A STUDY AND EVALUATION OF DNA-SEQUENCE-ASSEMBLY PROGRAMS

ANUKOON WISITSORA-AT

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE
(COMPUTER SCIENCE)
FACULTY OF GRADUATE STUDIES

MAHIDOL UNIVERSITY
2005

ISBN 974-04-6109-3

Thesis
Entitled

A STUDY AND EVALUATION OF DNA-SEQUENCE-ASSEMBLY PROGRAMS

Anukoon Wisitsoraat
.....
Mr.Anukoon Wisitsoraat
Candidate

C. Pluempitiwiriyawej
.....
Asst. Prof. Charnyote Pluempitiwiriyawej, Ph.D.
Major Advisor

Chinae Thammarongtham
.....
Mr.Chinae Thammarongtham, Ph.D.
Co-advisor

Rassmi Hoon
.....
Assoc.Prof.Rassmidara Hoonsawat, Ph.D.
Dean
Faculty of Graduate Studies

Supachai Tangwongsan
.....
Assoc. Prof. Supachai Tangwongsan, Ph.D
Chair
Master of Science Programme in Computer Science
Faculty of Science

Thesis
Entitled

A STUDY AND EVALUATION OF DNA-SEQUENCE-ASSEMBLY PROGRAMS

was submitted to the Faculty of Graduate Studies, Mahidol University
For the degree of Master of Science (Computer Science)
on
May 10, 2005

Anukoon Wisitsoraat
.....
Mr.Anukoon Wisitsoraat
Candidate

C. Pluempitiwiriyawej
.....
Asst. Prof. Charnyote Pluempitiwiriyawej, Ph.D.
Chair

Chinae Thammarongtham
.....
Mr.Chinae Thammarongtham, Ph.D.
Member

.....
Miss.Supatana Auethavekiat, Ph.D.
Member

Duangdao Wichadakul
.....
Miss.Duangdao Wichadakul, Ph.D.
Member

Rassmidara Hoonsawat
.....
Assoc.Prof.Rassmidara Hoonsawat, Ph.D.
Dean
Faculty of Graduate Studies
Mahidol University

Amaret Bhumiratana
.....
Prof. Amaret Bhumiratana, Ph.D
Dean
Faculty of Science
Mahidol University

ACKNOWLEDGEMENTS

I would like to express my great appreciation to Dr. Charnyote Pluempitiwiriyawej, my thesis advisor, who guides me through steps to completion. I am grateful to Dr. Chinae Thammarongtham for his assistance and for serving as my co-advisor.

I am specially grateful to my parents and all teachers who have taught me. Finally, I would like to thank all my friends and colleagues whose helps I drew from on many occasions.

Anukoon Wisitsoraat

A STUDY AND EVALUATION OF DNA-SEQUENCE-ASSEMBLY PROGRAMS

ANUKOON WISITSORAAT 4337389 SCCS/M

M.Sc. (COMPUTER SCIENCE)

THESIS ADVISORS: CHARNYOTE PLEUMPITIWIRIYAWEJ, Ph.D.,
CHINAE THAMMARONGTHAM, Ph.D.

ABSTRACT

DNA sequencing is one of the most important techniques for life science in the era of genome research. The limitations of current biochemical techniques make DNA assembly, particularly for large-scale genomes, highly challenging since it is complicated, time-consuming, and expensive. Computer technologies can be used to align and assemble fragments. The alignment and the assembly processes are complex and incompletely solved due to repeated patterns, DNA evolutions and sequencing errors.

This thesis describes the underlying research, design and implementation of a new sequence assembly program, called *SEQASS*, to help in alignment and assembly of the whole genome sequences. SEQASS has been developed with respect to the framework presented in CAP3. Unlike CAP3, however, SEQASS uses BLAT, a BLAST-like tool, to accelerate pair-wise comparisons and improve assembly quality. SEQASS has been evaluated on the DNA sequences of mammal organisms, which were obtained from Genbank. The evaluation measurements include the total percent coverage, the total percent of identity, the number of good contigs, the average size of good contigs, the number of misassembled contigs, and the total run time. The output contig sequences were evaluated by comparing the sequences with the known whole genome input by BLAT. SEQASS performances were compared to other available programs including CAP3, CAP1, and TIGR. These evaluations showed that SEQASS is a useful tool in the alignment and assembly of whole sequence program and has good potential in the field of genome research.

KEYWORDS:DNA EQUENCE ASSEMBLY/DNA SEQUENCING/SEQASS/BLAT
156 P. ISBN 974-04-6109-3

การศึกษาและการประเมินค่าของโปรแกรมรวบรวมดีเอ็นเอ(A STUDY AND EVALUATION OF DNA-SEQUENCE-ASSEMBLY PROGRAMS)

อนุกูล วิศิษฏ์สรอรรถ 4337389 SCCS/M

วท.ม. (วิทยาการคอมพิวเตอร์)

คณะกรรมการควบคุมวิทยานิพนธ์ : ชาญยศ ปลื้มปิติวิริยะเวช, Ph.D., ชิเน ธำมรงค์ธรรม, Ph.D.

บทคัดย่อ

การเรียงลำดับดีเอ็นเอ (DNA sequencing) เป็นงานวิจัยทางด้านเทคโนโลยีชีวภาพที่สำคัญที่สุดอันหนึ่ง การต่อลำดับดีเอ็นเอสำหรับสิ่งมีชีวิตในปัจจุบันมีความยุ่งยากซับซ้อนและราคาแพงมากเนื่องจากข้อจำกัดของเคมีชีวเทคนิค จึงได้มีการนำเทคโนโลยีคอมพิวเตอร์มาใช้เพื่อช่วยในการจัดเรียงและรวบรวมดีเอ็นเอให้รวดเร็วและราคาถูกขึ้น อย่างไรก็ตามกระบวนการของการจัดเรียงและรวบรวมดีเอ็นเอด้วยโปรแกรมคอมพิวเตอร์นั้นยังต้องมีการวิจัยอีกมากเนื่องจากปัญหาที่ยังไม่สามารถแก้ได้ทั้งหมด ได้แก่ แบบรูปซ้ำซ้อน (repeated patterns) วิวัฒนาการของดีเอ็นเอ (DNA evolutions) และการลำดับผิด (sequencing errors)

วิทยานิพนธ์นี้อธิบายการออกแบบและพัฒนาโปรแกรมรวบรวมดีเอ็นเอใหม่ที่ชื่อว่า *SEQASS* ใช้เพื่อช่วยในการจัดเรียงและรวบรวมดีเอ็นเอให้มีความถูกต้องสูงในระดับเดียวกับโปรแกรมที่มีขายอยู่ในเชิงพาณิชย์ โปรแกรม SEQASS ถูกพัฒนาขึ้นมาตามหลักการที่ใช้ในโปรแกรม CAP3 แต่โปรแกรม SEQASS ก็แตกต่างจาก CAP3 ตรงที่ได้มีการใช้ BLAT ซึ่งเป็น BLAST-like tool เพื่อเร่งการเปรียบเทียบระดับคู่ (pair-wise comparisons) ให้เร็วขึ้น พร้อมทั้งปรับปรุงคุณภาพผลการรวบรวมดีเอ็นเอ โปรแกรม SEQASS ได้ถูกประเมินค่าโดยใช้ลำดับของดีเอ็นเอของสิ่งมีชีวิตที่เลี้ยงลูกด้วยนมหลายชนิดซึ่งได้มาจากฐานข้อมูลของ Genbank ตัววัดที่ใช้ในการประเมินค่าได้แก่ เปอร์เซ็นต์ของการครอบคลุม(total percent coverage) เปอร์เซ็นต์ของความเหมือน (Total percent of identity) จำนวนและขนาดเฉลี่ยของคอนทิกที่ดี (the number and average size of good contigs) จำนวนของคอนทิกที่เสีย (the number of misassembled contigs) และเวลาที่ใช้ในกาทำงาน คอนทิกที่ได้ได้ถูกประเมินค่าโดยการเปรียบเทียบกับสายดีเอ็นเอรวมที่รู้มาก่อนโดย BLAT คุณภาพของโปรแกรม SEQASS ได้ถูกประเมินค่าและเปรียบเทียบกับโปรแกรมอื่นที่สามารถหามาได้ เช่น โปรแกรม CAP3 CAP1 และ TIGR. ในการประเมินค่าเราพบว่าโปรแกรม SEQASS เป็นเครื่องมือที่มีประโยชน์ในการจัดเรียงและรวบรวมดีเอ็นเอ รวมถึงเป็นเครื่องมือที่มีศักยภาพสูงในงานวิจัยทางด้านเทคโนโลยีชีวภาพด้วย

156 หน้า ISBN 974-04-6109-3

CONTENTS

LIST OF TABLES

LIST OF TABLES(CONT.)

LIST OF TABLES(CONT.)

LIST OF FIGURES

LIST OF FIGURES(CONT.)

LIST OF FIGURES(CONT.)

CHAPTER 1
INTRODUCTION

1.1 Problem statement

The discovery of the famous DNA in 1953 [1] has led to enormous progress in biology and biotechnology researches. A new research area called *bioinformatics* has been introduced. It combines biological, computational, and informatic sciences for genetic code exploration. The informatic sciences also play an important role because the accumulation of large amount of data from various genome research centers has necessitated international databases for nucleic acids and proteins [2]. One of the most interesting, on-going bioinformatic researches is the DNA sequencing [2-26] from which the genomic information is discovered. It is the fact that knowing a complete DNA sequence of an organism can help us to understand its functions and mechanisms. Although many organisms e.g., *Bacterio-phage* λ [3], *Cytomegalovirus* [4], etc., have been completely sequenced, most of them are in short-scale (i.e., hundreds or thousands base pairs in length). The limitations of current biochemical sequencing techniques (e.g., radioactive sequencing [5], dye-labeled sequencing [6], and electrophoresis sequencing [7]) make genome sequencing of long-scale organisms (i.e., millions base pairs in length) highly challenging since it is complicated, time-consuming, and expensive.

In 1982, F. Sanger [8] has introduced a genome sequencing technique, called *shotgun sequencing*. Its idea is to shear an unknown DNA sequence of an organism into a large number of fragments each of which can be sequenced individually. Each fragment is randomly cloned into a universal cloning vector. Then, the cloned pieces are randomly taken by biochemical sequencing techniques, which will generate a number of decoded sequences, called *reads*. Next, these reads are ordered, based on overlaps in the genetic code. Finally, these overlapping reads are assembled into the complete sequence.

With the advanced computer technologies, several shotgun sequence assembly algorithms and programs have been developed to reconstruct the original sequence from DNA fragments. The basic reconstruction process involves finding pairs of overlapping fragments, merging as many fragments together as possible, and creating a consensus sequence from the merge fragments. Some earlier DNA sequence assembly programs such as XBAP [9], Autoassembler [10], GCG [11], FALCON [12], and several others [13-17] were developed based on this general algorithm. These programs work reasonably well. Each of them has its advantages and disadvantages. However, they do not typically perform satisfactorily on the reconstruction of an order of larger scale magnitude because they were not designed to handle complicating factors.

Recently, several more advanced shotgun sequence assembly programs e.g., CAP3 [18], ARCHENE [19-20], CELERA [21], TIGR [22], and PHRAP [23] have been developed to make ordering and assembling of reads to be faster, more accurate, and less expensive. The basic details of these algorithms will be described in Chapter 3. However, these programs still need further improvement because of the following complicated, incompletely-solved problems [2,24]:

- *Repeated patterns*. A repeated pattern, which appears in each fragment, causes a number of potential overlaps and makes the assembling process difficult. A sequence may be incorrectly assembled due to the repeated pattern.
- *DNA evolution*. The evolution of DNA involves the insertion, the deletion, and the substitution of genetic codes in a DNA sequence. The ordering of fragment reads requires matching of genetic codes in the fragment reads. When evolution occurs, the *approximate* matching between fragment reads is desirable.
- *Large number of fragment reads*. The large number fragment reads requires a large amount of computation time and resources, which is proportional to the quality of matching and assembling sequences.
- *Unsequenceable fragments*. Some DNA fragments from each genome are impossible to sequence. The unsequenceable fragments cause incomplete sequence.

- *Chimeric reads*. A chimeric read is an experimental error wherein fragments from two or more different parts of the genome are combined together into a single read.
- *Sequencing errors*. Various biochemical sequencing techniques used for DNA decoding are often prone to errors due to uncertainty of data and incorrect interpretation. Decoded DNA fragments often contain erroneous codes especially near their both ends. This also leads to incorrect assembly and less assembly coverage because overlapping between reads can be hidden or be incorrectly found.

1.2 Goal of the thesis

In this thesis, we provide knowledge to research community in Thailand for better understanding of the state-of-the-art DNA sequence assembly program. We have studied and compared various assembly programs, e.g., CAP3, ARCHENE, CELERA, TIGR, and PHRAP. In addition, we have developed a new DNA sequence assembly program called *SEQASS*. Our program will follow the framework presented in CAP3 [18], which has shown high assembly quality, moderate complexity, and straightforward methodology. Unlike CAP3, however, we use BLAT [33], a BLAST-like tool, to accelerate pair-wise comparisons and improve assembly quality. Our program can (a) identify the occurrences of insertion, deletion, and substitution of genetic codes, (b) determine potential overlaps between pairs of fragment reads by approximate matching methodology, (c) align and assemble the fragment reads to form a whole genome sequence with a minimum error and maximum coverage. Our program can be a basis of future development of more advanced DNA sequence assembly for Thai-bioinformatics research communities to reduce Thailand's foreign software technological dependency. A quantitative analysis of the performance of our DNA sequence assembly program has been performed. The evaluation is based on the number of errors of matching and the number coverage of the whole-genome sequence.

1.3 Organization of the thesis

The rest of this thesis is organized with following topics.

Chapter II *Review of DNA sequence assembly* gives a general review of DNA sequence assembly processes and algorithms. Basic genome sequence assembly strategies and methods are explained.

Chapter III *Study of DNA sequence assembly programs* provides an extensive review of DNA sequence assembly programs. Algorithms of recently developed genome sequence assembly programs including CAP3, ARCHENE, CELERA, PHRAP, and TIGR are described.

Chapter IV *SEQASS program and main modules* explains the implementation of shotgun sequence assembly developed in this research work. Data preprocessing, dynamic programming algorithm for computation of overlaps, and process of joining reads into contigs are described in details.

Chapter V *Module evaluation and discussions* shows the DNA assembly results of shotgun sequence program of the raw read data of different organisms. Part of assembly results and all detailed statistics at each processing step are discussed.

Chapter VI *Qualitative analysis of shotgun sequence assembly results* evaluates and analyzes the assembly results of shotgun sequence assembly program. The performances of program in term of the number/size of contigs, the percent identity of contigs, the percent genome covered and the number of misassemblies are discussed. Finally, the performance of SEQASS is compared to several other available programs, including CAP3, CAP1, and TIGR.

Chapter VII *Conclusion and future work* concludes the study results of shotgun sequence assembly program and provides suggestion for future work.

Appendix *Basics of molecular biology* provides a brief review of molecular biology for DNA. Biological chemistry of DNA, RNA, and protein molecules are explained.

CHAPTER 2
REVIEW OF DNA SEQUENCE ASSEMBLY

Since the work in this thesis is involving with DNA sequence assembly programs, we provide the reader with a brief introduction on genome sequencing strategies and then general methods and algorithms of DNA sequence assembly.

2.1 Genome sequencing strategies

Genome sequencing refers to the process of decoding an unknown genome sequence. Two commonly used strategies of genome sequencing are the sequencing by hybridization (SBH) and the shotgun sequencing.

In SBH strategy [24], a single strand DNA sample is labeled with a radioactive or fluorescent material and then hybridized with a sequencing chip that is a complete two-dimensional array of known k-tuple DNA probes. Due to major experimental and mathematical difficulties, SBH method is not practical for large genome sequencing.

In shotgun sequencing [2], a genome is broken into small fragments. The fragments are then cloned, sequenced by a biochemical method, and assembled together by overlapping and clone-mate information. This method is generally divided into two approaches: the hierarchical shotgun (HS) and the whole genome sequence assembly (WGSA). In HS approach, the whole genome sequence is first divided into large hierarchical subsequences called *Bacterial Artificial Chromosomes* (BACs), which can be mapped to reconstruct the original genome. DNA sequencer is then used to assembly sequences in each BAC. In WGSA approach, DNA sequencer is directly used to build the whole genome from DNA fragments of the whole genome without a BAC map.

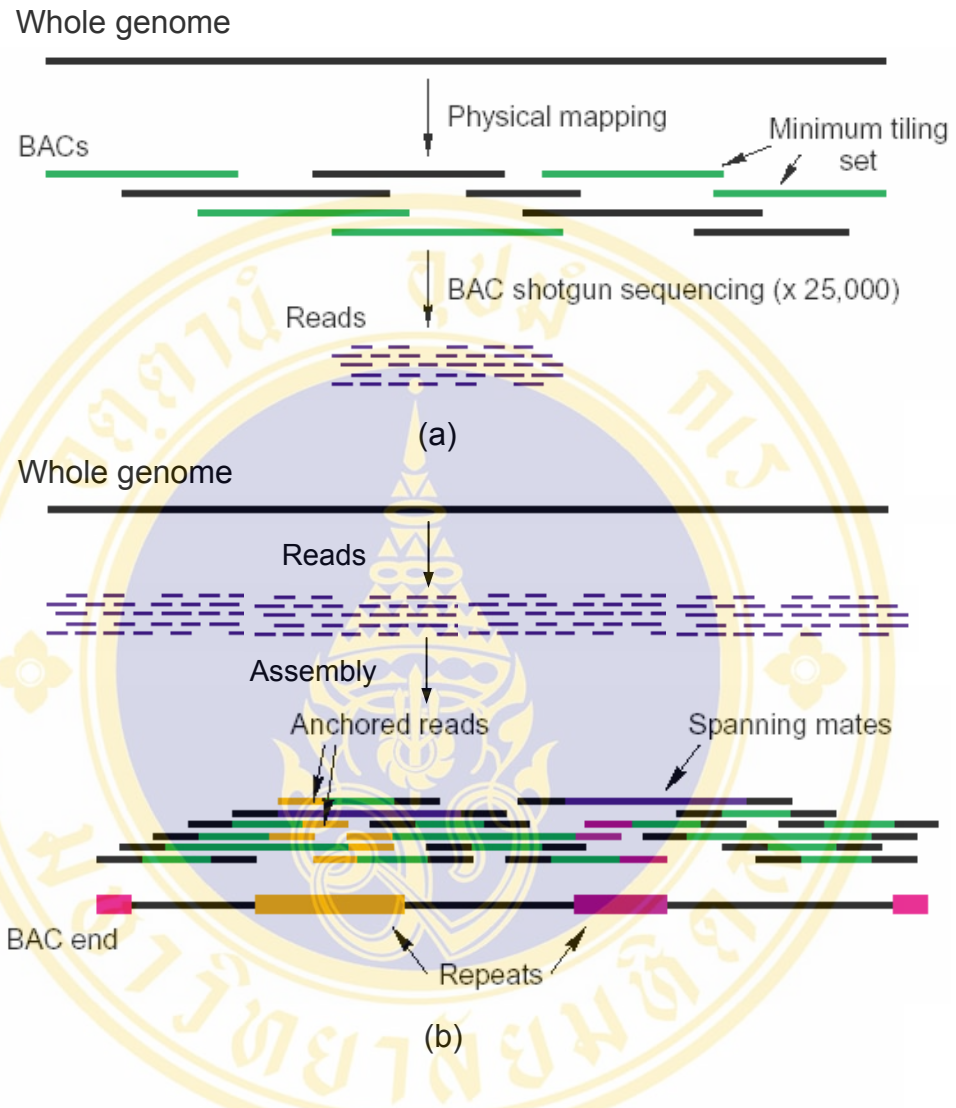

Fig. 2.1 Principles of (a) hierarchical shotgun and (b) whole genome sequence assembly approaches.

The concept of the HS and the WGSA approaches are illustrated in Fig. 2.1 (a) and (b), respectively. It can be seen that the WGSA approach is applied to the entire genome all at once, while the HS approach is applied to large, overlapping DNA fragments of known location in the genome. The WGSA approach requires enormous computational resources because of very large data size. In addition, the growing number of repeats in large genome would cause more misassemblies in the shotgun sequencing process. Although WGSA seems not appropriate for large DNA sequence assembly, it has recently shown to be more successful method for large DNA sequencing than the HS as more efficient shotgun sequence assembler has been

developed for large genomes [2]. The *Drosophila melanogaster* genome of a fruit fly with 135 Mbp was the largest genome successfully sequenced and assembled to date by the WGSA method [25], while the HS method has failed to complete the sequence assembly by the same time. The HS mapping technique is presently not popular for large DNA sequence decoding because it is much more time-consuming than the WGSA due to slow mapping construction and multiple shotgun sequencing processes. Therefore, the WGSA is currently the main choice for whole DNA sequence decoding and the main research on whole DNA sequence assembly has been focused on the development of more efficient algorithms for direct shotgun sequencing of very large genome.

2.2 General methods and algorithms of DNA sequence assembly

Genome sequence assembly process consists of the following five main tasks [24]: whole genome disassembly, cloning, decoding of DNA fragments, shotgun sequence assembly, and finishing. The first three tasks involve biological processing of DNA as shown in Fig. 2.2. In the last two tasks, computer technology plays a very important role.

2.2.1 Whole genome disassembly

The process of sequencing starts from a pure sample of a large number of copies of the source DNA, which are produced by polymerease chain reaction (PCR) of a few DNA samples. DNA sequence length of typical organism is typically 100 Kbps or much longer, which can not be directly decoded by electrophoresis. Thus, the strand of whole genome sample is cut into small DNA fragments. There are several methods for genome disassembly used in biology laboratory. First, sound (sonication) is used to randomly break a sample into a collection of DNA. Second, the sample was cut by passing into a nozzle under pressure (nebulation). Last, whole genome can be cleaved biologically by restriction enzymes. Restriction enzymes always cut DNA source at short specific patterns. A number of restriction enzymes are used to obtain DNA fragments over most patterns or locations. For DNA decoding, DNA fragments must have suitable size the range of ~100-1000 bps. DNA fragments with desired-size

can be selected from all fragments using size-separation under gel electrophoresis and excising a band of the gel.

