A Comparison of DNA Fragment Assembly Algorithms

Lishan Li and Sami Khuri
Department of Computer Science
San José State University
San José, CA 95192-0249, USA

Abstract

As more research centers embark on sequencing new genomes, the problem of DNA fragment assembly for shotgun sequencing is growing in importance and complexity. Accurate and fast assembly is a crucial part of any sequencing project and many algorithms have been developed to tackle it. Since the DNA fragment assembly problem is NP-hard, exact solutions are very difficult to obtain. In this work, we present four heuristic algorithms, which we designed, implemented and tested. We compare the algorithms and the data structures of the four heuristics and present results of our experiments. We also compare our results with the assemblies produced by the well-known packages: PHRAP and CAP3.

Keywords: Fragment assembly, shotgun, overlap, layout, consensus, optimization algorithm.

1. The DNA Fragment Assembly Problem

DNA fragment assembly is a technique that attempts to reconstruct the original DNA sequence from a large number of fragments, each several hundred base-pairs long. The DNA fragment assembly is needed because current technology, such as gel electrophoresis, cannot directly and accurately sequence DNA molecules longer than 1000 bases. However, most genomes are much longer. For example, a human DNA is about 3.2 billion nucleotides in length and cannot be read at once. The following technique was developed to deal with this limitation. First, the DNA molecule is cut at random sites to obtain fragments that can be sequenced directly. The overlapping fragments are then assembled back into the original DNA molecule. This strategy is called shotgun sequencing. Originally, the assembly of short fragments was done by hand, which is not only inefficient, but also error-prone. Hence, a lot of effort has been put into finding techniques to automate the shotgun sequence assembly. The general outline of most assembly algorithms is first to create a set of candidate overlaps by examining all pairs, followed by forming an approximate layout of fragments, and finally creating a consensus sequence. All existing methods rely on heuristics, since the fragment assembly problem is NP-hard. More specifically, assembling DNA fragments is divided into three distinct phases:

a) Overlap Phase - Finding the overlapping fragments. This phase consists in finding the best or longest match between the suffix of one sequence and the prefix of another. We compare all possible pairs of fragments to determine their similarity. Usually, the dynamic programming algorithm is used in this step to find semiglobal alignments.

b) Layout Phase - Finding the order of fragments based on computed similarity scores. This is the most difficult step because it is hard to determine true overlaps. After the order is determined, the progressive alignment algorithm is applied to combine all the pairwise alignments obtained in the overlap phase.

c) Consensus Phase - Deriving the DNA sequence from the layout. The most common technique used in this phase is to apply the majority rule in building the consensus. Other methods exist for finding the consensus, such as the use of probabilistic scores in PHRAP [5].

The DNA fragment assembly problem is NP-hard [11], therefore, it is not possible to find an exact algorithm that solves this problem and runs in polynomial time (unless P = NP). The complexity of the problem increases even further due to the following factors.

a) Unknown orientation: After the original sequence is cut into many fragments, the orientation is lost. The sequence can be read in either 5' to 3' or 3' to 5'. One does not know which strand should be selected. If one fragment does not have any overlap

with another, it is still possible that its reverse complement might have such an overlap.

b) Base call errors: There are three types of base call errors: substitution, insertion, and deletion errors. They occur due to experimental errors in the electrophoresis procedure. Errors affect the detection of fragment overlaps. Hence, the consensus determination requires multiple alignments in high coverage regions.

c) Incomplete coverage: It happens when the algorithm is not able to assemble a given set of fragments into a single contig.

d) Repeated regions: Repeats are sequences that appear two or more times in the target DNA. Repeated regions have caused problems in many genome-sequencing projects, and none of the current assembly programs can handle them perfectly [1].

e) Chimeras and contamination: Chimeras arise when two fragments that are not adjacent or overlapping on the target molecule join together into one fragment. Contamination occurs due to the incomplete purification of the fragment from the vector DNA.

