if there are two consecutive $\circ$s, the one on the right is aligned with a 'space'. The multiple alignment in Figure 2 is a natural generalization of this process. The priority order used here is $*$, $\circ$, and $\diamond$. When inserting the second interlacing (the one involving $\circ$s and $\diamond$s) into the first interlacing (the one involving $*$s and $\circ$s) we take the first $\circ$ and all the $\diamond$s (the lower priority symbol) to the left of it. This block is replaced with the first $\circ$ in the first interlacing. If there are any $\diamond$s to the left of the $\circ$ in this block, this $\diamond$ is moved to the left until it sees a symbol that has a lower priority than itself or until it sees a symbol it has previously jumped over. This process is repeated until the last $\circ$ in the second interlacing is inserted in the first one. If there are any $\diamond$s left to the right of the last circle in the second interlacing, they are inserted in accordance with the priority order.

Consensus When a consensus sequence is developed, we keep track of the number of bases in the sequences being aligned at each location. This number is called the *vote* received by that particular base. Once a multiple alignment is obtained the characters of the consensus sequence are those that receive the maximum vote at each location of the multiple alignment. The gaps at the extremities of any sequence are not considered in the voting.

The votes obtained by each constituent consensus sequence is used in the voting process of the multiple alignment of the consensus sequences. This property enables the algorithm to perform multiple alignment on a much finer scale than is possible with most existing algorithms.

Recursion

The consensus sequences of the clusters can be considered as a new collection of fragments. Hence, we repeat the procedure applied to the original collection of fragments. The overall system is shown in Figure 4.

In the beginning, the box denoted 'Fragments' contains the output obtained by shotgun sequencing. We compute the AMI profile for each fragment. In the box denoted 'Vector Quantization' the fragments are clustered using the Euclidean distance between their AMI profiles as a measure of similarity. Each cluster is processed using

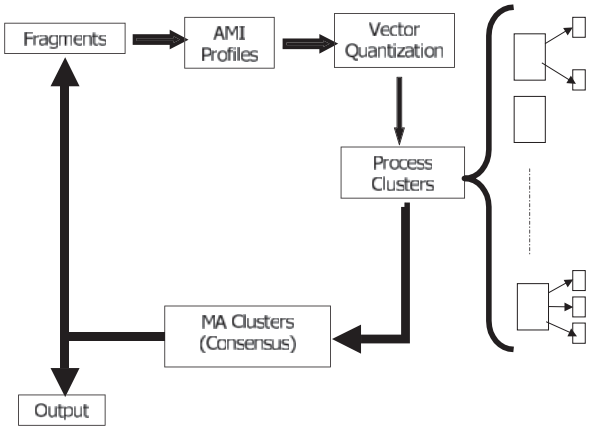

Fig. 4. Overall system.

the pairwise semiglobal alignment of its members and partitioned further if there are fragments that arise from distinct regions of the target sequence. In the next step we calculate the multiple alignment of the fragments in each cluster, keeping the 'vote' in each column to be used in later multiple alignments.

The set of consensus sequences is now considered as a new collection of fragments. This process is repeated until there is one cluster left, or until no new cluster is born after all clusters are processed. In the first case we have a final consensus sequence for the target, in the second case, we end up with a number of contigs that can be ordered arbitrarily.

RESULTS AND DISCUSSION

The proposed system was tested using two sets of simulations. In the first set, the goal was to explore the performance of the system with respect to the various parameters used in the algorithms. The sequences used in this set were two random sequences of length 50 000, the first 50 000 bases of two yeast chromosomes (GenBank Accession No: X59720 and D50617 respectively), and the first 100 000 bases of the second yeast chromosome. These target sequences were processed using GenFrag (Engle and Burks, 1994) which provides the fragment set (with random orientations and errors) intended to imitate the output of a real shotgun sequencing project. In all simulations the coverage is 6 and the average fragment length is 550.

In the second simulation set, we used the output of a real shotgun sequencing project. The sequence used is the complete sequence of Mus Musculus Chromosome 2 RP23-58L18 (GenBank Accession No: AC087332) which is 143 720 bp long and contains many repeats. The 5823 fragments used in the assembly programs were obtained from the Trace Archive (http://www.ncbi.nlm.nih.gov/Traces/).

