

**BARNACLE: AN ASSEMBLY ALGORITHM FOR
CLONE-BASED SEQUENCES OF WHOLE GENOMES**

BY VICKY CHOI

A dissertation submitted to the
Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Computer Science

Written under the direction of
Martin Farach-Colton
and approved by

New Brunswick, New Jersey

January, 2002

© 2002

Vicky Choi

ALL RIGHTS RESERVED

ABSTRACT OF THE DISSERTATION

BARNACLE: An Assembly Algorithm for Clone-based Sequences of Whole Genomes

by Vicky Choi

Dissertation Director: Martin Farach-Colton

The clone-based sequencing strategy is the strategy adopted by the International Human Genome Sequencing Consortium (IHGSC) to sequence the human genome. The data set generated by the clone-based approach consists of a set of Bacterial Artificial Chromosome (BAC) clones, typically 100-200Kb. Each BAC clone is individually sequenced and assembled. In general, a clone consists of a set of more or less non-overlapping fragments. The sequence assembly problem is to reconstruct the genome sequence by assembling the fragments. The primary difficulty of the problem is caused by repeats, which are sequences that appear two or more times in the genome. In addition, the assembly problem is further complicated by laboratory errors, such as chimeras and contaminations; by the draft quality of the sequences, which are sometimes inaccurate; and by polymorphism.

In this thesis, we propose an assembler, BARNACLE, that is based on a mathematically justifiable (unlike previous work) approach for assembling the sequences. First, we assemble the consistently overlapping fragments, which is a necessary condition for the true assembly. Then we use the clone-overlaps to resolve fragment-level inconsistencies. Making use of the fact that the clone graph must be interval, we detect and remove the repeat-induced overlaps. Furthermore, this also allows us to detect chimeric clones. In

order to resolve the non-interval graphs, we design an efficient algorithm which is based on a divide-and-conquer method. We then use the interval representation of BAC clones to order and orient the subassemblies. Finally, the additional information derived from plasmid-read, EST and mRNA data is used for orientation purposes. Moreover, unlike the other two existing algorithms, we are able to detect several types of errors in the input data.

We illustrate our approach by assembling the public working draft of the human genome, which consists of a set of finished and draft BAC clones produced by IHGSC. It takes 3 minutes on a Pentium III (933 MHz) computer for BARNACLE to assemble the April freeze of the public working draft of the human genome. We present and compare our results on the April freeze with the two public assemblies: NCBI's assembly and GIGASSEMBLER's assembly. We also present our findings of suspected input errors in this data set.

Acknowledgements

I am indebted to many people whom I can never thank enough. It is impossible to express my gratitude in words, but I will try to at least list their names here.

First and foremost, I thank my medical care team, which includes the doctors, nurses and therapists at St Mary's Hospital of Mayo Clinic, Rochester, Minnesota led by Dr. Michael Bannon and Dr. Scott Zietlow. Regretfully, I can not remember most of their names. Those I remember: my primary doctor, Dr. Michael Bannon; my ICU doctors, Dr. Scott Zietlow and Dr. Jeremy Brink; my orthopedic doctor, Dr. Mark Dekutoski; my rehabilitation doctor, Dr. Nnamdi Nwaogwugwu; my nurse, Diane Wroblewski; the social worker, Susan Predmore; my then occupational therapist and now my very special friend Rebecca McClintock and many, many more ... I would also like to thank the tireless visits of "strangers", including Jean and Herbert Ensz, Jean and Howard Helms, Sam and Susan Huang, Suei-Ching and Kai-Nan An. I would also like to thank my incredible friends Yanbin Luo, Amishi Gandhi, Grace Lo, Tony Milas, Ana Milanova, Sarah Koskie, XiaoHui Xin, Siu-Lok Choi, Alex Ng and my MPhil advisor Mordecai J. Golin. I would like to thank Dean Harvey Waterman of the Graduate School and his wife Judith Waterman, Professor Michael Fredman, Valentine Rolfe, Maryann Holtsclaw, and many more professors and graduate students in the Department of Computer Science, Rutgers University. I do not know how to express my gratitude, but each and every bit of their love has channeled into this work.

I am very thankful to my wonderful advisor Martin Farach-Colton. Without his trust, support, and guidance, this thesis would not have been possible. I thank the Program in Mathematics and Molecular Biology for the PMMB fellowship. I thank Dannie Durand for her early offer of a post-doctoral position. I thank Craig Nevill-Manning for providing the Pentium III computer for finishing up this project. I thank

the following people for their critical reading and comments of a journal version of this thesis: JinSheng Lai, Knut Reinert, Granger Sutton, Dannie Durand, Navin Goyal, Sachin Lodha and Wojciech Makalowski. I thank Navin Goyal, Sarah Koskie, Sachin Lodha and Akshay Vashist for helping me with the thesis writing and preparing the presentation. I thank my PhD thesis committee members: Martin Farach-Colton, Michael Fredman, Craig Nevill-Manning and Dannie Durand. Special thanks to Dannie Durand for her detailed comments.

I thank Rutgers University for providing such a wonderful environment. I thank Professors Michael Fredman, Endre Szemerédi, János Komlós and Jeff Kahn for their excellent theoretical computer science and discrete mathematics classes. I thank Professor Eric Allender for being such a great graduate director, our “super-mom” Valentine Rolfe, very competent Maryann Holtsclaw, always warm-hearted Geralyn Colvill and Kate Goelz. I thank the Student Bible Study Group of Rutgers Community Christian Church for the fellowship. I thank all my great friends Amishi, Ana, Annalisa, Petr, Sarah, Tony, XiaoHui, Yanbin at Rutgers throughout these years and my very special friend Govind.

Last but not least, I thank my wonderful parents for their unconditional love, and my brother, my sister, my nephew Billy and niece Christina for their love.

Dedication

To my medical care team at St Mary's Hospital, Mayo Clinic, Rochester, Minnesota

Table of Contents

Abstract	ii
Acknowledgements	iv
Dedication	vi
List of Tables	x
List of Figures	xi
1. Introduction	1
2. Biological Background	3
2.1. Genetic Background and Terminology	3
2.2. DNA Cloning	4
2.3. Reading DNA	6
2.4. Shotgun Sequencing	6
2.5. Sequencing the human and other whole genomes	7
3. Graph Theory Background	8
3.1. Basic Notations and Definitions	8
3.2. Characterizations of Interval Graph	9
3.3. Interval Graph Recognition Algorithms	10
3.4. Blocks of Interval Graphs	12
3.5. Lexicographic Breadth First Search (LBFS)	13
4. The Sequence Assembly Problem	15
4.1. Introduction	15
4.2. Details of Input	18

4.2.1. Sequence Information	18
4.2.2. Overlap Information	20
OVERLAPPER	22
4.2.3. Orientation Information	25
4.3. Our BARNACLE Assembler	27
5. The Algorithm	30
5.1. High level description of the algorithm	30
5.2. Details of each step	30
5.2.1. Classify Fragments	34
5.2.2. Assemble consistent overlapping MFs into subcontigs	35
5.2.3. Merge good subfragments into subcontigs	38
5.2.4. Detect and resolve conflict chromosome assignments	39
5.2.5. Construct a BAC graph from subcontigs	41
5.2.6. Resolve the inconsistent overlaps	41
5.2.7. Resolve non-interval components	42
The 5-SWEEP LBFS Algorithm	44
Identifying a forbidden subgraph	45
Overview of the method for resolving non-interval components	51
The Divide-and-Conquer Procedure	57
5.2.8. Obtain the ordering of BACs in the interval representation	59
5.2.9. Orient Subcontigs	60
5.2.10. Assign coordinates to subcontigs and order subcontigs by sorting lexicographically	61
5.2.11. Detect FNs	64
5.2.12. Detect FPs	66
5.2.13. Adjust the ordering of subcontigs	66
5.2.14. Orient subcontigs according to plasmid-read, EST, mRNA data	67
5.2.15. Derive a consensus sequence for each contig	69

6. Results and the comparison with the public assemblies	70
6.1. Input of the April freeze	70
6.2. Results on the April freeze	71
6.3. Comparison with the two public assemblies: NCBI's assembly and GI- GASSEMBLER's assembly	72
6.4. Examples of Suspected Errors Detected	75
7. Future Work and Remarks	79
References	83
Vita	85

List of Tables

5.1. The high level description of the algorithm.	32
5.2. The pseudocode of assembling consistent overlapping MFs into subcontigs.	37

List of Figures

2.1. DNA cloning. DNA to be cloned is inserted into a plasmid (a small, self-replicating circular molecular DNA found in bacterial cells). When the recombinant plasmid is introduced into bacteria, the insert will be replicated along with the rest of the plasmid.	5
3.1. Two interval graphs and their corresponding realizations.	9
3.2. The forbidden subgraphs of an interval graph.	11
4.1. Shotgun DNA sequencing. First, random shotgun sequencing is applied to the BAC to obtain sufficient sequence reads. Then sequence reads are assembled into fragments. Finally, the gaps are closed to obtain the finished sequence.	16
4.2. The originally proposed clone-based approach. A BAC library is constructed by fragmenting the target genome into BAC-sized segments. Then a physical map of the clones is constructed. According to the map, a minimum tiling set of clones (shown as solid lines; the remaining clones are shown as dotted lines) is selected and individually subjected to shotgun sequencing (see Figure 4.1).	17
4.3. Two types of valid overlaps: (1) dovetail overlap vs. (2) complete containment. Because of the draft quality of sequences, end errors are allowed (shown as dashed lines).	22
4.4. The alignment graph of two sequences s and t , where s dovetail overlaps with t	23
4.5. Contigs and subcontigs. A contig of three overlapping BACs and its corresponding ordered and oriented subcontigs.	27

4.6. True vs. repeat-induced overlaps. Consider the overlap of two fragments A and B. There are two possible inferences. (1) The overlap is true, where the fragments are from the overlapping segments of the genome. (2) The overlap is induced by repeated sequence R. 28

5.1. The idea behind our algorithm. Conceptually, the algorithm consists of three steps. (A) Fragments are “conservatively” assembled into subcontigs. The details of this step constitute steps 1-6 of the algorithm. (B) Consistent repeat-induced overlaps and chimeric clones will destroy the interval property of the clone graph. Resolving this step corresponds to step 7 of the algorithm. (C) According to the interval realization of clones obtained from the interval clone graph, subcontigs are oriented and ordered in steps 8-10 of the algorithm. Remark: Steps 11-15 of the algorithm, which use additional information to adjust the ordering and correct the orientation, are not shown in this figure. 31

5.2. Fragment representation. A fragment f is represented by $[1, m]$, where m is the length of f . The reverse complement of f is represented by $[m, 1]$. 31

5.3. Dovetail overlap. The overlap of f and g : $f, [a_1, a_2], g, [b_1, b_2]$. Assume $a_1 < a_2$ and $a_2 < m_f - \epsilon_f$. If $b_1 < b_2$, then $b_1 < 1 + \epsilon_g$ else $b_1 < m_g - \epsilon_g$. We call a_1 the end point of g on f , and b_2 the end point of f on g 33

5.4. Containment overlap. The overlap of f and g : $f, [a_1, a_2], g, [b_1, b_2]$. Assume $a_1 < a_2$. If $b_1 < b_2$, the sequence of g is contained in f , otherwise the reverse complement of g is contained in f . Note that $|b_1 - b_2| < m_g - \epsilon_g$. 33

5.5. Neighbors of a fragment (orientation not shown). The fragments in $L(f) = \{l_1, l_2\}$ ($R(f) = \{r_1, r_2, r_3\}$ resp.) are ordered by their end point on f . Thus l_1 is the rightmost left neighbor of f and r_1 is the leftmost right neighbor of f 35

5.6.	Consistent vs. inconsistent overlap (orientation not shown). (1) The overlap of $\{f, g\}$ is consistent because the consistency condition is satisfied, i.e., $R(g) \sqsubseteq R(f) \cup \{f\}$ and $L(f) \sqsubseteq L(g) \cup \{g\}$. (2) The overlap of $\{f, g\}$ is not consistent because $R(g) \not\sqsubseteq R(f) \cup \{f\}$. (3) The overlap of $\{f, g\}$ is not consistent because $L(f) \not\sqsubseteq L(g) \cup \{g\}$	36
5.7.	Assembling consistent overlapping maximal fragments. First, f_1 and f_2 are assembled; then f_2 and f_3 , f_3 and f_4	38
5.8.	G is a GoodSub, while B is a BadSub.	38
5.9.	Merge GoodSubs into Subcontigs. In this subcontig of $\{f_1, f_2, f_3, f_4\}$, the GoodSubs a, b of f_1 , c of f_3 are put back into the subcontig.	39
5.10.	A BAC graph constructed from subcontigs. Fragments r_1, \dots, r_4 belong to BAC R; b_1, b_2, b_3 are fragments of BAC B and g is a fragment of BAC G. There is an edge of BAC R and BAC B because of several overlaps of fragment r_i and b_j . Similarly there is an edge of BAC R and BAC G because of the overlaps of fragments g and r_3 or r_4 in the subcontig.	42
5.11.	Resolving inconsistent overlaps according to the connectivity of the BAC graph. The same BAC fragments are shown in the same color. Fragments r_i (b_i, g_i resp.) are fragments of BAC R (B, G resp). Assume that BAC R is adjacent to BAC B. (1) There are inconsistent overlaps between fragment $\{b_1, r_2\}$ and $\{b_1, g_1\}$. Since BAC R is adjacent to BAC B, the overlap of $\{b_1, r_2\}$ is chosen and the overlap $\{b_1, g_1\}$ is discarded as a repeat-induced overlap. (2) Fragment r_3 is a BadSub because fragment g_2 overlaps with r_3 but not b_2 which contains r_3 . Since BAC R is adjacent to BAC B, r_3 is made to become a GoodSub of b_2 . That is, the overlap of r_3 and g_2 is discarded as a repeat-induced overlap.	43
5.12.	Suspicious FNs. The overlaps of f and g_2 , f and h_2 are inconsistent because g_2 and h_2 do not overlap. However, $\mathcal{B}(f)$ overlaps with both BACs $\mathcal{B}(g_2)$ and $\mathcal{B}(h_2)$. This would suggest an FN of g_2 and h_2	44
5.13.	The “bow” of $\{b_1, bp, fv, b_3\}$	45
5.14.	The “bow” of $\{b_1, bp, fv, b_3\}$ in each type of forbidden subgraph.	46

5.15. Depending on the coverage of the sequences, there are three types of a chimera. Type (c1): The two parts of the chimera are far apart in one single contig. Type (c2): Two parts of the chimera lie in two different contigs. Both of them lies in the middle of the contig. Type (c3): Two parts of the chimera lie in two different contigs. One part of the chimera lies in the middle of the contig; the other part lies in the end of another contig.	49
5.16. Three cases of a chimera of Type (c2) shown in Figure 5.15. The forbidden subgraph is of Type II or Type III. Note that there might be several overlapping forbidden subgraphs caused by the chimera.	50
5.17. Depending on the coverage of the sequences, there are three types of a repeat. Type (r1): the repeat occurs in the same contig which would cause a cycle. Type (r2): the repeat occurs in two different contigs. Both of them lie in the middle of the contig. Type (r3): the repeat occurs in two different contigs. One of them is in the middle of its contig; the other lies in the end of its contig.	51
5.18. Three cases of a repeat of the Type (r2) shown in Figure 5.17 and their corresponding forbidden subgraphs caused.	52
5.19. A Type I forbidden subgraph. One of the four edges would be FP or one of the diagonals $\{a, c\}$ and $\{b, d\}$ would be FN. For the RV case, all four vertices are equally likely.	53

5.20. A Type II forbidden subgraph. The vertices $\{e_1, e_2, e_3\}$ are the AT-triple, also called outer vertices; $\{b_1, b_2, b_3\}$ are called inner vertices. The edge of $\{b_i, e_i\}$ is called the outer branch; $\{c, b_i\}$ is the inner branch. Observe that if one of the outer branches is FP, say $\{c, b_i\}$, then removing the branch $\{c, b_i\}$ would result in the interval for b_i being a subinterval of the interval for c in the interval realization. Thus it is more likely that the noise occurs in the inner branch. Also, although when all of the vertices of the forbidden subgraph are I-critical, (for the same reason of being a subinterval in the realization,) c is more likely to be the problematic one. Therefore, if we have to remove one I-critical vertex in order to resolve the component (for the RV case), c should be chosen. 54

5.21. A Type III forbidden subgraph. We define inner and outer vertices. The vertices $\{e_1, e_2, e_3\}$ are called outer vertices; $\{b_1, b_2, b_3\}$ are called inner vertices. The edge $\{e_i, b_i\}$ is an outer branch; $\{b_i, b_{i+1}\}$ is an inner branch. Because if the FP is an outer branch, the inner vertex would become subinterval, we assume the noise is more likely caused by the inner branch than by the outer branch. But if we have to remove one I-critical vertex in order to resolve the component and all three b_i s are I-critical, we arbitrarily choose one of them. 54

5.22. Depending on the connectivity of the graph, the removed BAC is classified as the more suspicious chimera and the less suspicious chimera. Type (ms1) and Type (ms2) are the more suspicious chimera, while Type (ls) is the less suspicious chimera. 55

5.23. Depending on the connectivity, there is a possibility that the more suspicious chimera is not a chimera but rather contains a repeat. For case 1, in which there are two contigs after removing the BAC, it is very likely it is a chimera. Even though there might be a gap in one contig, it is unlikely that it would be a repeat for otherwise it would result in a large gap in the other BAC. However, when one contig is connected by the removed BAC, i.e. the removal of the BAC would result in three or more contigs, it is possibly due to a repeat as shown in case 2.	56
5.24. Two possible orientations. Orient each subcontig such that it reflects the correct ordering. In this case, it is the left one.	60
5.25. The condition of adjacent subcontigs. The corresponding end BACs of the adjacent subcontigs must be either the same or overlapping.	61
5.26. An end BAC is missing, resulting an incorrect coordinate for the subcontig.	62
5.27. The subcontigs ordering and the corresponding interval realization.	63
5.28. The subcontigs ordering and the corresponding interval realization. The second order will be used if the coordinates $[\pi(e_k), \phi(a_k), \phi(e_k)]$ is used for a subcontig.	64
5.29. The ordered subcontigs are $S_1 = [B - t, 1, 4]$, $S_2 = [B, 10, 10]$, $S_3 = [B + 1, 4, 11]$, $S_4 = [B + s, 11, 15]$, where $\pi(10) = B$, $\pi(11) = B + 1$	65
5.30. For end-blocks, which have only one adjacent block, we permute the BACs within the end block according to the subcontig assemblies such that the warping of the BACs is minimized.	65
5.31. FNs detection. No matter how we order the subcontigs, the subcontig in the box will violate the adjacency condition. This is due to a FN as the arrow shows.	66

- 6.1. An example of a suspicious chimera. The suspicious chimera BAC *AL138797.8* has 19 fragments. Four fragments (total length of 110,361bp) overlap with one middle segment of a contig; another 14 fragments (total length 44,852bp) overlap with one middle segment of another contig. Below, the BAC in NCBI's build, 5 fragments were absent, 10 of the 14 fragments became singletons (i.e. the overlaps were discarded). 75
- 6.2. An example of a chromosome misassignment. BAC *AC051655* was assigned to chromosome 17 according to the annotation at GenBank record. (1)BARNACLE detects and corrects the chromosome assignment of BAC *AC051655*. Independent followup shows BAC *AC051655* shares an STS marker which was used to assign chromosome for BACs *AP002349* and *AC007030*. (2) The corresponding BACs in NCBI's build. 76
- 6.3. An example of wrong nt-pairs. According to the annotation of the 11 BACs of 8p21.3-p22 anti-oncogene of hepatocellular colorectal and non-small cell lung cancer (*AB020858* . . . *AB020868*), the NCBI's "ngtable", which is then used to generate nt-pairs, is generated for these BACs as shown in (1). However, our "conservative" assembly detects that all the BACs are in the wrong orientation/strand. For example, the wrong orientation of *AB020858.1* and *AB020859.1* (and hence the nt-pair) was detected by a draft BAC *AC072058.1* (as shown in (2)). Similarly, we have five other draft BACs to support the suspicion of wrong orientation of the other BACs. (3) is the corrected annotation of these BACs. . . . 77
- 6.4. An example of a chimeric fragment. According to our FN detection, we aligned *AP001275.3~11* against all its overlapping BACs. The alignments of *AP001275.3~11* and *AC010854.3~1* indicate that *AP001275.3~11* consists of two sequences separated by more than 50Kb. 78

7.1. Sequence overlap vs. fingerprint overlap. (1) There are 3 sequence overlaps, but they are incompatible. At least one of them is FP. (2) Treated as fingerprint overlaps (whether two fragments overlap) would result in an incorrect interval representation of the 3 BACs. 82

Chapter 1

Introduction

The fundamental blueprint upon which life is based is encoded in a kind of molecule called *deoxyribonucleic acid (DNA)*. A DNA molecule consists of two strands which are tied together in a helical structure. Each strand is represented by a string over the alphabet $\{A, C, G, T\}$, called a *DNA sequence*. Each character in a DNA sequence is called a *base*, *base pair* (bp) or *nucleotide*.

The process for determining the sequence of nucleotides of a region of DNA is called *DNA sequencing*. The result of such a sequencing experiment is called a *read*. With the current sequencing technology, one can directly determine a read of at most 1000bp (on average 500bp) at a time. To determine the sequence of much longer stretches of DNA (large-scale sequencing), researchers employ the *shotgun DNA sequencing* [14] strategy, in which a random sample of sequencing reads are collected from a target DNA sequence and then assembled by a computer program to infer the target sequence. The computational problem of reconstructing the target sequence by assembling the sequences from the data set is called the *sequence assembly problem*.

The primary difficulty of the problem is caused by repeated regions or *repeats*, which are sequences that appear two or more times in the target sequence. As the size of target sequence becomes larger, the repeats are expected to be more problematic and thus the sequence assembly problem would become harder. The complete set of DNA sequences of an organism is called a *genome*. The genome of higher organisms is expected to be repeat-rich. For example, more than half of the human genome is repeated sequences, which include large 50-500Kb duplicated segments with high sequence identity ($\geq 98\%$) [6].

For whole genomes, there are several different proposed sequencing approaches [27].

Two approaches for sequencing the human genome have been pursued. One is the *whole-genome shotgun sequencing*, adopted by Celera Genomics [16]. The other is the *hierarchical shotgun sequencing*, also referred to as *clone-based* or *BAC-by-BAC*, used by the International Human Genome Sequencing Consortium (IHGSC) [6].