Over the past decade a number of fragment assembly packages have been developed, such as Phrap [5], TIGR assembler [13], STROLL [2], CAP3 (Contig Assembly Program) [6], Celera assembler [9], and EULER [11]. In our work, we design, implement and run four programs based on different computational methods to tackle the problem. We direct the reader who is interested in this work and who would like to know more about our algorithms, their design and implementation, to Li's work [8].

The remainder of this work is organized as follows. Section 2 introduces our first algorithm, Genetic Algorithm (GA). Its fitness functions were inspired by Parsons' GA [10]. Section 3 presents the Greedy Algorithm inspired by Elloumi's Approximation Algorithm [3] that constructs and orders the Best Set of Maximum Weight Contigs (BSC) instead of ordering fragments. Section 4 introduces the Structured Pattern Matching Algorithm. It was inspired by Kim's pattern matching approach [7], in which we first construct a detailed map of the problem at hand and then determine the target sequence based on a fingerprinting scheme. Our experiments show that this algorithm is the most accurate and fastest among our four algorithms. Section 5 discusses the Clustering Heuristic Algorithm, which uses a clustering technique to order fragments and progressively builds the fragment layout. Finally, we give our experimental results in Section 6 followed by our conclusion.

2. DNA Fragment Assembly Using the Genetic Algorithm

The most challenging step in "overlap-layout-consensus" DNA fragment assembly is to order the fragments. Since finding the exact order of the fragments is an extremely slow process, heuristic techniques, such as Genetic Algorithm (GA) can be used. Our GA heuristic was inspired by the work of Parsons et. al. [10]. The GA population consists of a set of individuals. Each individual represents one possible alignment. The search space for the fragment assembly problem is the set of all possible solutions in the population. We use the permutation representation with integer number encoding. Permutation representation requires special operators to make sure that we always get legal solutions. In order to maintain a legal solution, the two conditions that must be satisfied are: first, all fragments must be presented in the ordering, and second, no duplicate fragments are allowed in the ordering.

The fitness function measures the quality of the alignment and finds the one that yields the best score. It is applied to each individual and it should guide the genetic algorithm towards the optimal solution. We implemented two fitness functions. Fitness function F1 sums the overlap score for adjacent fragments in a given solution. The goal here consists in finding the permutation of fragments (an individual in the population) that has the highest score.

$$F1(I) = \sum_{i=0}^{n-2} w(f[i], f[i+1])$$

The second fitness function, F2, not only sums the overlap score for adjacent fragments, but also sums the overlap score for all other possible pairs.

$$F2(I) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} |i-j| \times w(f[i], f[j])$$

This fitness function penalizes solutions in which strong overlaps occur between non-adjacent fragments in the layouts. The objective of F2 is to minimize the overlap score. The overlap score in both F1 and F2 is computed using the semi-global dynamic programming algorithm.

In genetic algorithms, operators are applied to a population of individuals to create a new population. In our assembler, we employ three such operators: selection, crossover, and mutation. We use ranking selection mechanism, in which the GA first sorts the individuals based on their fitness values and then selects the individuals with the best fitness score until the specified population size is reached.

We implemented two crossover operators: order-based and edge-recombination. The purpose of the crossover operator is to allow partial solutions to evolve in different individuals and then combine them to produce a better solution.

The mutation operator is used for modification of single individual. We use the swap mutation operator, which randomly selects two positions from a permutation and then swaps them. It is a good operator for permutation problems such as ordering fragments.

3. DNA Fragment Assembly Using The Greedy Algorithm

Our Greedy Algorithm is based on the Best Set of Maximum Weight Contigs Approach [3]. The algorithm considers unknown orientation and missing fragments. The first step of the algorithm is to construct the Best Set of Maximum Weight Contigs (BSC). The complexity of this step is $O(n^2l^2)$, where n is the number of fragments and l is the average length of fragments. The second step of the algorithm is to order the Maximum Weight Contigs (MWC) of BSC based on contig overlaps order. The complexity of this step is $O(m^2l^2)$, where m is the number of MWCs.