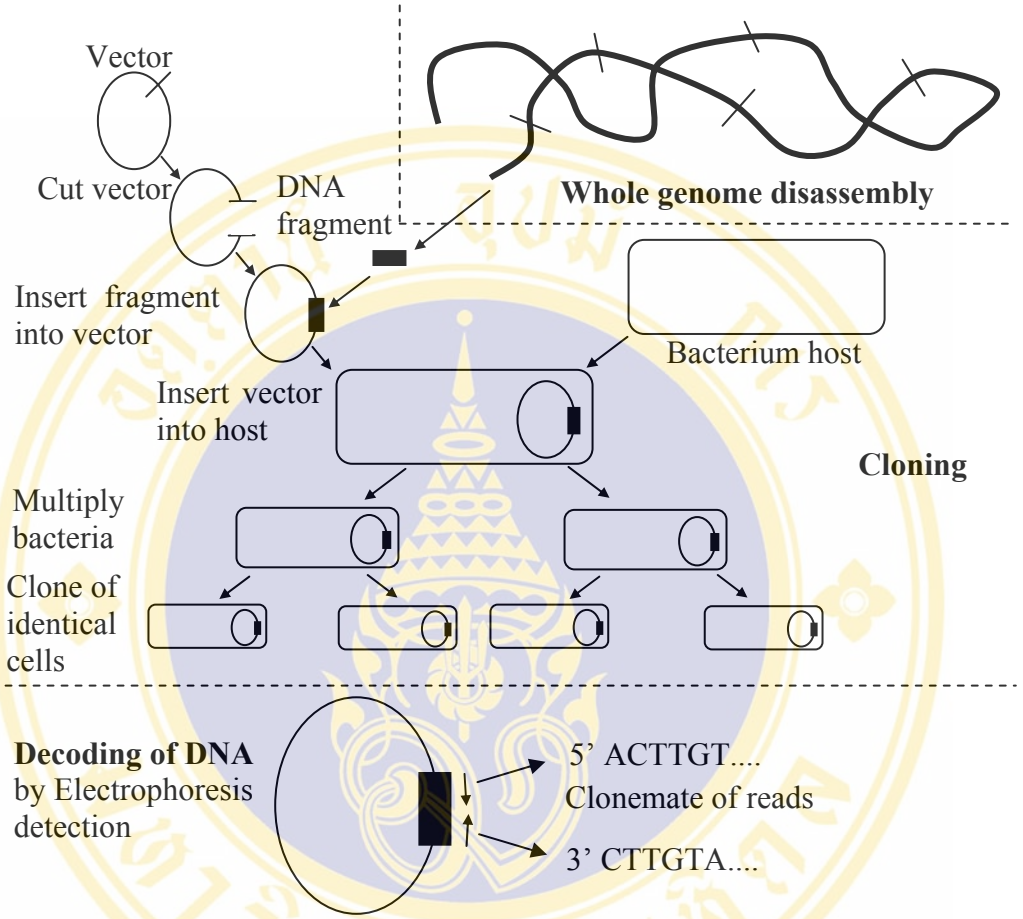

Fig. 2.2 Biological processing of DNA for shotgun sequence assembly.

2.2.2 Cloning

Before decoding of DNA fragments, DNA fragments must be amplified multiple times because genome sequencing will be performed by overlapping assembly and multiple number of DNA fragments at different parts of genome is required to cover most of the whole genome. From reported sequencing experiment, eight-fold over sampling of read with an average length of 600 to 700 base pairs by cloning is needed to achieve 99% of genome covered by reads after the initial sequencing phase.

Cloning is a biological process used to perform recombinant DNA amplification. Cloning process as shown in Fig. 2.2 begins with cloning vector or clone. Typically, cloning vector is a virus or a bacterium that can infect a bacterial

host. Vectors can be categorized into three types: *Lamda*, *Plasmid*, and *Cosmid*. The *Lamda* is a virus vector. The *Plasmid* is a bacterial virus vector. The *Cosmid* is a combination of Lamda and Plasmid. Vector itself is normally circular sequences of DNA reads. It has a known sequence of base pairs and can accept a clone insert of foreign DNA. To start cloning, vector is cut with a restriction enzyme and the size-selected DNA fragment was inserted into the vector. At most one fragment is inserted at predetermine point called the cloning site that has two ends. The fragment at this point is called *insert* and the collection of insert is called a *clone library*. Clones and both ends of each clone insert are then multiplied by inserting into a bacterial host. The bacterial host with inserted vector is then fed under suitable condition leading to self-multiplication and increasing the number of reads necessary for genome overlapping assembly process.

2.2.3 Decoding of DNA fragments

The DNA fragments in the insert vector is decoded to obtain the nucleotide content by a biochemical process called *gel electrophoresis*. In electrophoresis [6-7], biochemical sequencing reaction produces a collection of geometrically distributed copies of every prefix of the given DNA strand such that the last nucleotide or base A, C, G or T of each prefix is known. This material passes through a permeable gel under applied voltage. The distributed material is then decoded to determine the sequence of nucleotides along one end of the source strand by a technician or a combination of a laser, charge-coupled device (CCD) detector, and software. The length of DNA sequence that can be decoded is limited as the size *ratio* (between consecutive prefixes) approach one because the number of long prefix is diminishing geometrically [7]. In typical DNA sequence decoding, the average length of DNA is approximately 500 bps and the maximum length of DNA is about 1000 bps. The decoded result of DNA fragment is now called a *read*. Within each insert vector, a pair of reads, which is called *clonemate*, are obtained from both ends of insert in opposite direction, regarding the so-called forward-reverse constraint. The clonemate should be specified by a range of distance. The specified range of distance is the associated with the nominal length of insert in the vector, which is approximately

known for each type of vector. The set of clonemate pairing read data is extremely valuable for guiding the assembly process and correctly ordering a contiguous sequence, so called a *contig*.

2.2.4 Shotgun sequence assembly

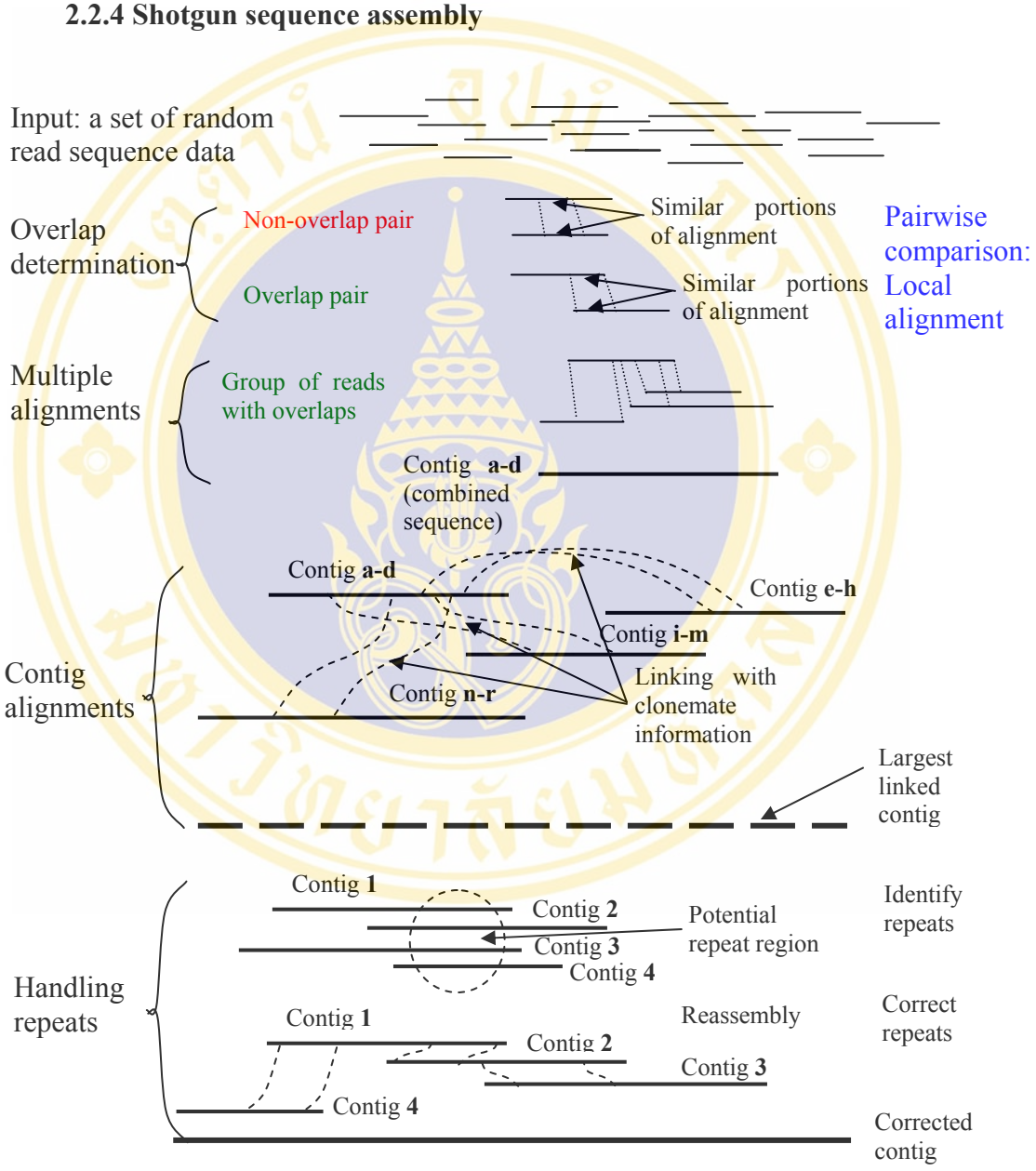

Fig. 2.3 Concept of basic shotgun sequence assembly.

The fragment reads obtained from biological process are then assembled together by a process called the *shotgun sequence assembly*. The main principle of shotgun assembly is to combine all reads into contigs based on similarity of overlap

between reads. As shown in Fig. 2.3, the shotgun sequencing process contains the following four main steps [2]: overlap determination, multiple alignment, contig alignment, and handling repeats.

Overlap determination

Each pair of reads is compared to determine similarity between them. After that, the position and identity percentage of similar parts in the pair of reads are used to determine the possibility of correct overlap. The overlap determination is sometimes called *local alignment* since it performs pairwise comparison. Several important programming algorithms have been developed to efficiently compare two sequences and identify their similarity. Major approaches that have been used to identify the similarity of reads are the following.

- *Greedy merging heuristics*: The main idea of greedy merging heuristic is to find the shortest common superstring of a set of sequences. This approach has been used in *BLAST* [28-32] and *BLAT* [33], which are among the most efficient and fastest tool for DNA pairwise comparison. Both tools rapidly scan for relatively short matches (hits) and extend these into high-scoring pairs (HSPs). However, BLAT differs from BLAST in some significant ways [33]. First, BLAST builds an index of the query sequence and then scans linearly through the database, but BLAT builds an index of the database and then scans linearly through the query sequence. Second, BLAST triggers an extension when one or two hits occur in proximity to each other, while BLAT can trigger extensions on any number of perfect or near-perfect hits. Last, BLAST returns each area of homology between two sequences as separate alignments, while BLAT stitches them together into a larger alignment.
- *Dynamic programming*: The concept of dynamic programming is to calculate a two-dimensional matrix similarity score between each pair of reads [24, 35-36]. The similarity score is accumulated so that we can trace local or global similarity between the DNA fragment pair by adding a positive score for each identity and subtracting penalty scores for each mismatch and insertion or deletion (indel). Thus, the maximum cumulative

similarity score in the matrix can identify the position of best matching between the pair and region of similarity can be determined by properly tracing through the matrix score. Dynamic programming takes much more computational time than BLAT but it can produce the most optimal alignment.

- *Eulerian path*: This method is to create a virtual Sequencing by Hybridization (SBH) problem by breaking the read into overlapping *n*-mers where *n*-mer is a substring of length *n* from the original sequence [24]. Next, the assembler builds a directed deBruijn graph in which each edge correspond to an n-mer from one of the original sequence reads. The source and destination nodes correspond respectively to the *n*-1 prefix and *n*-1 suffix of the corresponding *n*-mer. An overlap represents a path that uses all the edges that is an Eulerian path.

Multiple alignments

After potential overlaps are determined for all pairs, the groups of reads that are found to be potentially correct overlap are merged together successively to form contigs. The greedy graph algorithm is the most commonly used method for contig construction [36-38]. This approach constructs a graph in which *nodes* represent reads and *edges* indicate the corresponding reads overlap. Each contig is represented as a simple path, which is a path through the graph that contains each node at most once. To construct contigs, two reads are initially merged and form the combined sequence. Next reads are then successively merged into the combined sequence. For each starting read called *root,* a tree is constructed to keep all fragments it contains. In addition, the sub-fragments, which are contained in another fragments and is called *child* fragment, are identify for each main fragment in the tree. In each merging step, mismatches and indels in the sequence are determined. After merging all reads into the group, the mismatches and the indels in the combined sequence are finally identified from the collected statistics. The process of final identification of mismatches and indels in the combined sequence is called “the consensus of multiple alignments”. The consensus of multiple alignments is then defined as “contig”.

Contig alignment

The set of contigs obtained from multiple alignments is then linked together by some set of constraint information including clonemate information. This information provides the constraint that the approximate distance between mate pair, which must be of opposite direction, must be within a specific value, which is corresponding to the size of cloning vector. This can be used to link between contigs containing the clonemate. The contig are repeatedly connected together by the constraint information until the set of the largest possible contigs is obtained.

Handling Repeat

In the basic assembly, two overlapping reads wherein a suffix of one is a prefix of another are assumed to be originated from the same region. This assumption is invalid for repeat reads because they cannot be identified from distinct places in the genome. Thus, difficult assembly work mainly comes from repeats of reads and repeats of overlapping between reads. Thus, the program must identify repeats, preferably during the assembly process to avoid mistakes caused by over collapsing repeat copies and to correctly assemble as many repeats as possible to reduce the amount of human labor involved in finishing complete genome. Following are methods for repeat detection

- *Large overlapping reads* is usually used to identify potential repeats. This simple idea is useful but it is only valid for uniformly sampled genome data. In reality, genome data may not be uniformly sampled due to some limitations of the DNA sampling process and low-copy repeats, which appear only a few times in a genome, may escape detection because they do not appear to be statistically oversampled.
- *Tangles* are complex areas in the graph constructed by the assembler using Eulerian path technique. Information contained in tangles can be used to guide experiments to resolve the repeat.
- *Clone-mate information* generated in cloning process can also be used to detect areas that have been incorrectly assembled due to repeats. Such areas usually contain many instances of clone mates that were assembled either too close or too far from each other, or whose relative orientation is

incorrect. When repeats are widely separated in the genome, clone-pairing data can resolve them effectively from reads whose mates are anchored in the neighboring non-repetitive areas.

- *Minor differences between repeats* can be found to distinguish the copies from one another. In the absence of sequencing errors, a single nucleotide difference between two copies of a repeat is enough to distinguish them.
- *Clusters based on shared differences* in the reads can be used to statistically resolve the repeats. This approach assumes that sequencing errors are independent and, therefore, that an identical position difference in multiple reads is likely to be a real difference typical of that copy of the repeat.
- *Directed sequencing experiments* and a *jumpstart assembly* process is used to solve repeats that can not be solved by all information sources described above. In directed sequencing experiment, researchers amplify stretches of DNA anchored in unique areas around the repeat. The information from the experiment will then be used for jump start assembly, in which the assembly program assembles a mixture of contigs and reads. In the jump start assembly, the program should join the contigs and reads such that the contigs do not break up in the process and the number of contigs does not increase.

2.2.5 Finishing

In practice, imperfect coverage, repeat and sequence errors produce a number of contigs. The task of finishing is to close gap between contigs. The program for this task is called scaffolder. There may also be gaps between scaffolders, which are needed to be further finished manually by an expert.

CHAPTER 3
STUDY OF DNA SEQUENCE ASSEMBLY PROGRAMS

The work in this thesis focuses on the study and the evaluation of DNA sequence assembly programs. Many programs have been introduced and are currently under development. These programs provide different assembly qualities since they use different detailed algorithms. This chapter is dedicated to provide the reader with brief introduction and algorithms used in CAP3, ARCHENE, CELERA, TIGR, and PHRAP sequence assembly programs.

3.1 CAP3

CAP3 [18] is the most recent generation of the CAP sequence assembly program developed by Haung, et al. CAP3 has several notable features. First, the CAP3 can clip the 5' and the 3' low quality regions of read. Second, it uses quality base values created from the PHRED program to compute the overlaps between reads, align multiple read sequences, and generate a consensus sequence. Last, CAP3 uses forward-reverse constraints to correct assembly error and link contigs. In addition, CAP3 can correct some of false clonemate information. It should be noted that the incorrect clonemate information often occurs because of errors in cloning and DNA decoding whose error often happens in lane tracking process of electrophoresis detection.

3.1.1 Outline of CAP3's Assembly Algorithm

There are three phases in CAP3 program. In the first phase, the 5' and the 3' poor regions of each read are identified and removed. Then, the overlaps between reads are computed. In the second phase, reads are joined to form contigs in decreasing order of overlap scores. In the last phase, a multiple sequence alignment of

reads and consensus contig sequences are constructed by using a quality value of each base. Contigs are finally linked together by forward-reverse constraints.

3.1.2 Details of CAP3 Assembly Algorithm

Fast identification of pairs of reads with an overlap

This method begins finding a pair of reads with an overlap. An overlap means two symmetric overlap between two reads. Let f_1 ,f_2, … be the input reads in given orientation and let r_x be the reverse complement of read f_x. The two symmetric overlap between read f_x and f_y where x<y means that an overlap between read f_x and f_y is symmetric to the one between read r_x and r_y and an overlap between read r_x and f_y to the one between read f_x and r_y . An overlap alignment between two reads is simplified as an order chain of segment pairs, where each segment pair corresponds to an ungapped portion of sufficient length of the alignment. A largest–scoring chain of segment pair between two sequences is computed by the BLAST-like tool. A pair of reads has a potential overlap if the reads contain a chain of similarity score greater than a score cutoff. A special and efficient approach used to find potential overlap is to concatenate all reads with a special character inserted every read boundary into a combined sequence denoted as *F*. High-scoring chains of segment pairs between a read f_x and *F* are computed to find potential overlap pairs of read f_x and f_y and pairs of reads r_x and f_y where x<y. For each pair of reads with a potential overlap, a minimum band of diagonal in the dynamic programming matrix is determined to cover all the chains of score greater than the cut off between the reads. The band of diagonal for a pair of reads with a potential overlap is then used to efficiently compute an actual overlap.

Clipping of low-quality regions

Both the base quality values and the sequence similarity are used to compute the 5’ and the 3’ clipping position of reads as follows. A good region of reads is defined as sufficiently long region of high-quality region that has sufficiently high similarity with another high-quality region. The 3’ clipping position of a read is the maximum of 3’ end positions of good regions of the read. The 5’ clipping position of a read is the minimum of 5’ end positions of good regions of the read. The start and the end positions of good regions can be computed from a minimum band of diagonal matrix of an optimal local alignment for each pair of read with a potential overlap.

In the CAP3 program, the local alignment algorithm of Smith and Waterman [24] is generalized by using a set of base quality values. A base quality value, q, is defined as $-10\log_{10}(p)$, where p is the estimated probability of error that can occur in the base. This approach can be considered as a scoring scheme of matches, mismatches and gap penalties. Let a positive integer m be a match score factor, a negative integer n be a mismatch score factor, and let a positive integer g be a gap extension of penalty factor. A match at bases of quality values q_1 and q_2 is given a score of m*minimum(q_1 ,q_2). A mismatch at base of quality values q_1 and q_2 is given a score of n*minimum(q_1 ,q_2). A base of quality values q_1 in a gap is given an extension score of -g*minimum(q_1 ,q_2),where q_2 is the quality value of the base in the other sequence immediately before the gap and the quality value of the base immediately after the gap otherwise. The similarity score of an alignment is the sum of score of matches, mismatches, and gaps.

Computation and evaluation of overlaps

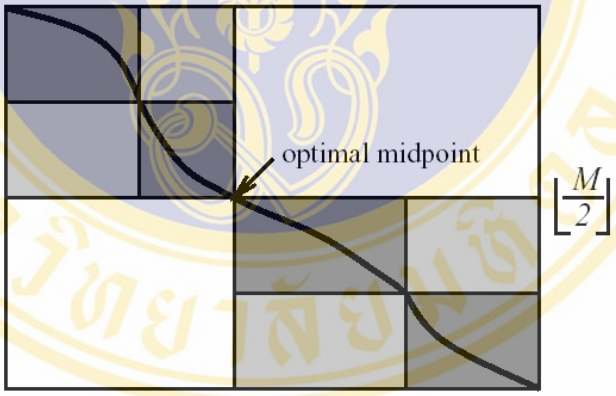

Fig. 3.1 Concept of band diagonals global alignment using a divide-conquer technique used in CAP3

There are two types of alignment. One is the local alignment that represents similar region and another is the global alignment that defines the maximum similarity score between reads. The local alignment was used to compute the 5'-3' clipping positions of reads and clean read that contains only high-quality regions were generated. In this step, the global alignment between the clean reads is used to determine the actual overlap. The global alignment with maximum similarity score is computed over the band of diagonals, centered at the start position of the optimal

local alignment. A divide-conquer technique is used to perform the band computation in linear space as conceptually illustrated in Fig. 3.1. The length, the similarity score, and the percent identity are defined to be the characteristic values of the alignment.

For overlapping evaluation, we use five measures: 1) The length, 2) the similarity score, 3) the percent identity, 4) the differences of the overlap at base of high quality values, and 5) the difference rate of overlap. If any of these measures is failed to pass the prescribed requirements, the overlap is not considered in the construction of contigs. The first three measures determine whether the overlap satisfies the minimum requirements on length, percent identity, and similarity score. The fourth measure examines the differences of the overlap at base of high quality values. If the overlap contains a sufficient number of differences at bases of high-quality values, then overlap is probably false. The fifth measure is examined with respect to the sequencing error rates of two regions involved in the overlap. The error rate of any region of a read is estimated using the error vector method [37].

Use of constraints in construction of contigs

There are four steps for using of constraints in construction of contigs. First, an initial layout of read is obtained by the greedy method that joins the overlapping reads in the decreasing order of overlap scores. Second, the quality of the current layout is assessed by checking constraints. Then, correction is made to correctable regions with the most unsatisfied constraints. Last, the second and the third steps are repeated until all constraints are used. The output contig linking information is then obtained.

Construction of alignments and consensus sequence

Multiple sequence alignment is used to finalize the construction of a contig. The reads are aligned in the increasing order of position in the contig. After global alignment of read, consensus sequence based on a quality base value is computed for the contig. After alignment, each position in a contig is called *column*. Each column consists of various base characters from all reads that are overlapped at the position. The quality value of a column alignment to use for determination of consensus base is calculated as followed.

Let the column consist of k nonblank characters c_i, $1 \le i \le k$. Each c_i is one of A,C,G,T,N and -, where N denotes any base other than the four base regular and - is a

gap symbol, Let q_i be the quality values of c_i. If c_i is a gap symbol, then q_i is quality value of the base immediately before c_i in the same row if such a base exists and quality value of the base immediately after c_i otherwise. Let $q_s(d)$ denote the average quality for substitution involving base type d of the read, let q_d denote the average quality value of deletion, and let q_n denote the average quality for insertion and d is a regular base. The quality values of each column are defined as

$$q_s(d) = \left[\left(\sum_{1 \le i \le k \text{ and } c_i = d} q_i \right) - \left(\sum_{1 \le i \le k \text{ and } c_i \ne d} q_i \right) \Big/ k \right] \qquad q_d = \left(\sum_{1 \le i \le k \text{ and } c_i \ne - } q_i \right) \Big/ k$$

$$q_n = \left(\sum_{1 \le i \le k} q_i \right) \Big/ k \qquad q_s(N) = -q_n$$

It should be noted that the quality values of the gap symbols in the column are excluded from the calculation of the average quality value for deletion. Any substitution involving base character N is considered as a mismatch. The score alignment of substitution, insertion and deletion are defined as follows. Consider a substitution involving the column of the block and a base d of the read with quality value q_r. If $q_s(d) > 0$ then the substitution is considered as a match and its score is m* min ($q_s(d)$, q_r) where the positive integer m is a match score factor. If $q_s(d) \le 0$, then the substitution is consider as mismatch and its score is n*min($-q_s(d)$, q_r) where n is a mismatch score factor.