In this thesis, we propose an algorithm, BARNACLE, for assembling sequences from the data set generated by the clone-based approach, with application to the human genome.

The thesis is organized as follows. Chapter 2 introduces biological background necessary to understand the problem. Chapter 3 introduces the graph theory background that is used to solve the problem. Chapter 4 describes the problem. Chapter 5 presents the algorithm in detail. Chapter 6 presents the results of the algorithm when applied to the input of April freeze of the public working draft of the human genome. Chapter 7 is the future work and remarks.

Chapter 2

Biological Background

In this chapter, we introduce the biological background necessary to understand the problem.

2.1 Genetic Background and Terminology

DNA strands

DNA is a double-stranded helix. Each strand is a sequence of nucleotides. There is an orientation along each strand, determined by the chemistry of its nucleotides' two ends, where the initial end is customarily described by the symbol 5', and the terminal end by 3'. The two strands are complementary, with A paired with T and C paired with G, and in opposite orientation. For example,

(5')	(3')
ACCATGGTGCACCTGACTCCTGAGGAG	
	TGGTACCACGTGGACTGAGGACTCCTA
(3')	(5')

That is, one is the reverse complement of the other. Thus, given one strand, one can readily infer the other strand.

Polymorphism

A *polymorphism* is a region of the genome that varies from individual to individual in a population. To be called a polymorphism, at least two variants of that region should be present in a significant number of people in the population.

The Human Genome

The human genome consists of 24 chromosomes: chromosomes 1, . . . , 22, X and Y . Chromosomes 1, . . . , 22, are called autosomes; X and Y are sex chromosomes. Most cells in humans contain 23 pairs of chromosomes. In case of a female, a cell contains 22 pairs of autosomes and a pair of X chromosomes; in case of a male, it contains 22 pairs of autosomes and one X chromosome and one Y chromosome. The size of the whole human genome is estimated to be around 3×10^9 base pairs [23]. The smallest chromosome (Y) contains ~ 50 Mb ($= 10^6 bp$); the largest chromosome 1 has ~ 250 Mb [23].

STS

A sequence region which is characterized by a unique pair of length 18 substrings 200-1000 bp apart is called a Sequence-Tagged-Site (STS).

GenBank

GenBank is a DNA sequence database maintained by the National Center for Biotechnology Information (NCBI).

2.2 DNA Cloning

In order to determine the sequence, one needs a large number of copies of the source DNA. One way of copying DNA is by *DNA cloning*. See, e.g., [23] for a more detailed description. A given piece of DNA is inserted into a *vector* to form a recombinant DNA molecule (See Figure 2.1). Vectors are DNA molecules originating from viruses, bacteria and yeast cells. They accommodate various sizes of foreign DNA fragments ranging from a few Kb for bacteria vectors (plasmids and cosmids) to 1Mb for yeast vectors (yeast artificial chromosomes). The piece of DNA to be copied is called an *insert*. Following introduction into suitable host cells, the inserted DNA gets replicated along with the host cell DNA. We then kill the host and dispose of the rest, keeping only the inserts in the desired quantity. The procedure thus effectively *cloned* a pure sample of the given insert. If the Bacterial Artificial Chromosome (BAC), which is

a genetically engineered plasmid is used as vector, the recombinant DNA molecule is called a *BAC clone*. The term BAC or BAC clone is also used to refer to the inserted DNA itself. The term will be used in this way in the rest of this thesis.

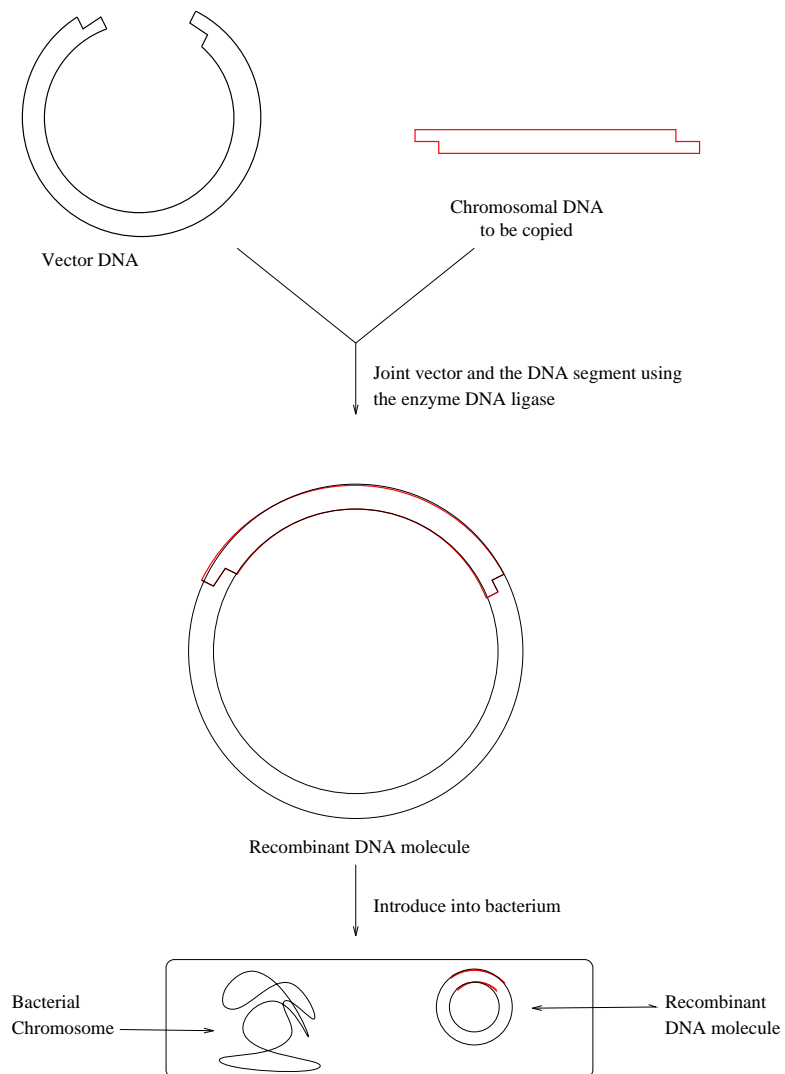


Figure 2.1: DNA cloning. DNA to be cloned is inserted into a plasmid (a small, self-replicating circular molecular DNA found in bacterial cells). When the recombinant plasmid is introduced into bacteria, the insert will be replicated along with the rest of the plasmid.

2.3 Reading DNA

The current method of determining the sequence of nucleotides involves first producing a collection of all prefixes of the given DNA strand such that the last nucleotide of the prefix is known. Then the sequence read is determined by a process called *gel electrophoresis*, which separates the prefixes by their length. The most commonly used approach is called *dideoxy sequencing* (also called chain-termination or Sanger method), in which dideoxynucleotide triphosphates act as the terminators for producing each prefix.

2.4 Shotgun Sequencing

The basic shotgun protocol [14] starts with a pure sample of a large number of copies of the source DNA whose sequence is to be determined, e.g. in the case of BACs, typically 100-200Kb. The protocol proceeds as follows: (Refer to [27] for a detailed description.)

1. Randomly shear the sample either using sound or passing it through a nozzle under pressure, which produces a uniformly random partitioning of each copy of the source strand into a collection of DNA fragments.
2. Select fragments using size separation under gel electrophoresis and then simply excising a band of the gel containing the desired size.
3. Clone the size-selected fragments (using DNA cloning technique) to obtain sufficient copies of each fragment.
4. Obtain sequence reads by dideoxy sequencing each fragment.

Since the shotgun DNA sequencing was devised by Sanger and his colleagues [14] in the early 1980s, the size of the source obtained by direct shotgun sequencing has increased over time, from 5 to 10 Kbp in 1980s to 40 Kbp (cosmid size) in early 1990. In 1995, making use of pair reads (mates), which are in opposite orientations and at a distance from each other approximately equal to the insert length, the whole genome of *H. Influenzae* with 1.8Mbp was sequenced. This achievement inspired Jim Weber and Gene Myers to propose whole-genome shotgun approach for the human genome [27].

2.5 Sequencing the human and other whole genomes

The Human Genome Project (HGP) was initiated in 1988 to sequence the entire human genome, which would provide an essential database to facilitate research in biochemistry, physiology, cell biology, and medicine. (See [10, 20] for its potential impact.) The HGP officially began in the United States in 1990. The sequencing strategy used is called the *hierarchical shotgun sequencing*, also referred to as *clone-based*, *BAC-by-BAC* or *clone-by-clone*. An alternative approach, called the *whole-genome shotgun sequencing*, was adopted by Celera Genomics [16] to sequence the human genome in 1998. Serving as a pilot project of the whole-genome shotgun approach, sequencing of the genome of *Drosophila* with length $\sim 120\text{Mb}$ was done by Celera Genomics, in collaboration with the Berkeley *Drosophila* genome Project (BDGP), in 2000. On February 2001, two draft sequences of the human genome produced by the two approaches were announced and published in *Nature* [6] and *Science* [16]. The goal is to finish sequencing of the human genome by 2003.

Chapter 3

Graph Theory Background

In this chapter, we introduce the graph theory background that is used to solve the problem.

3.1 Basic Notations and Definitions

Let $G = (V, E)$ be a simple undirected graph. For $A \subseteq V$, the *subgraph* induced by A is defined to be $G_A = (A, E_A)$, where $E_A = \{uv \in E : u \in A, v \in A\}$.

A vertex $v \in V$ is called an *articulation point* (AP) if the removal of v would partition the graph into disjoint connected components.

An *ordering* of V is a bijection $\phi : V \rightarrow \{1, \dots, n(= |V|)\}$. The ordering is also denoted by $(\phi^{-1}(1), \dots, \phi^{-1}(n)) = (v_1, \dots, v_n)$.

For $v \in V$, define neighborhood $N(v) = \{u \in V : uv \in E\}$ and $N[v] = N(v) \cup \{v\}$.

We say that vertices u and v are *indistinguishable* if $N[u] = N[v]$.

Let R be an equivalence relation on V defined by uRv iff $N[u] = N[v]$. Each equivalence class of R is called a *block* of G .

Definition 1 *A graph G is called an interval graph if there is one-one correspondence between its vertices and a set of intervals on the real line such that two vertices are adjacent in G iff their corresponding intervals overlap.*

The set of intervals assigned to the vertices of G is called an interval representation or realization of G .

Figure 3.1 show two examples.

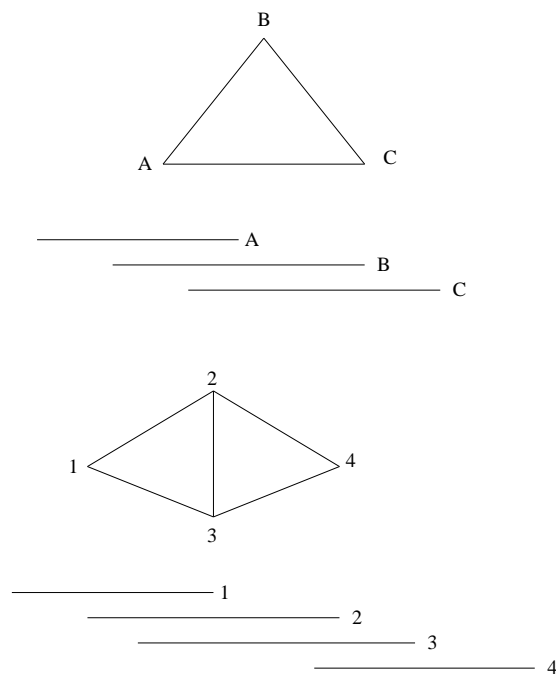


Figure 3.1: Two interval graphs and their corresponding realizations.

3.2 Characterizations of Interval Graph

Definition 2 A graph is chordal if it does not contain an induced cycle of size greater than 3.

Definition 3 An asteroidal triple (AT) is an independent triple of vertices such that between each pair of vertices in the triple there is a path that the third vertex is not a neighbor of that path.

A graph is asteroidal if it contains an AT.

For example, II, III, IV and V in Figure 3.2 are asteroidal. In each case, the vertices of the AT-triple are circled.

Observe that it is necessary that an interval graph be chordal and AT-free. To see this, suppose on the contrary that a graph contains an induced cycle of length k , $k \geq 4$, say $a_1 a_2 \dots a_k a_1$. Let I_1, \dots, I_k be the corresponding intervals in the realization. Then we have I_1 and I_3 disjoint, and I_2 overlaps with I_1 and I_3 , but not intervals I_j for $j > 3$, while these intervals I_j connect I_1 and I_3 . This is impossible.

Next suppose that G contains an AT (a_1, a_2, a_3) . The intervals I_1, I_2, I_3 are mutually disjoint. WLOG, suppose I_2 separates I_1 and I_3 . Then I_2 meets any path connecting I_1 and I_3 . That is, a_2 is a neighbor of any path of a_1, a_3 contradicts to the definition of asteroidal triple.

In fact, this necessary condition is also sufficient. This was proved by Lekkerkerker and Boland [26].

Theorem 1 *A graph is interval iff it is chordal and asteroidal triple-free.*

A less well-known characterization, is expressed in terms of forbidden subgraphs, also given in the same paper [26], stated below in Theorem 2.

Theorem 2 *A graph is interval iff it has no (induced) subgraph which is isomorphic to one of the graphs I, II, III, IV and V shown in Figure 3.2.*

We will use this characterization to help to resolve the non-interval graph.

3.3 Interval Graph Recognition Algorithms

There are several linear-time interval graph recognition algorithms [3, 25, 22, 21, 8]. The first four algorithms [3, 25, 22, 21] are based on the following theorem.

Theorem 3 *A graph $G = (V, E)$ is an interval graph iff its maximal cliques can be linearly ordered such that, for each vertex v , the maximal cliques containing v occur consecutively.*

The fifth algorithm [8]¹ is based on the following characterization.

Theorem 4 *A graph $G = (V, E)$ is an interval graph iff there is an ordering ϕ of V such that for any $u, v, w \in V$, with $\phi(u) < \phi(v) < \phi(w)$,*

$$uw \in E \implies uv \in E. \tag{3.1}$$

¹A flawed version of this algorithm appeared in [9].

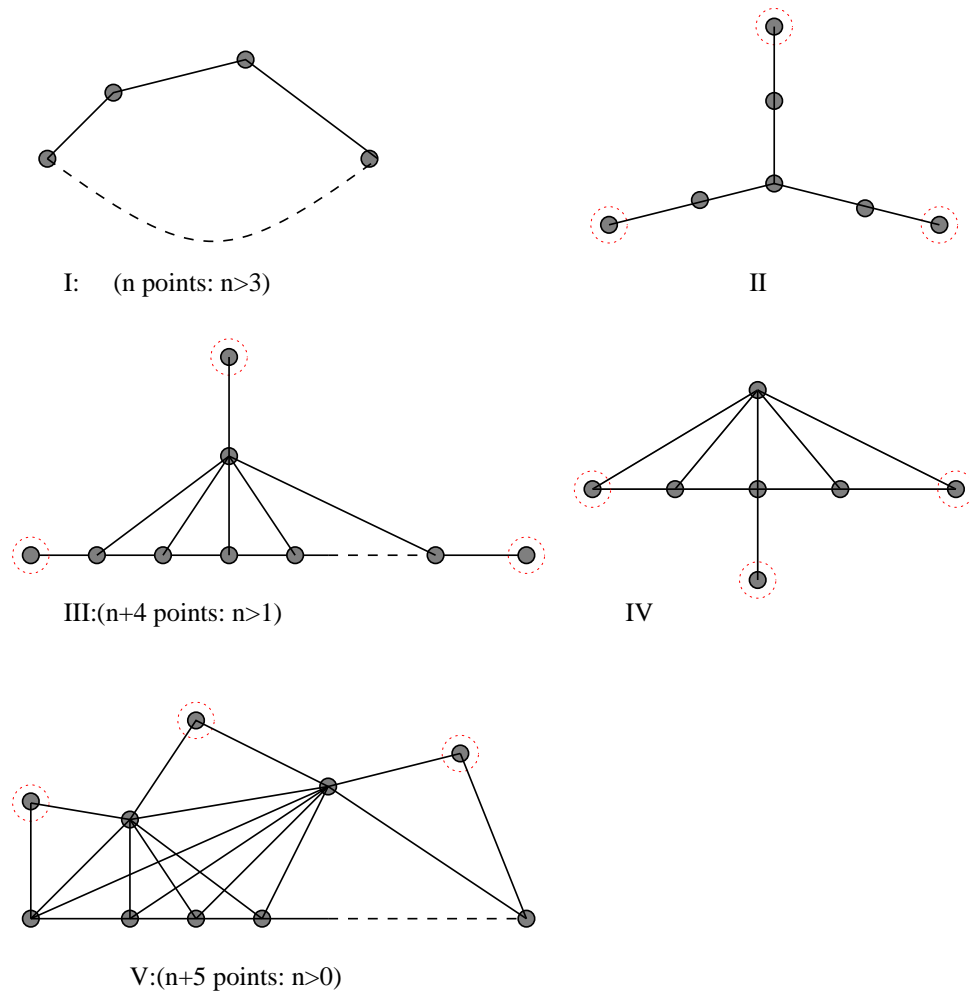


Figure 3.2: The forbidden subgraphs of an interval graph.

We call any ordering of V satisfying Condition 3.1 of Theorem 4 an *I-ordering*. Before we sketch the proof of Theorem 4, we establish a relationship between the I-ordering and the interval realization by observing another condition equivalent to (3.1).

Let $R_\phi[v] = \{u \in N[v] : \phi(u) \geq \phi(v)\}$, i.e. the set of the right neighbors of v w.r.t ϕ . Let $\gamma(v)$ be the largest vertex in $N[v]$, i.e. $\phi(\gamma(v)) \geq \phi(w) \forall w \in N[v]$.

Observation 1 *Let $G = (V, E)$, and ϕ a linear ordering of V . Then the following two conditions are equivalent:*

1. For any $u, v, w \in V$, with $\phi(u) < \phi(v) < \phi(w)$,

$$uw \in E \implies uv \in E.$$

2. For any $v \in V$,

$$R_\phi[v] = \{u : \phi(v) \leq \phi(u) \leq \phi(\gamma(v))\} \tag{3.2}$$

Note the Condition (3.2) $R_\phi[v] = \{u : \phi(v) \leq \phi(u) \leq \phi(\gamma(v))\} \iff \phi(R_\phi[v]) = [\phi(v), \phi(\gamma(v))]$, i.e, the right neighbors of v are consecutive, called this condition “right-nbr-consecutive”(RNC) condition.

Proof. Suppose G is interval, let \mathcal{I} be its interval realization. Then order the vertices by the left end point of its corresponding interval in \mathcal{I} . It is not difficult to verify such an ordering satisfies the Condition (3.1).

Conversely, suppose we have an ordering ϕ that satisfies the Condition (3.1). Represent each $v \in V$ by the interval $[\phi(v), \phi(\gamma(v))]$. From the Observation, it is easily seen that this is an interval realization for G . [For $u, v \in V$ with $\phi(u) < \phi(v)$, $uv \in E \iff \phi(v) \in [\phi(u), \phi(\gamma(u))] \iff \phi(v) \in [\phi(u), \phi(\gamma(u))] \cap [\phi(v), \phi(\gamma(v))] \neq \emptyset.$ ■

3.4 Blocks of Interval Graphs

Definition 4 *An interval graph is proper if the set of intervals in the realization can be chosen to be inclusion-free (i.e. no subintervals).*

Let $G = (V, E)$ be a proper interval graph. A *straight enumeration* of G is a linear ordering of the blocks in G , such that for each block, the block and its neighboring blocks are consecutive in the ordering.

The first and last blocks of a straight enumeration are called *end-blocks*. The rest of the blocks are called *inner-blocks*.

Theorem 5 *A (connected) proper interval graph has a unique ordering of blocks in G up to its full reversal.*

3.5 Lexicographic Breadth First Search (LBFS)

Lexicographic Breadth First Search (LBFS) was introduced to recognize chordal graph [28]. As the name suggests, LBFS is a special kind of BFS (Breadth First Search), in which ties are broken lexicographically. More precisely, each vertex is labeled by a bit vector of length $n(= |V|)$ bits, initially set to 0s. After the i th vertex is visited, set all its neighbors' i th bit to 1. Then the vertex with lexicographically largest label among unvisited vertices is chosen to be the $(i + 1)$ vertex to be visited. State formally:

```

Procedure LBFS( $G$ ):
Input: a graph  $G = (V, E)$ 
Output: an ordering  $\phi$  of  $V$ 
Initialize all vertices of  $V$  to be 0s;
for  $i = 1 \dots |V|$  do
    pick an unnumbered vertex  $v$  with lexicographically the largest lable;
     $\phi(v) = i$ ;
    for each  $w \in N(v)$  do
        update the  $i$ th bit of the label of  $w$  to 1;

```

The ordering ϕ of V produced by an LBFS procedure is called an *LBFS-ordering* of V . The implementation of LBFS can be done in $O(|V| + |E|)$ time. We do not actually calculate the labels, but instead keep the unnumbered vertices in lexicographic order. One implemenation of LBFS is shown below.

Implementation of procedure LBFS(G):

Input: a graph $G = (V, E)$

Output: an ordering ϕ of V

1. $L = \{\{V\}\}$;
2. for $i = 1 \dots |V|$ do
 3. $v =$ the first element of the first set in L ;
 4. remove v ;
 5. $\phi(v) = i$;
 6. split and replace each $L_j \in L$ into $N(v) \cap L_j$ and $L_j \setminus N(v)$;
 Note: put $N(v) \cap L_j$ in front of $L_j \setminus N(v)$;
 7. discard empty sets;

The LBFS-ordering is characterized by the following property [4].

Theorem 6 *An ordering ϕ of V is an LBFS-ordering iff for all vertices a, b, c of G with $\phi(a) < \phi(b) < \phi(c)$, $ac \in E$, $ab \notin E$ implies the existence of d with $\phi(d) < \phi(a)$ such that $db \in E$ and $dc \notin E$.*

Chapter 4

The Sequence Assembly Problem

In this chapter, we introduce some background and related work on the sequence assembly problem. After describing the details of input, we formally state the problem for our assembler, BARNACLE.