The Greedy Algorithm computes contig overlaps rather than fragment overlaps. The advantages are twofold: it enables us to take only the true overlaps into account and it gives a better guarantee for finding the orientation of the fragments. The major steps of our Greedy Algorithm are:

A) Construction of BSC: The algorithm takes as input a set of fragments and outputs a best set of contigs stored in a vector template. The linked list template and the vector template are the major data structures used in this step. The linked list is composed of the overlap scores sorted in descending order. Each item contains a pair of fragment IDs and their overlap score. The first fragment is from the forward direction and the second fragment is the reverse complement of another fragment that has overlap length above some threshold with the first fragment. The vector represents a set of contigs. Each Contig object contains a pair of fragments and their overlap weight.

B) Ordering of Contigs: The time complexity of this step is $O(m^2l^2)$, where m is the number of contigs and l is the average length of the fragments. The algorithm takes as input the best set of contigs (output of the first step) and outputs a list representing the contigs ordering. The algorithm makes use of a vector representing the contigs information (the output of the first step).

4. DNA Fragment Assembly using Structured Pattern Matching

The Structured Pattern Matching Algorithm is based on a technique called hybridization fingerprinting that is usually used by biologists to deduce the overlap information among DNA clones from biological probes. DNA clones are exact copies of a particular part of a genome and are much longer than fragments [7].

To tackle the DNA fragment assembly problem, our algorithm divides the task into three phases. The first phase is called probe matching. Instead of using biological probes, we randomly select short probes (e.g. 12 bps) from each fragment. We then use exact pattern matching in determining the relative positions (i.e. probes occurrences) of the input fragments, including their reverse complements. Thus, each fragment is represented as an ordered set of probes and associated interprobe distances rather than a sequence of nucleotides. The second phase is called overlap map construction. It constructs a detailed map to show how fragments are ordered and how they align. We first determine how fragments overlap based on the probes occurrences obtained from the previous phase. Contigs consisting of a set of fragments are then constructed in a greedy fashion, guided by a heuristic measure of fragment alignments. However, it does not solely rely on scoring pairwise fragment alignments; instead, each fragment is dynamically scored against each contig. Comparing a fragment to a contig exploits the multiple coverage characteristics of shotgun sequencing data. The third phase is called sequence determination. It is relatively straightforward since all the information we need is available from the second phase.

The time complexity of the Structured Pattern Matching algorithm is approximately linear in the length of the target sequence. The efficiency is due to the compact encoding representation of fragments.

We now discuss the three phases of the algorithm in more detail:

1) Probe matching to identify probe occurrences in input fragments. It takes as input a set of DNA fragments and each fragment's reverse complement. The output of this phase consists of fragments represented as ordered set of probe occurrences and not of nucleotides (as with other traditional algorithms).

2) Overlap map construction:
Once the probes are selected and detected, the Structured Pattern Matching algorithm greedily constructs an overlap map based on the patterns of

probe occurrences information obtained in the previous step. The following steps describe the overview of the overlap map construction:

Step 1: Construct a pairwise overlap table that records overlap lengths for all possible pairs.

Step 2: Select the fragment pair with the highest score and construct an initial contig containing just these two fragments.

Step 3: The new contig is added to the set of contigs and rescored against all remaining fragments.

Step 4: The process continues, adding the best-scoring fragment to the contig and updating the modified contig's scores against all remaining fragments. If no fragment exhibits a significant overlap score, the process terminates, and a new contig is constructed.

3) Sequence determination.

Once the overlap map is completed, a consensus sequence is generated using all the available information. The time complexity is linear. From the overlap map, we know the order of the fragments in each contig and we also know that their relative left positions occur in the map, which makes the sequence determination straightforward.

5. DNA Fragment Assembly using the Clustering Heuristic Algorithm

The traditional three steps: overlap, layout, and consensus, are used in this algorithm. We use the semiglobal alignment algorithm to find all possible pairwise overlaps. When the overlaps are determined, we use a greedy heuristic in the layout phase to find the multiple sequence alignment among a set of fragments. We take the pair of fragments with highest overlap as the starting point. The layout is constructed by successively adding the fragment that has the highest overlap with the assembled fragments. This algorithm takes the unknown orientation into account. The idea is based on the clustering concept. It means that the fragment that is newly added into the alignment has the best overlap with either the last fragment or with the first fragment in the current alignment. Each fragment is progressively added into the existing alignment until no fragment is left.