The score of a gap is the gap open score plus the gap extension score of each element in the gap. Let a positive number g be gap extension penalty factor. The quality value q_d in deletion gap is $-g$*min(q_d, q_r) where q_r is the quality of a base of the read immediately before or after the gap. The extension score of a base of the read with quality value q_r in an insertion gap is $-g$*min(q_n, q_r), where q_n is the average insertion quality of a column of the block immediately before or after the gap.

Results

The CAP3 takes a file of sequence read in FASTA format, a file of quality values in FASTA format, and a file of forward-reverse constraint as input and produce output file in CAP format. For evaluation, CAP3 program were tested on the four data sets with default parameter values. CAP3 is demonstrated to correct data errors with supported constraint information.

3.2 ARACHNE

ARACHNE [19-20] is a whole-genome shotgun assembler. It assembles genome sequence using paired-end whole-genome shotgun reads. It is a large software system with relatively complex architecture.

3.2.1 Outline of ARACHNE's Assembly Algorithm

Input Data

This program analyzes read sequence, which are obtained from both ends of plasmid clones that are paired forward and reverse reads. The mean and the standard deviation of the insert length of each plasmid clone can be defined in the program. The program trims reads to eliminate terminal regions of extremely low quality and remove reads containing very little high-quality region. It also trims known vector sequences and eliminates known contaminants.

Overlap detection and alignment: Sort and extend method

This program uses a sort and an extend method to find overlap. The sort and the extend methods involve producing a sorted table of *k*-letter subword (*k*-mer) together with its source that is the read in which it occurs and its position is within the read. This program eliminates *k*-mers that occur with extremely high frequency because they normally correspond to high copy, high-fidelity repeated sequence in the genome. This program identifies all instances of read pairs, which share one or more overlapping *k*-mer, and uses three steps to align pairs of read. The first step merges overlapping shared *k*-mers, the second step extends the shared *k*-mers to alignment, and the last step refines alignment by dynamic programming.

Error and correction

This program detects and corrects sequence error by generating multiple alignments among overlapping reads. The error of base is identified and corrected based on overwhelming base occurrence and base quality. Repeated sequences are identified by occurrence of each alternative sequence with no overwhelming vote for one alternative. This program similarly corrects insertion or deletion of sequence error. As the reads are corrected, corresponding changes are made to the alignment.

Identification of paired pairs

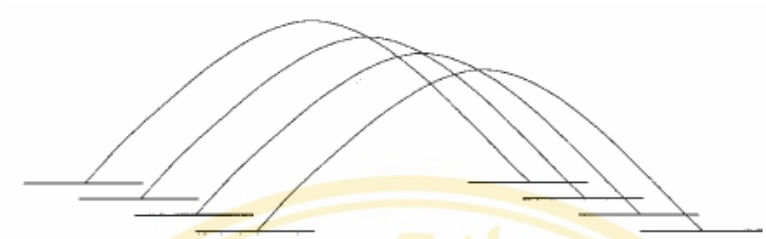

Fig. 3.2 Use of paired pairs to identify and merge true overlaps in ARACHNE

This program searches for instances of two plasmids of similar insert size with sequence overlaps occurring at both ends (reads of both clonemates overlap to each other at both ends). These instances are referred to as paired pairs. This program uses paired pair information with successive read overlaps to identify true overlaps and merge them together as conceptually illustrated in Fig. 3.2. In case of paired pairs from within a large repeat (larger than the insert size of plasmids), this program discards such an instance. The collection of paired pairs is merged together into contigs, and consensus sequence is formed. The resulting contigs are treated as large reads.

Contig assembly

In absence of repeats, construction of contigs is straightforward, but in case of repeats, a false overlap occurs from different copies of a repeat. This software identifies potential repeat boundaries and avoids assembling contigs across these boundaries. Some of boundaries that are merged safely contain reads called *dominated read* and subread that is fully included in the other reads called *containment read*.

Detection of repeat contigs

Repeat contigs are defined as contigs in which nearly identical sequence from distinct regions and they may collapse together. There are two ways to identify repeat contigs. First, repeat contigs have an abnormal coverage of contigs. This can be assessed by using the log-odd ratio that a contig of a given length and density of read represent a unique sequence versus being composed of read derived from two copies

of a repeat. Second, they have conflict link to the other contigs. It indicates that repeat contigs have conflict link to multiple distinct non-overlapping contigs.

Creation of supercontigs

There are two types of contigs. One is repeat contigs and the other is unique contigs. The program uses the forward-reverse links from plasmid reads to order and orient the unique contigs in longer layouts called *supercontigs* (*scaffolds*). In addition, it uses at least two forward-reverse links to merge contigs and avoid chimeric plasmids.

Filling gaps in supercontigs

There are two types of gap occurring in contigs layout. One is gap from repeat contigs and the other is gap from an insufficient number of shotgun reads. This program uses repeat contigs to fill gaps. It begins finding a path of pairwise overlap contigs and follows by the use of the forward-reverse link from supercontigs to find contig in a gap.

Consensus derivation and postconsensus merger

The layout of overlapping reads is converted to a consensus sequence with quality score. After forming consensus sequence, the program merges overlapping adjacent contigs into supercontigs and combines overlapping across spurious repeat boundaries.

3.2.2 Details of ARCHENE's assembly algorithm

Initial processing of reads

This program trims read using base quality values. This initial processing finds the longest contigs such that expect error in the sequence is less than 5%. Trimming was performed so that no base of quality less than 10 is within 12 bases of either end. After trimming, if less than 50 bases remain, the read is discarded.

The alignment module

This program contains four phases in alignment modules. First, the program accepts input sequence to identify all instances of the relevant *k*-mer. Each instance of the *k*-mer record consists of four entries the *k*-mer, the sequence number, the position of the *k*-mer in the sequence, and the reverse complement flag. This flag tells us that it is *k*-mer sequence or complement sequence. The records from phase 1 are placed in

vector and entered into phase 2. In the second phase, the program screens a number of contigs and combines the inherent redundant reads into the longer, imperfect, but gap-free partial alignment. The program maintains a list of extend *k*-mer list.

In the third phase, full alignment lists are created from the extended *k*-mer list. For each pass of sequence, it forms a directed graph whose vertices are extend *k*-mer hits associated to the sequence pair and whose edges correspond to pairs of extend k-mer hits that can likely be combined into single, good partial alignment. In the last phase, each full alignment is refined via dynamic programming under the assumption that the new alignment is closed to old one. Base quality is used to assign a number of mismatch, insertion, and deletion. The number of all types are summed and divided by $(\text{overlap}/100)^{1.5}$. The alignment of read whose penalty score exceeds 100 is discarded.

Detection of chimeric read

The read that contains genomic sequence from two disparate locations are termed *chimeric*. A chimeric read *c* can be found under the conditions that read *c* have a good transitive overlap to read *z*, but no good overlap between read *c* and *z* are found. In addition to chimeric read, there are *dead-end* reads that have no good overlap beyond its left or right but with some good transitive overlap significantly beyond this end. This read occurs from concatenating one long read and one short read. Chimeric and dead-end read are considered low quality and are discarded.

Contig assembly

In contig construction, the repeat potential boundaries and the dominated read are identified. The repeat boundaries come from missed overlapping read. The dominated read occurs in read that is maximal with the set of its extended reads and a superset of others extended reads in the contig. In the first round, read merging are performed up to repeat boundaries. A second round of merging is then performed after repeat boundaries are recomputed with the dominated read constraint.

Detecting repeat(Super) contigs

There are two methods of detecting repeat. First, a repeat contig is found from log-odds ratio with given density of reads. The contig with log-odd ratio of less than 1 is considered a repeat. Second, a repeat is determined from the consistency of

forward-reverse link. The relation between contig *A* and *B* has two parameters, One is the mean distance denoted by $d(A,B)$, the other is the standard deviation of distance denoted by $Err(A,B)$, There are two conditions for detecting repeat contig.

1) If $d(A,B) < -2000^{-4} * Err(A,B)$ then the contigs are marked repeat.
2) The contig *A* is linked to contig *B* and *C*. If the overlap length between contig *B* and contig *C*, $d(B,C) < -2000^{-4}$, then contig *A* is marked as a repeat.

Supercontig assembly

Super contigs are built incrementally as illustrated in Fig. 3.3. In this process, the following steps are performed.

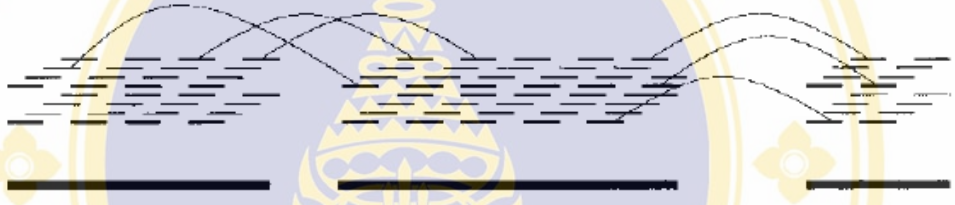

Fig. 3.3 Concept of super contig assembly used in ARACHNE

1) Extract from *Q* the pair of super contigs, S_1 and S_2, with the highest priority score.
2) Merge S_1 and S_2 to create supercontig *T*.
3) Remove all pairs in *Q* where one of the two supercontigs is either S_1 or S_2.
4) Find all supercontigs *W* that shared forward-reverse links to *T*, and insert (*T*,*W*) into *Q*.
5) Apply rules 1 and 2 for marking repetitive supercontigs, *T* or any supercontig *W* linked to *T*. For any supercontig marked as repetitive, remove all pairs in Q containing it.

Filling gaps in supercontigs

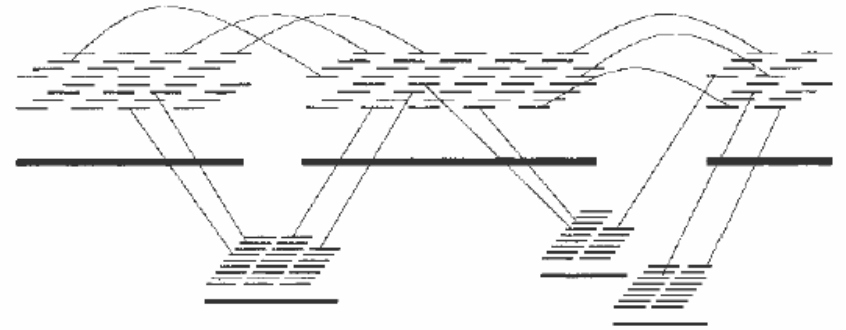

Fig. 3.4 Concept of gap filling used in ARACHNE

For every pair of contigs, *A* and *B*, that are consecutive in a supercontig *S*, the gap between *A* and *B* can be filled as conceptually illustrated in Fig. 3.4 under the following conditions:

1) If (*A*, *B*) is in E_{PATHS}, a path that has already been computed for *A* and *B*, then simply fill the gap with the contigs in the path.
2) Otherwise, find every contig *T* that shares forward-reverse links with *W* such that the links *T* is positioned between *A* and *B* in *W*. The set of all such contigs is called the set of targets, V_T. G_{PATHS} of a path from *A* to *B* that uses only nodes in V_T is created by breadth-first search. If a path is found in G_{PATHS}, which corresponds to a path *p* in *G* from *A* to *B*, then the ordered contigs in *p* is used to fill the gap between *A* and *B*.

Consensus derivation

The consensus derivation algorithm consists of three steps, generating multiple alignment from a list of oriented reads, approximate placement of reads measured from the beginning of the contig, and the preexisting pairwise alignment between reads. The merge starts from an initial read that has no bases to its left and the merge continues base by base to the right with the consistent alignment. The process continues until there are no base to merge. Normally, all reads in the contig are used, yielding single consensus sequence for the contig.

Results

The ARACHNE program was tested with simulated data. There are two steps to simulate data. First, data are selected from known genome at random location. The data compose of forward-reverse reads. Second, the read from previous step was assigned realistic quality score and error by pairing it with a real read from BAC. However, the data still fall short of being completely realistic in several parts. First, it lacks of potential cloning. Second, they don't have symmetrically poor sequence. Last, some of region, which is difficult to clone, are not found in these sequences.

Using simulated data from *H. influenzae* , *S. cerevisiae* , *C. elegans*, *D. melanogaster* and from the 21st and 22nd human chromosome, the result of evaluation can be summarized as follows:

- The contigs coverage is 97%-98% at full coverage, the read usage is 92%, and the maximum of multiple usage is 1%.
- In short contigs, the N50 contig length L such that 50% of the bases are in contigs of size *L* or greater is of interest. The N50 contig length is ~ 350 kb for full coverage and 17 kb for half coverage.
- For full coverage, the overall nucleotide accuracy is better than 99%.
- For full coverage, almost error comes from deletion and insertion.
- This program requires 8.4 GB of RAM and take 21 hour of runtime on single processor (667 MHz Compac Alpha).

3.3 CELERA

CELERA [21] is a sequence assembly program with some interesting features. It calculates the DNA sequence characteristics and uses double-barreled sequencing methods to maximize the assembly qualities.

3.3.1 Outline of CELERA's shotgun DNA sequencing process

CELERA employs two main concepts for DNA sequencing

DNA sequence characteristics

In DNA sequence, a number of repeats can occur. The characteristics of repeats are length, copy number, and fidelity between copies. The fragmentation of the solution into a collection of gap-separated contigs will increase at least linearly with the source size for a fixed level of sequencing coverage.

Double-barreled shotgun sequencing

In DNA sequencing, the average size of the insert (DNA fragment that is inserted into vector before cloning) is 2*L* or longer, where *L* is the average length of a read. Both ends of decoded insert is called *pairmate*, which are always in opposite orientation. The distance between pairmate is equal to the length of insert, which is

approximately known for the type vector used. The concept of this scheme can be illustrated in Fig. 3.5.

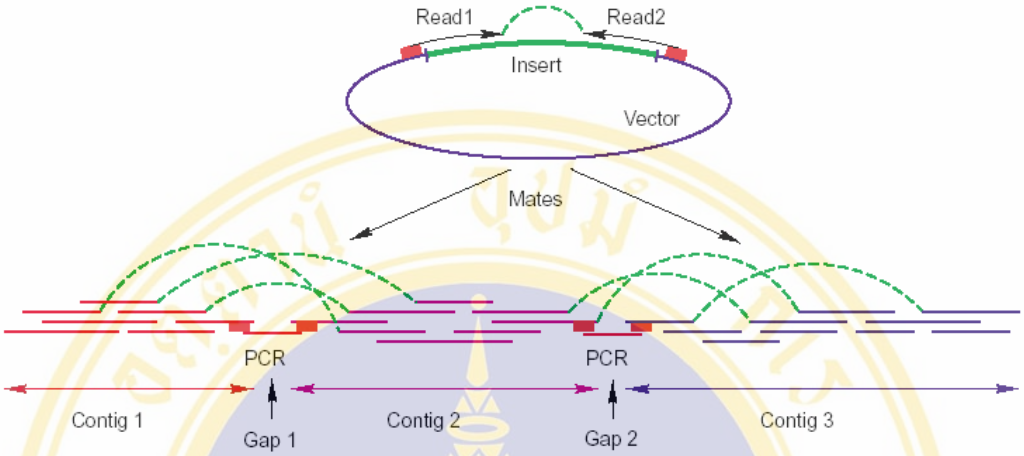

Fig. 3.5 Concept of double-barreled shotgun sequencing used in CELERA

In principle, this information can be used to link contigs. However, this information may sometimes not be used if there are sampling error before entering into vector. A little of the benefits of having long end-sequenced insert is lost in hybrid schemes where a sizable fraction of a read library contains a single read and where the paired reads from inserts over the distribution of insert length skews toward the shorter length.

3.3.2 Details of CELERA's shotgun DNA sequencing process

CELERA combines three main approaches, the clone by clone, the sequence-tagged connector, and the whole-genome shotgun approaches.

The clone by clone approach

The clone-by-clone approach consists of two steps. First, BAC inserts are built from breaking whole genome sequence. Last, shotgun sequencing is then used to sequence insert of BACs, which cover the genome. BACs are combined together with overlaps between BACs based on finger print data about each insert. This approach often fails to complete the whole genome assembly due to high cost, less-efficient process of building BAC map.

The sequence-tagged connector approach

This approach involves sequencing of the ends of BACs. In contrast to clone-by-clone approach, BAC is produced by partial digestion with restriction enzyme so that ends of BACs are not random and most of BACs are not sequenced. A few of BACs are selected as seeds of an ordered clone by clone walk and each BAC clone is shotgun-sequenced. Shotgun sequence of two overlapping BACs was performed in each direction to determine next BACs in each direction to sequence. This method is based on clone-by-clone walks across genome as each clone in each walk is sequenced. Complexity of multiple BACs still made this method difficult to succeed.

The whole-genome shotgun approach

In current practice of shotgun sequencing, 6X to 7X coverage of sample read was used to shotgun BAC sequence. A number of reads is needed because the end of read at 10-15% are too noisy to detect overlap. In Celera, only the high-quality part of read and 10X coverage are used to compensate for a shorter length. In the given genome's repetitive nature, clonemate information of insert is needed to assembly sequence, particularly solving any repeats whose length is shorter than distance between mate pair. If the repeat are longer, a series of inter-marker that requires assembling the sequence between a pair of STS marker or BAC end sequence will be used to solve this assembly problem.

The fragment assembly problem

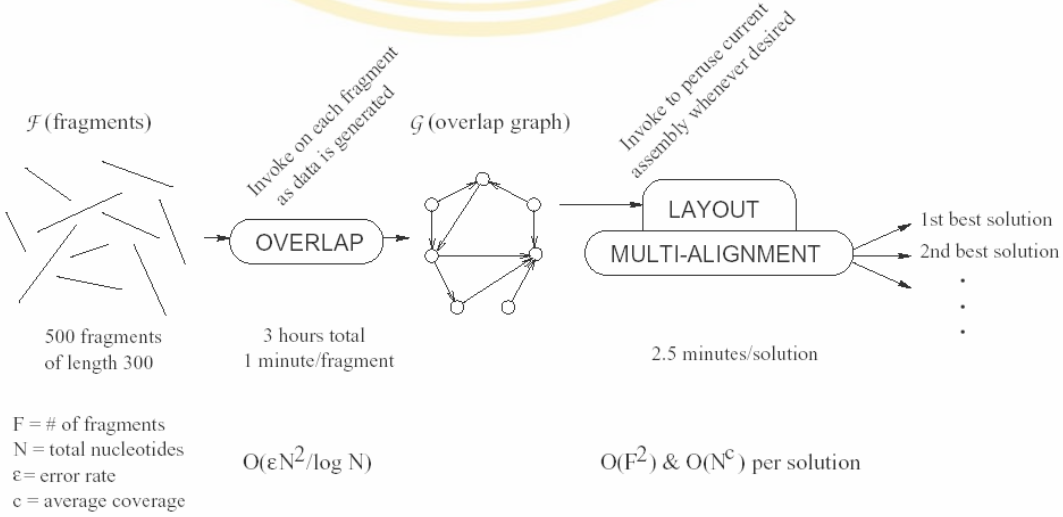

Fig. 3.6 Concept of fragment assembly algorithms used in CELERA

The main concept of Celera's fragment assembly algorithms can be illustrated in Fig. 3.6. It is a graph based algorithm with three-phase of assembly. The following are definition and details of Celera's method.

Definitions

G : Length of target sequence, $\overline{L}$: Average length of sequence read,

R : Number of sequencing reads in shotgun data set,

N : $R\overline{L}$,total number of base pair sequence

$\bar{l}$: Average length of a clone set, $\bar{c}$: N/G Average sequence coverage

$\overline{m}$: $R\bar{l}$ /2G average clone or map coverage

The important characteristics of data are as follows.

Incomplete coverage: These are due to both stochastic natures of sampling and cloning. This cause gap appeared in the reconstructed sequence.

Sequence error: From gel electrophoresis, there are errors at the end of reads. There are errors less than 1% in the first 500 bases. If the number of base increases, error will increase 10% to 15%.

Unknown orientation: DNA is a double-strand helix. In sequence reconstruction, strands are sequenced from orientation of insert.

The collection of input reads is defined as $F=\{f_i\}_{i=1}^{R}$. The assembly layout, ε, is the string of {A,C,G,T} and collection , R, contains the pair of integer (s_i , e_i) $\in[1,R]$ such that

-If $s_i < e_i$ then f_i can be aligned to the substring $S[s_i, e_i]$ with $< \varepsilon|f_i|$ difference.

-If $s_i > e_i$ then f_i can be aligned to the substring $S[e_i, s_i]$ with $< \varepsilon|f_i|$ difference.

The string S represents the reconstruction of the source strands and the integer graph indicates the substring of S that gives rise to each read. The order of s_i and e_i encodes the orientation of the fragment read in the layout whether f_i was sampled from S or its complement strand. The parameter $\varepsilon \in[0,1]$ models the maximum error rate of the sequencing process. The sequencing is divided in three phases, overlap, layout, and consensus.

In the overlap phase, each read is compared to every other reads. Given sequencing data error, overlap is accepted if overlapping difference between the pair is within a bound value of ε. For high comparing speed, method of finding exact common substring of some length k based on a hashing scheme is used for overlap detection. In the overlap process, an overlap graph, in which every vertex models a read and every edge of an overlap between both reads, is produced.

In the layout phase, pairs (s_i, e_i) of positions for every fragment are found. There are many algorithms for solving the layout problem including greedy algorithm with optimal factor, simulated annealing genetic algorithm, relaxation method based on generation either spanning forest or weight matching in order of score, chordal graph collapsing with a reduction to speed Eulerian tour, and greedy algorithm with quality base value in the case of repeated segments.

In consensus phase, the consensus base at each position in the reconstructed sequence whose coverage is two or greater are determined. The result can be obtained from the layout using methods including window sweep optimization, Hidden Markov model, gradient-descent algorithm, and round robin realignment. If the sampling is perfectly uniform, expected statistics of the results should be:

- $1\text{-}e^{-C}$ of the source strand covered by some read.
- Fe^{-C} in the coverage of the source.
- Gap free sequences or contigs of the average length (L/C)

The percentage of the genome covered depend on C and the maximum of gap is at C=1 and is declined at C>1.

3.4 TIGR

TIGR [22] is an advanced sequence assembly program with special features.

3.4.1 Outline of TIGR assembler's algorithms

The algorithms of TIGR assembler are as follows:

1) Perform pairwise fragment comparisons for the entire data set to generate a list of potential fragment overlap. Use the distribution of number of potential overlap for each fragment to label fragments as repeats or non-repeats.

2) Start with a non-repeat fragment as the initial assembly seed or a repeat fragment if no non-repeat fragment is left and quit if no fragments remain.