4.1 Introduction

The sequence assembly problem is to reconstruct the genome sequence by assembling sequences from a data set. The data set we are working with consists of a set of Bacterial Artificial Chromosome (BAC) clones, typically 100-200Kb, generated by the clone-based approach. We will use ‘BAC’, ‘clone’ and ‘BAC clone’ interchangeably. Each BAC clone is individually sequenced and preassembled. In general, a clone consists of a set of more or less non-overlapping *fragments* (see Figure 4.1), which are the preassemblies of the shotgun reads of each BAC clone produced by some assemblers, such as PHRAP (Green, unpublished). A clone is called a finished clone if it consists of only one fragment and it has accuracy of at least 99.99%, otherwise it is called a draft clone. As we mentioned in Chapter 1, the primary difficulty of the assembly problem is caused by repeats. In addition, the assembly problem of the clone-based approach is further complicated by laboratory errors, such as chimeras and contaminations; by the draft quality of the sequences, which are sometimes inaccurate; and by polymorphism.

The set of BAC clones, finished and draft, produced by IHGSC is called the *public working draft of the human genome*. There are two other algorithms for assembling the public working draft of the human genome. One is GIGASSEMBLER [24] from UC Santa Cruz and the other is NCBI’s unpublished algorithm [19].

It is worthwhile to point out that the clone-based strategy used is not the originally

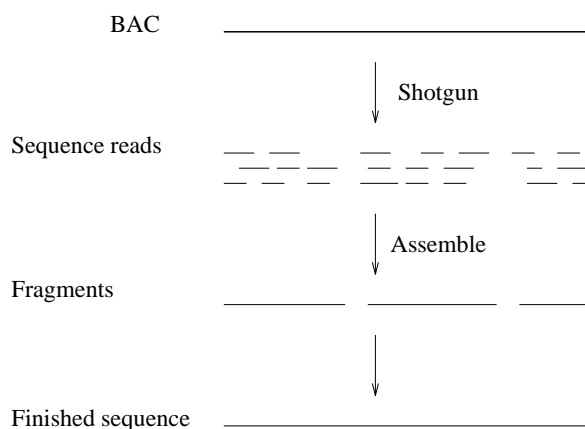


Figure 4.1: Shotgun DNA sequencing. First, random shotgun sequencing is applied to the BAC to obtain sufficient sequence reads. Then sequence reads are assembled into fragments. Finally, the gaps are closed to obtain the finished sequence.

proposed “clone by clone” strategy, in which a physical map giving the clone ordering is first produced, and then a minimum tiling set of clones in the map is selected and individually subjected to shotgun sequencing (see Figure 4.2). The idea is “mapping first then sequencing”. Mapping is done by assembling the BAC clones, whose sequences are to be determined, based on their *fingerprint* overlaps. The fingerprint overlaps are generated by comparing the fingerprints of BACs. A fingerprint of a BAC is supposed to describe unique information contained in the BAC. There are several types of fingerprints: restriction length digests [17], restriction map [30], oligo probe hybridization [12] and STS probes [18]. In practice, the physical map of the clones based on fingerprint overlaps is constructed concurrently with the sequencing [6]. The ordering of the sequenced clones is either unknown or not accurately known.

The fingerprint-based physical map [7] was employed by GIGASSEMBLER to assist in the sequence assembly, although the clone ordering was only roughly determined in the map [24]. NCBI does not make use of the fingerprint map. Instead, several STS marker maps are used for chromosome assignments. Both algorithms are based on greedy approaches to assemble sequences in which the best overlap is assembled first, where *best* is defined by some score functions used to assign weights for overlaps. Since the genome is highly repetitive, these score functions are unlikely to yield the

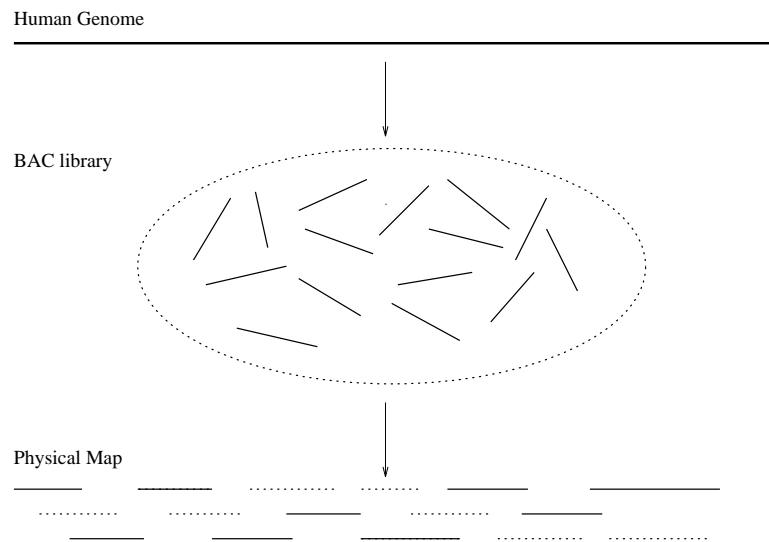


Figure 4.2: The originally proposed clone-based approach. A BAC library is constructed by fragmenting the target genome into BAC-sized segments. Then a physical map of the clones is constructed. According to the map, a minimum tiling set of clones (shown as solid lines; the remaining clones are shown as dotted lines) is selected and individually subjected to shotgun sequencing (see Figure 4.1).

true assembly [2]. Without the fingerprint clone contig to restrict BACs to, the NCBI algorithm first formulates a Maximal Interval Subgraph (MIS) problem to obtain a BAC ordering and then assembles the sequences of overlapping BACs. This approach is natural given the history of the Genome Project, with its original physical-mapping-first approach. However, this top-down approach suffers from the fact that if a BAC is misplaced, the mistake in the assembly is unrecoverable.

Our assembler, BARNACLE, uses the same input data as NCBI, that is, sequences from BAC clones augmented with chromosome-assignment. Part of the novelty of our approach is to assemble the genome bottom-up, without making use of fingerprint-based physical map of the clones. That is, we use extensive sequence-level overlaps to suggest BAC overlaps. This approach allows us to order BACs with very high confidence. We are able to achieve much more reasonable final sequence-level assemblies. Also, instead of using score functions to attempt to resolve the repeat problem, our way of resolving the repeat problem is by considering the consistency of overlaps and the interval graph formalism which are mathematically justifiable. In addition, this enables

us to detect inconsistencies in the underlying data. We remark that this error detection feature does not exist in the other algorithms.

4.2 Details of Input

In this section, we describe the input for BARNACLE, which includes the sequence information of BACs and fragments, overlap information and orientation information.

4.2.1 Sequence Information

A *BAC* consists of a contiguous stretch of DNA from a chromosome. Each BAC consists of a set of *fragments*, which are produced by an assembler, such as *PHRAP*, of the shotgun reads of subclones of the BAC (see Figure 4.1)¹.

The associated data of each BAC consists of the estimated length of the BAC, the phase of the BAC (defined below), the number of fragments, the sequence of each fragment and the chromosome of the BAC, if assigned.

Example 1

accession #	estimated length	phase	chr	number of fragments
AC002092.1	95456	1	17	4
	fragment acc#	start	end	length
	AC002092.1~1	1	888	888
	AC002092.1~2	889	46200	45312
	AC002092.1~3	46201	84925	38725
	AC002092.1~4.1	84926	95170	10245

Example 1 shows BAC *AC002092.1* which has an estimated length 95456bp and four fragments. Fragment *AC002092.1~1* is the sequence from 1 to 888 in the GenBank record of *AC002092.1*.

¹Notice that we only know the sequence of each of its fragments but we do not know the actual sequence of each BAC.

Sometimes the length of the BAC is estimated by taking the sum of the fragments' lengths plus (number of fragments - 1) times a constant, where (number of fragments - 1) is the number of *gaps* the BAC has and the constant is an estimate gap length.

Phases and additional fragment information

Depending on the raw data coverage (which is the average number of times that a basepair has been sequenced), BACs are categorized into three phases. Phases 1 and 2 are the draft sequences; the finished sequences comprises phase 3. For a phase 1 BAC, the fragments of the BAC are not necessarily disjoint, and the order and orientation of fragments are in general unknown. For a phase 2 BAC, the fragments are disjoint and the order of fragments is known. A phase 3 BAC is a finished sequence, i.e. only one fragment. In addition, for some phase 1 BACs, some partial fragment order information and end fragment information are also available. See Example 2.

Example 2

AL354895.5~1	AL354895.5	1	10864	1,1/5	?
AL354895.5~2	AL354895.5	10965	13132	1,2/5	?
AL354895.5~3	AL354895.5	13233	16145	1,3/5	?
AL354895.5~4	AL354895.5	16246	18439	1,4/5	?
AL354895.5~5	AL354895.5	18540	33466	1,5/5	?
AL354895.5~6	AL354895.5	33567	41015	2,1/3	?
AL354895.5~7	AL354895.5	41116	54375	2,2/3	?
AL354895.5~8	AL354895.5	54476	57517	2,3/3	?
AL354895.5~9	AL354895.5	57618	64286	3,1/2	?
AL354895.5~10	AL354895.5	64387	66845	3,2/2	?
AL354895.5~11	AL354895.5	66946	88981	?	?
AL354895.5~12	AL354895.5	89082	92846	?	?
AL354895.5~13	AL354895.5	92947	97618	?	?
AL354895.5~14	AL354895.5	97719	116211	?	?
AL354895.5~15	AL354895.5	116312	118464	?	?

AL354895.5~16	AL354895.5	118565	135271	?	?
AL354895.5~17	AL354895.5	135372	141729	?	?
AL354895.5~18	AL354895.5	141830	147926	4,1/2	?
AL354895.5~19	AL354895.5	148027	157164	4,2/2	T7,R

In example 2, fragment *AL354895.5~19* is an end fragment, next to vector T7. There are 4 groups which we know the order of fragments. First group consists of 5 fragments, which are the first 5 fragments and they are consecutive. Similarly for the other 3 groups.

Orientation

As already mentioned, DNA molecules are double strands. Fragments can come from either strand. In general, we do not know which strand a particular fragment belongs to. However, whatever strand it comes from, the sequence read always goes from 5' to 3'. Thus it is either the fragment as given or its reverse complement (in reverse orientation) the substring of the genome. That is, each fragment can be used either in the direct or the reverse orientation.

4.2.2 Overlap Information

In this section, first we define overlaps (which will be used for assembling sequences), and then describe an efficient algorithm, called OVERLAPPER, for searching overlaps.

Alignment

Let s be a sequence. Then $|s|$ denotes the length of s , and $s[i]$ denotes the i th element of s . A substring of s from the a th element to the b th element is denoted by $(s, [a, b])$. A space is denoted by a special symbol “-”.

Let s and t be sequences over alphabet Σ . A (global) *alignment* of s and t maps s and t to sequences s' and t' such that

1. $|s'| = |t'|$;

A local alignment s', t' of s and t is an *overlap* if (1) s' and t' are effixes, or (2) $|s'| > |s| - \epsilon_s$ or $|t'| > |t| - \epsilon_t$. The former is called the *dovetail overlap* while the latter is called *complete containment* as shown in Figure 4.3.

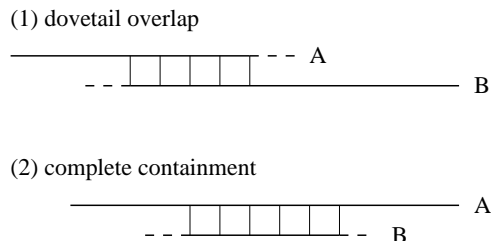


Figure 4.3: Two types of valid overlaps: (1) dovetail overlap vs. (2) complete containment. Because of the draft quality of sequences, end errors are allowed (shown as dashed lines).

Based on local alignments of all fragment sequences against all fragment sequences by some alignment algorithms, we further process the two types of overlaps (remark: because of the draft quality of the sequences and polymorphism, some overlaps might be corresponding to several pieces of local alignments, instead of one local alignment. For these cases, we need to join the pieces together in order to obtain the valid overlaps.) , with sequence identity ζ at least 97% and end-allowed-errors ϵ taken to be 50bp for finished sequences and the minimum of the 10% of the fragment length and 1000bp otherwise.

Also, according to the annotation of some finished sequences in the GenBank, some overlaps are generated. These are called *nt-pairs*. These overlaps include 0bp overlaps, i.e., the fragments do not overlap but are consecutive to each other.

OVERLAPPER

In this section, we describe an efficient algorithm for aligning highly similar sequences, called OVERLAPPER, for detecting overlaps. The details of the algorithm can be found in the unpublished manuscript [5].

Idea

Consider two sequences s and t with a dovetail overlap (the same idea applies to complete containment overlaps), that is, there is an effix s' of s that is almost identical to an effix t' of t . The alignment graph of s and t will look like Figure 4.4. Observe that

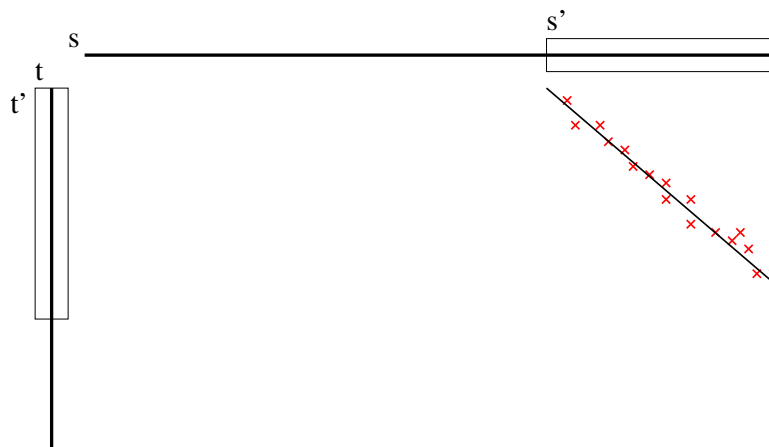


Figure 4.4: The alignment graph of two sequences s and t , where s dovetail overlaps with t .

the regions of the desired alignment (i.e. the overlap) are highly similar, the regions of interest should have many hits, which are exact matches of w -mers (a string of length w). To find the hits efficiently, we use the *dictionary matching* method as in BLAST [1]. First build a look-up table (*the dictionary*) of all w -mers of the subject sequence, and then “look up” each w -mer in the query sequence to see if and where it occurs in the subject sequence.

There are two approaches to the dictionary matching problem. First, encode each w -mer into an integer of $[1..4^w]$, that is, each w -mer can be used as an index into an array of size 4^w . The i th entry of the array contains a list of all occurrences of the corresponding w -mer in the query sequence. Second, build an Aho-Corasick tree (also called the DFA in BLAST) of all w -mers in $O(nw)$ time. Both data structures allow searching hits by scanning a sequence of length m in $O(m + H)$ time, where H is the number of hits. After finding the hits, unlike BLAST, we do not extend the hits to HSPs (defined in BLAST).

To efficiently find the regions of the overlap, which should have many hits, we organize the hits in their corresponding diagonals. Due to the indel, the hits of regions are not in a single diagonal. Instead, they should be in a narrow band. Merge the hits in neighboring diagonals into a band. For each band, if it contains a dense region (i.e. with many hits), a k -band alignment (or X -drop, or the “greedy” algorithm [29] if it is linear gap) is triggered to find the overlap.

Details of OVERLAPPER

Input: a query sequence Q of length n , a subject sequence dfs of length m

Output: the (almost) optimal alignments of the highly similar substrings of Q and dfs

Data Structure

- An array D : $D[i]$ contains a list of hits in the diagonal i , for $-m \leq i \leq n$; D is called the *diagonal array*.
- An array $Band$: $Band[t] = D[t..t+k-1]$, for $-m \leq t \leq n-k+1$. For fixed τ , a band is *significant* if its size is more than τ .
- A list SB : contains i if $Band[i].size > \tau$; SB is the list of all significant bands.

Algorithm

1. Build a look-up table of all w -mers in dfs ;
2. Scan Q to find hits, append each hit in its corresponding diagonal :

$$D[j-i] \leftarrow (i, j) \text{ if } dfs[i..i+w-1] = Q[j..j+w-1];$$

Time : $O(|Q| + \text{total number of hits})$.

Note since we scan Q sequentially, the first coordinate i of each diagonal is in increasing order.

3. Find all *significant* bands:

- (a) for $-m \leq t \leq n - k + 1$, compute $Band[t].size = \sum_{i=0}^{k-1} D[t + i].size$; if $Band[t].size > \tau$, append t to the list SB ;

Time: $O(n + m)$;

- (b) eliminate the overlapping bands : scan through the list SB , delete the smaller overlapping bands in $|SB|$ time.

- (c) merge $D[t..t + k - 1]$ by first coordinate to $Band[t]$ for each $t \in SB$.

Time: $O(\text{total number of hits in the bands})$.

4. For each significant band, find 'dense' region:

4.1 Scan through the hits in the band, count the consecutive hits whose first coordinate is within $[w + 1, \chi]$, maintain the maximum count.

Time = $O(\text{total number of hits in the bands})$.

4.2 If maximum count $> \tau$, align the region bidirectionally around a seed (e.g. the central point of the region, which can be obtained while scanning in the above step) by a k -band alignment algorithm (or X -drop, or the "greedy" algorithm if it is linear gap), stop when falls below X of the best score yet seen.

4.2.3 Orientation Information

There are sequences from paired-end plasmid reads, ESTs and mRNAs. These sequences are aligned against all fragment sequences. Based on these alignments, we further process the relative orientation of fragment pairs. Each paired-end plasmid read consists of two sequences in opposite orientation, one forward and one reverse. According to the alignments with each plasmid pair, we derive the relative orientation of fragment pairs. See Example 3.

Example 3

Plasmid Pair : {G33-617D11.f, G33-617D11.r}

Fragment Information:

AP001919.2~1	AP001919.2	1	49377
AC020735.5~8	AC020735.5	18463	27385

Alignment Information:

G33-617D11.f	AP001919.2	99.83	1	602	9041	8439
G33-617D11.r	AC020735.5	99.83	1	575	20594	20020

According to the alignments in Example 3, fragment AP001919.2~1 is in opposite orientation with the forward sequence G33-617D11.f; fragment AC020735.5~8 is in opposite orientation with the reverse sequence G33-617D11.r. Thus the relative orientation of AP001919.2~1 and AC020735.5~8 is -1 , i.e., opposite.

Each mRNA (similarly EST) consists of a set of disjoint exons (see, for example, [5] for a detailed description). According to the alignments with each mRNA, we derived the relative orientation of fragments pairs. See Example 4.

Example 4

mRNA: AB000267.1

Fragment Information:

AC060812.3~11	AC060812.3	134632	169945
AL158047.6~1	AL158047.6	1	165274

Alignment Information:

AB000267.1	AC060812.3	100.00	1	133	169081	169213
AB000267.1	AL158047.6	97.13	131	373	140031	140274

In Example 4, fragment AC060812.3~11 contains an exon of AB000267.1 from 1 to 133; fragment AL158047.6~1 contains the next exon of AB000267.1 from 131 to 373. (Ideally, the exons should be disjoint, but due to the sequence errors, we allow 25bp distance between exons.) Thus the relative orientation of AC060812.3~11 and AL158047.6~1 is 1, i.e., the same.

4.3 Our BARNACLE Assembler

A *contig* is a contiguous region that is covered by a set of overlapping BAC clones. A *subcontig* is a contiguous region that is covered by a set of overlapping fragments.

Given a set \mathcal{F} of fragments and a set \mathcal{O} of overlaps between fragments, by an assembly of \mathcal{F} we mean fragments are positioned relative to each other (in a linear order) according to some overlaps in \mathcal{O} such that the sequences of each overlapping region are almost the same (subject to the allowed-errors).

The assembly of the fragments which cover the subcontig is called the *subcontig assembly*. The contig assembly is defined similarly. When there is no danger of confusion, we will also refer to the subcontig (resp. contig) assembly as subcontig (resp. contig). Because of the gaps between fragments of a BAC, a contig is an ordered and oriented set of subcontigs (see Figure 4.5).

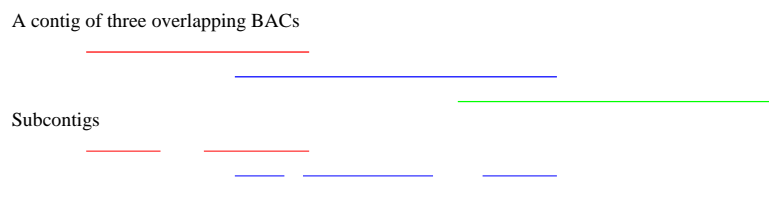


Figure 4.5: Contigs and subcontigs. A contig of three overlapping BACs and its corresponding ordered and oriented subcontigs.

We say an overlap is *true* if the fragments are derived from the overlapping segments of the genome, otherwise the overlap is *repeat-induced* (see Figure 4.6) [13]. A repeat-induced overlap is due to a repeated region, which occurs in two or more different location of the genome. The overlaps which are not true are called *False Positives* (FPs). In other words, repeat-induced overlaps are FPs. On the other hand, the overlaps which are true but not detected are called *False Negatives* (FNs). FPs and FNs are called noise. In addition, due to laboratory errors, there is another type of noise, called *chimeras*, in which two separate sequences are mistakenly put together to form a BAC.

Given a set of fragments, because of the noise, there might be many assemblies. The

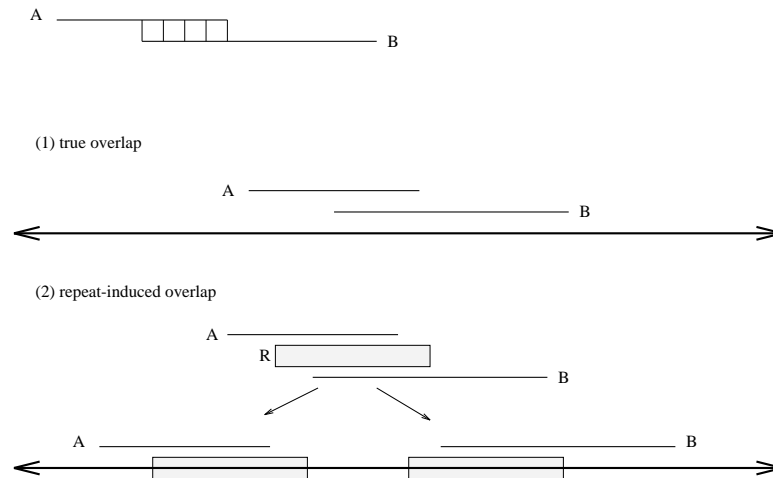


Figure 4.6: True vs. repeat-induced overlaps. Consider the overlap of two fragments A and B. There are two possible inferences. (1) The overlap is true, where the fragments are from the overlapping segments of the genome. (2) The overlap is induced by repeated sequence R.

goal is to construct *the true assembly*. The assembly problem would be straightforward if we could divine all true overlaps, i.e., if the data of overlaps were noise-free. The key objective is thus to clean up the noise as much as possible and assemble the fragments according to the true overlaps.