In what follows, we describe the main parts of the Clustering Heuristic Algorithm.

Step 1: Construct a score table for all possible pairs of fragments considering forward directions and reverse complements.

Step 2: Sort the score in descending order and insert all FragmentPair objects (a pair whose score is above some threshold) into a linked list. Each FragmentPair node contains a pair of fragment IDs and overlap score. If the score is positive, it indicates that both fragments are from the same strand. A negative score means that the two fragments are from different strands.

Step 3: Select the first node (a,b) in the linked list as a starting point to order the rest of the fragments. Set a to be the first fragment and b to be the last fragment in the current layout.

Step 4: Select the next node in the linked list and compare the pair of fragments with the first and the last fragments in the current layout. If the clustering is successful, the fragment ID joins the set. If it needs to be inserted in the front, we reset the first fragment in the layout. If it needs to be appended at the end, we reset the last fragment in the layout. Otherwise, we put the node in a temporary sorted linked list. The process continues until the current linked list is traversed. Note that only one possible direction for each fragment can be chosen.

Step 5: When the current linked list is being traversed, a new contig is created. We need to remove the nodes that contain IDs in the selected fragment set from the temporary linked list. Next, we reset the temporary linked list as the current linked list and continue from Step 3 until no more fragments are left. Each contig contains the list of ordered fragment IDs.

6. Experimental Results

We basically conducted experiments on two types of sequence data to test our various algorithms. The two types differ in the average size of fragments: we have fragments with average length of around 380 base-pairs, and fragments of size approximately 700 base-pairs long. The first fragment size is very close to the size of fragments that were used in the Human Genome Project, while the second fragment size of 700 bps reflects the size of sequences that modern sequencers can handle nowadays. We chose our target sequences from NCBI (http://www.ncbi.nlm.nih.gov). We used GenFrag [4] to generate the different data sets shown in Table 1. GenFrag is a UNIX/C application created to accept DNA sequence input and generate shotgun format cleavages with fragment overlaps, in order to test any reassembly application. In other words, GenFrag performs DNA sequence cleavage to simulate wet lab fragmentation. It takes a known DNA sequence and uses it as a parent strand from which to randomly generate fragments according to the criteria (mean fragment length and coverage of parent

sequence) supplied by the user. The target sequences in Table 1 are arranged in increasing sizes, from 3,835 bps to 77,292 bps. Target Sequence displays the accession number of the corresponding sequence, while the number between parentheses under Data Set Name represents the coverage at each position of the target sequence.

We now describe the sequences that are summarized in Table 1.

(1) Target sequence X60189: A human MHC class III region DNA with fibronectin type-II repeats HUMMHCFIB. Sequence length is 3,835 bps.

(2) Target sequence NM_007123: A human Usher type 2. Sequence length is 6,332 bps.

(3) Target sequence M15421: A human apolopoprotein HUMAPOBF. Sequence length is 10,089 bps.

(4) Target sequence NC_001453: A sea urchin (Echinoderm) entire mitochondrial genome. Sequence length is 15,650.

(5) Target sequence NC_001807: The human (Homo sapiens) entire mitochondrial genome. Sequence length is 16,571.

(6) Target sequence J02459: First 40% of base pairs from LAMCG, the complete genome of bacteriophage lambda. Sequence length is 20,100 bps.

(7) Target sequence BX842596: A Neurospora crassa (common bread mold) BAC. Sequence length is 77,292 bps.

As can be seen in Table 1, sequences with accession numbers X60189, M15421, and J02459 have fragments of lengths in the neighborhood of 380 bps. Sequences NM_007123, NC_001453, NC_001807, and BX842596 all have fragments with sizes around 700 bps.

The four fragment assembly algorithms were implemented in C++. Each program runs from the command line and takes the fragments' data set in FASTA format as input. The Genetic Algorithm also takes a configuration file with parameters. All the experimental runs were performed on a Dell Inspiron 4150 with a 512MB memory and 1.7 Ghz CPU.