3) Use potential overlap list to attempt merge between the current assembly and non repeat fragments when no potential overlaps with non repeat fragments remain for the match criteria and enforce clone length constraints when attempting to merge with a repeat fragments.

4) If due to a merge with a repeat fragment, a non-repeat fragment is added to the potential overlap list go to step 3.

5) When there is no fragment left on current potential overlap list, output information about the current assembly and go to step 2.

3.4.2 Details of TIGR assembler's algorithms

Pairwise Comparison

TIGR Assembler locates all *n*-mer oligonucleotide shared between fragment pair in order to find a potential overlap. The Smith-Waterman dynamic programming algorithm is then used to find potentially correct overlap.

Merging Fragment with Assemblies

A modified Smith-Waterman is used to evaluate a current assembly with a fragment. There are four criteria which is added in Smith-Waterman; minimum length of overlap, minimum similarity in the overlap region as a percent of the best possible score, a maximum length of overhang, a maximum number of local error. TIGR use criteria according to type of data. TIGR will increase criteria if fragment is repeat and does not meet clonemate constraint. On the other hand, TIGR will decrease these criteria if the fragment is not repeat and meet clone constraint. Tandem repeat are repeat region which are nearly identical and contiguous where there can be two or more copies. Tandem repeat requires sophisticated ways to be assembled correctly.

Building a Consensus Sequence

The consensus sequence is built by merge a fragment at a time to existing consensus sequence TIGR generate profile for alignment and consensus sequence. There are many rule for building consensus sequence in the following.

I) If the largest profile component is greater than two-thirds of the totals of the component output an upper case A C G or T or nothing in the case of gap.

II) If there are two non-gap components that are prevalent output a lower case two base ambiguity code (m ,r ,w ,s ,y or k)

III) If the prevalent component is greater than half of the total output a lower case a ,c ,g or t or nothing in the case of a gap

IV) Otherwise output a lower case n. In addition if a gap is the prevalent component but less than two-thirds of total then make the next symbol output be in lower case. This rule highlights any ambiguities by outputting lower case.

Repeat Region

Repeat regions cause misassembly of target. A repeat region is a contiguous piece of DNA that has a very high similarity to another piece of DNA in the target. Repeat region can have false overlap with fragment from another region. True overlap is the region share between two fragments, which span a contiguous piece of target DNA. A false overlap is detected when two non-contiguous fragments contain very similar region. The fragment from repeat region is identified by the number of potential overlap each fragment has based on pairwise comparison. TIGR Assembler determines the median number of potential overlap and any fragment with more than r x median potential overlap (r~1.4-1.6) is labeled as being from repeat. There are two approaches in assembly repeat region. One is increasing the stringency of the match criteria for repeat region and the other is strictly enforce clone length constraint for repeat fragment is detected by the differing sequence surrounding the repeat region. For longer identical repeat region, TIGR Assembler uses clone length constraints. For the very long repeat, repeat is solved by designing primer to walk through the repeat containing clone.

Concurrent Assemblies

TIGR can construct multiple assemblies simultaneously and use clone information to effectively span and correct repeat. Beside solving repeat region, clone information is used to solve weak joins and chimeras.

Optimal Clone Length

Two types of clone are used to construct assembly. One is a large number of shorter clones to minimize break in clonemate and the other is a much smaller number

of larger clone to span larger repeat region. The choice of optimal clone size is dependent upon the total size of the target DNA and the size of typical repeats.

3.5 PHRAP

PHRAP [23] is an advanced sequence assembly program with some special features.

3.5.1 Outline of PHRAP assembly

1) Read in sequence & quality data, trim off any near-homopolymer runs at ends of reads, and construct read complements.

2) Find pairs of reads with matching words. Eliminate exact duplicate reads. Do swat comparisons of pairs of reads that have matching words, and compute (complexity-adjusted) swat score.

3) Find probable vector matches and mark so they aren't used in assembly. Locate near duplicate reads, reads with self-matches, and matching read pairs that are "node-rejected" i.e. do not have "solid" matching segments.

4) Use pairwise matches to identify confirmed parts of reads; use these to compute revised quality values.

5) Compute LLR scores for each match (based on qualities of discrepant and matching bases). [Iterate above two steps]

6) Find best alignment for each matching pair of reads that have more than one significant alignment in a given region (highest LLR-scores among several overlapping).

7) Identify and remove probable chimeric and deletion reads.

8) Construct contig layouts, using consistent pairwise matches in decreasing score order (greedy algorithm). Consistency of layout is checked at pairwise comparison level.

9) Construct contig sequence as a mosaic of the highest quality parts of the reads.

10) Align reads to contig; tabulate inconsistencies (read/contig discrepancies) & possible sites of misassembly. Adjust LLR-scores of contig sequence.

3.5.2 Details of PHRAP assembler's algorithms

Phrap adjusted quality values/error probabilities and computes adjusted quality values for each read on the basis of read-read confirmation information, as follows. If a read is confirmed by an opposite-strand or different-chemistry (dye terminator vs. dye primer) read at a given position, that position is given a quality which is the sum of the two input read qualities; when more than one opposite-strand read confirms a given position, only the single highest quality from all opposite-strand matching reads is used. If qualities are related to error probabilities as described above, this procedure can be interpreted as computing an error probability for each base which is the product of the error probabilities for the two reads; since error profiles for opposite-strand or different chemistry reads are essentially independent, this is reasonable, although it is somewhat conservative in that it does not take into account same-strand matches.

Contig positions are assigned quality values equal to the highest adjusted quality of any read at that position, and then adjusted downward to take into account any discrepancies with other reads. As a result, when phred input qualities are used, the output phrap qualities associated to each contig position has a natural interpretation as (conservative) error probabilities. Such error probabilities provide an extremely useful guide to where editing or additional data collection is needed.

The phrap-adjusted qualities are used in computing "LLR scores" for each pairwise match between two reads. These scores take into account the qualities of the base calls in the reads, and are (approximate) log-likelihood ratios for comparing the hypothesis that the reads truly overlap to the hypothesis that they are from 95% similar repeats. The point is that discrepancies between overlapping reads are due to base-calling errors and thus tend to occur in low-quality bases, whereas reads from different repeats can have high-quality discrepancies that are due to sequence differences between the repeats; and the probability of the observed data under each hypothesis can be quantified using the interpretation of the phrap qualities in terms of error probabilities. A pairwise match tends to have a positive LLR score if the two reads overlap, whereas it tends to have a negative LLR score if the two reads are from different repeats (unless the repeats are nearly identical). Following are details in main steps.

0) The procedure used in both phrap and cross-match to find sequence matches is the following. First, any region at the beginning or end of a read that consists almost entirely of a single letter is converted to 'N's; such regions are highly likely to be of poor data quality which if not masked can lead to spurious matches. Reads are then converted to uppercase (in order to allow case-insensitive word matches). All matching words of length at least minmatch between any pair of sequences are then found, by

- Constructing a list of pointers to each position (in each sequence) that begins a word of at least minmatch letters not containing 'N' or 'X' and quicksorting the list.
- Scanning the sorted list to find pairs of matching words. For each such pair, a band of a specified width, centered on the diagonal defined by the matching words, is defined and overlapping bands are merged. Following construction and merging of bands, a "recursive" SWAT search of each band is used to find matching segments with score greater than or equal to minscore: "recursive" here means that if such a match is found, the corresponding aligned segments in each sequence are (conceptually) X'd out and the process repeated on the remaining portions of each sequence. This procedure allows detection of multiple matching pieces in different locations, and will usually find most copies of repeats.

1) In phrap, sequence pairs with score ≥ minscore are considered for possible merging. A critical issue here is the appropriate score matrix for SWAT. Setting the indel penalties slightly higher than the mismatch penalties gives better alignments by favoring mismatches in compression regions. Since SWAT uses profiles, one has the option of distinguishing different quality levels by use of different symbols (e.g. upper case for more accurate calls, lower case for less accurate calls) and setting the penalties appropriately; this would involve using the quality levels to adjust the symbols used in the reads, which should be done AFTER the word matching routine.

- Determine "confirmed" part of each read (i.e. the part which appears in a SWAT alignment against some other read; a read with the same name -- up to an internal '.' if any -- is not considered confirming). Probable chimeras are

detected as reads for which the confirmed part can be separated into two non-overlapping pieces, separated by at most *MAX_CHIMERA_GAP* bp such that the part confirmed by a given read lies in one or the other piece but not both; AND such that each piece has a prematurely terminating alignment with some other read. Chimeric reads may arise from i) chimeric clones; ii) gel mistracking across lanes; iii) unremoved sequencing vector; (iv) deletion clones. Reads having two non-overlapping confirmed pieces are often non-chimeras that are the only link between two non-overlapping contigs, and should be permitted in the assembly.

- Identify "strongly confirmed" regions in each read (having matches to a reverse sense read), deletions, and "rejected" alignments -- those that don't extend as far as they should, or that have mismatches involving high quality bases.

2) Sort all matching pairs by decreasing LLR score, and assemble layout by progressively merging pairs with high score. Consistency of merge is required: the SWAT alignment implies relative offset of one contig with respect to another, and hence a potential implied overlap. The SWAT alignment should extend over essentially the entire region of implied overlap, apart from an allowable gap. Probable deletion clones (reads that have two adjacent pieces matching two separated pieces of another read, and having no confirming read across the breakpoint), and chimeras are not used in any merges.

3) Construct contig "consensus" as a mosaic of individual reads. Strategy: ends of alignments, and midpoints of perfectly matching segments of sufficient length, define crosslinks - pursue crosslinks, which increase accuracy and extend read. Formally this is done by constructing a weighted directed graph whose nodes consist of (selected) positions in reads; there are bidirectional edges with weight 0 between aligned bases in overlapping reads, and unidirectional edges from 5' to 3' positions within a single read, with weight equal to the total quality of the sequence between the two nodes. Standard C.S. algorithms (dating back to Tarjan) permit identification of a path with maximal weight, in time linear w.r.t. number of nodes. The quality values for the resulting sequence are inherited from the read segments of which it is composed.

3.6 Summary and comparison of sequence assembly programs

The summary of algorithms in each shotgun step and comparisons of these programs are listed in Table 3.1. It can be seen that these programs except CELERA share some basic algorithms, including the use of dynamic programming, base quality scheme, greedy algorithm for multiple alignment, and forward-reverse constraints for contig alignment. However, these programs have different contig assembly and repeat detection scheme, which are advanced parts of the programs. While CELERA is significantly different from other program because the whole program is graph-based algorithm. ARACHNE and CELERA are designed in a large computer system that can handle large DNA data and cannot be used on a person computer system due to memory space limitation. From the various, reports [2, 18-20, 40], CAP3 has shown both time and space efficiency with the best assembly quality, while PHRAP is relatively inferior to other assembly programs.

Table 3.1 Summary and comparison of algorithms of DNA sequence assembly programs

Methodologies	CAP3	ARACHNE	CELERA	TIGR	PHRAP
Overlap Computation	BLAST and dynamic programming with base quality value	Sorting and extending method with Paired pair constraints	Eulerian graph method	N-mer oligonucleo-tide share between fragment pairs	String matching with SWAT type base quality value
Multiple Alignment	Greedy tree-graph algorithm with base quality constraint	Greedy tree - graph algorithm with base quality constraint	Genetic algorithm with optimal factor, simulated annealing and relaxation method	Greedy tree and graph algorithm with base quality constraint	Greedy tree and graph algorithm with SWAT base quality constraint
Contig Assembly	Linking contigs with forward-reverse clone-mate constraints	Detection of repeat and construction of Super-contig	Window sweep optimization, Hidden Markov model gradient–descent or round robin realignment	Repeat detection & correction by number of copy and clone-mate information	Tabulate read/contig discrepancies & possible misassembly
Performance [2, 18-20]	Higher than all other programs: low errors and good percent coverage	High but lower than CAP3: low errors and fair percent coverage	High but lower than Archene: short but high quality contigs	Better than Phrap: short but good quality contigs	Lower than CAP3, TIGR, and Celera: long poor quality contigs
Advantages [40]	-Time&space efficient algorithms -High assembly quality	-Time efficient algorithm -High assembly quality -Large DNA assembly	-Time efficient algorithm - Good assembly quality -Large DNA assembly	-Space efficient algorithms -High assembly quality	-Space & time efficient algorithm
Disadvantages [40]	-Perform poorly for some low-quality data	-Produce low percent coverage -Require a large computer system	-Produce many short contigs -Require a large computer system.	-Produce too many short contigs -Relatively small DNA assembly	-Produce few long but poor quality contigs

CHAPTER IV

SEQASS PROGRAM AND MAIN MODULES

4.1 Outline of SEQASS program

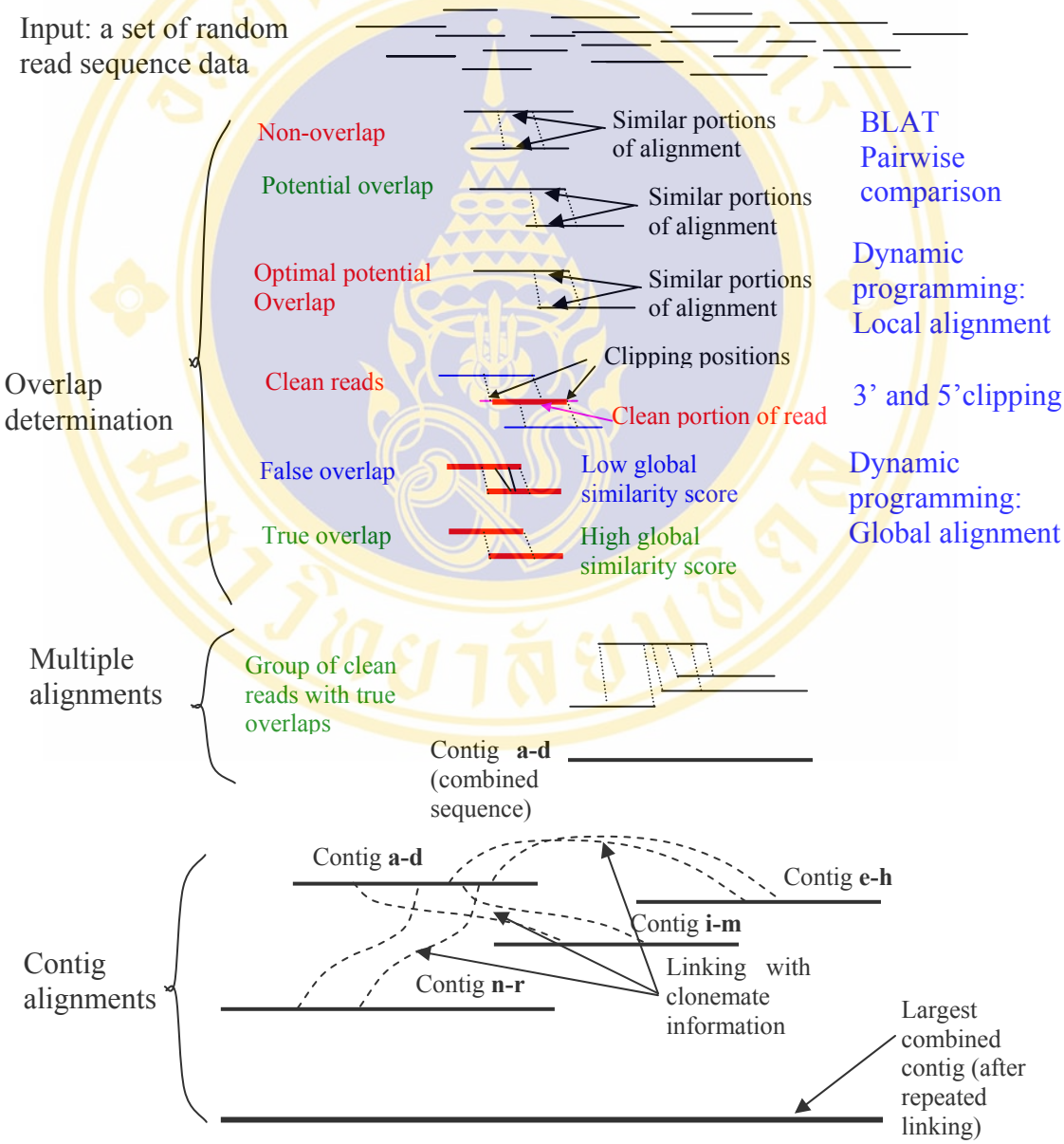

Fig. 4.1 Outline of SEQASS program

In this research, we have developed a new DNA sequence assembly program, called *SEQASS*. It follows the framework presented in CAP3 [18], which has shown high assembly quality, moderate complexity and straightforward implementation. Unlike CAP3 that uses *BLAST* [28-32], SEQASS uses *BLAT* [33] to accelerate pairwise comparison and improve assembly quality during the local alignment. Fig. 4.1 outlines the three main processes of DNA sequencing in SEQASS: Overlap determination, multiple alignment, and contig alignment.

In SEQASS, the overlap determination process can be briefly explained as follows. First, BLAT is used to quickly determine local alignments and produce a set of potential overlaps. Second, overlapping length constraints are applied to filter overlappin pairwise reads. Third, base quality score and dynamic programming technique are used find optimal local alignment whose positions are utilized to perform clipping read-ends with poor base quality. Finally, other constraints including aligned position, minimum identity, and maximum similarity score of dynamic programming are employed to identify if the clipped reads and associated global alignment are actually correct overlaps.

The multiple alignments apply greedy tree and graph algorithms to construct contigs. The simultaneous alignments of multiple overlaps are achieved with position, mismatch, and indel statistical tracking methods. The consensus alignment is executed based on CAP3's quality base scoring scheme and maximum occurrence of based character found at mismatches and indels. In the contig alignments, contigs are linked together by tracking clonemate information. Clonemate information will be automatically detected from the data description.

4.2 DNA data files

To assemble DNA sequences, our SEAASS program takes into account a set of reads and a set of base quality values. They can be obtained from Genbank genome databases, which are freely available at www.ncbi.nlm.nih.gov.

```
>gnl|ti|127084480 name:avt18h08.y1 accession:AC110665 mate:127084448
mate_name:avt18h08.x1 template:avt18h08 end:F
TTCAGAGCTGAGCTCCATGTGGTGGATTCCTTATCGGGTTCTTTCCCCCTCAGAGCCCATCTCTCCCTGG
TCACCATTTCAAGAAGAGCTGATTAAAGCATAGCAATAGAGCTCAATGTTCCAGGAAACTCCCCAGGGCC
TGGCCAGGAGCCTTAAGATAGGAGGTACCTACAACTTGAGTCCAAATCCTTGCATTATCAGATACTTCCA
GCCAGGGTGAGGACCAGCTAGTGACACGCACAAATTTTACTGACCACAAGCTACACCCAGCTCTGGAATA
ACAGCCTCCTATCCCCATCCCATGGTTATTCTAGAGGAAGCAAAGAGGAGGGGCATCTTAACTTCAAAGG
TGAACCTATGTTAACATTGAGAAGAGGCTAGGTGTTTCATTGGTGGTGTTTTCCTTTGAATGTAAGTGCA
TGAAAGTCAATCCCTGTAAACTTGAGGTGAGGTATTGCACAAACTGCTTGTTGTTCTGGTTGATCTCTCC
AGCCCTTCTTTTTTGGCTAGATTCAAGCTCCCCCATTTCACTGGACCACCCGTAGTGCTTGAGTGATCGT
CTAGTCTTCAGTCTCACCTTGCCTGCACTGTCAAAGTACTCTCCCTAAAAACTGTAATTCAACTGGGATT
TCCCGTAACCTTCATTTGTTCCTGATCACCCTCAGGATATAGTCAAAGTTCTTCACCGTTACCTACCCAN
AGTCTT
```

(a)

```
>gnl|ti|127084480 name:avt18h08.y1
 7 11 13 12  9  9 10 17 17 26 23 23 18 24 19 15 12 24 24 40 40 51 51 23 23 10 10 10 24 24 24 18 10 10 10  9 12 14 27 32
40 40 44 44 46 46 46 46 35 35 35 35 35 39 56 40 39 35 35 35 35 56 51 56 56 56 51 51 56 40 40 40 40 40 40 40 40 40 40
40 40 40 40 56 56 56 56 56 56 56 51 51 51 51 51 51 56 40 40 40 40 40 40 40 40 40 40 51 51 51 51 40 40 40 40 40 40 45
45 40 40 40 39 39 39 40 40 40 45 45 45 45 45 51 51 40 40 51 51 51 51 56 51 51 51 39 39 39 39 39 39 51 56 56 56 40 40
40 40 40 40 51 51 51 51 45 51 51 51 51 51 56 56 56 56 56 45 43 43 43 43 43 51 51 51 51 51 51 51 51 56 56 56 56 56 56
56 56 56 56 43 43 43 43 43 43 56 56 56 56 56 56 56 56 56 56 56 45 43 43 43 43 43 43 43 43 56 56 56 56 56 56 43 43 43
43 43 43 56 51 51 51 51 51 51 56 56 56 56 56 56 56 56 56 51 51 51 51 51 51 51 51 51 56 56 56 56 56 56 56 51 51 51 51
56 51 51 51 51 51 51 56 56 56 56 56 56 51 45 45 45 45 45 51 56 56 56 56 56 51 51 51 51 51 51 56 56 56 56 56 56 56 56
56 56 56 56 56 56 56 56 56 56 56 56 56 56 51 51 51 51 51 51 56 56 56 56 56 56 56 56 56 56 43 43 43 43 43 43 56 56 56
51 51 45 45 45 45 51 51 51 51 51 51 51 51 51 51 51 56 56 56 56 56 56 51 51 51 51 45 45 51 51 51 56 56 56 56 56 56 51
56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56 56
56 56 56 56 56 56 56 51 51 51 51 45 45 45 45 45 45 45 51 56 56 51 51 51 51 51 51 51 51 51 51 51 56 56 56 56 56 56 51
51 51 51 51 51 56 56 56 51 51 51 51 51 51 56 51 51 51 45 45 45 51 45 45 56 51 51 51 56 56 56 56 56 56 56 56 56 56 56
46 46 42 37 37 40 40 40 40 56 56 56 51 45 45 45 45 45 40 40 37 37 37 40 45 45 45 45 51 51 51 45 46 51 51 51 51 56 56
56 56 56 56 56 56 56 56 56 56 56 56 51 51 51 51 51 51 56 56 51 51 46 40 40 46 48 46 40 40 32 32 32 32 29 25 29 48 48 40
40 40 40 32 32 29 28 29 34 40 40 40 40 40 40 40 40 40 40 34 29 29 34 40 40 40 44 40 40 40 46 46 44 40 40 40 40 48 29
22 27 25 27 27 27 34 32 32 27 27 29 24 18 20 25 27 32 32 40 48 48 29 29 29 29 32 32 32 32 25 25 22 29 37 27 27 23
18 18 20 24 24 27 19 27 27 29 25 25 29 29 25 25 29 20 25 25 25 18 22 19 12 12 19 15 22 25 29 29 18 27  4  0 4 21 21 21
15 15
```

(b)

Fig. 4.2 (a) A snapshot of sample read sequence of *Canis familiaris* and (b) the corresponding quality score

The set of reads is contained in a FASTA file and the set of based quality value is contained in a qscore file. To evaluate our SEQASS program, an assembled whole genome is required and used to compare to the output of the program. Fig. 4.2 shows a snapshot of each data file for *Canis familiaris*. The file consists of two main parts: the description line and data value lines. The description line always begins with '>' character. The description normally consists of following information: sequence name (e.g. gnl|ti|127084480), sequence code (e.g. avt18h08.y1), template (clone) name, and end type (forward or reverse). The remainder of the description line comprises additional information such as the data base name (accession number) and clonemate name/code. The sequence name for each is the standard Genbank

nomenclature while sequence code is derived from template name with extension implying forward or reverse end. The accession number is the reference number for the read data set. Clonemate name/code are useful for contig alignment. It should be noted that clonemate name/code is optional and may not exist in some read data. The data value line only consists of one of 5 characters, 'A', 'C', 'G', 'T', and 'N' where 'N' is the unidentified base of DNA. It should also be noted that each FASTA file from Genbank contains only a single read per file with ".fasta" extension.