Table 1. Simulation results on data with no base call errors ($\epsilon = 0$) and $\approx$ 3% base call errors ($\epsilon \approx 0.03$). $|C|$: length of the final consensus, m: matches between the target and consensus

	$\epsilon = 0$		$\epsilon \approx 0.03$	
Sequence	$\|C\|$	m	$\|C\|$	m
Random1	50 000	50 000	50 436	49 766
Random2	50 000	50 000	50 477	49 744
Yeast1	50 000	50 000	50 456	49 756
Yeast2	50 000	50 000	50 411	49 780

Table 2. Performance as a function of base call errors, ϵ. $|C|$: length of the final consensus, m: matches between the target and consensus

ϵ	Length	# fragments	$\|C\|$	m
0.05	50 000	546	50 741	49 527
0.04	50 000	546	50 545	49 703
0.03	50 000	546	50 411	49 780
0.02	50 000	546	50 246	49 865
0.01	50 000	546	50 134	49 929
0	50 000	546	50 000	50 000

Table 3. The input (number of fragments: Fr) and output (number of clusters: Cl) parameters for the vector quantizer at each iteration (I); the maximum, minimum, average and standard deviation values of the number of fragments in clusters; and the number of clusters that are further partitioned due to clustering error

I	Fr	Cl	Max	Min	μ	σ	Split
1	546	64	14	5	8	4.35	52/64
2	252	32	14	5	7	4.79	27/32
3	181	16	19	7	11	5.29	13/16
4	141	8	25	10	17	6.92	7/8
5	101	4	30	15	25	6.92	4/4
6	75	2	46	29	37	10.44	2/2
7	35	1	35	35	35	0	

The data used in both simulation sets were also assembled using three popular fragment assembly programs (PHRAP 0.990319 Green, 1994; CAP3 Huang and Madan, 1999; and STROLL 1.2 Chen and Skiena, 1997) for purpose of comparison.

The results of the first simulation set are tabulated in Table 1 where $|C|$ denotes the length of the final consensus sequence and m denotes the number of matches in an optimal global alignment between the target sequence and the final consensus sequence. In all cases the program was able to suggest one final consensus sequence. The length of the AMI profile vector used in simulations is 16. Increasing the size of the profile vector beyond 16 had little or no effect on the final answer. The initial number of clusters was 64. At each iteration, the number of clusters produced by the vector quantizer was halved. Six iterations were used for all cases.

All four of the sequences in Table 1 are 50 000 bp long and the initial number of fragments used (the output of GenFrag) is 546 for all cases. The results show that when no error is introduced, the algorithm is able to construct the target sequence with 100% similarity in all cases. When approximately 3% error is introduced the similarity between the target sequence and the final consensus sequence is over 99% and the length of the final consensus sequence is within 1% of the target sequence. This holds true for all cases.

Table 2 presents the performance of the proposed algorithm as a function of error rate. The test sequence chosen for this simulation is the *yeast2* sequence. As expected, the length of the consensus sequence converges to the length of the target sequence as the error rate decreases. The similarity of the consensus sequence to the target sequence is also inversely proportional to the error rate and is greater than 99% in all cases. We also tested the performance of the proposed algorithm as a function of increasing coverage. However, the results did not suggest a significant improvement, and hence are not included here.

Table 3 shows the number of clusters at each iteration when the target sequence is *yeast2* with an error rate of 3%. The minimum, maximum, average, and standard deviation values for number of fragments in clusters are also provided. The final column shows the number of clusters that are split due to *clustering error*. For example at the first iteration, out of the 64 clusters obtained by the vector quantizer eight of them had fragments that arose from the same region of the target sequence and, therefore, did not require further partitioning. The number of splits can be reduced by starting with a larger number of clusters in the first iteration. These results suggest that at early stages of the algorithm the average value of the number of fragments in a cluster is close to coverage providing a balanced classification. This can also be observed from the relatively small standard deviation.