Definition 5 *Two BACs/fragments are agreeable if their chromosome assignments are the same or at least one of them is Unknown.*

Formally, the input to BARNACLE consists of a set of BACs, overlaps of agreeable fragments and relative orientation of fragment pairs. The output consists of a set of contigs, and for each contig, its assembly and a consensus sequence deriving from the assembly.

To measure the quality of the assembly, we introduce the definition of *warp*:

$$\text{warp} = \frac{\text{assembled BAC length}}{\text{estimated BAC length}}.$$

Intuitively, assuming that the estimated BAC length is accurate, then if the assembly is good, the assembled BAC length should be close to the estimated length, thus the warp should be close to 1. In other words, the number of BACs with warp around 1 will

be a good indicator whether the assembly is good or not. However, in some cases, the estimated BAC length might not be accurate, e.g., a BAC of length around 180Kb but an estimated length of around 6Kb (which is total length of fragments of the BAC). Thus to better measure the assembly, we make use of the upper bound of a BAC's length, e.g., for the human genome, the BAC length is upper bounded by 250Kb. One can then measure the assembly by the number of misassembled BACs, whose warp > 1.5 and their assembled length is more than 250Kb.

Chapter 5

The Algorithm

In this chapter, we first illustrate the basic idea of the algorithm, then present the high level description of the algorithm. Each step of the algorithm, is explained in detail in the individual subsections of Section 5.2.

5.1 High level description of the algorithm

The basic idea of the algorithm is illustrated in Figure 5.1. The high level description of the algorithm is shown in Table 5.1. Each step is explained in detail as an individual subsection of the next section.

5.2 Details of each step

Before we go into the details of each step, we introduce our representations for fragments and overlaps, and notation of oriented subsets.

Fragment Representation

For each fragment f , let m_f denote the length of f . Recall that each fragment can be in one of the two possible strands/orientations (either as given or its reverse complement). The positive orientation (as given) is represented by $[1, m_f]$, while the reverse complement of f is represented by $[m_f, 1]$. In either representation, the first base is called the left end of f while base at position m_f is called the right end of f (See Figure 5.2). Notice that the left (right resp.) is relative to the given orientation (and not its own orientation).

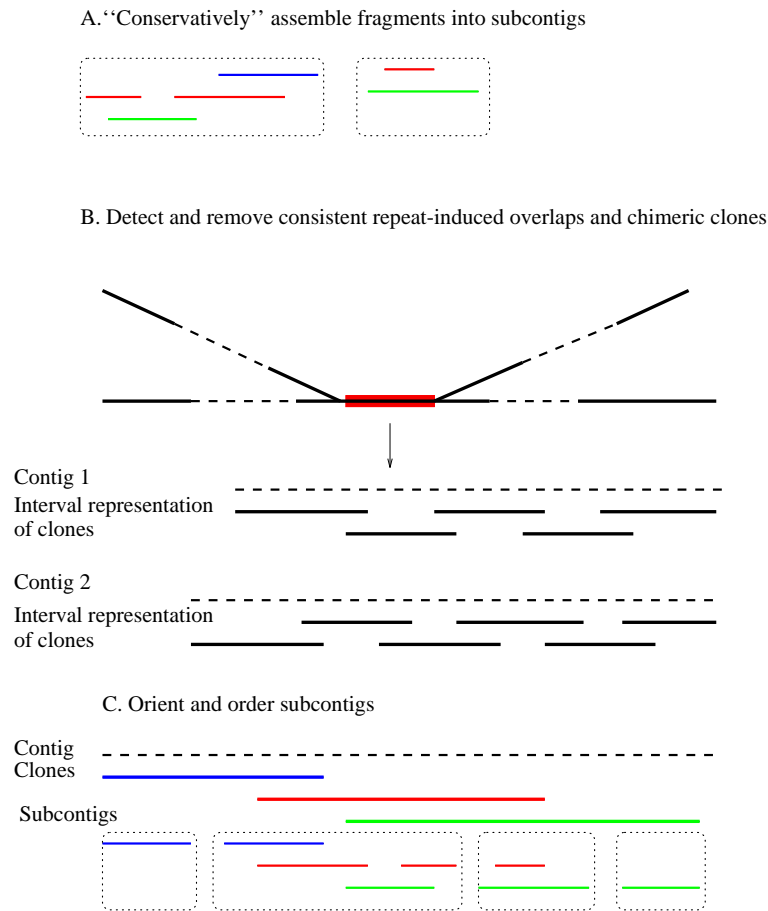


Figure 5.1: The idea behind our algorithm. Conceptually, the algorithm consists of three steps. (A) Fragments are “conservatively” assembled into subcontigs. The details of this step constitute steps 1-6 of the algorithm. (B) Consistent repeat-induced overlaps and chimeric clones will destroy the interval property of the clone graph. Resolving this step corresponds to step 7 of the algorithm. (C) According to the interval realization of clones obtained from the interval clone graph, subcontigs are oriented and ordered in steps 8-10 of the algorithm. Remark: Steps 11-15 of the algorithm, which use additional information to adjust the ordering and correct the orientation, are not shown in this figure.



Figure 5.2: Fragment representation. A fragment f is represented by $[1, m]$, where m is the length of f . The reverse complement of f is represented by $[m, 1]$.

- A. **“Conservatively” assemble fragments into subcontigs.**
1. Classify fragments: singleton, subfragment and maximal fragment.
 2. Assemble consistent overlapping maximal fragments into subcontigs.
 3. Put back *good* subfragments to subcontigs.
 4. Detect and resolve conflicting chromosome assignments.
 5. Construct a BAC graph from subcontigs.
 6. Resolve inconsistent overlaps according to the connectivity of the BAC graph.
- B. **Detect and remove consistent repeat-induced overlaps and chimeric clones.**
7. For each component G_i of the BAC graph, if G_i is not interval, resolve the component by removing repeat-induced overlaps or suspicious chimeric BACs.
- C. **Orient and order subcontigs.**
8. Obtain the interval representation of BACs.
 9. Orient subcontigs.
 10. Assign coordinates to subcontigs and order subcontigs by sorting lexicographically.
 11. Detect the potential false negatives and remove the involved fragments.
 12. Detect false positives (consistent repeat-induced overlaps that do not destroy the interval property).
- D. **Adjust the ordering and correct the orientation of the subcontigs using additional information.**
13. Adjust the order of the subcontigs according to the extra fragment information.
 14. Orient subcontigs according to the relative orientation of fragment pairs generated from plasmid reads, ESTs and mRNAs data.
 15. Derive a consensus sequence for each contig from the assembly of maximal fragments of the contig.

Table 5.1: The high level description of the algorithm.

Overlap Representation

Because of the property of reverse complementarity, if fragments f and g overlap, the reverse complement of f overlaps with the reverse complement of g . For each fragment pair, it suffices to know where the overlap is and what the relative orientation of the fragments is.

Let ϵ_f be the end-allowed-error for overlaps with f . For example, $\epsilon_f = \min\{10\% \cdot m_f, 1000\}$ for the public working draft of the human genome. Given an fragment pair f, g , based on the local alignments of f and g , we preprocess at most one valid overlap between them : $f, [a_1, a_2], g, [b_1, b_2]$, by checking whether the alignment could be a dovetail overlap (see Figure 5.3) or a containment (see Figure 5.4). Let $\delta(f, g)$ denote

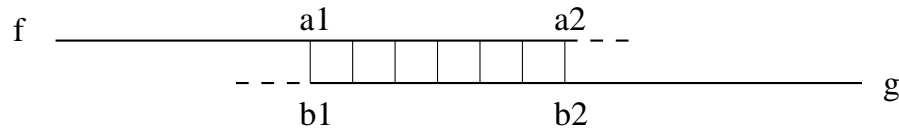


Figure 5.3: Dovetail overlap. The overlap of f and g : $f, [a_1, a_2], g, [b_1, b_2]$. Assume $a_1 < a_2$ and $a_2 < m_f - \epsilon_f$. If $b_1 < b_2$, then $b_1 < 1 + \epsilon_g$ else $b_1 < m_g - \epsilon_g$. We call a_1 the end point of g on f , and b_2 the end point of f on g .

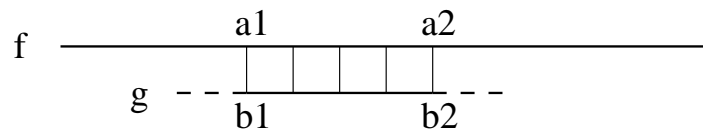


Figure 5.4: Containment overlap. The overlap of f and g : $f, [a_1, a_2], g, [b_1, b_2]$. Assume $a_1 < a_2$. If $b_1 < b_2$, the sequence of g is contained in f , otherwise the reverse complement of g is contained in f . Note that $|b_1 - b_2| < m_g - \epsilon_g$.

the relative orientation of f and g in the overlap. Without loss of generality, assume that $a_1 < a_2$. If $b_1 < b_2$, then both f and g are in the same orientation, otherwise they are in the opposite orientation; that is, either f overlaps with the reverse complement

of g or g overlaps with the reverse complement of f . Thus,

$$\delta(f, g) = \begin{cases} 1 & : (a_2 - a_1)(b_2 - b_1) > 0 \\ -1 & : (a_2 - a_1)(b_2 - b_1) < 0 \end{cases}$$

Oriented Subset

For $f \in \mathcal{F}$ the set of all fragments,

$$N(f) = \{g \in \mathcal{F} : g \text{ overlaps with } f\}.$$

That is, $N(f)$ is the set of fragments which overlap with f . Recall that there is relative orientation of an overlapping fragment pair. To check if a fragment overlaps with two or more different fragments in the same relative orientation, we introduce the concept of oriented subset.

Definition 6 For $f, g \in \mathcal{F}$, define $N(f) \sqsubseteq N(g)$ (pronounced as $N(f)$ is an oriented subset of $N(g)$) if for all $a \in N(f)$, $a \in N(g)$ and $\delta(f, a) = \delta(f, g) \cdot \delta(g, a)$.

Thus, $N(f)$ is an oriented subset of $N(g)$ if every oriented fragment that overlaps with f also overlaps with g in the same orientation.

5.2.1 Classify Fragments

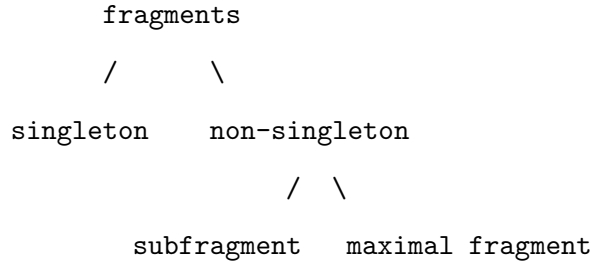
Let \mathcal{F} be the set of all fragments and let \mathcal{O} be the set of all overlaps between fragments.

Definition 7 A fragment $f \in \mathcal{F}$ is a singleton if it has no overlaps with other fragments in \mathcal{F} , otherwise it is a non-singleton.

A non-singleton fragment $f \in \mathcal{F}$ is a subfragment if there exists a fragment $g \in \mathcal{F}$ such that f is completely contained in g . For example, g in Figure 5.4 is a subfragment.

A non-singleton fragment $f \in \mathcal{F}$ is a maximal fragment if it is not a subfragment in \mathcal{F} .

Fragments are classified as follows:



5.2.2 Assemble consistent overlapping MFs into subcontigs

Let \mathcal{MF} be the set of all maximal fragments of \mathcal{F} .

For $f \in \mathcal{MF}$, define

$$L(f) = \{g \in \mathcal{MF} : g \text{ overlaps with the left end of } f\},$$

$$R(f) = \{g \in \mathcal{MF} : g \text{ overlaps with the right end of } f\}.$$

Fragments in $L(f)$ ($R(f)$ resp.) are called left (right resp.) neighbors of f (See Figure 5.5).

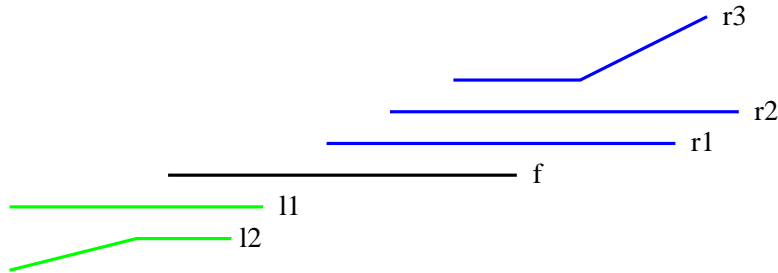


Figure 5.5: Neighbors of a fragment (orientation not shown). The fragments in $L(f) = \{l_1, l_2\}$ ($R(f) = \{r_1, r_2, r_3\}$ resp.) are ordered by their end point on f . Thus l_1 is the rightmost left neighbor of f and r_1 is the leftmost right neighbor of f .

Definition 8 Let f, g be two maximal fragments and f overlaps with g such that $f \in R(g)$ (and $g \in L(f)$). We say f and g overlap consistently if

$$\left\{ \begin{array}{l} R(g) \subseteq R(f) \cup \{f\} \\ L(f) \subseteq L(g) \cup \{g\} \end{array} \right. \quad (5.1)$$

Condition (5.1) is called the *consistency condition*. See Figure 5.6 for an example of a consistent overlap versus inconsistent overlaps.

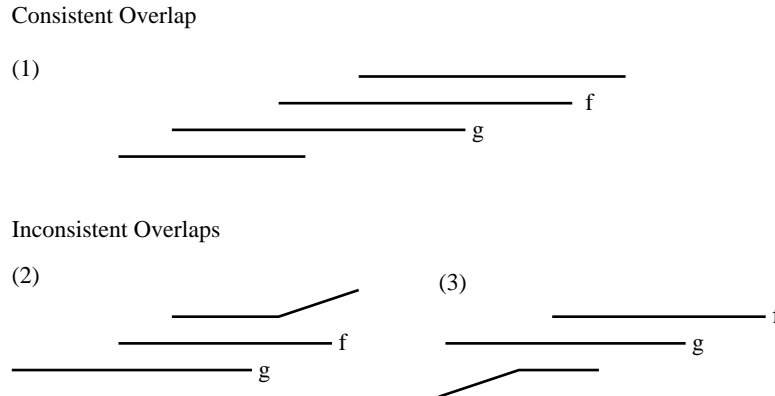


Figure 5.6: Consistent vs. inconsistent overlap (orientation not shown). (1) The overlap of $\{f, g\}$ is consistent because the consistency condition is satisfied, i.e., $R(g) \subseteq R(f) \cup \{f\}$ and $L(f) \subseteq L(g) \cup \{g\}$. (2) The overlap of $\{f, g\}$ is not consistent because $R(g) \not\subseteq R(f) \cup \{f\}$. (3) The overlap of $\{f, g\}$ is not consistent because $L(f) \not\subseteq L(g) \cup \{g\}$.

Suppose that $g \in L(f)$ (and $f \in R(g)$), then for the overlap of f and g to be consistent, it is necessary that f be the leftmost right neighbor of g and g be the rightmost left neighbor of f . The leftmost right neighbor and the rightmost left neighbor of f are called *consecutive* neighbors of f . Therefore, to identify the overlaps that are consistent with a fragment, it is sufficient only to check the consistency condition for the fragment with its consecutive neighbors.

For efficient implementation, all fragments in $L(f)$ ($R(f)$ resp.) are sorted according to their end points in f , for each maximal fragment f . The maximal fragments are then incrementally assembled into subcontigs by checking if the consistency condition is satisfied for the consecutive fragments. The pseudocode is shown in Table 5.2.2. See Figure 5.7 for an example. Note that consecutive fragments which do not satisfy the consistency condition are not assembled. These inconsistent overlaps are resolved later in Section 5.2.6.

The time complexity of this step is $O(\log \Delta \cdot E_M)$ (sorting) + $O(\Delta \cdot E_M)$ (assembling) = $O(\Delta \cdot E_M)$ time, where $N_M(f) = \{g \in \mathcal{MF} : \{f, g\} \in \mathcal{O}\}$ and $\Delta = \max_{f \in \mathcal{MF}} |N_M(f)|$, and $E_M = \sum_{f \in \mathcal{MF}} |N_M(f)|$, $f \in \mathcal{MF}$.

```

1. for all  $f \in \mathcal{MF}$ 
2.   unvisit( $f$ );
3. while there exists an unvisited MF do
4.   let current-fragment an unvisited MF
      * left neighbor *
5.   IsLeft=1;
6.   if (IsLeft) then
7.     let nbr be the 1st left neighbor of the current fragment;
8.   else
9.     let nbr be the 1st right neighbor of the current fragment;
10.  while current-fragment and nbr are consistently overlapping do
11.    assemble current-fragment and nbr;
12.    IsLeft = IsLeft *  $\delta(\text{current-fragment}, \text{nbr})$ ;
13.    current-fragment=nbr;
14.    if (IsLeft) then
15.      let nbr be the 1st left neighbor of the current-fragment;
16.    else
17.      let nbr be the 1st right neighbor of the current-fragment;
      * similarly for right neighbor *

```

Table 5.2: The pseudocode of assembling consistent overlapping MFs into subcontigs.

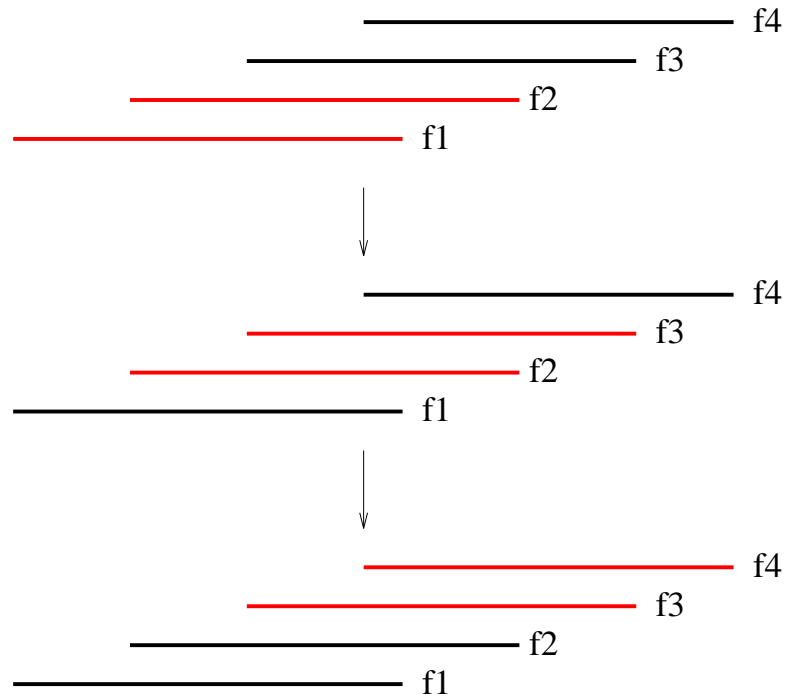


Figure 5.7: Assembling consistent overlapping maximal fragments. First, f_1 and f_2 are assembled; then f_2 and f_3 , f_3 and f_4 .

5.2.3 Merge good subfragments into subcontigs

Before we describe how subfragments are assembled, we introduce the definition for good subfragments (GoodSub), whose overlap is analogous to the consistent overlap of maximal fragments.

Definition 9 A subfragment b is a GoodSub if for each fragment p which contains b , $N(b) \sqsubseteq N(p)$, otherwise b is a BadSub.

See Figure 5.8 for an example. Any overlaps of a fragment and a GoodSub are also called *consistent* overlaps.



Figure 5.8: G is a GoodSub, while B is a BadSub.

We merge GoodSubs into the corresponding subcontigs. That is, for each fragment in a subcontig, if it has a GoodSub, we put the GoodSub into the subcontig. See Figure 5.9 for an example.

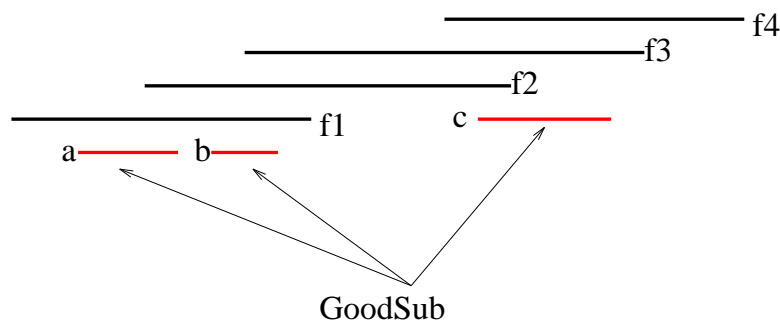


Figure 5.9: Merge GoodSubs into Subcontigs. In this subcontig of $\{f_1, f_2, f_3, f_4\}$, the GoodSubs a, b of f_1 , c of f_3 are put back into the subcontig.

5.2.4 Detect and resolve conflict chromosome assignments

In this section, we first introduce the scheme adopted to assign chromosome for the public working draft of the human genome. Then we describe how conflicting chromosome assignments are detected and resolved in this data set.

For the public working draft of the human genome, a BAC is assigned to a chromosome based on:

- STS: presence of at least two STSs that have been mapped to the same chromosome;
- GenBank: annotation on the submitted GenBank record;

Otherwise, the chromosome of the BAC is *Unknown*.

Notice that an unknown chromosome BAC is allowed to overlap with any BACs associated with any (known or unknown) chromosomes. Thus, we might have a subcontig with two or more known chromosome fragments connected by some unknown chromosome fragments. Also, an unknown chromosome BAC might have fragments in different chromosome subcontigs. In either case, the unknown chromosome BAC is

called *conflicting chromosome BAC*. The conflicting chromosome BACs might be due to noise (repeat-induced or chimeric BAC) or chromosome misassignments.

If the chromosome assignments are of high confidence, then clearly we have repeat-induced overlaps or chimeras. Chromosome assignments based on STS natures are considered more reliable than those based on annotations in the GenBank records.

Identify conflicting chromosome BACs

Notice that if an unknown chromosome BAC is not conflicting, then it overlaps with at most one known chromosome BAC. If there is one known chromosome, we assign the chromosome to the BAC. However, if an unknown chromosome BAC is conflicting, then either it overlaps with two different chromosome BACs or it overlaps with another conflicting chromosome BAC. To identify conflicting chromosome BACs, we use a greedy approach to assign unknown chromosome BACs until a conflict occurs. Once a conflict is detected, all the original unknown chromosome neighbors of the conflicting BAC become conflicting also.