4.3 SEQASS main modules

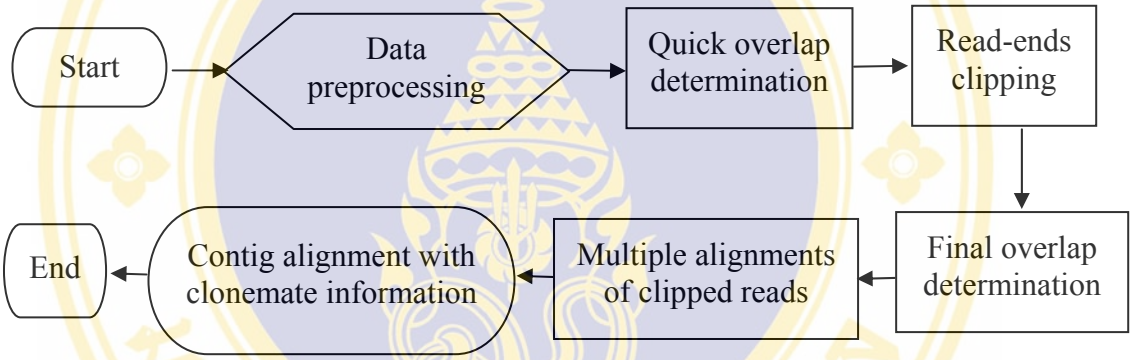

Fig. 4.3 Six main modules of SEQASS program.

Our shotgun sequence assembly program has been developed. The program, the source code, the installation guide, the user manual, and other related documents are recorded in the CD-ROM attached to this thesis. Fig. 4.3 shows the top-level flow chart of the SEQASS program, which consists of six main modules: 1) Data preprocessing, 2) quick overlap determination, 3) read-ends clipping, 4) final overlap determination, 5) multiple alignments of clipped reads, and 6) contig alignment with clonemate information.

4.3.1 Data preprocessing

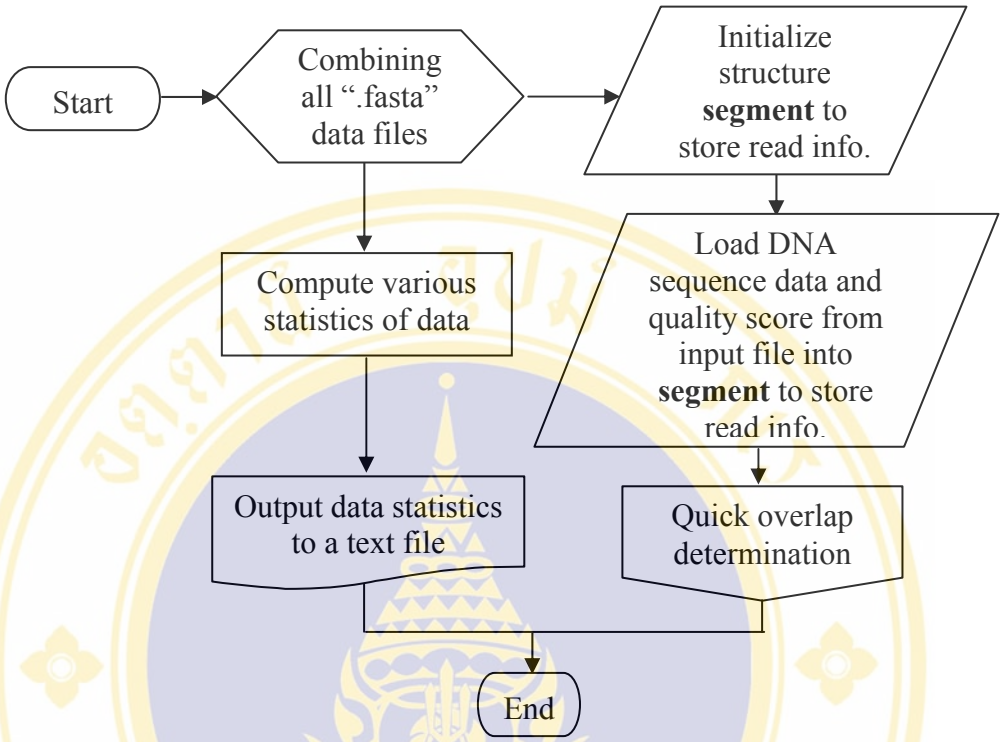

Fig. 4.4 The data preprocessing module

The input data for preprocessing are FASTA files of all reads and corresponding base quality value files. The outputs produced from preprocessing are the combined FASTA files of all reads and another file containing statistics of the read data. The flow chart of data preprocessing is shown in Fig. 4.4. The data preprocessing stage of the program contains following operations:

- Combining all ".fasta" data files into a single all-read-sequence file. The name of all-read data file can be specified by user.
- The read data and quality score information are loaded into an array of structure named ***segment.*** The segment structure carries all information of a read segment and contains members as described and listed in Table 4.1.

Table 4.1 Description for members of structure ***segment***.

name	Character pointer to the string of sequence name
len	Interger storing the length of sequence name
id	Long integer storing read identification number from sequence code
seqa	Character pointer to the string of unclipped read data
lengtha	Integer storing the length of the unclipped read
reva	Character pointer to the string of the reverse compliment of unclipped read
qscr	Pointer to the base quality score of unclipped read
rscr	Pointer to the reverse of base quality score of unclipped read
mid	Clonemate identification number determined from clonemate code
seq	Character pointer to the string of clipped read data
length	Integer storing the length of the clipped read
rev	Character pointer to the string of the reverse compliment of clipped read
fvclp	5' clipping position of the read
thrclp	3' clipping position of the read
kind	Kind of segment (1=Overlap, 0= Containment)
list	Link list of overlapping edges

- During data preprocessing steps, various statistics of data are calculated and reported as a text output file ("inpstat.txt"). These include the total number of reads, the length of shortest read, the length of longest read, the average length of reads, the total combined length of all reads, and the total number of each base character (A, C, G, T, N) of all reads.

4.3.2 Quick overlap determination

It should be stressed that the operation in this section of SEQASS is entirely different from the operation of the same stage in CAP3. The flow chart of SEQASS quick overlap determination is shown in Fig. 4.5.

BLAT is used to quickly determine possible overlaps among all data. We obtained c source code of standalone BLAT program from http://genome.ucsc.edu. After studying the program, the BLAT source code is then modified to be included in our program as a library. The BLAT library is created under our header file "blatlib.h" with the associated c source code library "blatlib.c", which are stored in CD-ROM attached to this thesis. We have made two modifications from the original BLAT source code

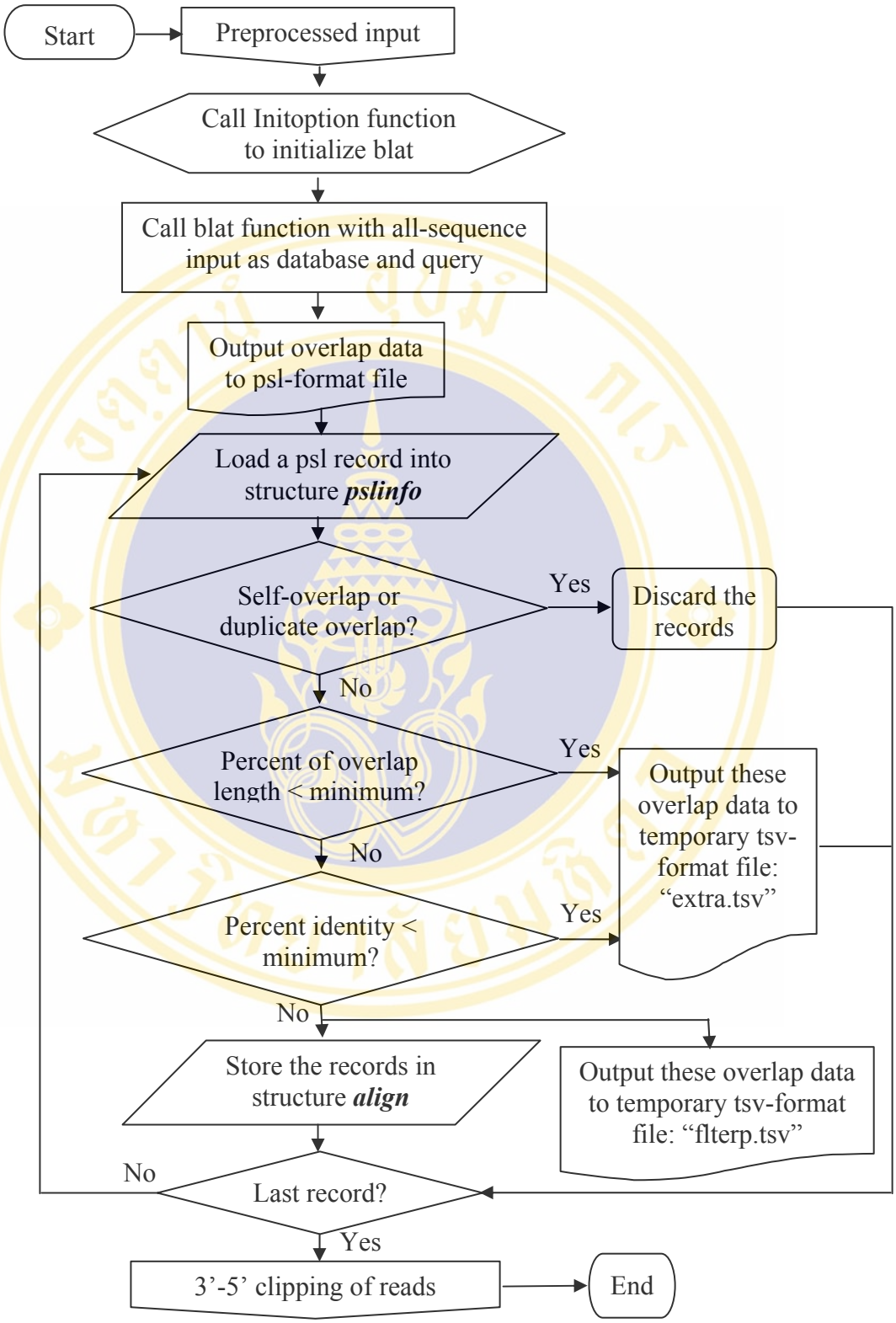

Fig. 4.5. The quick overlap determination module

- Initoption(char *dbFile, char *queryFile, char *outName) function is created to replace option initialization of blat. Original BLAT routine use argument from command line variables. This function creates virtual command line variables from input database file name (dbFile), input query output file name (queryFile), and output file name (outName) with a fixed option of "-ooc=11.ooc", which indicates the use of 11mers patterns to acclerate sequence search. "11.ooc" is the 11mers pattern file, which must be present at run time.
- Addition of memory disallocation in blat routine. The original blat(char *dbFile, char *queryFile, char *outName) routine contains hidden bugs in memory allocation. The hidden bugs caused serious memory shortage and unexpected shutdown problem to our SEQASS program. After analyzing the blat routine, it was found that the 'dbSeqList' variable, which is the pointer to large memory of data base sequence list, was allocated memory with no memory disallocation in the original routine. Thus, we add memory disallocation for this large variable and the problem is diminished.

The blat function is then called to perform quick pattern search on all read data and determine all possible overlaps. To minimize time for blat operation, all-sequence-data is searched with itself. The input data base file (dbFile) and input query file (queryFile) are the same, which is the combined all-sequence file. In blat operation, query is searched with respect to database (or reference) sequences. Upon blat operation, the output is generated in PSL format, which is a spreadsheet-compatible tab/comma delimited form. Thus, the output file should be named with '.psl' extension. The PSL format contains records of overlaps that contain overlap information in the data fields listed in Table 4.2. Each PSL record is temporarily loaded into memory in structure ***pslinfo*** and following processing are performed.

Table 4.2 Description for members of structure ***pslinfo***.

match	Number of matched bases found in the overlap
mismatch	Number of mis-matched bases found in the overlap
repmatch	Number of repeat matches found in the overlap
Ns	Number of unidentified bases found in the overlap
Qgapcnt	The number of gaps[1] found in the query side of the overlap
Tgapcnt	The number of gaps found in the reference side of the overlap
Qgapbase	The number of gap bases found in all gaps of the query sequence
Tgapbases	The number of gap bases found in all gaps of the reference sequence
strand	The direction of query in the overlap. '+' or '-' strand means query is in forward or reverse (complement) direction, respectively
Qname	Name of the query sequence
Tname	Name of the reference sequence
Qsize	The number of bases of the query sequence
Tsize	The number of bases of the reference sequence
Qstart *Qend*	The start and end base position in the query side of overlap. The position is calculated with the query sequence in forward direction.
Tstart *Tend*	The start and end base position in the reference side of overlap.
Blckcnt	The number of blocks[2]
Blcksize	The array of numbers of bases in blocks of the query side of the overlap
qstart	The array of start base position in blocks of the query sequence
tstart	The array of start base position in blocks of the reference sequence
ovlng	The overlap length
pcid	The percent identity of overlap

Note: 1. Gap is the insertion or deletion ('-') interposed the middle of overlap.
2. Block is the group of continuous bases with no insertion or deletion: region between gaps found in the overlap.

- Remove self-overlaps and duplicate-overlaps. Self-overlap, overlap of sequence **a** to **a**, and duplicate-overlap, overlap between sequence **a** to **b** and **b** to **a**, are produced because all-sequence-data is searched about itself. These overlaps are removed by comparing *Qname* and *Tname*. It must be noted that *Qname* and *Tname* are only stored with their integer parts. The redundant overlaps is simply eliminated by selecting only the case that *Qname* is greater that *Tname*.
- Filter out overlaps with overlap length less than minimum overlapping percentage. The minimum overlapping percentage can be specified by user and the default value is 20%. The minimum overlapping length

(*pslovlngmin*) is calculated from the percentage of *Qsize* or *Tsize* whichever is greater. The overlap length (*ovlng*:field) is estimated from the PSL data from the relationship: *ovlng* = *Tend-Tstart+ Qgapbase*. If *ovlng* is less than *pslovlngmin*, the overlaps will not be used in the calculation 3'-5' clipping of read but they are output to a temporary file "extra.tsv" because these records will be used for final overlap determination.

- Filter out overlaps with percent identity less than minimum percent identity (*pcidmin*). The minimum overlapping percentage can be specified by user and the default value is 90%. The percent identity (*pcid*) is calculated from the percentage of *match* to *ovlng*. If *pcid* is less than *pcidmin*, the overlap will not be used in 3'-5' clipping of read.
- The overlaps satisfied all minimum requirement are stored in a temporary tab-delimited file (flterp.tsv) and those not satisfied were saved in another temporary tab-delimited file (extra.tsv). Key overlap data are then stored in structure ***align***, containing data fields as listed in Table 4.3, for 3'-5' clipping and final overlap determination.

Table 4.3 Description for members of structure ***align***.

qix	The index of query sequence of the overlap
tix	The index of reference sequence of the overlap
stari	The start base position in the query side of overlap
endi	The end base position in the query side of overlap
starj	The start base position in the reference side of overlap
endj	The end base position in the reference side of overlap
strand	The direction of query in the overlap. 1 or 0 strand means query is in forward or reverse (complement) direction, respectively

4.3.3 The read-ends clipping

Clipping the 3' and the 5' of reads are performed to cut off poor quality region of reads. The satisfied overlaps, which were stored in structure **align**, are used to calculate 3' and 5' clipping positions of reads as followed. A good region of reads is defined as sufficiently long region of high-quality region that have sufficiently high similarity with another high-quality region. The 3' clipping position of a read is the maximum of 3' end positions of good regions of the read. The 5' clipping position of a read is the minimum of 5' end positions of good regions of the read. The start and

end positions for each pair of overlapping reads of the satisfied overlap are computed from a minimum band of diagonal matrix of an local alignment with base quality values. It should be noted that the start and end positions for each pair of overlapping reads of the satisfied overlap that were calculated by BLAT were discarded because they are not optimally aligned positions.

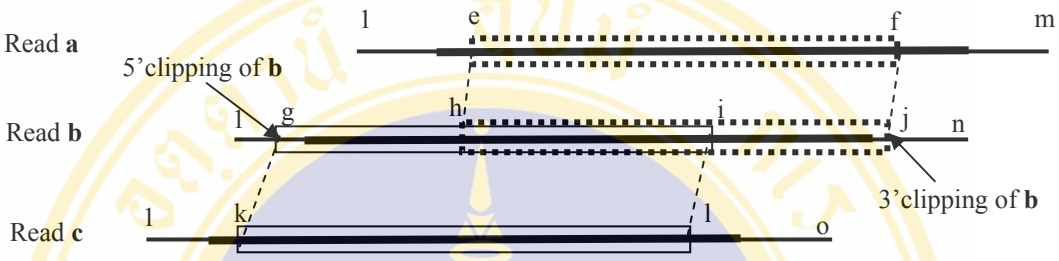

Fig. 4.6 Computation of the 5' and 3' clipping positions of read **b**

The schematic representation of the 3' and the 5' clipping position for read **b** is illustrated in Fig. 4.6. In this example, read **b** has high local similarities to reads **a** and **c**. A pair of broken lines shows the start and end positions of a similarity determined from local alignment. The 5' clipping position is the minimum of local similarity positions g (g<h) and the 3' clipping position is the maximum of local similarity positions j (j>i) It should be noted that thick lines indicate the high-quality regions of read.

In the implementation as summarized in the flow chart of Fig. 4.7, the 3' and the 5' clipping positions for each read that are stored in the *thrclp* and *fvclp* field in ***segment*** structure were iteratively calculated through satisfied overlaps by comparing the new start and end overlapping position of the read with the present 3' and 5' clipping positions. After the 3' and the 5' clipping positions for all reads are determined, the information of clipped reads is generated. These information includes *seq*, the pointer to the start position of clipped read, *rev*, the pointer to the start position of clipped read reverse complement, and *length*, the length of clipped read sequence. The information will be then used for the final overlap determination.

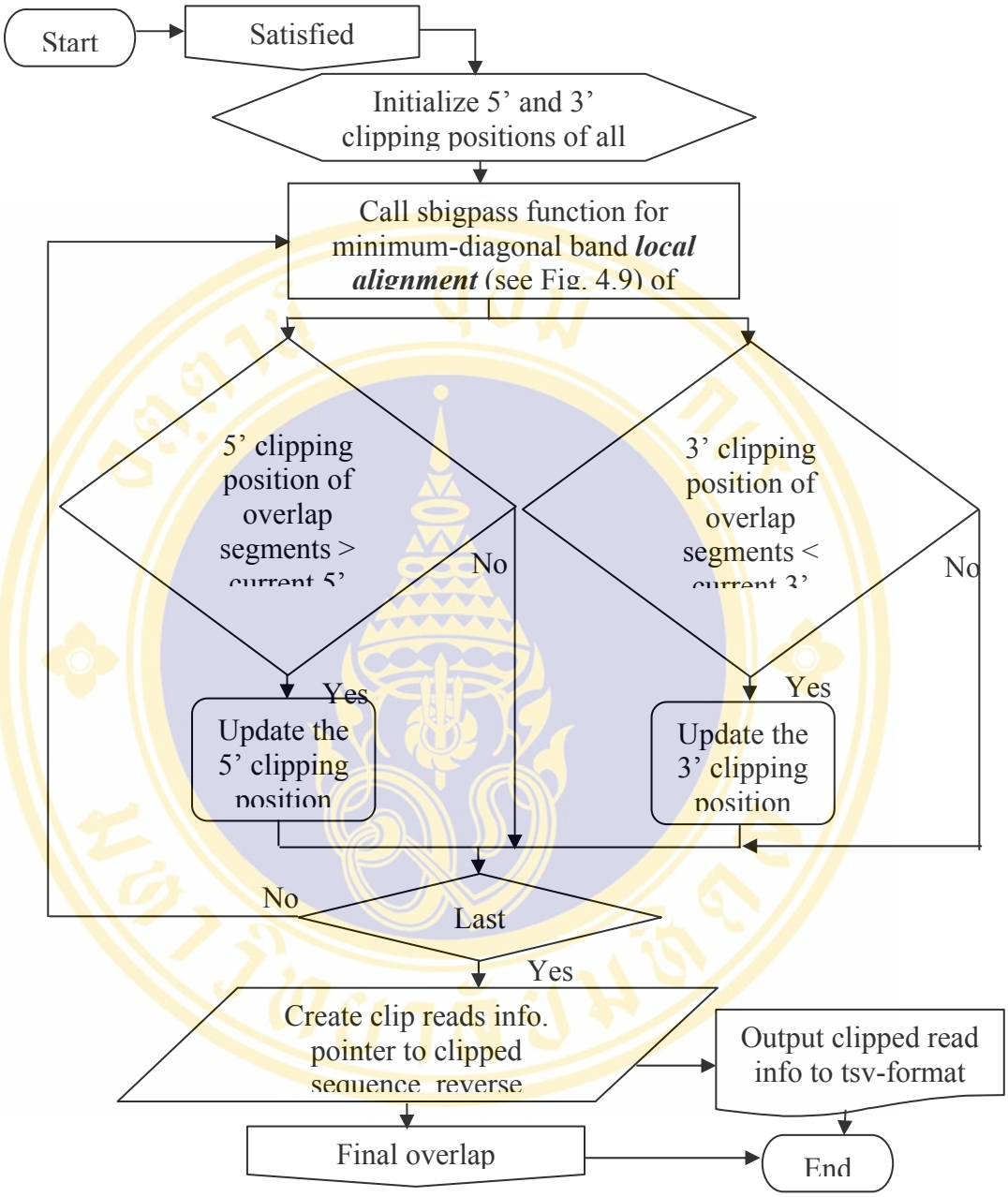

Fig.4.7 The read-ends clipping module

Local alignment by dynamic programming

Local alignment by dynamic programming is the best method to determine optimum position of overlaps that yield the best similarity. Local alignment by dynamic programming is a complicated and time-consuming but very useful and important algorithm. This section provides the reader the details from basic to band-diagonal local alignment algorithm that is finally implemented in SEQASS program.

Basic local alignment

The purpose of local alignment algorithm is to find the optimal common and similar segment of the two read sequences, which can be represented schematically in Fig. 4.8 [10].