The effect of the AMI profile vector size and the initial number of clusters are shown in Tables 4 and 5. In all cases the algorithm produced one final consensus sequence. The target sequence is the first 50 000 or 100 000 bases of *yeast2* subject to an error rate of 3%.

Table 4 shows that the overall performance of the proposed system increases with the size of the AMI profile vector. However there is a certain saturation point beyond which no improvement is observed. The saturation occurs around 16 when the sequence length is 50 000 and 32 when the sequence length is 100 000.

The effect of the initial number of clusters used in the 'Vector Quantization' step is similar to that of the

Table 4. Performance as a function of AMI profile vector length, R, using a fixed initial number of clusters, J. Target sequence length, $|T| = 50\,000$ bp ($J = 32$) or 100 000 bp ($J = 128$)

R	$\|T\| = 50\,000$	$\|T\| = 100\,000$
4	48 785	97 556
8	49 120	98 131
12	49 652	98 426
16	49 780	98 613
24	49 793	98 855
32	49 826	99 084
48	49 826	99 103
64	49 826	99 103

Table 5. Performance as a function of initial number of clusters, J, using a fixed AMI profile vector length, R. Target sequence length, $|T| = 50\,000$ bp ($R = 16$) or 100 000 bp ($R = 32$)

J	$\|T\| = 50\,000$	$\|T\| = 100\,000$
32	49 717	98 755
48	49 755	98 871
64	49 780	98 963
96	49 796	99 036
128	49 804	99 084
192	49 804	99 117
256	49 804	99 117
384	49 804	99 117

size of the AMI profile vector. The system performance increases with increasing number of clusters. This is due to the algorithm's ability to produce better clusters when the initial number of clusters is large. This affects the final consensus sequence as the clusters in each iteration become less vulnerable to clustering errors. As in the case of AMI profile vector size, there is a limit to this improvement and one can heuristically pick a large value that produces satisfactory results. Finally it should be noted that the effect of AMI profile length is more significant than the effect of the initial number of clusters.

We also tested the proposed algorithm against some of the existing popular fragment assembly programs. Again the test sequence was *yeast2* with 3% base call errors and with no errors. All of the algorithms generated a number of contigs as their output failing to suggest one final consensus sequence. The results are shown in Table 6.

In the case of no errors, the proposed system outperforms the existing procedures by producing one final consensus sequence identical to the target sequence. All three existing programs produce 2 contigs (PHRAP and CAP3 have identical results) with a high total number of matching bases (11 errors in the case of PHRAP and CAP3 and 33 errors in the case of STROLL). Note that in all cases

Table 6. Comparison of fragment assembly programs (P: PHRAP, C: CAP3, S: STROLL, Pr: Proposed) on data with no base call errors and $\approx$ 3% base call errors

		$\epsilon = 0$		$\epsilon \approx 0.03$	
	Contig	$\|C\|$	m	$\|C\|$	m
P	1	44 014	43 993	10 007	9 980
	2	6 008	5 996	34 106	33 890
	3			6 022	5 976
	Sum	50 022	49 989	50 115	49 846
C	1	44 014	43 993	10 006	9 980
	2	6 008	5 996	11 535	11 332
	3			22 637	22 531
	4			6 003	5 946
	Sum	50 022	49 989	50 191	49 789
S	1	5 907	5 870	9 942	9 913
	2	43 999	43 997	33 844	33 724
	3			5 863	5 803
	Sum	49 906	49 867	49 649	49 440
Pr	1	50 000	50 000	50 411	49 780

STROLL tends to underestimate the total length whereas all remaining programs (including the proposed system) result in overestimates.

In the case of erroneous data, the performance of all four of the programs are very close. When the total number of matching bases are compared, the maximum difference between any two programs is about 0.0032% hence negligible. It should be noted that the contigs produced by the existing approaches are overlapping for both error rates. Therefore this leads to slight overestimates of the total number of matching bases. There is no mis-assembly in any of the cases. As in the case of no errors, the proposed system is able to produce one final consensus sequence. This is not true of the other programs.