Resolution of conflicting chromosome assignments

Once the set of all conflicting BACs has been identified, we first identify the potential chromosome misassignments by determining whether unassigning GenBank assignments (i.e. become unknown) will eliminate the conflicts.

1. Assign unknown as much as possible and identify the conflicts;
2. Unassign the GenBank assignment neighbors which overlap with the conflicting chromosome BACs;
3. repeat the above.

Note that we unassign the GenBank assignments and then reassign them, this might result in changing the chromosome assignments. Check if there are any known agreeable overlaps after the change, if there are, make the change, otherwise leave the chromosome as it is. After changing some chromosomes and the corresponding set of overlaps,

we assume the remaining conflict chromosome problems are due to noise, repeats or chimeras.

The problem of resolving the conflicts due to noise is complicated by the fact that there are several possible problematic BACs. Without any other further evidence, we solve this problem progressively. First, resolve the more confident conflicting BACs, e.g. the BAC overlaps with more than one known chromosome BACs directly, by evaluating the length of overlaps to decide if they are due to repeats or due to chimeras. If the length of overlaps is less than 25Kb, we classify it as a repeat otherwise a chimera. For the repeats, we remove the overlaps; for the chimeras, we remove the BACs. After these confident conflicting BACs are resolved, some conflicts might become non-conflicts and we can then assign the chromosome. For the remaining conflicting BACs, that is, each of them overlaps with one known chromosome BAC and another conflicting BAC, we evaluate the length of overlaps with the known chromosome to decide if they are due to repeats or due to chimeras.

5.2.5 Construct a BAC graph from subcontigs

For $f \in \mathcal{F}$, let $\mathcal{B}(f)$ denote the BAC of f . Denote the set of all the overlaps in the subcontigs by \mathcal{G} . That is, \mathcal{G} consists of the transitive closure of the consistent overlaps. To resolve the inconsistent overlaps, we first construct a BAC graph $G = (V, E)$ according to the overlaps in \mathcal{G} , where $V = \{B_1, \dots, B_m\}$ and $\{B_i, B_j\} \in E$ if there is an overlap of a fragment of B_i and a fragment of B_j in a subcontig. That is, two BACs B_i, B_j overlap if there is an overlap between their fragment pair. See Figure 5.10 for an example.

5.2.6 Resolve the inconsistent overlaps

The inconsistent overlaps are resolved according to the connectivity of the BAC graph. See Figure 5.11 for an example. Let f, g be consecutive fragments (e.g., f is the leftmost right neighbor of g). Suppose that the overlap of f and g fails to satisfy the consistency condition (5.1). Then if $\{\mathcal{B}(f), \mathcal{B}(g)\} \in E$, i.e., there is at least one fragment pair of $\mathcal{B}(f)$ and $\mathcal{B}(g)$ in \mathcal{G} , make the overlap of f and g satisfy (5.1) by identifying and

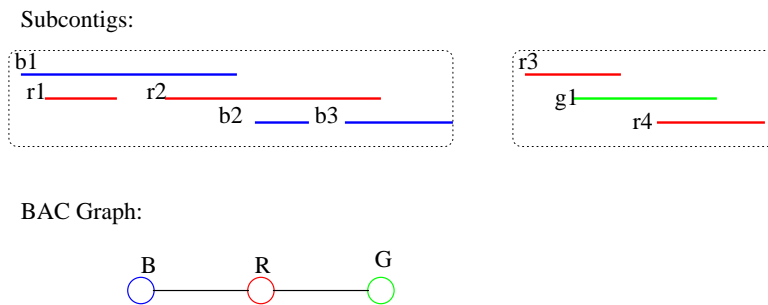


Figure 5.10: A BAC graph constructed from subcontigs. Fragments r_1, \dots, r_4 belong to BAC R; b_1, b_2, b_3 are fragments of BAC B and g is a fragment of BAC G. There is an edge of BAC R and BAC B because of several overlaps of fragment r_i and b_j . Similarly there is an edge of BAC R and BAC G because of the overlaps of fragments g and r_3 or r_4 in the subcontig.

eliminating the repeat-induced overlaps (FPs). After the FPs are eliminated, some overlaps might become consistent. Update the assemblies of subcontigs and the BAC graph; repeat the above step until no more inconsistencies can be resolved. For the remaining inconsistencies which do not have connectivities, we resolve them according to the length of overlaps.

Remark: some inconsistencies might suggest possible FNs. For example, see Figure 5.12, suppose g is the leftmost right neighbor of f , and f is the rightmost left neighbor of h . However, g and h do not overlap. Thus, the overlaps of f and g , f and h are inconsistent. But $\{\mathcal{B}(f), \mathcal{B}(g)\} \in E$ and $\{\mathcal{B}(f), \mathcal{B}(h)\} \in E$, this might suggest that there is an FN between g and h . In this case, our current algorithm breaks ties by choosing the longer overlap one. A future work might include exporting the suspicious FN pairs and further examining their alignments.

5.2.7 Resolve non-interval components

By definition, if our subcontig assemblies are good, the BAC graph thus constructed must be interval. However, since the consistent overlaps are only a necessary condition for the true overlaps, there might be some consistent repeat-induced overlaps in the subcontigs. In addition, there might be some chimeric BACs. That is, the subcontig assemblies might contain noise and thus the BAC graph might not be interval. Before

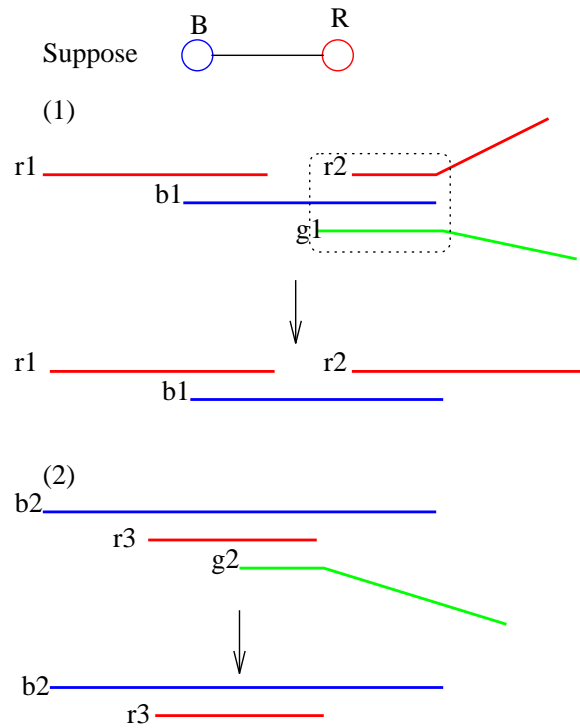


Figure 5.11: Resolving inconsistent overlaps according to the connectivity of the BAC graph. The same BAC fragments are shown in the same color. Fragments r_i (b_i , g_i resp.) are fragments of BAC R (B, G resp.). Assume that BAC R is adjacent to BAC B. (1) There are inconsistent overlaps between fragment $\{b_1, r_2\}$ and $\{b_1, g_1\}$. Since BAC R is adjacent to BAC B, the overlap of $\{b_1, r_2\}$ is chosen and the overlap $\{b_1, g_1\}$ is discarded as a repeat-induced overlap. (2) Fragment r_3 is a BadSub because fragment g_2 overlaps with r_3 but not b_2 which contains r_3 . Since BAC R is adjacent to BAC B, r_3 is made to become a GoodSub of b_2 . That is, the overlap of r_3 and g_2 is discarded as a repeat-induced overlap.

we can obtain an interval realization of the BACs to orient and order subcontigs, we need to resolve these non-interval components.

Recall that there are five different linear time interval graph recognition algorithms (see Section 3.3), which check whether the input graph is interval, and if it is, output an I-ordering of its vertices. For our purposes, which include obtaining an I-ordering if the graph is interval and otherwise identifying the forbidden subgraph of the non-interval graph, we employ the 5-SWEEP LBFS algorithm. Since the algorithm is based on LBFS, if the input graph is not interval, the ordering so produced by the algorithm fails only when it reaches a forbidden subgraph. This enables us to identify the forbidden

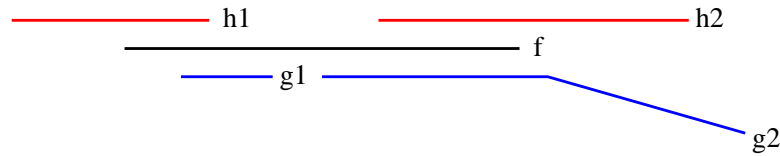


Figure 5.12: Suspicious FNs. The overlaps of f and g_2 , f and h_2 are inconsistent because g_2 and h_2 do not overlap. However, $\mathcal{B}(f)$ overlaps with both BACs $\mathcal{B}(g_2)$ and $\mathcal{B}(h_2)$. This would suggest an FN of g_2 and h_2 .

subgraph.

Before we describe how the non-interval components are resolved, we sketch the 5-SWEEP LBFS algorithm.

The 5-SWEEP LBFS Algorithm

This algorithm consists of two passes: first, an ordering of the vertices is produced; second, determine whether the graph is interval by checking whether the ordering is an I-ordering.

5-SWEEP LBFS Algorithm

1. Perform 5 sweeps of LBFS

Input: a graph $G = (V, E)$.

Output: an ordering ϕ of V .

The ordering was produced by 5 sweeps of LBFS, with the later LBFS using the ordering of the previous LBFS for tie-breaking.

2. Determine whether G is interval by checking whether ϕ is an I-ordering of V , that is, whether all vertices of V satisfy the right-nbr-consecutive (RNC) condition w.r.t. ϕ .

To implement this efficiently, we construct for each vertex a right neighbor adjacency list and arrange the elements in increasing order. This can be easily implemented in linear time by reversing the order of vertices and redistributing their neighbors. Thus, for each $v \in V$, $RN(v) = \{u_1, \dots, u_k\}$, where

$\phi(v) < \phi(u_1) < \dots < \phi(u_k)$, and check that the elements are consecutive, i.e., that for each $v \in V$, $\phi(u_{i+1}) = \phi(u_i) + 1$ with $u_0 = v$.

Identifying a forbidden subgraph

Suppose G is not interval. Let fv be the smallest vertex in the ordering that fails to satisfy the right-nbr-consecutive (RNC) condition, i.e. $\exists i, u_i, u_{i+1} \in RN(fv)$ such that $\phi(u_{i+1}) > \phi(u_i) + 1$. Denote the smallest such u_{i+1} by b_3 , and let $b_1 = \phi^{-1}(\phi(u_i) + 1) \notin RN(fv)$. Since $\phi(b_1) < \phi(b_3)$, by the LBFS property (Theorem 6), there exists an a with $\phi(a) < \phi(fv)$ such that $\{a, b_1\} \in E$. Since $\phi(a) < \phi(fv)$ and fv is the smallest failing vertex, a satisfies the right-nbr-consecutive (RNC) condition. Thus we have $\{a, fv\} \in E$ because $\phi(a) < \phi(fv) < \phi(b_1)$. Let bp be the largest such vertex. Thus, for each non-interval component, we have identified these four vertices $\{b_1, bp, fv, b_3\}$ as shown in Figure 5.13. Note that $\{b_1, fv\} \notin E$ and $\{bp, b_3\} \notin E$. We call the path through of b_1, bp, fv , and b_3 a *bow*. Figure 5.14 illustrates the bow in each type of forbidden subgraph.

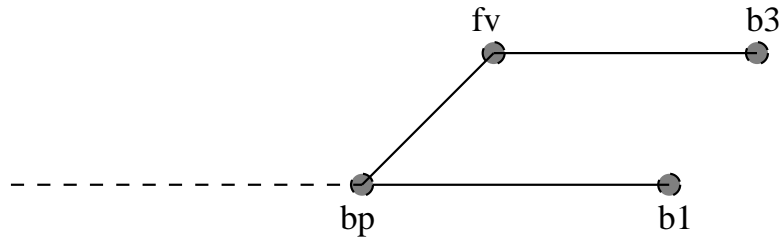


Figure 5.13: The “bow” of $\{b_1, bp, fv, b_3\}$.

Based on the connectivity around the bow, we identify a corresponding forbidden subgraph which contains the bow as follows:¹

1. $\{b_1, b_3\} \in E$: The forbidden subgraph is of Type I:

¹There might be more than one forbidden subgraph which contain the bow, as is often the case for the vertex corresponding to a chimeric BAC.

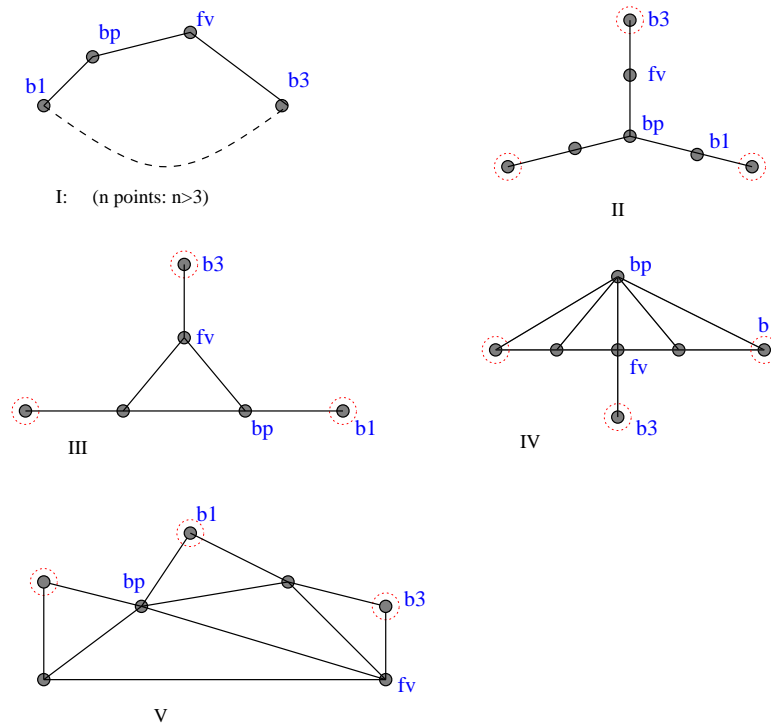


Figure 5.14: The “bow” of $\{b_1, bp, fv, b_3\}$ in each type of forbidden subgraph.

$fv \text{--} b_3$

| |

$bp \text{--} b_1$

2. $\{b_1, b_3\} \notin E$: There are several possibilities:

(a) $bp \neq 1$ (i.e. bp has a left neighbor): In this case, one of the following holds.

i. fv and bp have no common left neighbors. Let b_0 be such a largest left neighbor of bp . The forbidden subgraph is of Type II:

b_3

|

fv

|

$e_0 \text{--} b_0 \text{--} bp \text{--} b_1 \text{--} e_1$

ii. Let b_0 be such a smallest common left neighbor of fv and bp . Then the forbidden subgraph may be of Type III, Type IV, or Type V:

$$\begin{array}{c}
 b_3 \\
 / \\
 fv \\
 / \mid \\
 b_0 \text{--} b_p \text{--} b_1
 \end{array}$$

A. Type III ($\exists e_0 \in Adj[b_0]$ such that $\{e_0, b_p\} \notin E$):

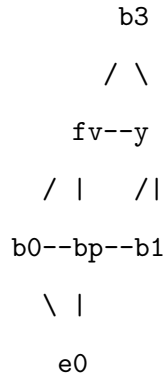
Either $\{b_0, b_1\} \notin E$:

$$\begin{array}{c}
 b_3 \\
 / \\
 fv \\
 / \mid \\
 e_0 \text{--} b_0 \text{--} b_p \text{--} b_1 \\
 \text{or } \{b_0, b_1\} \in E: \\
 b_3 \\
 / \\
 fv \\
 / \mid \\
 e_0 \text{--} b_0 \text{--} b_p \\
 \backslash \mid \\
 b_1 \text{--} e_1
 \end{array}$$

B. Type IV ($fv < y < b_1$ and $\{y, b_3\} \notin E$):

$$\begin{array}{c}
 / \\
 fv \text{--} y \\
 / \mid / \mid \\
 b_0 \text{--} b_p \text{--} b_1 \\
 \backslash \mid \\
 e_0
 \end{array}$$

C. Type V ($fv < y < b_1$ and $\{y, b_3\} \in E$):



- (b) $bp = 1$: The forbidden subgraph is of Type I of a cycle. In this case, we flip the graph and traverse from the other end.

Question: Which vertices/edges of the forbidden subgraph are the cause of the noise?

Theoretically, the subcontig assemblies, which constitute the forbidden subgraph, might not be sufficient to identify the noise. For example, if the forbidden subgraph is a 4-cycle as shown in Figure 5.19, the noise could be an FN of the diagonal or one of the four edges could be FP. To check whether it is an FN, the alignments of the involved BACs need to be further examined. To check whether one of the four overlaps is repeat-induced, one could consider the resulting interval realization and the subcontig assemblies to access the possibility of repeats. For instance, suppose that the edge $\{a, d\}$ in Figure 5.19 is repeat-induced, and that the graph becomes interval if the edge $\{a, d\}$ is removed. Then consider the interval realization of the graph. The interval for vertex d would be a subinterval of the interval for vertex c ; that is, d would have had to overlap c heavily. Thus, for the edge $\{a, d\}$ to be an FP, the length of overlap of $\{a, d\}$ should be short and the length of overlap of $\{c, d\}$ should be long with respect to the length of BAC d (such that $\{a, c\}$ does not overlap). Ideally, to confidently identify the noise, the involved BACs and the corresponding subcontig assemblies of the forbidden subgraph should be exported for further examination (and/or to request additional biological information about them to assist in identifying the noise). Without additional information, we heuristically identify the noise based on the structure of the forbidden subgraph and the subcontig assemblies.

Before we describe the idea of resolving the non-interval components, we analyze each type of the noise and its effect on the graph structure.

Analyzing Noise

There are three types of noise: chimeras, false positives and false negatives.

Chimeras Recall that a chimera arises when two separated sequences from the genome join together to form a BAC that consequently is no longer a contiguous sequence of the genome. We say the chimera consists of two parts, one for each contiguous sequence. Depending on the coverage of sequences (and hence the connectivity of the BAC graph), there are three types of a chimera as illustrated in Figure 5.15. The forbidden subgraph thus caused by each type depends on the connectivity of

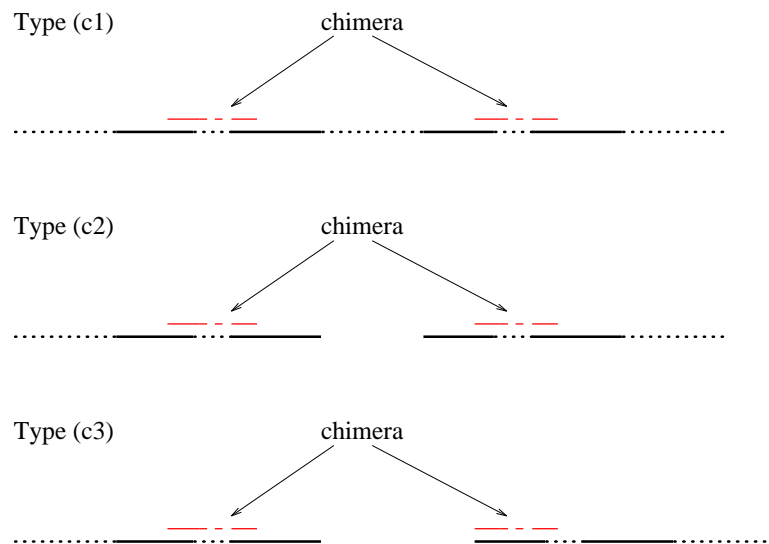


Figure 5.15: Depending on the coverage of the sequences, there are three types of a chimera. Type (c1): The two parts of the chimera are far apart in one single contig. Type (c2): Two parts of the chimera lie in two different contigs. Both of them lies in the middle of the contig. Type (c3): Two parts of the chimera lie in two different contigs. One part of the chimera lies in the middle of the contig; the other part lies in the end of another contig.

the contig. For example, the three possible cases of Type (c2) are illustrated in Figure 5.16.

FPs As in the chimera case, depending on the coverage of the sequences, there are three types of a repeat as illustrated in Figure 5.17. The corresponding forbidden subgraphs caused by the repeat in Type (r2) is illustrated in Figure 5.18.

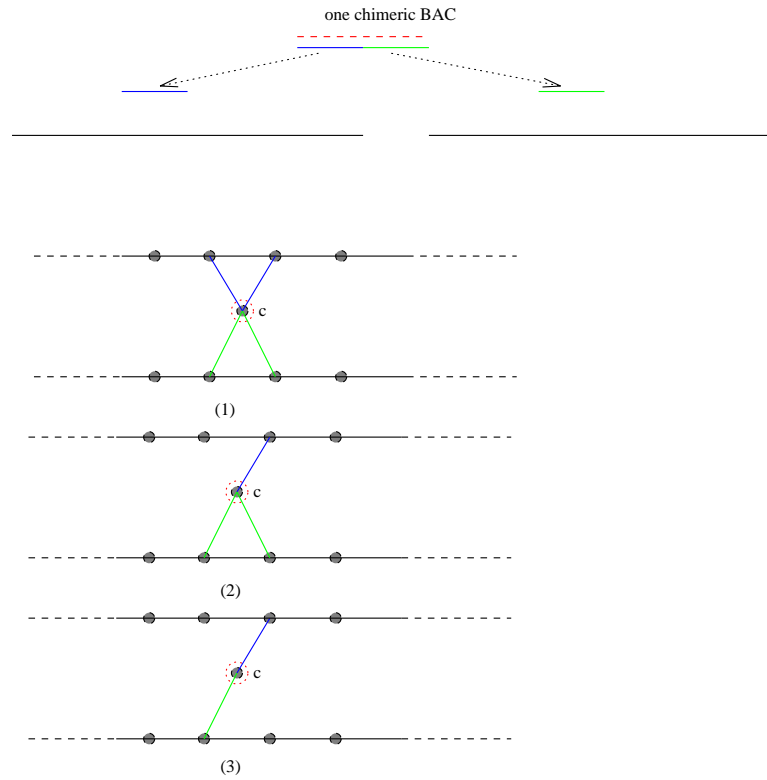


Figure 5.16: Three cases of a chimera of Type (c2) shown in Figure 5.15. The forbidden subgraph is of Type II or Type III. Note that there might be several overlapping forbidden subgraphs caused by the chimera.