We evaluated each assembly result in terms of the number of contigs assembled. Since we obtain fragments from a known target sequence, we can compare our assembled consensus sequence with the target. Table 2 gives a summary of the results. As for the running times of the four algorithms, the times ranged from a few seconds to several hours. An overlap length of 30 was used as a threshold for all four algorithms and packages.

The Structured Pattern Matching Algorithm runs much faster than the other three algorithms. The time difference becomes even larger as we move to larger data sets. The Greedy Algorithm seems to perform poorer than the other three algorithms in finding the number of contigs. It is also the slowest of all four algorithms, sometimes requiring up to 6 hours to run to completion.

Table 1. Sixteen data sets generated with GenFrag

Target Sequence	Sequence Length (bps)	Data Set Name	Number of Fragments	Average Fragment Length (bps)	Coverage
X60189	3,835	X60189(4)	39	395	4
		X60189(5)	48	386	5
		X60189(6)	66	350	6
		X60189(7)	68	387	7
NM_007123	6,332	NM_007123(4)	37	691	4
		NM_007123(7)	64	680	7
M15421	10,089	M15421(5)	127	398	5
		M15421(6)	173	350	6
		M15421(7)	177	383	7
NC_001453	15,650	NC_001453(4)	90	708	4
		NC_001453(7)	157	676	7
NC_001807	16,571	NC_001807(4)	95	694	4
		NC_001807(7)	166	674	7
J02459	20,100	J02459(7)	352	405	7
BX842596	77,292	BX842596(4)	442	708	4
		BX842596(7)	773	703	7

Since the Greedy Algorithm, the Clustering Algorithm and the Genetic Algorithm were substantially slower than the Pattern Matching Algorithm , we decided not to run the three algorithms with BX842596 (of size 77,292 bps).

For data sets X60189(4), X60189(5), NM_007123(7), M15421(6), M15421(7) and NC_001453(7) all four algorithms assembled fragments as well as the commercially available packages: PHRAP and CAP3. For data sets X60189(6), X60189(7), and NM_007123(4) only the Greedy Algorithm produced two contigs while all other programs and packages gave only one contig. For sequences M15421(6) , M15421(7) and NC_001453(7) all algorithms produced two contigs.

As can be seen in Table 2, for 14 data sets out of the 16 data sets, the results obtained by the Pattern Matching Algorithm are the same as PHRAP's. The Pattern Matching Algorithm performed as well CAP3 for 14 data sets, and outperformed CAP3 with M15421(5).

We conclude that all four algorithms obtain excellent results on small problem instances. For the larger data sets, there is some variability in the number of contigs found by our four algorithms. Our Pattern Matching Algorithm competes pretty well with PHRAP and CAP3 for all the sizes of fragments we considered in this work.

7. Conclusion

The DNA fragment assembly is a very complex problem in computational biology. Since the problem is NP-hard, the optimal solution is extremely difficult to find. Hence, there are many computational techniques that attempt to find good solutions for this problem. In this work, we designed, implemented, tested, and analyzed four different algorithms in detail.

The algorithms use different techniques to tackle the DNA fragment assembly problem. For smaller data sets, all four algorithms got the same result in approximately the same running time. However, the results and the performance vary as the data sets become larger. For larger data sets, i.e., 50 or more fragments, the performance of the algorithms ranking from best to worst is: Structured Pattern Matching Algorithm, Clustering Heuristic Algorithm, Genetic Algorithm, and finally the Greedy Algorithm.

We would like to add that although the algorithms were influenced by techniques and algorithms that are found in the literature, our design and implementations vary from the original algorithms. Many features advanced by the authors of the original algorithms were either ignored or completely modified by our design and implementation. Therefore, our conclusions are valid only for our interpretations

Table 2. Results of the four algorithms and two packages with the sixteen data sets