A problem encountered during sequencing is that of repeats. When there are repeating regions in the target sequence, the fragment assembly algorithms tend to overcompress the final consensus by combining the repeat regions. As suggested by earlier work (Chen and Skiena, 1997), the assembly programs are helpless when the length of the repeat region is greater than the fragment length. Although the proposed algorithm does not provide a special solution to this problem, the idea of statistical clustering handles it in a primitive way. If the fragments that belong to the repeat region happen to be placed in different clusters then the proposed system successfully reconstructs the target. Unfortunately, that may not always be the case. To investigate this property, we tested our program on sequences that have repeats. In cases where the fragments that came from repeat regions were placed

Table 7. Comparison of fragment assembly programs on real BAC data set. The column labels denote the total number of contigs, total number of significant contigs, and the average percent similarity between the significant contigs and the corresponding segments in the target

Program	Contigs	s. contigs	% similarity
Proposed	6	6	0.976
PHRAP	133	1	0.988
CAP3	49	17	0.969
STROLL	25	8	0.961

in the same cluster, the resulting consensus sequence was much shorter than the target sequence. However, when we ran the program and forced it to use alternative clusterings, some of the simulations provided the correct answer. Noting that the length of the target sequence can be estimated with at most 10% error (Setubal and Meidanis, 1997), by repeatedly running the program and discarding the cases where the final consensus is well short of the expected target length, we can handle the case of sequences with repeats. Although this issue needs to be addressed more thoroughly in future work, the existing system does provide a practical solution to the fragment assembly problem with satisfactory performance.

In the second set of simulations we used a long, repeat rich DNA segment. The target sequence was the 143 720 bp long Mus Musculus Chromosome 2 RP23-58L18 that contains many repeats. The 5823 fragments used in the assembly programs are the real shotgun sequencing outputs and were obtained from the Trace Archive of the NCBI website. In keeping with common practice we assumed we could estimate the length to within 10% of the actual length. In some cases the large number of repeats caused overcompression resulting in sequences of length around 50 000 bp. As this is substantially less than the smallest estimate of the target length, we discarded these results. Following the approach described above we ran the simulation until the total length of the contigs were within 10% of the target sequence, suggesting a reasonable output. The first reasonable output was found in the third simulation and for similar (repeat rich) target sequences a reasonable output was always found within the first five simulations. In the case of Mus Musculus Chromosome 2 RP23-58L18 we obtained a reasonable output in six of ten simulations. The AMI profile vector length used in simulations was 32 and the initial number of clusters was 1024. In each of the 'reasonable' six cases, the final contigs covered all the bases in the target sequence. The overlap between consecutive contigs were not sufficient for any further merges. In Table 7 the average of these six simulations together with the results for the other programs is presented.

The programs PHRAP, CAP3, and STROLL produced 133, 49, and 25 contigs out of which the number of significant contigs were 1, 17, and 8 respectively. In all four programs the significant contigs completely spanned the target sequence. In Table 7 we list the average percent similarity between the significant contigs and the corresponding segments in the target sequence. The proposed algorithm resulted in a much smaller number of contigs with a comparable percent similarity.

Finally, we should note that, except for PHRAP, none of the programs (including the proposed algorithm) resulted in mis-assembly. The only significant contig produced by PHRAP was 153 142 bp long. The first 18 060 base pairs of this contig matched the end of the target sequence (125 661–143 720) while the end of the contig (26 878–152 669) matched the beginning of the target (1–125 788).

CONCLUSIONS

In this paper we propose a new approach to solve the fragment assembly problem. We substantially reduce the size of the problem by clustering fragments. The measure of similarity used for clustering is the Euclidean distance between the AMI profiles of the fragments. This turns out to be an efficient measure as AMI profiles are robust to errors and remain the same when calculated for the reverse complement of a sequence. By using a divide-and-conquer approach we are able to process a feasible number of fragments at a time and calculate the consensus sequence on a finer scale in a shorter time. The expected number of fragments in a cluster is N/J where N is the number of fragments and J is the number of cluster. This reduces the complexity by a polynomial of J in overlap and an exponential of J in remaining phases (as the problems are NP-complete). The simulation results are very promising both for artificial and real data sets. The algorithm reconstructs the target sequence with over 99% similarity and within 1% of its length with a base call error rate of up to 5%.