FNs We expect FNs to be rare at this stage. The FNs usually are due to the draft quality of the sequences or due to polymorphism. But since each draft BAC has several fragments, two BAC overlaps usually would have several fragment pairs overlap (and therefore missing some overlaps will not affect the BAC edges). Thus FNs are more likely due to short finished BAC sequences (as from the real case). The forbidden subgraph thus caused will be a Type I (4-cycle) graph.

Remark: Forbidden subgraph introduced by repeats and chimeras look similar in the BAC graph level. It is difficult to tell a repeat from a chimera without further examination. Also, even if the noise that causes the forbidden subgraph is due to a repeat, in some cases we can not tell which part of the clone is the repeat.

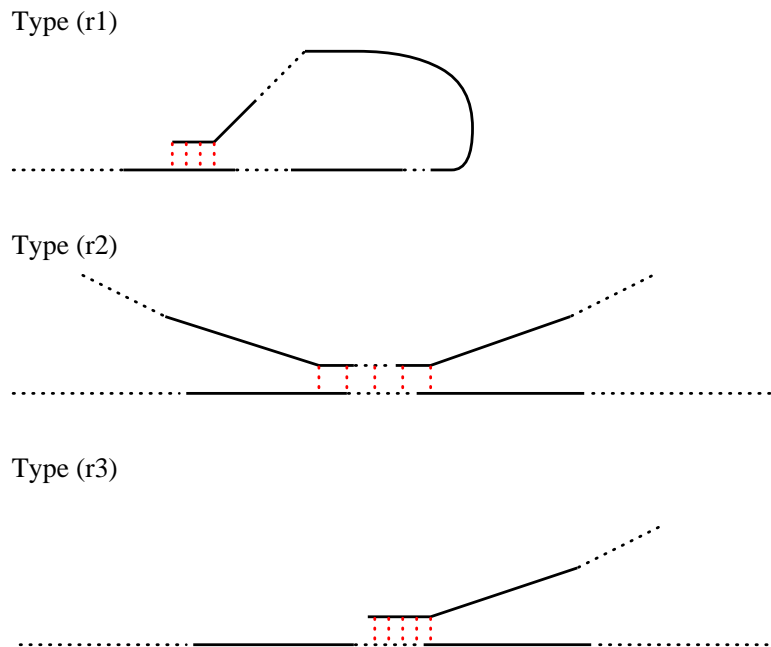


Figure 5.17: Depending on the coverage of the sequences, there are three types of a repeat. Type (r1): the repeat occurs in the same contig which would cause a cycle. Type (r2): the repeat occurs in two different contigs. Both of them lie in the middle of the contig. Type (r3): the repeat occurs in two different contigs. One of them is in the middle of its contig; the other lies in the end of its contig.

Overview of the method for resolving non-interval components

Suppose the non-interval component contains only a chimeric BAC. Then the non-interval graph would become interval if the chimera were removed. Motivated by this idea, we define a vertex $v \in V$ to be *I-critical* if $G|_{V \setminus \{v\}}$ is interval. Notice that if G contains only one forbidden subgraph, then all of the vertices of the forbidden subgraph are I-critical because the removal of any of them would destroy the forbidden graph and thus make the graph interval. Also, note that each of the involved vertices of an FN is also I-critical; and if one of the sequences of the repeat occurs only in one BAC, it would cause only one I-critical vertex.

Given a non-interval component G_i , we identify a forbidden subgraph; and check whether at least one of the vertices of the forbidden subgraph is I-critical. If so, we say G_i is fixable.

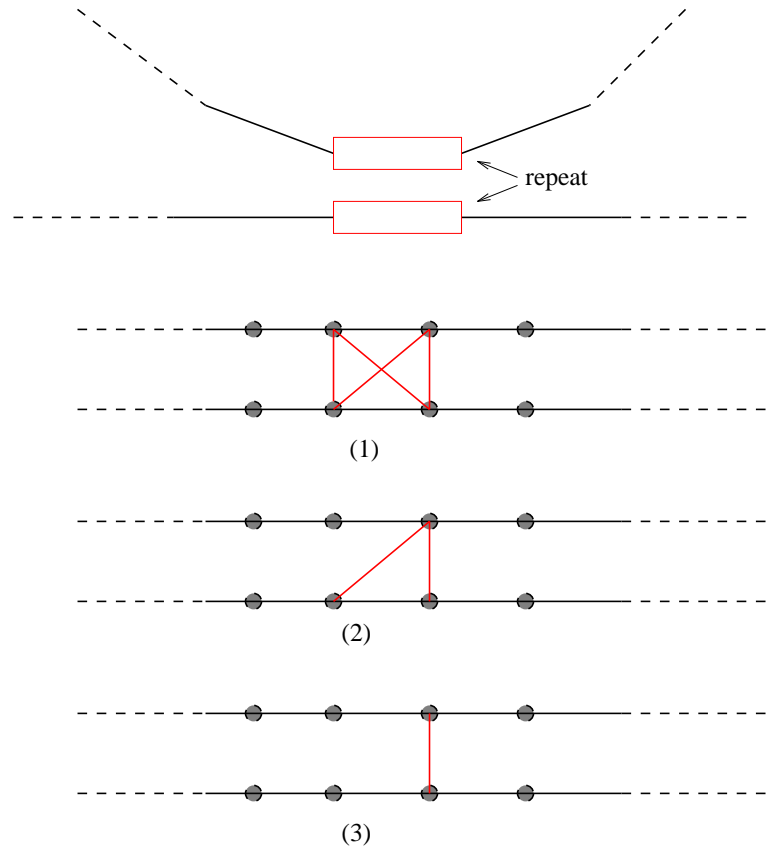


Figure 5.18: Three cases of a repeat of the Type (r2) shown in Figure 5.17 and their corresponding forbidden subgraphs caused.

For the non-fixable component, we employ a divide-and-conquer method by dividing the graph according to some articulation points (APs) so that each subcomponent is fixable. After fixing the subcomponents, the non-fixable component would become fixable as the AP would become I-critical. When there are no APs (and no I-criticals), this is likely because the noise involves more than one BAC. In this case, we choose one of the two breaking points depending on the connectivity. More discussion of articulation points is contained in a section below.

To resolve the fixable component (which can always be resolved by removing an I-critical vertex), a heuristic based on the structure of the forbidden subgraph and the corresponding assemblies is developed. In general, a fixable component G is resolved by one of the following:

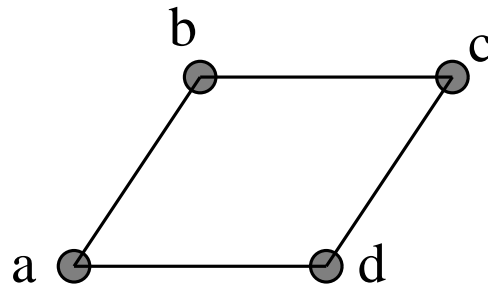


Figure 5.19: A Type I forbidden subgraph. One of the four edges would be FP or one of the diagonals $\{a, c\}$ and $\{b, d\}$ would be FN. For the RV case, all four vertices are equally likely.

- FN: adding an FN edge; or
- FP: removing FP edges due to an identified repeat; or
- RV: removing a vertex which is either a suspicious chimera or contains an unidentified repeat.

Based on analyzing each forbidden subgraph, we heuristically identify the possible FNs, FPs and RVs. Figure 5.19 gives an analysis of Type I forbidden subgraphs. Figure 5.20 shows Type II and Figure 5.21 shows Type III. Note that Type IV and V are similar to Type III (we omit their analysis).

Suspicious Chimeras

For the RV case, we further analyze the relationship of the removed (I-critical) vertex with the resulting interval graph to classify the noise type: more suspicious chimera, less suspicious chimera or unclassified.

We say the removed BAC is a *more suspicious chimera* if it is of the Type (ms1) or Type (ms2) shown in Figure 5.22, or a *less suspicious chimera* if it is of Type (ls) shown in Figure 5.22. Notice that this type might correspond to Type (c3) in Figure 5.15 or Type (r3) in Figure 5.17. Thus the removed BAC could be a chimera or could contain a repeat. The more suspicious chimera is much more unlikely to be due to repeats. See

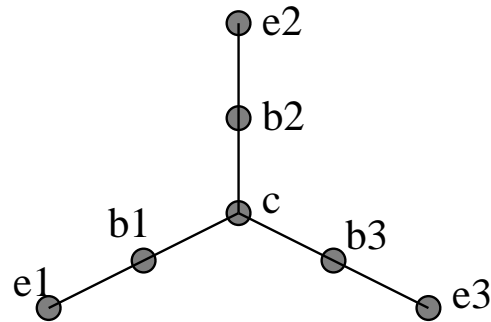


Figure 5.20: A Type II forbidden subgraph. The vertices $\{e_1, e_2, e_3\}$ are the AT-triple, also called outer vertices; $\{b_1, b_2, b_3\}$ are called inner vertices. The edge of $\{b_i, e_i\}$ is called the outer branch; $\{c, b_i\}$ is the inner branch. Observe that if one of the outer branches is FP, say $\{c, b_i\}$, then removing the branch $\{c, b_i\}$ would result in the interval for b_i being a subinterval of the interval for c in the interval realization. Thus it is more likely that the noise occurs in the inner branch. Also, although when all of the vertices of the forbidden subgraph are I-critical, (for the same reason of being a subinterval in the realization,) c is more likely to be the problematic one. Therefore, if we have to remove one I-critical vertex in order to resolve the component (for the RV case), c should be chosen.

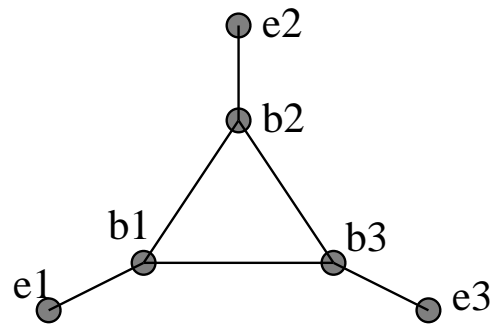


Figure 5.21: A Type III forbidden subgraph. We define inner and outer vertices. The vertices $\{e_1, e_2, e_3\}$ are called outer vertices; $\{b_1, b_2, b_3\}$ are called inner vertices. The edge $\{e_i, b_i\}$ is an outer branch; $\{b_i, b_{i+1}\}$ is an inner branch. Because if the FP is an outer branch, the inner vertex would become subinterval, we assume the noise is more likely caused by the inner branch than by the outer branch. But if we have to remove one I-critical vertex in order to resolve the component and all three b_i s are I-critical, we arbitrarily choose one of them.

Figure 5.23 for the analysis of Type (ms2) for the possibility of containing repeats. To ensure correctness, follow-up by genome centers of these cases are necessary.

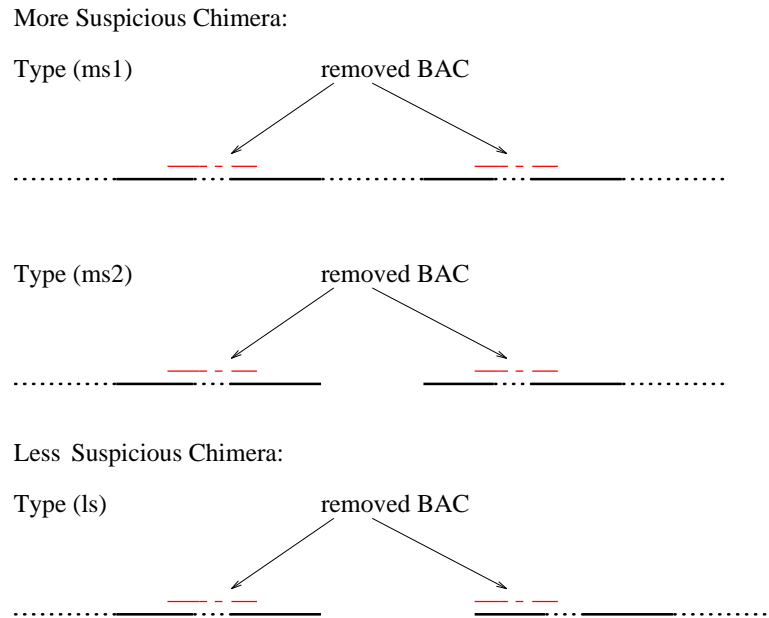


Figure 5.22: Depending on the connectivity of the graph, the removed BAC is classified as the more suspicious chimera and the less suspicious chimera. Type (ms1) and Type (ms2) are the more suspicious chimera, while Type (ls) is the less suspicious chimera.

Choice of Articulation Points

As described above, given a non-fixable graph, we employ a divide-and-conquer method to partition the graph so that each subcomponent is fixable. The natural idea is then to partition the graph according to some *articulation points* (APs). But how should the AP be appropriately chosen so that the graph can be resolved efficiently and correctly, i.e. so that each subcomponent can be resolved independently?

If the AP is not a vertex of a forbidden subgraph, removing the AP will not effect resolution of the subcomponents. Suppose that the AP is a vertex of a forbidden subgraph. Assuming that only one vertex is involved in the noise, then if AP is not the problematic vertex, the removal of the AP will make the problematic vertex become non-problematic (the forbidden subgraph will be destroyed by the removal of the AP)

More Suspicious Chimera:

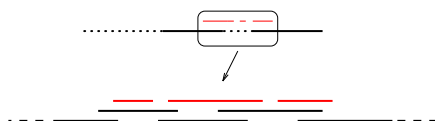
Type (ms2)

removed BAC

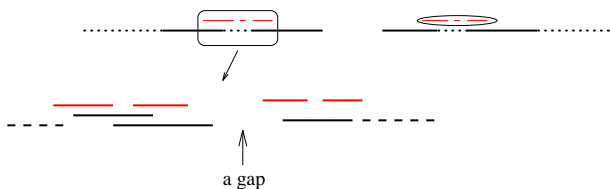


1. There are only two contigs after removing the BAC, i.e. the contig itself is connected.

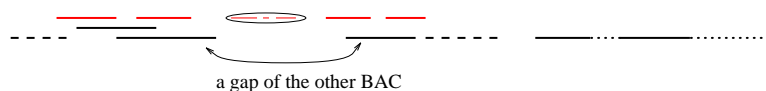
(1.1) Both of the two parts of the BAC lie in the middle of a long subcontig



(1.2) There is a gap of at least one part of the BAC:



So the other part of the BAC might be a repeat, i.e., it would be as follows:



2. One contig is connected by one part of the chimera, i.e., the removal of the part would disconnect the contig:

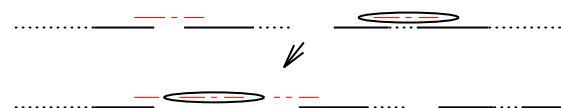


Figure 5.23: Depending on the connectivity, there is a possibility that the more suspicious chimera is not a chimera but rather contains a repeat. For case 1, in which there are two contigs after removing the BAC, it is very likely it is a chimera. Even though there might be a gap in one contig, it is unlikely that it would be a repeat for otherwise it would result in a large gap in the other BAC. However, when one contig is connected by the removed BAC, i.e. the removal of the BAC would result in three or more contigs, it is possibly due to a repeat as shown in case 2.

in the subcomponent. The problematic vertex becomes “visible” only when the AP is put back. Thus, in this case, the removal of AP will not affect the resolution of the subcomponent. However, when there is more than one vertex involved in the noise, the removal of the AP might affect the resolution of the subcomponent. In this case, where there are no I-critical vertices, it is very likely that one of the problematic vertices will be an articulation point (e.g. repeats with more than one BAC as shown in (1) of Figure 5.18). Thus in this case, and in general, we would choose the articulation point closest to the failing vertex.

Also, when there are no APs (i.e. no I-critical vertices and no APs), how should we resolve the non-interval component? We choose the failing vertex (fv) or breaking point (bp) of the detected bow as the partition point. The removal of these vertices might destroy a forbidden subgraph and thus result in the graph being fixable.

The Divide-and-Conquer Procedure

The following is the procedure for resolving the non-interval components.

FIX_NON-INTERVAL(G)

Input: non-interval graph G

Output: interval graph G'

1. If G is non-interval, identify the bow $\{b_1, bp, fv, b_3\}$ and from it identify the forbidden subgraph;
2. If at least one of the vertices of the forbidden subgraph is I-critical, FIXABLE(G);
3. Otherwise, let u be the first articulation point(AP) after the breaking point bp . When there is no such articulation point, if $\{bp, fv\}$ disconnects the graph, $u := fv$, else $u := b_3$.
4. Remove u , and recurse on each component G_i by FIX_NON-INTERVAL(G_i). Now all components are interval, i.e. u will become I-critical, resolve the graph by FIXABLE(G);

$G' = \text{FIXABLE}(G)$

Input: non-interval graph G with at least one I-critical vertex

Output: interval graph G'

1. Identify the bow and a forbidden subgraph containing the bow (note: at least one of the vertices of the forbidden subgraph is I-critical);

(a) $\{b_1, b_3\} \in E$: Type I forbidden subgraph;

- i. If one of $\{fv, b_1\}$'s BAC lengths is less than $10K$, check whether fixable by the FN $\{fv, b_1\}$.
- ii. Else if one of $\{bp, b_3\}$'s BAC lengths is less than $10K$, check whether fixable by the FN $\{bp, b_3\}$.
- iii. Let BL be the minimum overlap length of potential FPs of the 4 edges, $\{\{fv, bp\}, \{fv, b_3\}, \{b_1, bp\}, \{b_1, b_3\}\}$. If $BL < 25K$, remove the FP, otherwise $\text{REMOVERV}(v)$, where v is a vertex of the forbidden subgraph and is I-critical.

(b) $\{b_1, b_3\} \notin E$:

- i. $\{fv, b_0\} \notin E$: Type II of the forbidden subgraph.
 - A. Let OB be the minimum overlap length of potential FPs of the 3 outer branches.
 - B. Let IB be the minimum overlap length of potential FPs of the 3 inner branches.
 - C. If $OB < IB$ and $OB < 8K$, remove the branch.
 - D. Else if $IB < 20K$, remove the branch.
 - E. Otherwise $\text{REMOVERV}(v)$, where v is a vertex of the forbidden subgraph and is I-critical.
- ii. $\{fv, b_0\} \in E$: Type III of the forbidden subgraph;
 - A. Let BL be the minimum overlap length of potential FPs around the 3 inner points.
 - B. If $BL < 20K$, remove the branch;

- C. Else REMOVE $RV(v)$, where v is a vertex of the forbidden subgraph and is I-critical.

REMOVING $RV(v)$

Input: non-interval graph G , where v is I-critical

Output: interval graph G'

1. Let d be the number of interval components of G if we remove v .
2. For each component G_i , $i = 1 \dots d$, let OL_i be the length of overlap of v with G_i .
3. If $OL_i < 25K$, remove all the overlaps.
4. Classify the type of v : if v is in the middle of 2 contigs, it is a more suspicious chimera; if v is in the middle of one contig and at the end of another contig, it is a less suspicious chimera; otherwise, unclassified noise (would be FNs).

5.2.8 Obtain the ordering of BACs in the interval representation

After all non-interval components are resolved, we apply the 5-SWEEP LBFS algorithm to the interval graph to obtain an interval representation and an I-ordering ϕ of BACs. Also, from the interval representation, we enumerate BACs by blocks, which will be used to assign coordinates for subcontigs and also reorder the BACs of end-blocks.

Recall that blocks are defined for proper interval graphs. To assign blocks for a general (not necessarily proper) interval graphs, based on the I-ordering, we identify the subBACs whose interval in the realization is a subinterval of another interval in the realization. Then restricting to the graph without those subBACs (i.e. the proper interval graph) blocks are enumerated. This can be easily implemented by the definition of block, that is, we increment the block index once the BAC's neighbors are different from the previous. Then for each subBAC, we use the block of the BAC which contains it. Thus, for each BAC B , we have its rank $\phi(B)$ and its block $\pi(B)$ in the interval representation.

5.2.9 Orient Subcontigs

As we mentioned before, each fragment can be used in either direct or reverse orientation. Each subcontig can have either orientation. In this section, we will use the ordering and the block enumeration of BACs to orient subcontigs. Intuitively, if a subcontig is “long” enough, we orient the subcontig such that the ranks of BACs in the subcontig are in “increasing” order, see Figure 5.24.



Figure 5.24: Two possible orientations. Orient each subcontig such that it reflects the correct ordering. In this case, it is the left one.

For each subcontig S_k , let

$$\begin{cases} a_k = \mathcal{B}(u) & \text{with } u \in S_k, \phi(\mathcal{B}(u)) \leq \phi(\mathcal{B}(w)), \forall w \in S_k \\ e_k = \mathcal{B}(u) & \text{with } u \in S_k, \phi(\mathcal{B}(u)) \geq \phi(\mathcal{B}(w)), \forall w \in S_k \end{cases}$$

i.e., a_k (e_k resp.) is the smallest (largest resp.) BAC of S_k . Call a_k and e_k the *end* BACs of subcontig S_k . Define $\Delta_{blk}(S_k) = \pi(e_k) - \pi(a_k)$. If $\Delta_{blk}(S_k) > 0$ orient the subcontig such that the blocks are increasing, otherwise orient the subcontig according to the rank of fragments in the subcontig. Specifically, let $F_{\pi(a_k)}$ ($T_{\pi(a_k)}$ resp.) be the starting (end resp.) position of a BAC of block $\pi(a_k)$ in S_k . Then we orient S_k such that $F_{\pi(a_k)} < F_{\pi(e_k)}$ (flip S_k if $F_{\pi(a_k)} > F_{\pi(e_k)}$ and $T_{\pi(a_k)} > T_{\pi(e_k)}$); If $F_{\pi(a_k)} > F_{\pi(e_k)}$ and $T_{\pi(a_k)} < T_{\pi(e_k)}$, orient according to F_{a_k} instead of $F_{\pi(a_k)}$. That is, if $\Delta_{blk}(S_k) = 0$ orient the subcontig such that the ranks are increasing, i.e. according to F_{a_k} , F_{e_k} etc.

We use $\Delta_{blk}(S_k)$ to measure the correctness of the orientation of S_k . If $a_k e_k \notin E$, we are sure that the orientation is correct, or if $\Delta_{blk}(S_k) > 1$, we are quite confident of the orientation.

We call the orientation of S_k *Sure* if $a_k e_k \notin E$ or $\Delta_{blk}(S_k) > 1$. For the unsure orientation, we use additional information (including end fragments, plasmid-pair, EST, mRNA) to orient them in Section 5.2.14.