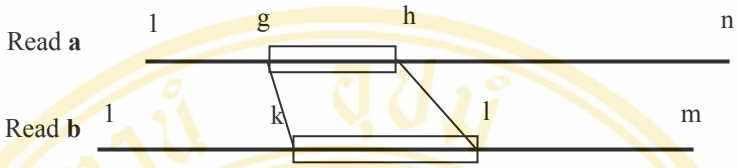

Fig. 4.8. Schematic representation of local alignment

In the aligned segment of the pair **a** beginning at a_g and ending at a_h, and **b** beginning at b_k and ending at b_l, a_g:b_k, a_{g+1}:b_{k+1}, …, a_{h-1}:b_{l-1}, and a_h:b_l are called *matched* pairs of letters. If the two matched letters are equal, the match is called *identity*. If the two matched letters are unequal and either one of them are not unidentified letter, N, the match is called *mismatch* or *substitution*.

Otherwise, the match is called *indel*, which stands for insertion or deletion. Insertion and deletion is the unidentified letter that is inserted to and deleted from the segment to obtain an optimum alignment between the pair. The process of basic local alignment consists of four detailed steps, the calculation of similar matrix, the calculation to output aligned position, the evaluation of the validity of the alignment, and recalculation of similar matrix to output other possible alignments that can be the best alignment. The overall flow chart of local alignment is summarized in Fig. 4.9.

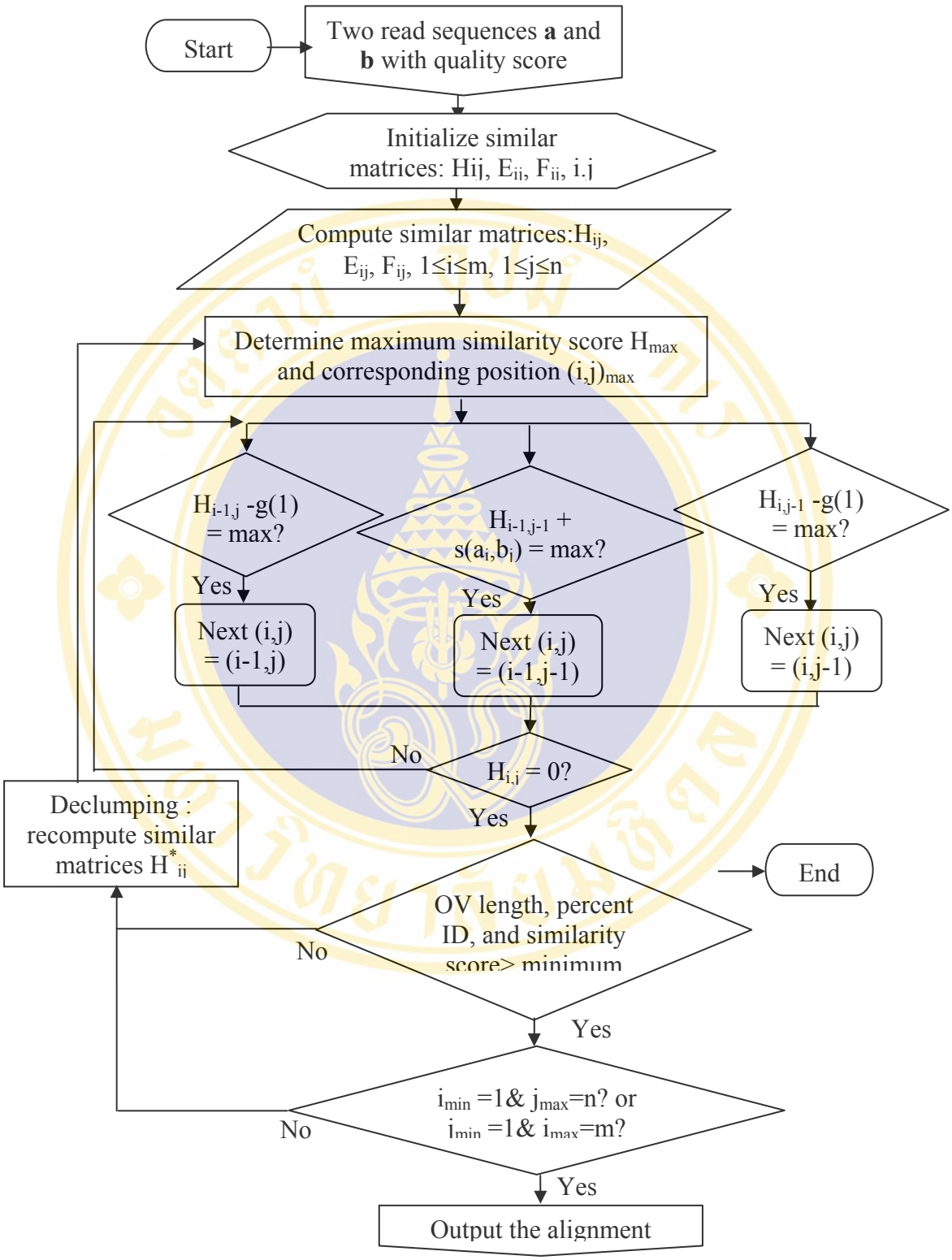

Fig. 4.9 The local alignment step in the read-ends clipping module

Calculation of similarity matrix

The general mathematical statement of the local alignment problem between two read sequences **a** and **b** is to find the maximum similarity score H(**a**, **b**) defined by:

$H(\mathbf{a}, \mathbf{b}) = \max\{S(a_g a_{g+1} \cdots a_{h-1} a_h, b_k b_{k+1} \cdots b_{l-1} b_l): 1 \le i \le j \le n, 1 \le k \le l \le m\}$

where $S(a_g a_{g+1} \cdots a_{h-1} a_h, b_k b_{k+1} \cdots b_{l-1} b_l)$ is the similarity score of aligned segment of the pair **a** beginning at a_g and ending at a_h, and **b** beginning at b_k and ending at b_l, and the general form of similarity score matrix element is proposed and defined according to:

$S_{i,j} = S(a_g a_{g+1} \cdots a_{h-1} a_h, b_k b_{k+1} \cdots b_{l-1} b_l)$: $i = h-g+1$, $j = l-k+1$ with $S_{0,0} = 0$, $S_{0,j} = -g(j)$, and $S_{i,0} = -g(i)$ and

$$S_{i,j} = \max\begin{Bmatrix} S_{i-1,j-1} + s(a_i b_j) \\ \max_{1 \le k \le j}\{S_{i,j-k} - g(k)\} \\ \max_{1 \le l \le i}\{S_{i-l,j} - g(l)\} \end{Bmatrix}$$

where $s(a_i,b_j)$ and g(k) are the similarity score function and indel penalty function. The similarity score function gives additive score for a match and negative penalty score for a mismatch, which is normally taken the form

$$s(a_i, b_j) = \begin{cases} +t : a_i = b_j \\ -s : a_i \ne b_j \ne N \end{cases}$$

where t and s are constants with typical values between 1 to 2. The indel penalty function provides a penalty score of k consecutive indels g(k) in the segment and is normally taken the form:

$g(k) = \alpha+\beta(k-1)$

where α and β are constants and typical values are $\alpha = \beta = 2$

The calculation of the maximum similarity score H(**a**, **b**) based on the definition for all possible values of g,h,k,l takes the number of alignment and use the upper time limit of $O(n^3m^3)$, which is too slow. The following improved algorithm utilizing three matrices, H, E, and F, has been proposed [10]:

$H(\mathbf{a}, \mathbf{b}) = \max\{H_{k,l}: 1 \le k \le n, 1 \le l \le m\}$

Let $g(k) = \alpha+\beta(k-1)$, $E_{i,j} = F_{i,j} = H_{i,j} = 0$ when $i \cdot j = 0$

$E_{i,j} = \max\{H_{i,j-1}-\alpha, E_{i,j-1}-\beta\}$, $F_{i,j} = \max\{H_{i-1,j}-\alpha, E_{i-1,j}-\beta\}$

$H_{i,j} = \max\{0, H_{i-1,j-1}+ s(a_i,b_j), E_{i,j}, F_{i,j}\}$

The three matrices are initialized and calculated recursively. To illustrate the algorithm, two example sequences **a** = GCTCTGCGAATA and **b** = CGTTGAGATACT, similarity function with t=2, s=1, and indel function with $\alpha = \beta = 2$ are used for the calculation. The calculated H matrix based on this algorithm is shown in Fig 4.10.

b \ a	N	G	C	T	C	T	G	C	G	A	A	T	A
N	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	2	0	2	0	0	2	0	0	0	0	0
G	0	2	0	1	0	1	2	0	4	2	0	0	0
T	0	0	1	2	0	2	0	1	2	3	1	2	0
T	0	0	0	3	1	2	1	0	0	1	2	3	1
G	0	2	0	1	2	0	4	2	2	0	0	1	2
A	0	0	1	0	0	1	2	3	1	4	2	0	3
G	0	2	0	0	0	0	3	1	5	3	3	1	1
A	0	0	1	0	0	0	1	2	3	7	5	3	3
T	0	0	0	3	1	2	0	0	1	5	6	7	5
A	0	0	0	1	2	0	1	0	0	3	7	5	9
C	0	0	2	0	3	1	0	3	1	1	5	6	7
T	0	0	0	4	2	5	3	1	2	0	3	7	5

Fig. 4.10. First similar matrix and local alignment of two example sequences, **a** and **b**.

Calculation to output aligned position (traceback)

After the similarity matrix is calculated, the positions of local alignment in both sequences must be determined. The general method is called *traceback*. In traceback algorithm, the aligned position is always started from the position (n,m) where $H_{n,m}$ is maximum. Other aligned positions are traced back from (n,m) to (0,0) under the following conditions. From a position (i,j), there are four possibilities for the next alignment position that can be determined according to

(1) if $H_{i-1,j}$ -g(1) is maximum, move to (i-1,j)

(2) if $H_{i-1,j-1} + s(a_i,b_j)$ is maximum, move to (i-1,j)

(3) if $H_{i,j-1}$ -g(1) is maximum, move to (i,j-1)

(4) if $H_{i,j} = 0$, terminate the alignment

The traceback results for this example are also shown in Fig. 4.10 as the series of dark-bordered square boxes. The traceback starts from (n,m) = (10,12) to (3,5) with

the length of alignment of 8 letters. Most tracebacks passed from (i,j) to (i-1,j-1) position with an exception of tracing back from (7,9) to (7,8), which is passing from (i,j) to (i, j-1). This special traceback indicates the existence of indel in this pair of alignment. The local alignment in this case can be written schematically as

a = G C T C T G C G A A T A
| | | | | | |
b = C G T T G A G N A T A C T

Evaluation of the validity of the alignment

The alignment is evaluated in order to determine if it is the potentially correct alignment based on the following criteria.

1) The alignment must be a potential overlap. In any alignment, one of the pair (**a** or **b**) is in front position of the other and it is called the head of alignment while the other is called the tail of alignment. *The aligned pair is a potential overlap if and only if the first matched position of the tail is the beginning position, which is one, and the last matched position of the head is its ending position, which is equal to its length.* The first local alignment in this example in this example is not a potential overlap because the first matched position of **b**, which is the tail of this alignment, is not started from its beginning position.

2) If the condition of potential overlap is satisfied, the overlap section is then check based on three measures: minimum overlap length, minimum percent identity, and minimum of the maximum similarity score. The percent identity is the ratio of the number of identity to overlap length calculated in percentage. These measures can be set by the user and can be determined based from experiment of a set data. These values are normally data-dependent. Typical value of minimum overlap length, minimum percent identity, and minimum of the maximum similarity score are 4, 50%, and 2, respectively. For this example, the alignment contains 6 identities, 1 mismatch, and 1 indel, thus having overlap length of 8, percent identity of 75% and maximum similarity score of 9, which are well satisfied the typical requirement but the condition of potential overlap for this alignment is

previously failed. Thus, this alignment is not a valid alignment and recalculation to search for other possible alignment should continue.

Recalculation of similar matrix to output other possible alignments (declumping)

The first alignment that was output is the first possible alignment between the pair with the highest maximum similarity score but it is found not to be a potential overlap and hence it cannot be the correct alignment. However, there are still other possible alignments between the pair with lower maximum similarity score that has no matches or mismatches in common with the alignment already output. In order to find other possible alignments, the concept of recomputing the similarity matrix, not allowing matches (identities or mismatches) already used, is employed. This recomputation technique is called *declumping*. This method is used so that the effects of all alignments in the clump, which is a set of alignments that share one or more matches with the alignment that already output, are removed. The recomputation procedure is as followed.

The recomputation is started from the upper leftmost position of a match in the alignment (g,h), which is also the position where the traceback is terminated. The recomputed similar matrix is denoted as H^* which satisfies

$H^*_{i,j} = H_{i,j}$ where $i<k$ or $j<l$

$H^*_{k,l} = \max\{0, H^*_{k-1,l} - g(1), H^*_{k,l-1} - g(1)\}$

where (k,l) must be the positions of matches in the alignment that is already output and to be declumped.

Starting from row g and position (g,h), the old $H_{g,h}$ is replaced by $H^*_{g,h}$ according to the new definition. Next, the remaining matrix elements along row g (g, h+1, …) $H^*_{g,l,(l>h)}$ are recomputed *based on the old definition using three matrices*. The recomputation along the row is proceeded until $H^*_{g,l,(l>h)} = H_{g,l,(l>h)}$ that is the recalculated element is the same as the previous element value because the remaining calculation will result in the same numbers.

For the next row g+1, the recalculation is started from (g+1, h+1) and $H^*_{g+1,l,(l>h)}$ is recomputed based on the old definition using three matrices and the recomputation along the row is proceeded to the position of the match (g+1,h') in this row of the alignment. The old similarity matrix $H_{g+1,h'}$ is replaced by $H^*_{g+1,h'}$

according to the new definition. Again, the remaining matrix elements along row g+1 (g+1, h'+1, …) $H^*_{g+1,l,(l>h')}$ are recomputed based on the old definition using three matrices and the recomputation along the row is proceeded until $H^*_{g+1,l,(l>h)} = H_{g+1,l,(l>h)}$ that is the recalculated element is the same as the previous element value. The recomputation of the remaining rows, g+2,… are proceeded in the similar manner as the row g+1. The recomputed similar matrix H^* based on this procedure is shown in Fig. 4.11.

b \ a	N	G	C	T	C	T	G	C	G	A	A	T	A
N	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	2	0	2	0	0	2	0	0	0	0	0
G	0	2	0	1	0	1	2	0	4	2	0	0	0
T	0	0	1	2	0	2	0	1	2	3	1	2	0
T	0	0	0	3	1	0	1	0	0	1	2	3	1
G	0	2	0	1	2	0	0	0	2	0	0	1	2
A	0	0	1	0	0	1	0	0	0	4	2	0	3
G	0	2	0	0	0	0	3	1	0	2	3	1	1
A	0	0	1	0	0	0	1	2	0	2	1	2	3
T	0	0	0	3	1	2	0	0	1	0	1	0	1
A	0	0	0	1	2	0	1	0	0	3	2	0	0
C	0	0	2	0	3	1	0	3	1	1	2	1	0
T	0	0	0	4	2	5	3	1	2	0	0	4	2

Fig. 4.11 Recomputed similar matrix and local alignment of sequences **a** and **b**

After the new similar matrix is obtained, another traceback process is employed to determine the next alignment with the next highest maximum similarity score. In this example, the new highest maximum similarity score of H^* is 5 at position (12,5). The second traceback results for this example are also shown in Fig. 4.11 as the series of dark-bordered square boxes. The traceback ends at position (7,1) with the length of alignment of 6 letters. Again, most tracebacks passed from (i,j) to (i-1,j-1) position with an exception of tracing back from (11,4) to (10,4), which is passing from (i,j) to (i-1, j). This special traceback also indicates the existence of indel in this pair of alignment. The local alignment in this case can be written schematically as

a = G C T N C T G C G A A T A
b = C G T T G A G A T A C T

The validity of this alignment is then evaluated. It can be seen that this is a potential overlap since the first matched position of **a**, which is the tail of this alignment, is started from its beginning position and the last matched position of **b**, which is the head of this alignment, is started from its ending position. Next, the overlap segment is checked for the minimum requirement of overlap length, percent identity, and maximum similarity score. It can be seen that this alignment contains 4 identities, 1 mismatch, and 1 indel, thus yielding overlap length of 6, percent identity of 66.7%, and maximum similarity score of 5. These satisfy typical minimum requirement. Although this alignment yields less percent of identity than the first alignment, this local alignment is potentially correct because a potential overlap is obtained upon this declumping calculation.

Thus, it may not be necessary to do more declumping and traceback calculation to search for other alignments and the recomputation of similarity matrix is terminated. It should be noted that the inserted indel has its pair character (A), which is presumably a possibly correct character at this position. However, the recomputation of similar matrix is generally continued more than twice until the best potential overlap is found. If the potential overlap can not be found over a number of iterations, the recomputation will be terminated under the condition that the maximum similarity score of the last iteration is increased from the previous iteration, which indicates that the recomputation results in the repetition of similar matrix.

Local alignment with heavy/light score and base quality value

The local alignment algorithm of Smith and Waterman is generalized to use heavy/light score and base quality values to improve the alignment quality [16]. This generalized scheme is implemented in SEQASS. First, the heavy/light score concept is introduced based on the fact that the both ends of sequence tend to contain a number of poor quality bases and hence the mismatched bases and indels in these regions should be lightly penalized because mismatchs or indels in these region are less strictly correct. Let POS5 and POS3 be the 5’ and 3’ boundary positions for light regions, the score for a match and negative penalty score for a mismatch of the similarity score function and indel penalty score function are modified to be

$$s(a_i, b_j) = \begin{cases} +t : & a_i = b_j \\ -s_h : a_i \neq b_j \neq N \text{ with POS5} \leq \text{i , j} \leq \text{POS3} \\ -s_l : a_i \neq b_j \neq N \text{ with i , j} < \text{POS5 or i , j} > \text{POS3} \end{cases}$$

where s_h and s_l are heavy and light mismatch scores and s_h is usually greater than s_l.

$$g(k) = \begin{cases} g_h = \alpha_h + \beta_h(k-1) \text{ with POS5} \leq \text{i, j} \leq \text{POS3} \\ g_l = \alpha_l + \beta_l(k-1) \text{ with i, j} < \text{POS5 or i, j} > \text{POS3} \end{cases}$$

It can be seen that mismatches and indels found before POS5 or after POS3 are subjected to - s_l and - g_l penalty and those between POS5 and POS3 are subjected to - s_h and - g_h. The heavy/light scoring help improving the alignment quality such that the poor quality base at both ends regions are likely to be removed from optimal alignment.

The base quality value concept has been developed to further enhance alignment quality. The quality value of a base is $q = -10\log_{10}(p)$, where p is the estimated error probability for the base. This value is used to weight scoring of match, mismatch and gap penalties of base involved. The quality value further help improving the alignment quality such that the poor quality bases in any regions are likely to be removed from optimal alignment positions. With the quality score, the score for a match and negative penalty score for a mismatch of the similarity score function and indel penalty score function are modified to be

$$s(a_i, b_j) = \begin{cases} +t * \min(q_i, q_j) : & a_i = b_j \\ -s_h * \min(q_i, q_j) : a_i \neq b_j \neq N \text{ with POS5} \leq \text{i , j} \leq \text{POS3} \\ -s_l * \min(q_i, q_j) : a_i \neq b_j \neq N \text{ with i , j} < \text{POS5 or i , j} > \text{POS3} \end{cases}$$

where q_i and q_j are bases quality scores of the base matched or mismatched bases.

$$g(k) = \begin{cases} g_h = \alpha_h + \beta_h(k-1) \text{ with POS5} \leq \text{i, j} \leq \text{POS3} \\ g_l = \alpha_l + \beta_l(k-1) \text{ with i, j} < \text{POS5 or i, j} > \text{POS3} \end{cases}$$

In this case q_i is quality score of the base that pair to the gap and q_j is the quality value of the base in the other sequence immediately before the gap and the quality value of the base immediately after the gap otherwise. In implementation, the

match, mismatch, and light mismatch score are set in 128x128 global matrices, V and V1. The 128 matrix dimension is used so that the ASCII value of a, c, g, t, n, A, C, G, T, and N can be set as indices of the matrix directly.

4.3.4 Final overlap determination

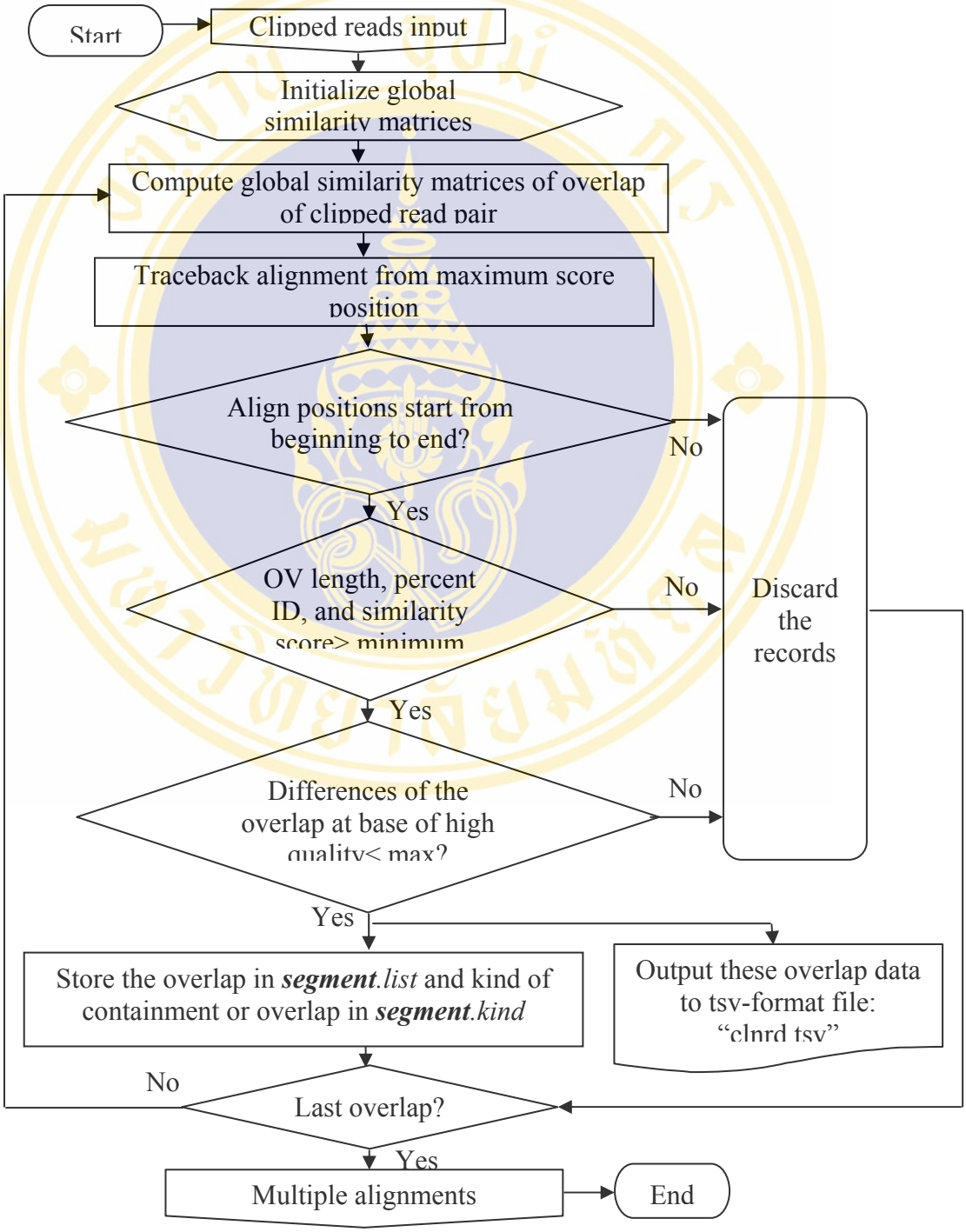

Fig. 4.12 The final overlap determination module

Overlap determination is finalized by performing global alignment between clipped reads in possible overlaps. The flow chart of final overlap determination is shown in Fig. 4.12. Global alignment is used to determine if two sequences of approximately the same size has high similarity. To check for a correct overlap, global alignment similarity matrix is calculated between the overlapping portions of both reads. The overlap is identify to be correct if the overlap length, percent identity, and maximum global similarity score values are higher than the minimum values and the differences of the overlap at base of high quality is lower than the allowed maximum value. The differences of the overlap at base of high quality are estimated as followed.