Data Set Name	PHRAP	CAP3	Greedy Algorithm	Clustering Algorithm	Genetic Algorithm	Pattern Matching Algorithm
X60189(4)	1	1	**1**	**1**	**1**	**1**
X60189(5)	1	1	**1**	**1**	**1**	**1**
X60189(6)	1	1	2	**1**	**1**	**1**
X60189(7)	1	1	2	**1**	**1**	**1**
NM_007123(4)	1	1	2	**1**	**1**	**1**
NM_007123(7)	1	1	**1**	**1**	**1**	**1**
M15421(5)	1	2	6	2	**1**	**1**
M15421(6)	2	2	2	2	2	2
M15421(7)	2	2	2	2	2	2
NC_001453(4)	3	3	6	3	3	3
NC_001453(7)	2	2	2	2	2	2
NC_001807(4)	2	2	7	2	3	3
NC_001807(7)	2	2	3	2	3	2
J02459(7)	1	1	3	**1**	2	**1**
BX842596(4)	6	6	N/A	N/A	N/A	7
BX842596(7)	2	2	N/A	N/A	N/A	2

(designing and implementations) and do not reflect in any performance of the original algorithms.

It is important to realize that we did not attempt to take into consideration measures to handle repetitive sequences. Repeats are very difficult to assemble and often cause the fragment assembly problem to assemble fragments that come from different locations of the target sequence. Most algorithms in the literature have specific operators to handle repeats. For example, it is often the case that the algorithm is preceded by a preprocessing step that discards repeats. Our four algorithms do not have any special operators for handling repeats. One of the successful attempts for handling repeats can be found in the sequencer based on graph theory by Pevzner et al. [12].

The major reason behind the faster running times for the Structured Pattern Matching Algorithm is that it uses multiple pattern matching to detect pairwise overlaps, while the other three algorithms use the dynamic programming algorithm. When the number of fragments becomes larger and the average fragment length longer, the dynamic programming runs very slowly. Therefore, we believe that designing a hybrid pairwise pattern alignment in the overlap phase of the Genetic Algorithm and the Clustering Heuristic Algorithm should improve the running times dramatically without any loss of accuracy.

8. References

[1] Chen, T. and Skiena, S.S. A case study in genome-level fragment assembly. Bioinformatics, 16, 494-500, 2000.

[2] Chen, T. and Skiena, S.S. Trie-based data structures for sequence assembly. The Eighth Symposium on Combinatorial Pattern Matching, 206-223, 1997.

[3] Elloumi, M. and Kaabi, S., "Exact and approximation algorithms for the DNA sequence assembly problem", *SCI in Biology and Medicine*, Volume 8, 1999.

[4] Engle, M.L. and Burks, C., "Artificially generated data sets for testing DNA fragment assembly algorithms", *Genomics*, 16, 1996.

[5] Phrap, http://www.mbt.washington.edu/phrap.documentation.html 1994.

[6] Huang, X. and Madan, A, "CAP3: A DNA sequence assembly program", *Genome Research*, vol. 9, 1999, pp. 868-877, 1998.

[7] Kim, S. and Segre, A. M, "AMASS: A structured pattern matching approach to shotgun sequence assembly", *Journal of Computational Biology*, 6(2), pp. 163-186, 1999.

[8] Li, L, "Computational Techniques for the DNA Fragment Assembly Problem", MS thesis, Computer Science Department, San Jose State University, CA, USA, May 2003.

[9] Myers, E. W., "Towards simplifying and accurately formulating fragment assembly", *Journal of Computational Biology*, 2(2), pp. 275-290, 2000.

[10] Parsons, R. and Johnson, M. E., "A case study in experimental design applied to genetic algorithms with applications to DNA sequence assembly", *American Journal of Mathematical and Management Sciences*, no. 17, pp. 369-396, 1995.

[11] Pevzner, P. A., *Computational molecular biology: An algorithmic approach*, The MIT Press, London, England, 2000.

[12] Pevzner, P. A., Tang, H. and Waterman, M. S. An Eulerian path approach to DNA fragment assembly. *Proc. Nat. Acad. Sci*., 98 (17), pp. 9748-9753, 2001.

[13] Sutton, G., White, O., Adams, M., and Kerlavage, A., "TIGR Assembler: A new tool for assembling large shotgun sequencing projects", *Genome Science & Technology*, no. 1, pp. 9-19, 1995.