Future work can focus on investigating different methods for clustering (as an alternative to k-means), different methods for measures of similarity (as an alternative to AMI profiles), and a strategy to handle repeats. Finally it should be noted that the proposed technique is not limited to fragment assembly of shotgun data but can be applied to other problems such as EST clustering and assembly.

REFERENCES

Bauer,M. (2001) *A distance measure for DNA sequences*, PhD thesis, University of Nebraska-Lincoln.

Burks,C., Engle,M.L., Forrest,S., Prsons,R.J., Soderlund,C.A. and Stolorz,P.E. (1994) *Automated DNA Sequencing and Analysis*. Academic Press, New York, pp. 249–259.

Chang,W. and Lawler,E. (1990) Approximate string matching in sublinear expected time. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*. pp. 118–124.

Chen,T. and Skiena,S.S. (1997) Trie-based data structures for sequence assembly. In *Proceedings of the 8th Symposium on Combinatorial Pattern Matching*. pp. 206–223.

Churchill,G., Burks,C., Eggert,M., Engle,M.L. and Waterman,M.S. (1993) Assembling DNA sequence fragments by shuffling and simulated annealing. *Technical report Los Alamos National Laboratory Los Alamos*, NM.

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

Engle,M.L. and Burks,C. (1994) GenFrag 2.1: new features for more robust fragment assembly benchmarks. *CABIOS*, **10**, 567–568.

Green,P. (1994) PHRAP documentation. http://www.phrap.org.

Gross,J. and Yellen,J. (1999) *Graph Theory and its Applications*. CRC Press, Boca Raton, FL.

Grosse,I., Herzel,H., Buldyrev,S.V. and Stanley,H.E. (2000) Species independence of mutual information in coding and non-coding DNA. *Phys. Rev. E*, **61**, 1–6.

Huang,X. (1992) A contig assembly program based on sensitive detection of fragment overlaps. *Genomics*, **10**, 18–25.

Huang,X. and Madan,A. (1999) CAP3: A DNA sequence assembly program. *Genomics*, **9**, 868–877.

Karp,R. and Rabin,M. (1987) Efficient randomized pattern-matching algorithms. *IBM J. Res. and Dev.*, **31**, 249–260.

Kececioglu,J.D. and Myers,E.W. (1995) Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, **13**, 7–51.

Kim,S. and Segre,A.M. (1999) AMASS: A structured pattern matching approach to shotgun assembly. *J. Comp. Biol.*, **6**, 163–186.

Luo,L., Lee,W., Jia,L., Ji,F. and Tsai,L. (1998) Statistical correlation of nucleotides in a DNA sequences. *Phys. Rev. Lett.*, **58**, 861–871.

MacQueen,J. (1967) Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium*. pp. 281–297.

Meidanis,J. (1993) Rethinking the DNA fragment assembly problem. In Baeza-Yates,R. and Ziviani,N. (eds), *Proceedings of First South American Workshop on String Processing*. UFMG, Belo Horizonte, Brazil, pp. 123–134.

Myers,E.W. (1995) Towards simplifying and accurately formulating fragment assembly. *J. Comp. Biol.*, **2**, 275–290.

Peltola,H., Soderlund,H. and Ukkonen,E. (1984) SEQAID: a DNA sequence assembling problem based on a mathematical model. *Nucleic Acids Res.*, **12**, 307–321.

Pevzner,P., Tang,H. and Waterman,M.S. (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl Acad. Sci. USA*, **98**, 9748–9753.

Sandelin,E. (2001) A Monte Carlo approach to sequence assembly. *Bioinformatics*, in press.

Sayood,K. (2000) *Introduction to Data Compression*, 2nd edn, Morgan Kaufmann, San Francisco, CA.

Setubal,J. and Meidanis,J. (1997) *Introduction to Computational Molecular Biology*, ITP.

Shannon,C.E. (1948) The mathematical theory of communication. *Bell Sys. Tech. J.*, **27**, 379–423.

Staden,R. (1982) Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequencing. *Nucleic Acids Res.*, **10**, 4731–4751.