Let an integer b be a high quality value cutoff and let an integer d be a quality difference score cutoff. For a difference of the overlap at bases of quality values q_1 and q_2, the score at the difference is $\max[0, \min(q_1,q_2), b]$. The use of the term zero implies that no difference is counted at bases of quality values less than b. The quality difference score of the overlap is the sum of scores at each difference. If the quality difference score of the overlap exceeds d, then the overlap is removed. The overlaps pass the requirements will be used in multiple alignment contig construction.

Table 4.4 Description for members of structure ***OVERLAP***.

number	The index of 3' segment of the overlap
host	The index of 5' segment of the overlap
ind	The index for reassembly
stari	The start base position in the 5' segment of overlap
endi	The end base position in the 5' segment of overlap
starj	The start base position in the 3' segment of overlap
endj	The end base position in the 3' segment of overlap
orienti	The orientation of the 5' segment of overlap: 0 = reverse 1 = forward
orientj	The orientation of the 3' segment of overlap: 0 = reverse 1 = forward
score	The global similarity score of overlap
length	The overlap length
match	The number of matched base in the overlap
kind	The kind of overlap: 0 = containment, 1 = overlap
next	The pointer to next ***OVERLAP*** structure in the linklist

In implementation, global alignment is performed by *bigpass* function based on overlap information of clipped reads that were stored in structure ***align*** and ***segment***. The function returns overlap length, percent identity, maximum global similarity score, and the differences of the overlap at base of high quality, which are then compared with the minimum constants of the program. The correct final overlaps that satisfy all requirements are then stored in *overptr* pointer, which is pointed to ***OVERLAP*** structure (with corresponding pointer *overptr*) that contains data fields as listed in Table 4.4.

For overlap list construction, each satisfied overlap is stored in *overptr* pointer and inserted into the field *list* of ***segment*** structure of 5' segment of the overlap. The field *list* is the pointer to the linklist of the overlap structure contained in a read segment. There are two main kinds of overlaps, containment and overlap. Containment is the overlap that one of segment is contained in the other. If the kind of overlap is containment, the *kind* field of the corresponding 5' segment of the overlap will also be assigned to be containment (*kind* = 0). For subsequent multiple alignment, each linklist is sorted according to the index of 3' segment of overlaps after all satisfied overlaps are filled in the linklists.

Global alignment by dynamic programming

Global alignment by dynamic programming is the ultimate methods to determine if both overlapping parts of overlap have the highest similarity. Global alignment by dynamic programming is similar to local alignment in the matrix formulation. Thus, it is not necessary to explain the details in alignment algorithm and only differences of global from local alignment are digested as followed.

The global alignment matrix can be generally calculated utilizing three matrices, H, E, and F, similar to the local alignment:

$H(\mathbf{a}, \mathbf{b}) = \max\{H_{k,l}: 1 \le k \le n, 1 \le l \le m\}$

Let $g(k) = \alpha+\beta(k-1)$,

$E_{i,j} = \max\{H_{i,j-1}-\alpha, E_{i,j-1}-\beta\}$, $F_{i,j} = \max\{H_{i-1,j}-\alpha, E_{i-1,j}-\beta\}$

$H_{i,j} = \max\{0, H_{i-1,j-1}+ s(a_i,b_j), E_{i,j}, F_{i,j}\}$

But for the matrix is initialized differently from local alignment according to

$E_{0,0} = F_{0,0} = H_{0,0} = 0$, $E_{i,0} = H_{i,0} = -g(i)$, and $F_{0,j} = H_{0,j} = -g(j)$

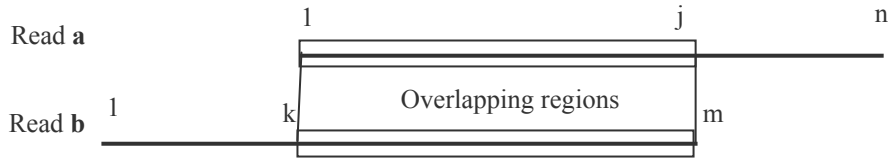

Fig. 4.13 Schematic representation of global alignment.

The three matrices are initialized and calculated recursively and the traceback to determine alignment output for global alignment is the same as that of local alignment. With this initialization, it will be required that the alignment extend from the beginning of both sequences to the end of both sequences as illustrated in Fig.4.13, while local alignment does not have this constraint. It should be noted again that global alignment is only computed in the predetermined overlapping region (found from local alignment) where size of both regions are similar.

4.3.5 Multiple alignment with greedy-tree algorithm

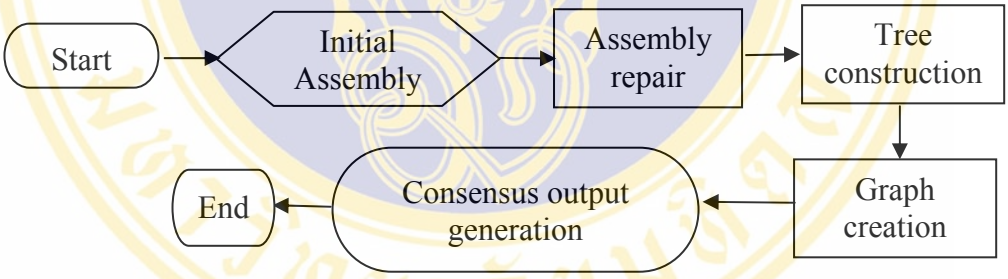

Fig. 4.14 The multiple alignments module

After correct overlaps among reads are determined, the overlaps are sorted in decreasing order of overlap scores. Greedy graph and tree algorithm is then used to merge them successively to form contigs. The c function source code of greedy algorithm is obtained from the first version CAP sequence assembly program [38]. The details of the source code operation were not available. We have studied by tracing and debugging the source code to understand the operation of the program. The description for greedy source code operation from our study and described as followed. The greedy-tree source code functions that are implemented in SEQASS

contain five main stages as shown in Fig. 4.14: Initial assembly, Assembly repair, Tree construction, Graph creation, and Consensus output generation.

Initial assembly

Table 4.5. Description for members of structure ***OVERLAP***.

isfive[2]	Two-element logical array indicates if 5' end of the contig is free (not overlap with other contigs): 1 = free contig (initial value) 0 = not free
isthree[2]	Two-element logical array indicates if 3' end of the contig is free: 1 = free contig (initial value) 0 = not free
orient[2]	Two-element logical array indicates orientation of 3' segment of the contig: 0 = reverse, 1 = forward
group	Contig serial number, which is the identification (id) number in the set of finally merged contigs: initial value = –1 = no initial group
next[2]	Two-element pointer array pointed to 3' adjacent segment of contig: initial value = 0 = no next contig
other	ID of segment on the other end of the contig
father	ID of father contig: initial value =–1 = no father contig
child	Id for the containment tree: initial value =–1 = no child contig
brother	ID for other containment tree contained in the father: initial value = –1 = no brother contig
node[2]	Two-element pointer array pointed to overlapping edges that are hosted by the contig: initial value = NULL = empty contig

Note: For all two element array, 0^{th} element=reverse complement and 1^{st} element=contig itself.

In the initial assembly, contig layout is first created by generating contig layout information for all reads in the structure ***CONS*** containing data fields as listed in Table 4.5. The flow chart of initial assembly, which is implemented in function *ASSEM()*, is shown in Fig. 4.15. First, all contigs are initialized to be empty with no father, brother, child, next, and nodes (overlapping edges). Assembly process to merge reads into contigs is then performed in two stages, the creation of layout information for containment contigs and noncontainment contigs, respectively.

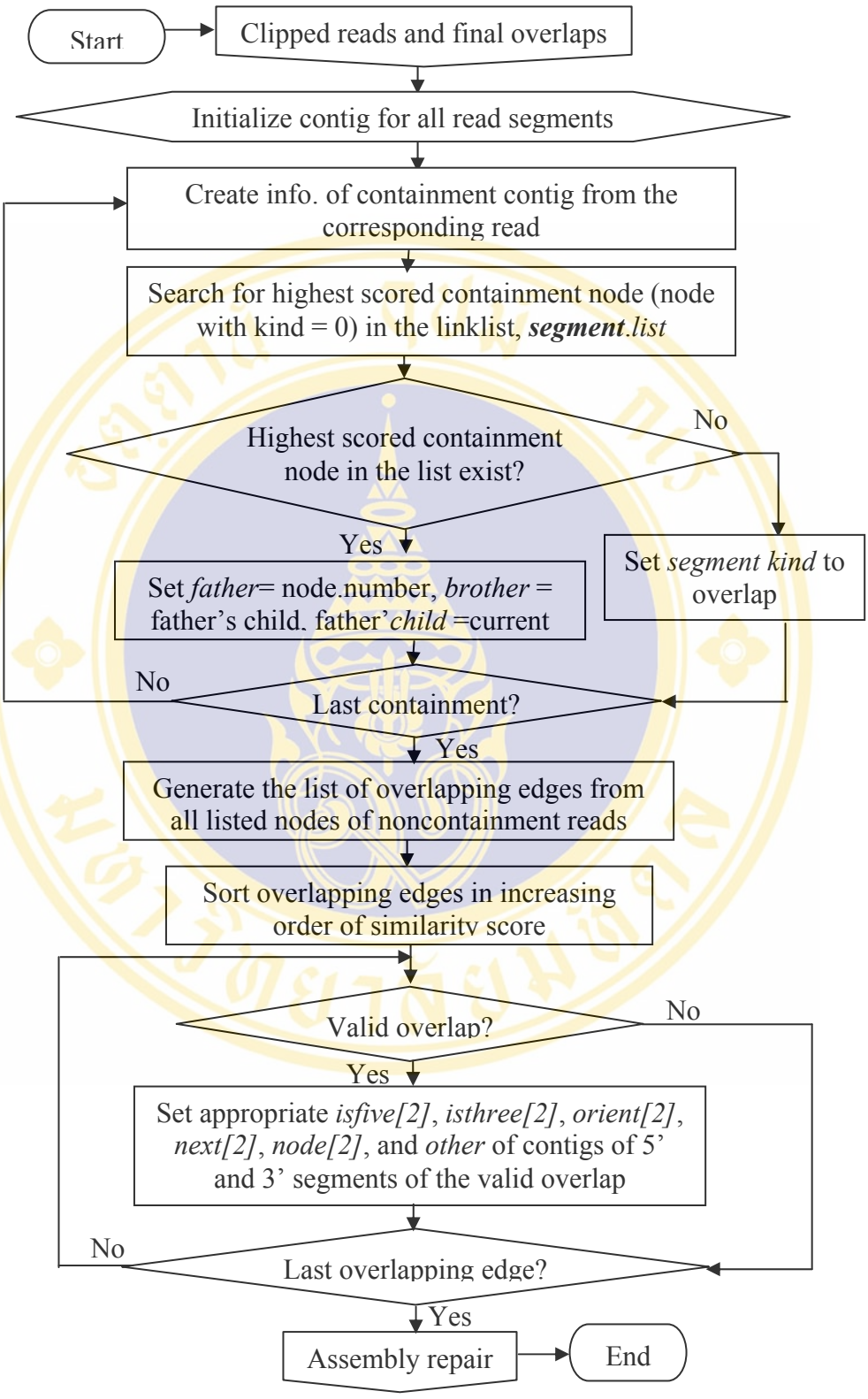

Fig. 4.15 The initial assembly step in the multiple alignments modules

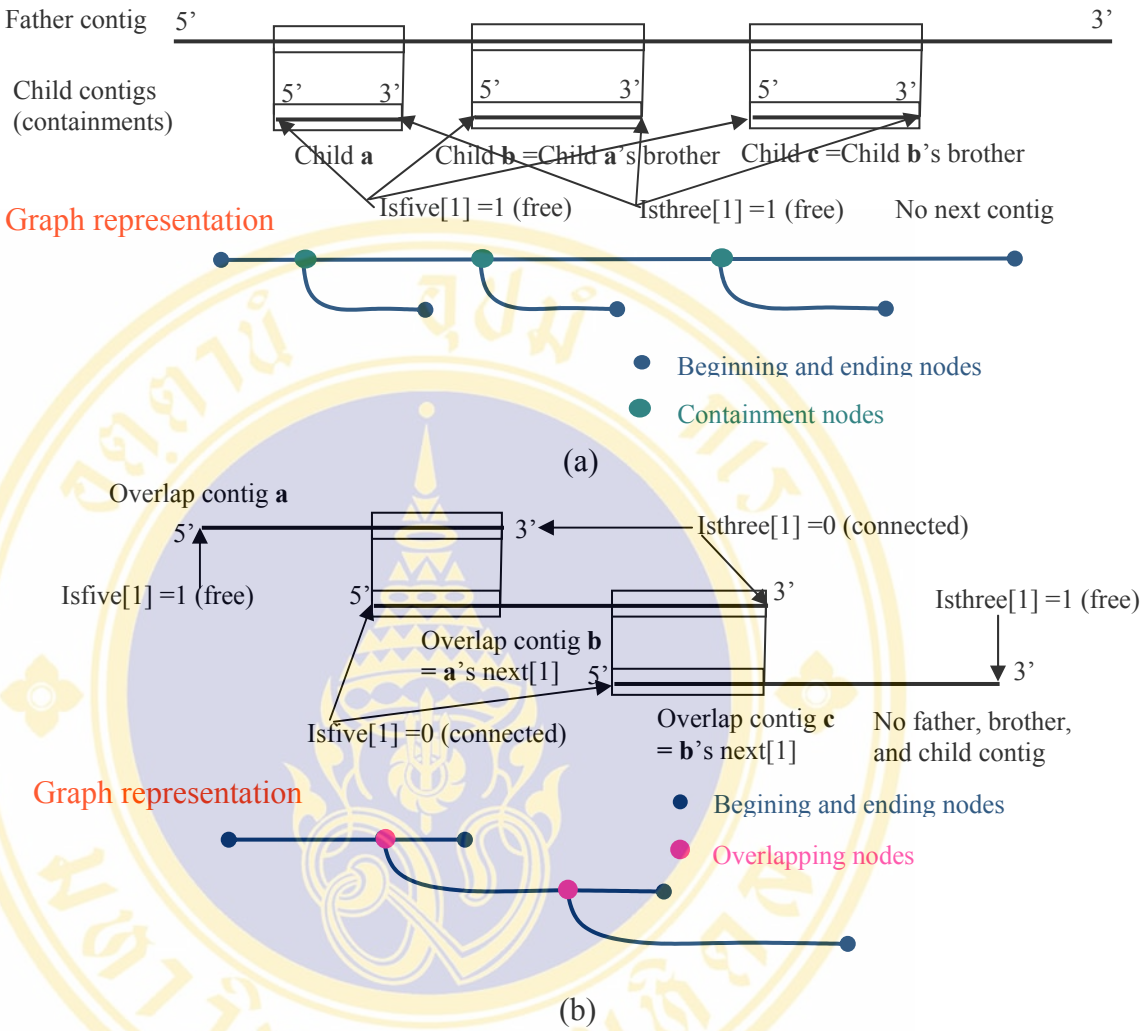

Fig. 4.16 The typical layout for (a) containmentand (b) noncontainment contigs.

The typical layout for containment and noncontainment contigs are illustrated schematically in Fig. 4.16. It should be noted that contig layout is only illustrated in forward direction for simplicity. The layout in reverse direction is simply the mirror of that in forward direction.

First, the information of containment contig is generated for and from the corresponding read (***segment*** with *kind* =0), obtained from the final overlap stage. The *father* contig, *brother* contig, *child* contig, and *node* of containment contig are identified from its overlap information (*list* field in ***segment*** structure: linklist of overlaps of the containment segment). Father contig for the containment contig is found by searching for containment node (node with kind = 0) in the linklist, ***segment***.*list*. The overlap containment pair of the node (***node***.*number*) is then set as the father contig, the nearest child (in the 3' direction) of father contig is set as brother

of current contig, and the current contig is set as the child contig of the father as illustrated in Fig. 4.16 (a). It can be seen that containment contig has no next contig in both directions and list of overlapping nodes is empty because is not overlap with other contig at both ends. If there are more than one containment nodes are found, the node with highest similarity score is selected as father contig. If no containment node is found for the containment segment, the kind of segment is changed to overlap (kind = 1) because the segment must have been incorrectly marked as containment due to false overlap information.

Next, the list of noncontainment overlapping edges is created by searching into linklists of overlaps of all noncontainment segments. Overlapping edges are then sorted in the decreasing order of overlap scores. The information of noncontainment contig, including *isfive[2]*, *isthree[2]*, *orient[2]*, *next[2]*, *node[2]*, and *other*, are updated to add overlapping segments into the contig by going through the list of overlapping edges in the decreasing order of overlap score. Both segments in each overlap will be added to 5' and 3' ends of their segment pair, if the overlap is determined to be valid. The overlap is valid, if 3' end of 5'segment (front segment of overlap) in the direction of overlap is free, 5'end of 3'segment (tail segment of overlap) in the direction of overlap is free, and the other end of contig is not the 3'segment of the overlap as illustrated in Fig. 4.16(b). It can be noticed that overlapping contig will have no father, child, and brother contig but have next overlap contigs at one end or both ends. The contig information of 5'segment and 3'segment are then updated for each valid overlap as followed.

The 3'segment ID is first set to be the *next* contig of the 5' segment contig (in the direction of 5' segment in the overlap) and 5'segment ID is set to be the *next* contig of the 3' segment contig (in the opposite direction of 3' segment in the overlap). The overlapping edge is then added to the *node* of 5' contig in the direction of 5' segment in the overlap and to the *node* of 3' contig in the opposite direction of 3' segment in the overlap. It should be noted that the *next* contig is corresponding the highest score *node.number*, which is the pair ID of overlapping segment of the host contig. Next, the *orient* of 5' segment (in the direction of 5' segment in the overlap) is set to be the orientation of 3'segment in the overlap and the *orient* of 3' segment (in

the opposite direction of 3' segment in the overlap) is set to be the orientation of 5'segment in the overlap. The *other* contig of the *other contig of 5' segment* is then assigned to be the *other* contig of the *other contig of 3' segment* and vice versa. Finally, the *isthree* (in the direction of 5' segment in the overlap) of the 5' segment contig is set to be not free (0) to indicate that the 3' end of 5'segment is now connected to another segment and the *isfive* (in the direction of 3' segment in the overlap) of the 3' segment contig is similarly set to be not free.

Assembly repair

The assembly must be repaired for conflicted overlap contigs. The flow chart for assembly repair is shown in Fig. 4.17. An overlap contig is in conflict if it was identified to be the overlap segment in the final overlap determination stage (kind =1) but after establishment of noncontainment contig layout, either 5' end or 3' end or both ends of the contig were found to be free (these ends are not overlapping to other segments). It should be noted that a correct and valid overlap contig should be overlapping to other segments on both ends.

In implementation (function *REPAIR()*), the list of conflicted segments are generated by checking the valid overlap contig conditions and stored in structure ***VX*** containing only there data fields *id* (segment identification number of the conflicted contig), *kind* (containment or overlap), and *list* (list of overlapping edges). After conflicts are identified, local and global alignments among the conflicted reads are performed to determine if they are overlaps or containments. The overlap information for the conflicted contigs including *kind* (containment or overlap) and *list* (list of overlapping edges) found from the additional local and global alignments are stored in structure ***VX***. Next, contig layouts for conflicted contigs are generated by reassembly, which is similar to the initial assembly. The reassembly process (in function *REASSEM*) to include conflicted reads also comprises of two stages, the creation information for containment contigs and noncontainment contigs, respectively.