Remark: we orient subcontigs before we order them because we need the orientation of subcontigs in order to compute warps (which is then used to adjust the ordering of end-blocks).

5.2.10 Assign coordinates to subcontigs and order subcontigs by sorting lexicographically

In this step, we use the ordering and the block enumeration of BACs to assign coordinates to subcontigs so that they can be ordered by sorting according to these coordinates lexicographically.

Observe that the corresponding end BACs of the adjacent subcontigs must be either the same or overlapping (see Figure 5.25). The necessary condition, which we call the

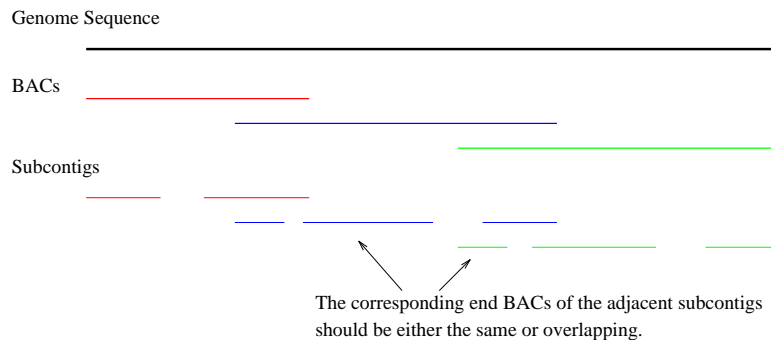


Figure 5.25: The condition of adjacent subcontigs. The corresponding end BACs of the adjacent subcontigs must be either the same or overlapping.

adjacency condition, for two adjacent subcontigs S_k and S_{k+1} , is

$$e_k = a_{k+1} \text{ or } e_k a_{k+1} \in E.$$

The natural way to assign coordinates for each subcontig S_k is then $[\phi(a_k), \phi(e_k)]$.

However, recall that there are *gaps* in BACs. A subcontig does not necessarily include fragments from each BAC in the corresponding region of the realization.

If the missing fragment of the subcontig does not correspond to a fragment of one of the end BACs, then the coordinate of subcontig is still correct. However, if the missing fragment corresponds to one of the end BACs, then the coordinate is incorrect, and

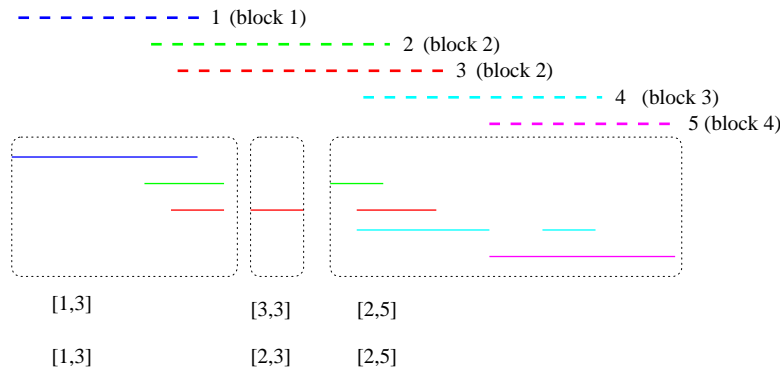


Figure 5.26: An end BAC is missing, resulting an incorrect coordinate for the subcontig.

hence the order of subcontigs is incorrect. For example, see Figure 5.26. Observe that if we sort subcontigs by $[\phi(a_k), \phi(e_k)]$, only missing the smallest end BAC will affect the ordering. Thus one way to amend the problem is to scan the sorted subcontigs from left to right, and move the misplaced one, which is detected by the adjacency condition, backward. Similarly, this can also be done in reverse by sorting subcontigs according to $[\phi(e_k), \phi(a_k)]$. We say the subcontigs are *ties* if the adjacency condition is always satisfied for any order of them. For example, suppose $\{2, 3\}, \{3, 4\} \in E$, then $[2, 4]$ and $[3, 3]$ are ties, for the adjacency conditions are satisfied for either of the orderings $[1, 3], [2, 4], [3, 3], [4, 4]$ and $[1, 3], [3, 3], [2, 4], [4, 4]$. But which one is the true one? See Figure 5.27 for the corresponding realization. If we assume that $[3, 3]$ misses the smallest end BAC, i.e. it should have been $[2, 3]$, then the order is $[1, 3], [3, 3], [2, 4], [4, 4]$ as shown in (1) of Figure 5.27. But if we assume that $[3, 3]$ misses the largest end BAC, i.e. it should have been $[3, 4]$, then the order is taken as $[1, 3], [2, 4], [3, 3], [4, 4]$ as shown in (2) of Figure 5.27. Without extra information, theoretically it is impossible to know which one is the case. Notice that if we use the second order, i.e., $[1, 3], [2, 4], [3, 3], [4, 4]$ as in (2) of Figure 5.27, the assembled length of BAC 3 is larger (while the assembled length of BAC 2 is smaller).

Subject to minimize the warping of BACs, which is defined by the ratio of the assembled BAC length and the estimated BAC length, the assembled length will be minimized if the missing one is of the same block BAC. For this reason, we use $[\pi(e_k), \phi(a_k), \phi(e_k)]$

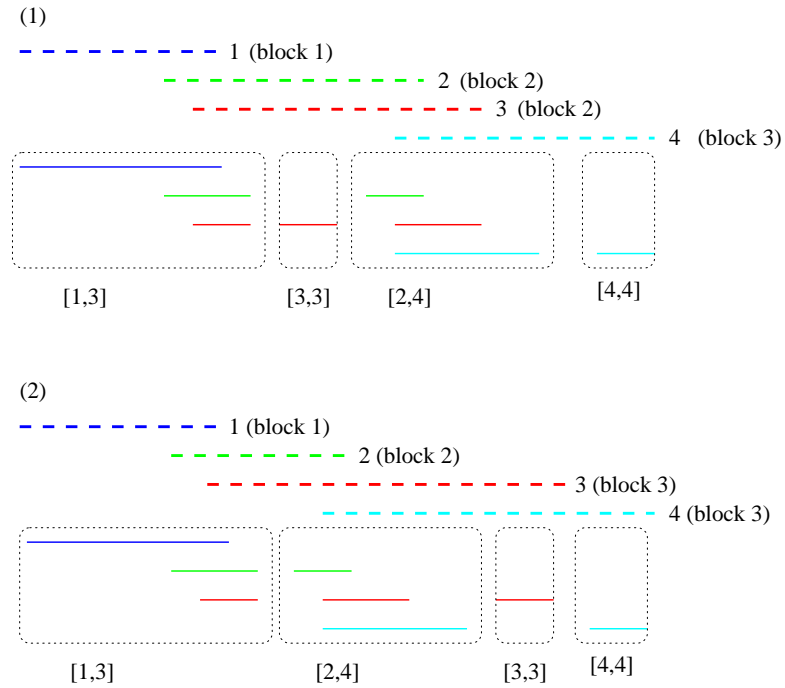


Figure 5.27: The subcontigs ordering and the corresponding interval realization.

as the coordinates of subcontig S_k . See Figure 5.28 for another example. In essence, we are assuming that the block of the missing end BACs are the same as the present one. As a result, if the block of the missing one is greater, then the ordering might not be correct, as shown in Figure 5.29.

For example, the following are ordered subcontigs:

$$S_1 = [B - t, 1, 4], S_2 = [B, 10, 10], S_3 = [B + 1, 4, 11], S_4 = [B + s, 11, 15]$$

where $\pi(10) = B$, $\pi(11) = B + 1$, as shown in the Figure 5.29. To correct this problem, we scan all the subcontigs from right to left and check if they satisfy the adjacency condition. If not, i.e. $e_k a_{k+1} \notin E$, but $e_{k+1} a_k \in E$, switch k and $k + 1$. In the above example, if we switch S_2 and S_3 , the adjacency condition is satisfied. This can be explained by a gap in BAC 11 and $\pi(10) < \pi(11)$. The subcontig $S_2 = [B, 10, 10]$ should really be $[B + 1, 10, 11]$.

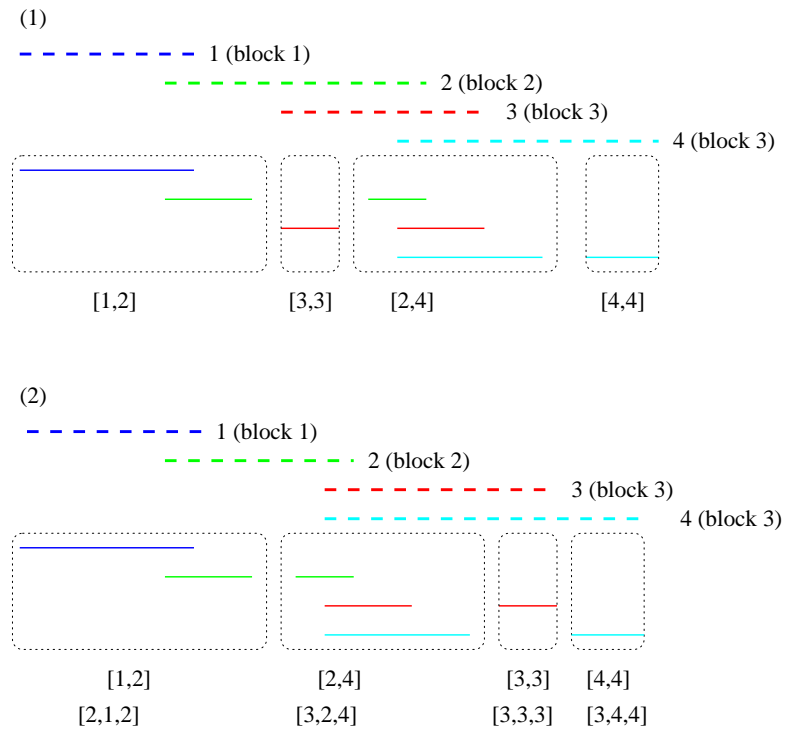


Figure 5.28: The subcontigs ordering and the corresponding interval realization. The second order will be used if the coordinates $[\pi(e_k), \phi(a_k), \phi(e_k)]$ is used for a subcontig.

Adjust the ordering of BACs in the end-blocks

Recall that by Theorem 5, the ordering is unique up to the permutation of vertices within each block. The ordering within each block is not known. Which permutation is the true one? Again without extra information, it is impossible to tell. Here we resort to the warping of BACs as a measurement. Note that for the inner blocks, it will not matter much as it is more likely determined by the assemblies automatically. But for the two end-blocks, we need to adjust the ordering because only one end overlaps with the adjacent block, see Figure 5.30. We permute the ordering in the end-blocks according to the assemblies with the adjacent block such that the warping is minimized.

5.2.11 Detect FNs

In the perfect, i.e. noise-free, world, we can adjust the ordering such that all adjacency conditions are satisfied. Accordingly, when some subcontigs cannot be ordered such

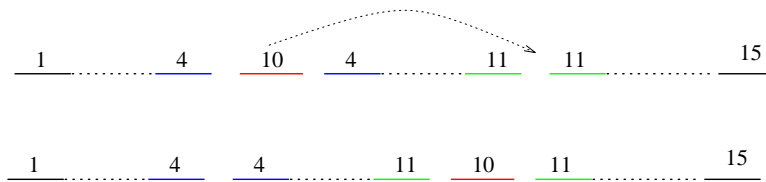


Figure 5.29: The ordered subcontigs are $S_1 = [B - t, 1, 4]$, $S_2 = [B, 10, 10]$, $S_3 = [B + 1, 4, 11]$, $S_4 = [B + s, 11, 15]$, where $\pi(10) = B$, $\pi(11) = B + 1$.

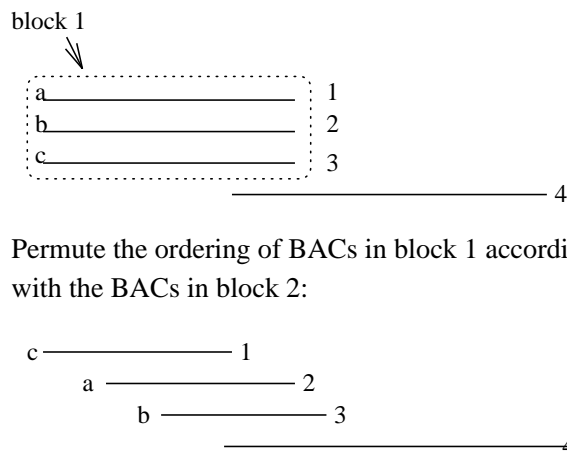


Figure 5.30: For end-blocks, which have only one adjacent block, we permute the BACs within the end block according to the subcontig assemblies such that the warping of the BACs is minimized.

that they satisfy the adjacency condition, it indicates either that the subcontig is misassembled due to the repeat-induced overlaps or that there are FNs (see Figure 5.31).

We expect the repeat-induced overlaps to be rare at this stage because of the clone graph being interval. To further verify the identification of the FNs, we aligned the involved fragments with their overlapping clones. Examining these alignments reveals several possible causes, which include the consequences of repeat-masking, low accuracy of some draft sequences, chimeric fragments or fragment misassemblies and polymorphism.

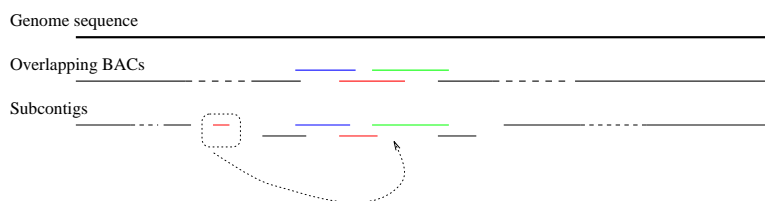
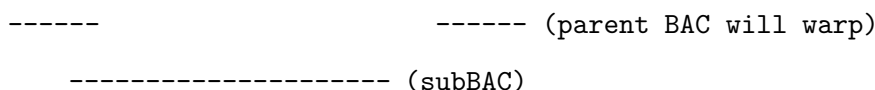


Figure 5.31: FNs detection. No matter how we order the subcontigs, the subcontig in the box will violate the adjacency condition. This is due to a FN as the arrow shows.

5.2.12 Detect FPs

There are possible repeat-induced overlaps which do not destroy the interval property of the graph, these can be the subBAC (or subinterval) in the interval realization. They can be detected because they would cause the parent BAC to be expanded (warping):



Thus, if a BAC A is warped and it has a subBAC S which only overlaps with the BAC partially, we remove the overlaps between A and S .

5.2.13 Adjust the ordering of subcontigs

If there are ties (defined above) in our ordering, we employ some additional information to break these ties. There are two types of additional information including (1) the identification of the end fragments of a BAC and (2) some partial order of some fragments within a BAC (see Section 4.2.1).

Adjust the ordering of subcontigs according to the end fragment information and correct the corresponding orientation

As always, the information (here, which fragment is the end fragment) might not be reliable. We test the reliability by checking whether we can adjust the order according to the information such that the adjacency condition is still satisfied. In other words, the end fragment information are only used to break ties. Also, the orientation is corrected such that the end fragment is the extreme fragment of the BAC in the assembly.

For each BAC B which has end fragment information, i.e., one or two end fragments, first according to their current position in the contig, we decide which one is the left end fragment and which one is the right end fragment. Then we orient the fragments so that they are the extreme fragments of the BAC. To ensure the reliability of the information, these adjustments (order and orientation) are always subject to the adjacency condition, i.e., whether we can adjust the order and orientation according to the information such that the adjacency condition is still satisfied.

Adjust the ordering of subcontigs according to the partial order fragment information and correct the corresponding orientation

This step is almost the same as the one above. For each group of the ordered fragments, we break “ties” according to the given partial order, also orientations are corrected accordingly. We do these steps in two sweeps, one from left to right and then right to left. (Because we only initialize a group when the fragment is the extreme. Note that subcontigs are lexicographically sorted by the increasing order of fragments in each subgroup. These two sweeps in general are sufficient for the partial order group ordering).

5.2.14 Orient subcontigs according to plasmid-read, EST, mRNA data

Note that the relative orientation of fragment pairs generated from plasmid-read, EST and mRNA are noisy. See Section 4.2.3 for how the pairs are generated.

First, observe that each fragment pair should be quite close (on average 6Kb apart for plasmid pairs; and assume that introns are not too long for ESTs/mRNAs), we exclude those fragment pairs whose corresponding BACs are not overlapping or the difference of their corresponding blocks is more than 1 for pairs generated from plasmid reads, or more than 2 for pairs generated from ESTs/mRNAs. Then we progressively orient subcontigs according to the information. That is, we first orient subcontigs with strongly supported information, and then less confident ones and so on.

We classify the confidence of the orientation of subcontigs into 4 levels:

$$\text{level} = \begin{cases} 3 & \text{if } a_k e_k \notin E \\ 2 & \text{if } \Delta_{blk}(S_k) > 1 \text{ or supported by an end fragment} \\ 1 & \text{if } \Delta_{blk}(S_k) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Note that for $level \geq 2$ we are quite confident of the correctness of the orientation. We say that the orientation of a subcontig with $level \geq 2$ is *Sure*. The relative orientation of fragment pairs is mainly used to orient the “unsure” (i.e. $level < 2$) fragments.

For each subcontig S_k , we organize all its relative orientations into a list of agreeing subcontigs (*Support*) and a list of disagreeing subcontigs (*Against*). Let $ns = |Support|$, the number of relative orientations of subcontigs which support its current orientation; $na = |Against|$, the number of relative orientations of subcontigs which are against its current orientation. We change the orientation of unsure subcontigs according to the relative orientations progressively as follows:

for level = 0 to 1 do

 for each subcontig S_k on level with $na > ns$

 if all the orientation of subcontigs in its Against list are Sure,

 flip the subcontig S_k and prompt its level to 2;

 for each subcontig S_k on level with $na > ns$

 if $|\{S_i : S_i \in \text{Against}[S_k] \text{ and } \text{level}(S_i) \geq 2\}| >$

$|\{S_i : S_i \in \text{Support}[S_k] \text{ and } \text{level}(S_i) \geq 2\}|,$

 flip the subcontig S_k and prompt its level to 2;

 for each subcontig S_k on level with $na > ns$

 if all the orientation of subcontigs in its Support list are unsure,

 flip the subcontig S_k and prompt its level to 2;

5.2.15 Derive a consensus sequence for each contig

We derive a consensus sequence for each contig from the assembly of the maximal fragments of the contig. When choosing a sequence to represent an overlapping region, the priority is given to higher phase sequences. For example, if an overlapping region consists of a phase 3 sequence and a phase 1 sequence, the phase 3 sequence is chosen to represent the overlapping region. If all the sequences in the overlapping region are of the same phase, then a middle point of the region is chosen as a transition point for going from one sequence to another.

Chapter 6

Results and the comparison with the public assemblies

In this chapter, we present the result of our algorithm when applied to the input of the April freeze of the public working draft of the human genome. In Section 6.1, we describe the details of the input of the April freeze. We present our results in Section 6.2. We compare the results with the NCBI's public assembly and GIGASSEMBLER's assembly in Section 6.3. Finally, in Section 6.4, we present examples of errors detected by BARNACLE in this data set.

6.1 Input of the April freeze

The sequence information with chromosome assignments, local alignments of all fragment sequences against all fragment sequences, local alignments of all fragment sequences against plasmid reads, ESTs and mRNA sequences were provided by NCBI. We further processed these local alignments to generate valid fragment pair overlaps and relative orientation of fragment pairs. The details of the April freeze are as following:

1. Sequence Information

phase	BACs	fragments	total length in Gbp	average number of fragments
1	19132	333676	3.2	17.44
2	1923	11159	0.3	5.80
3	10204	10204	1.2	1.00
Total	31259	355039	4.7	11.36

Among the 31259 BACs, 403 BACs with 1802 fragments were removed by NCBI because of complete containment. For comparison purposes, we also removed these BACs. That is, 30856 BACs with 353237 fragments were used as input.

2. Chromosome Assignment

The scheme employed by NCBI to assign a chromosome to a BAC is based on:

- STS: presence of at least 2 STS markers that have themselves been mapped to the same chromosome;
- GenBank: annotation on the submitted GenBank record;

Otherwise, the chromosome of the BAC is *Unknown*.

The STS maps used are GENETHON, MARSHFIELD, WI-YAC, GM99-GB4, SHGC-G3. The chromosome assignments of BACs are summarized as below:

STS	GenBank	Unknown
16626	12809	1824

3. **Overlap Information** There are 381,038 fragment pairs of the same chromosome or at least one of them an unknown chromosome. We generated these overlaps based on the local alignments of each fragment sequence against all other fragment sequences provided by NCBI. They actually first RepeatMasked the sequences, masking off the regions which are known from repeat libraries, e.g. Alu repeats, to avoid too many spurious alignments and then did the all-against-all sequence local alignments by NCBI's Megablast. There are 5087 nt-pairs.
4. **Orientation Information** Relative orientation fragment pairs are generated from paired-end plasmid reads, ESTs and mRNAs. There are 343,360 fragment pairs from plasmid reads, 276,105 from ESTs and 288,675 from mRNAs.

6.2 Results on the April freeze

We assemble 244,629 non-singleton fragments into 25101 subcontigs, with length 2.11 Gbp. From these subcontigs, we obtain a clone graph of 29019 BACs in 3047 connected

components, as shown below.

number of BACs	number of connected components
21347	2904 (Interval)
7672	143 (Non-interval)
29019	3047

Upon resolving non-interval components, 137 problematic BACs, which are either suspicious chimeras or contains unidentified repeats, are removed. The result is 28854 non-singleton BACs in 3296 interval components, as shown below.

	BACs	Fragments Used/Fragments	Contigs	Length in <i>Gbp</i>
singletons	1865	25114/25114	1865	0.270
non-singletons	28854	321968/324699	3296	2.611
	30719	347082/349813	5161	2.881

We remove 2731 ($= 349813 - 347082$) fragments based on our FN detection which indicates the fragments should be contained in some subcontigs.

6.3 Comparison with the two public assemblies: NCBI's assembly and GIGASSEMBLER's assembly

In this section, we compare our results with the two public assemblies: NCBI's assembly and GIGASSEMBLER's assembly:

NCBI's Assembly:

	BACs	Fragments Used/Fragments	Contigs	Length in <i>Gbp</i>
singletons	1829	28263/28263	1829	0.273
non-singletons	29027	300185/324974	3269	2.564
	30856	328448/353237	5098	2.837

The 24789 (= 353237 – 328448) fragments were absent from NCBI's assembly for reasons we do not know.

GIGASSEMBLER's Assembly:

Since GIGASSEMBLER made use of the fingerprint clone contigs, their “contigs” are not the same as our contigs as some of them might contain gaps between two neighbor clones which do not overlap.

BACs	GIGASSEMBLER's Contigs	Length in <i>Gbp</i>
30897	3718	2.7711

Out of the 30897 BACs, 30784 are the same as ours; but 113 BACs are different (some of them have different accession versions.) When comparing with their assembly, only the common 30784 BACs are considered.

We compare the assemblies by considering the warps (defined in Section 4.3) of BACs.

Warp	Our Assembly	NCBI's Public Assembly	GIGASSEMBLER's Assembly
≤ 1.5	28622	25987	26602
1.5 – 1.8	1249	1368	1248
1.8 – 2.0	334	598	568
2.0 – 5.0	507	2335	2127
5.0 – 10.0	7	436	193
> 10.0	0	132	46

Note that warp might not be a good measurement if the estimated BAC length is not accurate. Since each BAC's length should not be more than 250Kb, we compare the assemblies by BACs' assembled length after restricting warp > 1.5 :

Assembled BAC Length	Our Assembly	NCBI's Public Assembly	GIGASSEMBLER's Assembly
$250K - 300K$	679	869	817
$300K - 500K$	750	2074	1976
$500K - 700K$	34	673	539
$700K - 1M$	0	349	206
$1M - 2M$	0	296	121
$2M - 3M$	0	45	15
$3M - 13M$	0	31	5
Total	1463	4337 ¹	3679

6.4 Examples of Suspected Errors Detected

- Chimeras.** Among the 137 problematic BACs that we removed, 59 are the most suspicious chimeras, 59 are less suspicious chimeras which can be also due to repeats, and 19 contain unclassified noise which includes the possible FNs. See Figure 6.1 for an example of a most suspicious chimera.

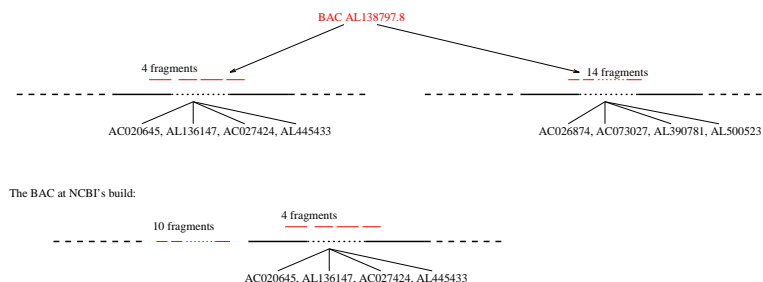


Figure 6.1: An example of a suspicious chimera. The suspicious chimera BAC *AL138797.8* has 19 fragments. Four fragments (total length of 110,361bp) overlap with one middle segment of a contig; another 14 fragments (total length 44,852bp) overlap with one middle segment of another contig. Below, the BAC in NCBI's build, 5 fragments were absent, 10 of the 14 fragments became singletons (i.e. the overlaps were discarded).

- Chromosome misassignments.** We detect and change 147 BACs' chromosome assignments. Independent follow-up shows that 78 of them have at least one STS supporting the suspicions. Many of them contains 2 or more STSs in the TNG map which was not used in this build by NCBI. See Figure 6.2 for an example.
- Wrong overlaps generated from the annotation of some finished sequences.** According to the annotation of GenBank records, NCBI generates some nt-pairs between finished sequences. The nt-pairs are believed to be true overlaps and thus would be chosen when there is a conflict. For this reason, both

¹We hypothesize that this number would be even worse if they had not thrown away the 24K fragments. Note while it is true that the warped BACs are misassembled, it is not necessary that the non-warped BACs are correctly assembled. Indeed, the suspicious chimeric BACs we detected are usually non-warped in NCBI's assembly because of the way their assembly is done.

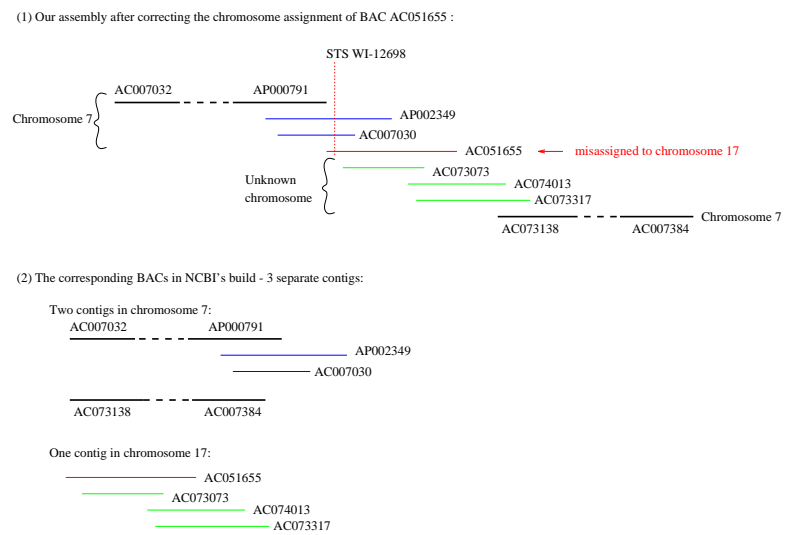


Figure 6.2: An example of a chromosome misassignment. BAC *AC051655* was assigned to chromosome 17 according to the annotation at GenBank record. (1) BARNACLE detects and corrects the chromosome assignment of BAC *AC051655*. Independent followup shows BAC *AC051655* shares an STS marker which was used to assign chromosome for BACs *AP002349* and *AC007030*. (2) The corresponding BACs in NCBI's build.

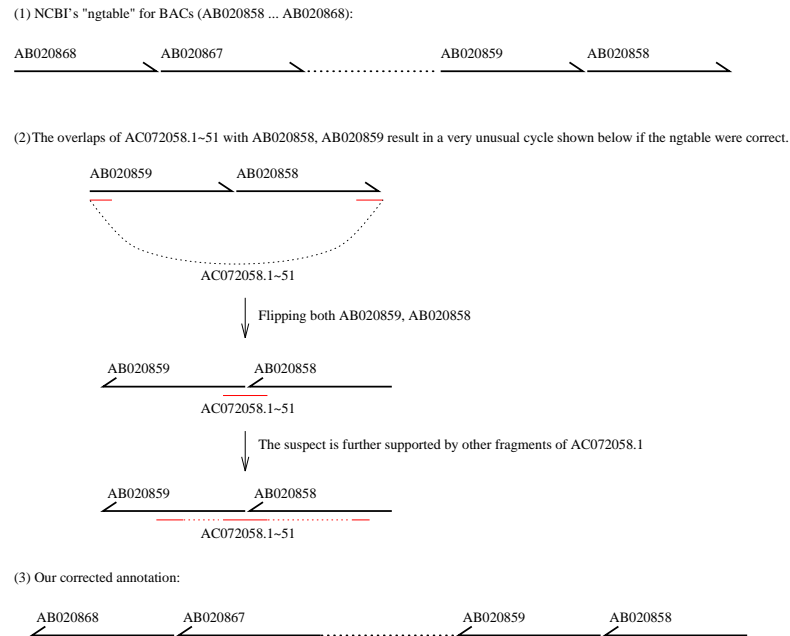


Figure 6.3: An example of wrong nt-pairs. According to the annotation of the 11 BACs of 8p21.3-p22 anti-oncogene of hepatocellular colorectal and non-small cell lung cancer (*AB020858* . . . *AB020868*), the NCBI's "ngtable", which is then used to generate nt-pairs, is generated for these BACs as shown in (1). However, our "conservative" assembly detects that all the BACs are in the wrong orientation/strand. For example, the wrong orientation of *AB020858.1* and *AB020859.1* (and hence the nt-pair) was detected by a draft BAC *AC072058.1* (as shown in (2)). Similarly, we have five other draft BACs to support the suspicion of wrong orientation of the other BACs. (3) is the corrected annotation of these BACs.

NCBI and GIGASSEMBLER assigned the highest score to these overlaps. However, the nt-pairs could be wrong. See Figure 6.3 for an example. This simple example shows that it is necessary to check the consistency of the underlying data in order to achieve a high-quality assembly.

- **Chimeric fragments or fragment misassemblies.** According to our FN detection, we aligned some involved fragments with their overlapping clones inferred by the algorithm. Relaxing sequence identity to 90% and end-allowed-errors to the minimum of 30% of the fragment length and 2000 bp for these alignments, 5283 more fragment pairs were identified. Examining some non-valid fragment pairs reveals low accuracy of fragment quality, fragment misassemblies or chimeric fragment (see Figure 6.4 for an example).

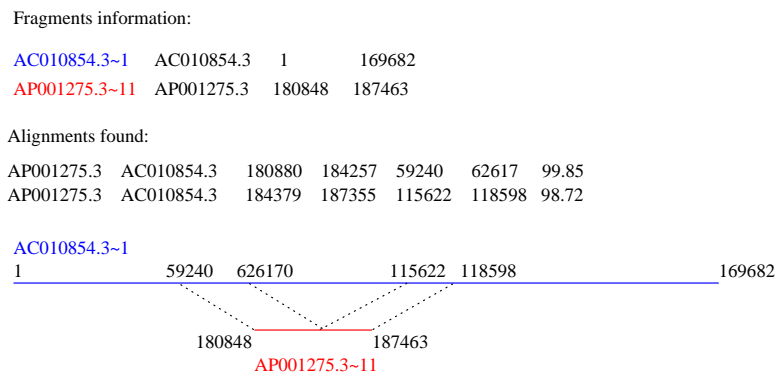


Figure 6.4: An example of a chimeric fragment. According to our FN detection, we aligned *AP001275.3~11* against all its overlapping BACs. The alignments of *AP001275.3~11* and *AC010854.3~1* indicate that *AP001275.3~11* consists of two sequences separated by more than 50Kb.

Chapter 7

Future Work and Remarks

Our algorithm is simple and efficient. It takes 3 minutes on a Pentium III (933MHz) computer to assemble the April freeze of the public working draft of the human genome¹. Most importantly, we believe the assembly produced by BARNACLE is quite truthful to the input data, as supported by the relatively low warp levels and the errors detected. Nevertheless, there are still many improvements that can be made to BARNACLE based on this framework.

Although the way we resolve the repeat problem by considering the consistency of overlaps and the interval graph formalism is mathematically justifiable, it is theoretically impossible to resolve some repeats based only on the sequence overlap information. For example, repeats which occur at one end of contigs due to the lack of coverage or repeats which are longer than an entire BAC cannot be resolved. More biological information is needed to resolve these cases. For instance, the 500Kb inverted repeat on chromosome Y is resolved by using the annotation of sequences in the GenBank. Hence, to resolve the repeat problem accurately, an iterative process involving collaboration with the sequencing centers and/or genome experts in repeats is necessary. Furthermore, there are various types of conflicts detected which require additional biological information to resolve them correctly. Thus, when collaboration with the sequencing centers and/or genome experts is possible, a future effort might output the problematic subassemblies, which cause conflicts, to request additional information for resolving them. With the additional information, we can then iterate the program (taking advantage of the efficiency of the algorithm) to reassemble the sequences. The conflicts

¹For this step, GIGASSEMBLER takes about two hours on a cluster of 100 Pentium III CPUs. We know that NCBI's assembler takes less than one day, but do not know the actual time.

detected include:

1. **Conflicting Chromosome Assignments:** There are three possible kinds of noise that can cause a conflicting chromosome BAC. These are chromosome assignments, repeats, and chimeras.
2. **Non-interval Connected Components:** There are three possible kinds of noise that can cause a forbidden subgraph. These are repeats, chimeras, and FNs. The forbidden subgraph together with the subcontig assemblies provide information as to what additional information is needed to identify the noise.

Also, the improvements might include the following:

1. **Preprocessing step:** The valid overlaps are generated from the local alignments of fragments. As remarked in Section 4.2.2, due to the draft quality and polymorphism, some valid overlaps might correspond to several disjoint pieces of local alignments. Better methods need to be developed to merge these pieces into valid overlaps. Also, currently when there is more than one valid overlap between a fragment pair, the longest overlap is chosen. This might not be the right choice. Additional information is needed for these cases. The relative orientations of fragment pairs generated from plasmid read, EST and mRNAs data (see Section 4.2.3) are noisy. Better methods of screening the noise need to be developed.
2. **FN detection:** Possible FNs can be detected while resolving inconsistent overlaps (see the remark in Section 5.2.6) and ordering the subcontigs violating the adjacency condition (see Section 5.2.10). The alignments of the involved fragments need to be examined to identify the overlaps. There are two kinds of FNs: fragment misassembly and polymorphism. In the public working draft of the human genome, the fragment misassembly problem is quite serious. If possible, one might want to take sequence reads as input directly and develop a two stages assembler: assemble sequence reads into fragments and assemble fragment into contigs. This way allow the second stage assembler (like BARNACLE) to directly verify whether some fragments are misassembled.

3. Ordering of the BACs: As mentioned in Section 5.2.10, the ordering of vertices of a proper interval graph is unique up to the permutation of vertices within each block. Thus the correct ordering of the the BACs involves both correctly identifying the subBACs (to make the interval graph proper) and the permutation of BACs within the same block. Once the ordering of the BACs has been determined from the interval realization of the BAC graph, we can refine the ordering from the subassemblies of subcontigs (thus the two problems, using the ordering of BACs to order subcontigs and refining the ordering of BACs using the assemblies of subcontigs, are intertwined). Currently, we adjust the ordering of BACs in the end blocks only if some of the BACs are warped. A better reordering method making better use of the assemblies of the subcontigs could be developed.
4. Resolving non-interval components: The heuristic used to resolve the non-interval component might be improved. For example, how should the articulation points be appropriately chosen to partition the graph? how can the noise (a vertex or a set of edges), which causes the forbidden subgraph, be better identified?
5. Chimeras: Since we consider only the overlaps between agreeable fragments, the chimeras with two sequences from two different chromosomes cannot be detected. To detect such chimeras, overlaps between all possible fragment pairs need to be considered. Also, for the suspicious chimeras, which might be due to repeats, follow-up (or error tracking) needs to be done for verification.

Finally, we remark that another advantage of our algorithmic approach is that we make better use of the available data. As it was also pointed out in [27], the clone-overlaps inferred from the sequences are more informative than from the fingerprint data. While the NCBI algorithm does not make use of the fingerprint-map as `GI-ASSEMBLER` does, the way NCBI makes use of the fragment overlaps to infer the clone overlaps is in the traditional, less informative physical-mapping clone overlap fashion. More precisely, in physical-mapping, one only knows whether two clones overlap by some common fingerprints, rather than at which ends they overlap, while such

information is explicit in the fragment overlaps. See Figure 7.1 for this important distinction. In particular, NCBI formulates a Maximal Interval Subgraph (MIS) problem (which is NP-hard) to obtain a clone ordering, where the weight of a clone-overlap is the summation of weights of all corresponding fragment pairs, which are assigned by some score functions. In contrast, we first “conservatively” assemble fragments, which uses the full information of fragment overlaps (and not just whether two fragments overlap), and then infer the clone overlaps from these subassemblies.

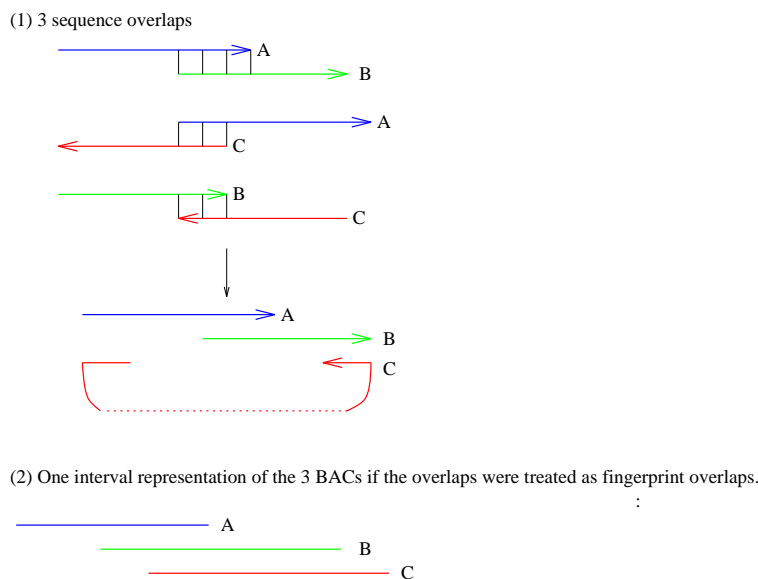


Figure 7.1: Sequence overlap vs. fingerprint overlap. (1) There are 3 sequence overlaps, but they are incompatible. At least one of them is FP. (2) Treated as fingerprint overlaps (whether two fragments overlap) would result in an incorrect interval representation of the 3 BACs.

A main goal of the Human Genome Project is to provide an accurate reference sequence of the euchromatic portions of all human chromosomes [15] and it is essential for an assembler to resolve repeats correctly in order to achieve accurate results [11]. It is our hope that future assemblers for clone-based sequences of whole genomes (including other higher organisms, such as the mouse, maize and rice, etc.) will be developed and improved based on this well-justified framework.

References

- [1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. A Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.
- [2] J. A. Bailey, A. M. Yavor, H. F. Massa, B. J. Trask, and E. E. Eichler. Segmental duplications: Organization and impact within the current human genome project assembly. *Genome Res.*, 11:1005–1017, 2001.
- [3] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and interval graph planarity using PQ-tree algorithms. *Journal of Computational System Sciences*, 13:335–379, 1976.
- [4] A. Brandstädt, F. Dragan, and F. Nicolai. LexBFS-orderings and powers of chordal graphs. *Discrete Mathematics*, 171:27–42, 1997.
- [5] V. Choi and M. Farach-Colton. Exon Search Tool, 2000. unpublished manuscript.
- [6] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.
- [7] The International Human Genome Mapping Consortium. A physical map of the human genome. *Nature*, 409:934–941, 2001.
- [8] D. G. Corneil, S. Olariu, and L. Stewart. The LBFS structure and recognition of interval graphs. Submitted for publication.
- [9] D. G. Corneil, S. Olariu, and L. Stewart. The ultimate interval graph recognition algorithm? (extended abstract). In *Symposium on Discrete Algorithms(SODA)*, pages 175–180, 1998.
- [10] National Research Council. *Mapping and Sequencing the human genome*. National Academy Press, 1988.
- [11] E. E. Eichler. Segmental duplications: What’s missing, misassigned, and misassembled - and should we care? *Genome Res.*, 11:653–656, 2001.
- [12] A. Coulson et al. Towards a Physical Map of Genome of the Nematode, *C. Elegans*. *Proc. Nat’l Academy of Science*, 83:7821–7825, 1986.
- [13] E. W. Myers et al. A whole-genome assembly of drosophila. *Science*, 287:2196–2204, 2000.
- [14] F. Sanger et al. Nucleotide sequence of bacteriophage λ DNA. *J. Molecular Biology*, 162:729–773, 1982.
- [15] F.S. Collins et al. New goals for the U.S. human genome project: 1998-2003. *Science*, 282:682–689, 1998.

- [16] J. C. Venter et al. The sequence of the human genome. *Science*, 291:1304–1351, 2001.
- [17] M. Olson et al. Random Clone Strategy for Genomic Restriction Mapping in Yeast. *Proc. Nat'l Academy of Science*, 83:7826–7830, 1986.
- [18] M. Olson et al. A Common Language for Physical Mapping of the Human Genome. *Science*, 245:1434–1435, 1989.
- [19] National Center for Biotechnology Information (NCBI). Available online URL: <http://www.ncbi.nlm.nih.gov/genome/guide/human/>.
- [20] E.D. Green. The human genome project and its impact on the study of human disease, 2001. To be published as Chapter 9 in the 8th edition of *Metabolic and Molecular Bases of Inherited Disease*.
- [21] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement with applications to transitive orientation, interval graph recognition and consecutive ones testing, 2000.
- [22] W.L. Hsu and T.H. Ma. Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM J. Comput.*, 28(3):1004–1020, 1999.
- [23] U.S. Department of Energy. Human Genome Program. Primer on molecular genetics, 1992. Available online URL: http://www.ornl.gov/TechResources/Human_Genome/publicat/primer/intro.html.
- [24] W.J. Kent and D. Haussler. Assembly of the working draft of the human genome with GigAssembler. *Genome Research*, 11, 2001.
- [25] N. Korte and R. H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989.
- [26] C. G. Lekkerkerker and J. C. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51:45–64, 1962.
- [27] E. W. Myers. Whole-genome DNA sequencing. *Computing in Science and Engineering*, 1(3):33–43, /1999.
- [28] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5:266–283, 1976.
- [29] U. Manber S. Wu, E. Myers and W. Miller. An $O(NP)$ Sequence Comparison Algorithm. *Information Processing Letters*, 35(6), 1990.
- [30] A. Akiyama Y. Kohara and K. Isono. The Physical Map of the *E. Coli* Chromosome: Application of a New Strategy for Rapid Analysis and Sorting of a Large Genomic Library. *Cell*, 50:495–508, 1987.

Vita

Vicky Choi

- 1989-93** Attended The Chinese University of Hong Kong, Hong Kong. Majored in Mathematics.
- 1993** B.Sc., The Chinese University of Hong Kong.
- 1993-95** Graduate work in Computer Science, The Hong Kong University of Science and Technology, Hong Kong.
- 1993-95** Teaching Assistant, Department of Computer Science, The Hong Kong University of Science and Technology.
- 1995** M.Phil in Computer Science, The Hong Kong University of Science and Technology.
- 1996** V. Choi, M. Golin. Lopsided trees: analyses, algorithms, and applications. Proc. of 23rd International Colloquium on Automata Languages and Programming (ICALP), pages 538-549.
- 1996-01** Graduate work in Computer Science, Rutgers, The State University of New Jersey.
- 1996-98** Teaching Assistant, Department of Computer Science, Rutgers, The State University of New Jersey.
- 1999-00** Graduate Assistant, Waksman Institute, Rutgers, The State University of New Jersey.
- 2000-01** (Supplemental) Visiting Pre-doctoral Fellow, National Center for Biotechnology Information (NCBI), US National Institutes of Health.
- 2001** V. Choi, M. Golin. Lopsided trees, I: Analyses. *Algorithmica*, 31:240-290.
- 2001** Program in Mathematics and Molecular Biology (PMMB) Fellowship.
- 2002** Ph.D. in Computer Science, Rutgers, The State University of New Jersey.