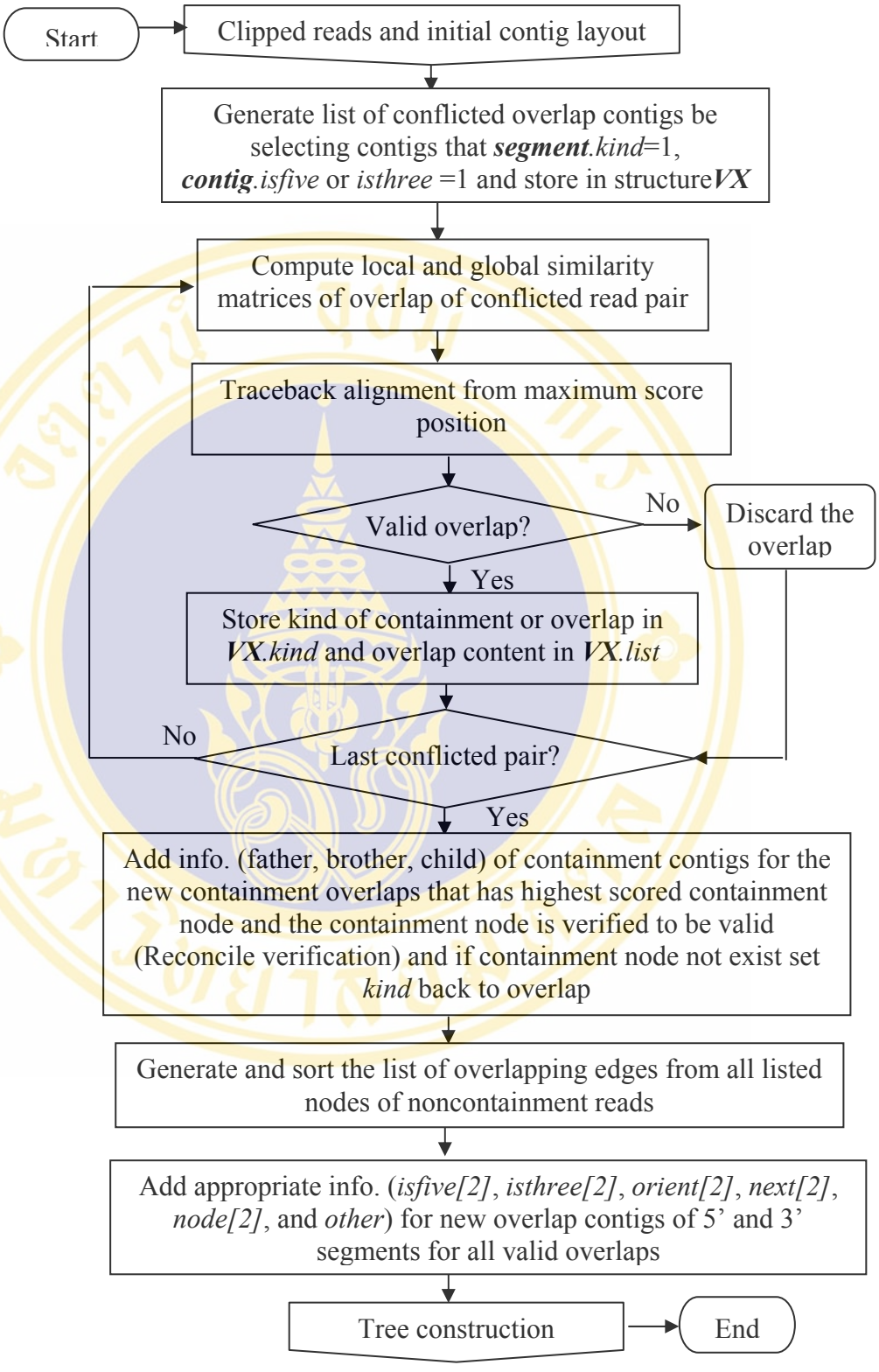

Fig. 4.17 The repair assembly step in the multiple alignments module

First, the information of containment contig is generated for conflicted segments that were determined to be containment in repair stage (those with field ***VX***.*kind*=0). The *father* contig, *brother* contig, *child* contig, and *node* of containment contig are identified from its overlap information (*list* field in ***VX*** structure: linklist of overlaps of the containment segment). Father contig for the containment contig is found by searching for containment node (node with kind = 0) in the linklist, ***VX***.*list*. The overlap containment pair of the node (***node***.*number*) is then set as the father contig, the child of father contig is set as brother of current contig, and the current contig is set as the child contig of the father. If there are more than one containment nodes are found, the node with highest similarity score is selected as father contig.

Unlike initial assembly, father and child containment pair in the reassembly stage must be further checked for proper containment because limited alignment in repair stage can lead to conflicted containment with existing contig. The validity of containment alignment is determined by the function *RECONCILE*. In RECONCILE stage, the father is identified to be invalid if father's contig.isthree and contig.isfive are free because it is not valid overlap contig. In addition, the father is also regarded to be invalid if father.isthree or isfive is not free but father's kind is containment (kind=0). Otherwise, father contig is valid and child will be checked for validity. Child contig is valid if the corresponding child.isthree or isfive is free. The validity of father and child are finaly confirmed by local alignment between father's next contig and child contig. The father and child pair is regarded as valid only if there is a proper overlap between father's next contig and child contig. Additional overlap found will be added to *list* in ***VX*** structure and included in corresponding overlap contigs. The containment contig information is then created for the valid containment and all invalid father and child are excluded from new containment creation. If no containment node is found for the containment segment, the kind of segment is changed to overlap (kind = 1) because the segment must have been incorrectly marked as containment due to false overlap information.

Next, the list of noncontainment overlapping edges is created by searching into linklists of overlaps of all conflicted segments that are found to be noncontainment in repair stage (those with field ***VX***.*kind*=0). The same as initial assembly, overlapping edges are sorted in the decreasing order of overlap scores and

the information of noncontainment contig, including *isfive[2]*, *isthree[2]*, *orient[2]*, *next[2]*, *node[2]*, and *other*, are updated (in the same way as initial assembly) to add valid overlapping segments into the contig by going through the list of overlapping edges in the decreasing order of overlap score.

Tree construction

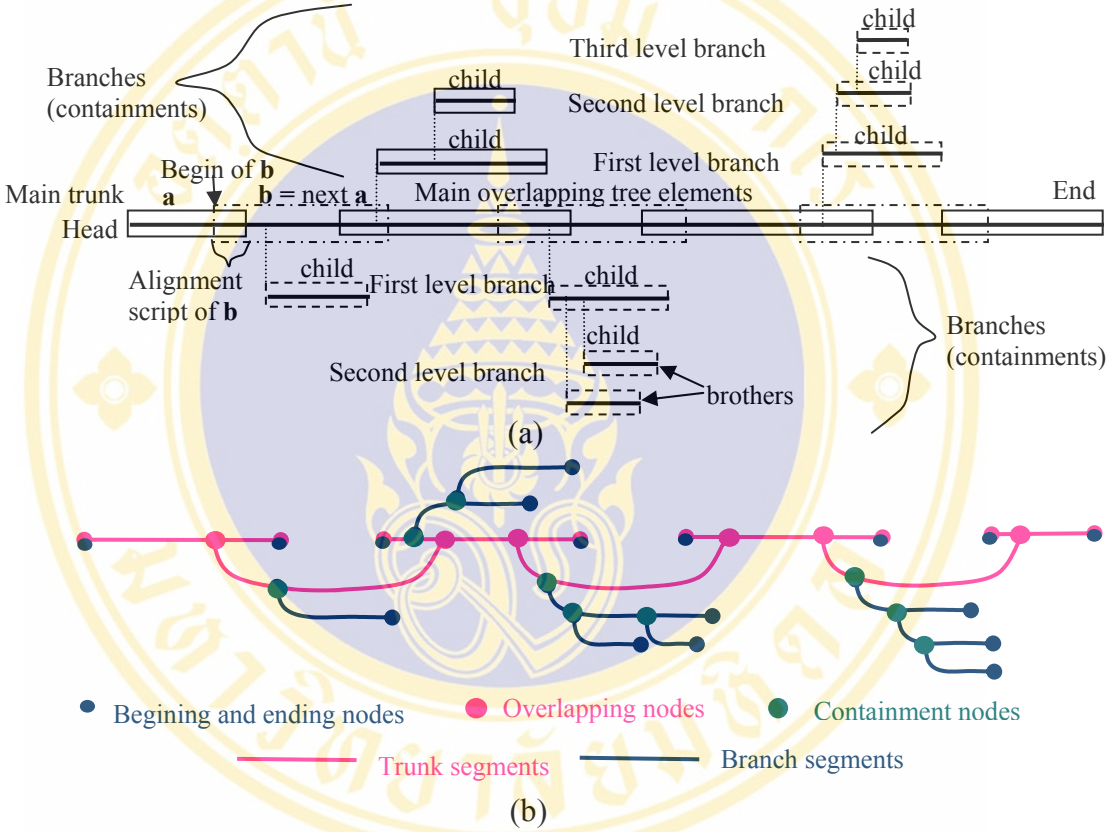

Fig.4.18 The typical (a) layout of a tree structure and (b) graph representation.

After the layout of contig is completely constructed and corrected, the final tree structure of the whole assembly is created from contig layout information of every read segment. The tree of assembled contig consists of some root (head) segments that are split and expanded into several branches. The whole tree structure is called graph, which will be subsequently created before output contig generation. The typical layout of tree structure and its graph representation are schematically illustrated in Fig. 4.18. The corresponding information of each tree element is stored in structure ***TTREE*** containing data fields as listed in Table 4.6.

Table 4.6. Description for members of structure ***TTREE***.

Head	Indicating if the segment is the head (root) of a tree: initial value = 0 (not a head)
Orient	Orientation of the root segment of the tree: 0 = reverse, 1 = forward
Begin	Start position of previous segment
Script	Pointer to alignment script. Alignment script is the alignment code that contains trace back information of the overlapping segment. The alignment script element for each match in the alignment is coded as zero. The alignment script elements for each k-deletion and k-insertion in the alignment are coded as –k and +k, respectively.
Size	The length of script corresponding to overlap length.
Next	The ID of next overlap segment in the head of overlap list
Child	The ID of child segment in the head of child list
Brother	The ID of next child segment of the father

From Fig. 4.18, it can be seen that the main branch (trunk) of the graph which consists of a series of overlapping tree elements. The overlapping elements are connected by the *begin* position (with respect to previous tree element) of *next* tree element as exemplified with element **a** and **b** in the figure. Branches that are split from the main trunk are all derived from successive containments of any main overlapping segment. It should be noted that there is no direct overlapping among child elements and each child terminates its own branch in the construction of tree. The graph representation (Fig. 4.18 (b)) is used to clearly describe connection and hierarchical relationships among tree elements. The overlapping segments of the trunk are connected via overlapping nodes and the branch segments are connected with the trunk or upper branch segment through containment nodes.

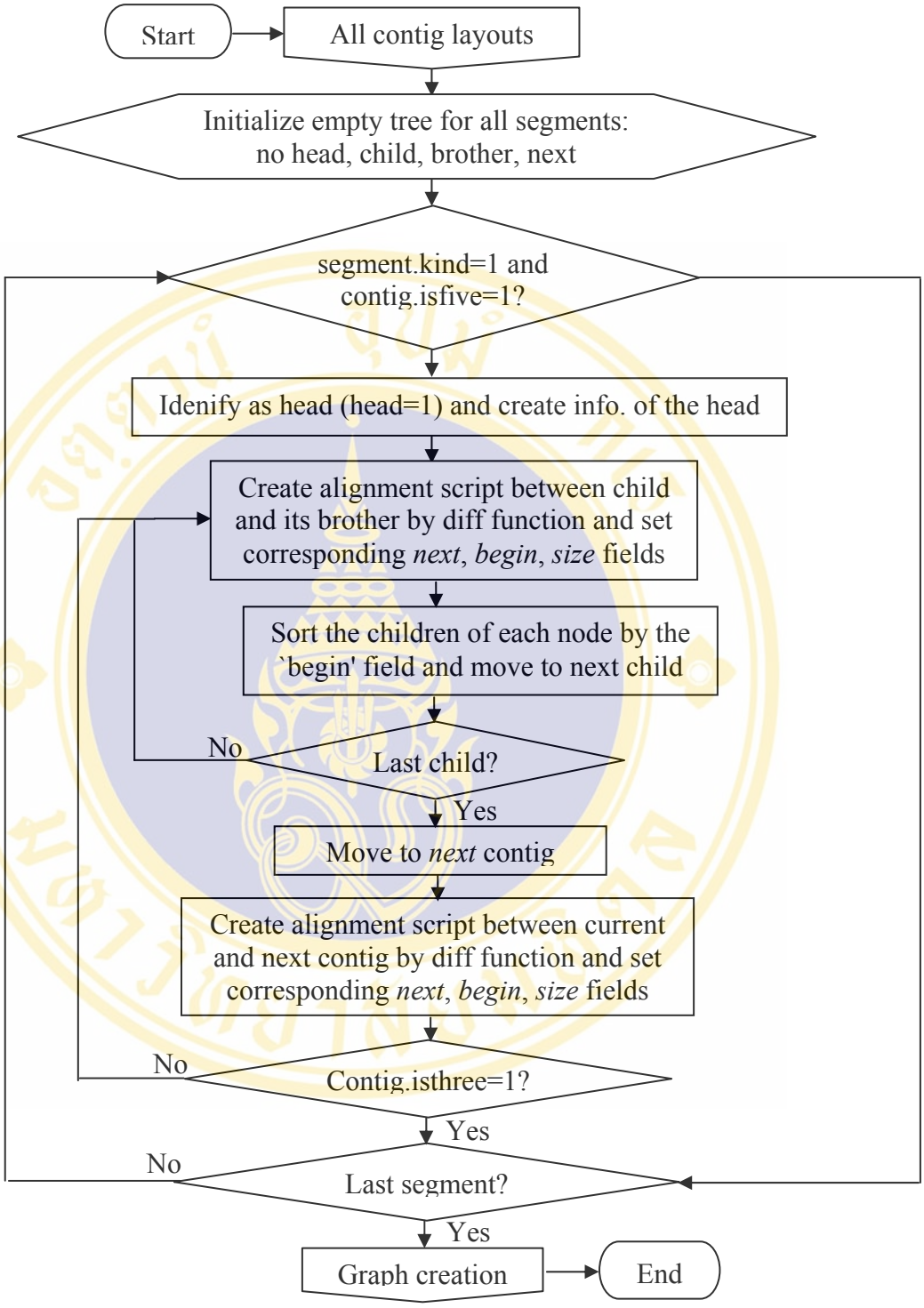

Fig.4.19 The tree construction step in the multiple alignments module

The tree information is generated from contig layout as illustrated in the flow chart in Fig. 4.19. First, each starting read of assembled contig, *head* or *root*, are identified by searching overlap contig structure whose 5' end is free and *head* field of

the tree is then set to 1. For each head segment, group contig is created. All fragments successively overlapping with the head segment are then added to the tree of the head in the increasing order of overlapping position (overlapping positions in the tree are sorted by *begin* field with SORT function). At the same time each child fragment contained in the parents that are overlapping in the tree are identified. For each overlap and containment, alignment script is generated from the start to end positions of the overlap by function diff, which utilize Macro functions for script coding of substitution, insertion, and deletion. Alignment scripts will be used for creation of final consensus output.

Graph creation

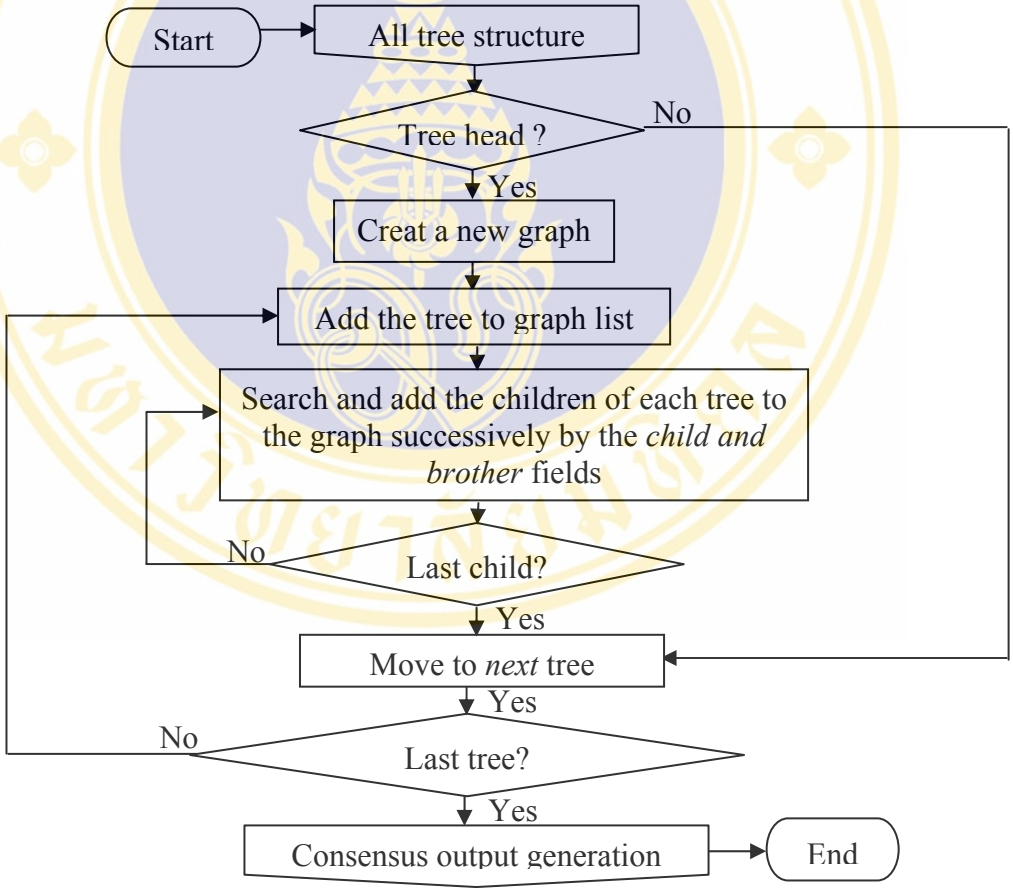

Fig. 4.20 The graph creation step in the multiple alignments module

A graph is the whole physical layout of a tree (corresponding to group contig) as previously illustrated in Fig. 4.18. Each contig is represented as a simple path, which is a path through the graph that contains each node at most once. In implementation as shown in the flow chart in Fig. 4.20, the graph of contig is

generated by outputting the lists of segment in the tree structure. The graph of contig is listed for every tree containing root segment, in which *head* field of tree is 1. The overlapping segments that follow the *head* are then listed with their corresponding orientations by going through the *next* field of the tree. In addition, the containment segments of each overlapping segment is listed right after the overlapping segment by outputting the list contained in the *child* field of the tree. In addition, the list contained in *brother* field of each *child* of the same father is also listed.

Consensus output generation

Table 4.7. Description for members of structure ***OUT***.

line	Pointer to one line of output segment.
a	Pointer to block in line.
c	Current character.
seq	Pointer to sequence.
length	Length of segment.
id	Index of segment.
s	Pointer to script vector.
size	Size of script vector
op	Current operation.
name	Pointer to the name of segment.
done	Logic variable: if merging operation of a segment is done, done=1
loc	Position of next segment symbol.
kind	Type of next symbol of segment.
up	Type of upper symbol of operation
dw	Type of lower symbol of operation.
child	Pointer to child subtree.
brother	Pointer to brother node.
father	Pointer to father node.
link	Pointer for operation linked list.

The output contig is finally constructed from the graph with tree information. The output is created as a linklist of row of fixed-width output line in structure ***OUT*** (with corresponding pointer *rowptr*) that contains data fields of information for input segment (node), merging operational tree, and output lines as listed in Table 4.7.

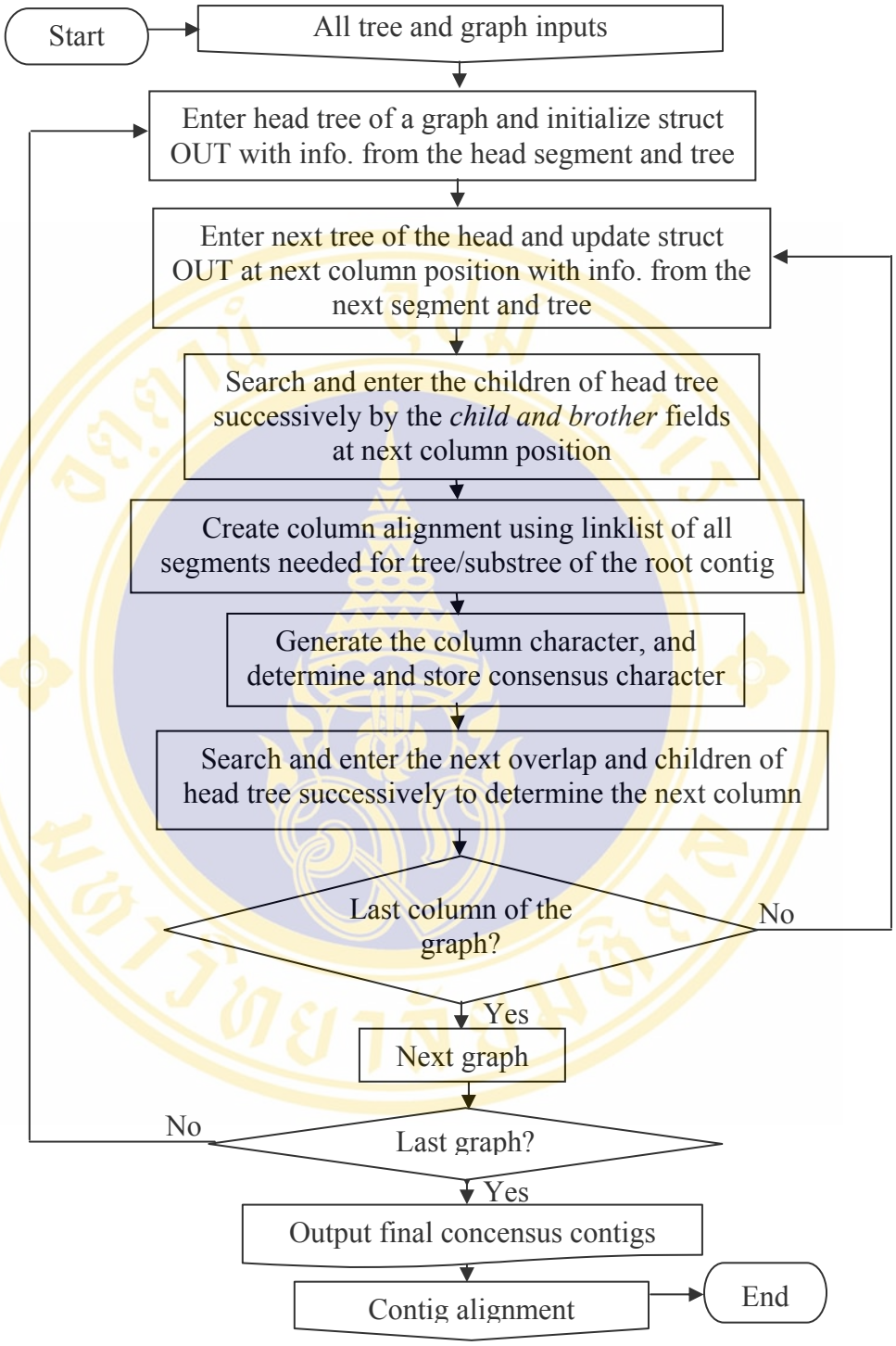

Fig. 4.21 The consensus output generation step in the multiple alignments module

The output of contig is generated by outputting the segment characters in the graph consecutively as illustrated in the flow chart in Fig. 4.21. Each character of the output is referred to as *column*, which would be produced from multiple alignment of overlapping character. The output of contig is generated for every tree containing *root*

segment. The information (including segment sequence, name, kind, alignment scripts, father/brother/child nodes, position of next segment, and operation) of root segment is first loaded and initialized into a temporary structure ***OUT*** variable called *work* by *ENTER* function. Next, the information (including segment sequence, name, kind, alignment scripts, father/brother/child nodes, position of next segment, and operation) of overlapping segments that follow the *root* are initialized and load using *ENTER* function by going through the *next* field of the tree. In addition, the containment segments of each overlapping segment are loaded using *ENTER* function right after the overlapping segment by loading the list contained in the *child* field of the tree. Moreover, the list contained in *brother* field of each *child* of the same *father* is also loaded using *ENTER* function.

Next, linklist of all segments needed for tree/substree of the root contig is created by *COLUMN* function that produces linklist in *link* field and returns *head* record as the first segment in the link list. All segments of the linklist are then aligned and the characters of segments overlapped at each column position are determined. The count of bases found at the column in temporary array variable sym with *sym[0]-sym[5]* are the count for A, C, G, T, N, and - base characters, respectively.

Finally, consensus bases are determined from the count of bases found at the column. The determination of consensus bases is based on the consensus formulation of CAP3, which is computed as followed. Let the column consist of k nonblank characters c_i, $1 \leq i \leq k$. Each c_i is in one of A,C,G,T,N and -, where N denotes any base other than the four base regular and - is a gap symbol, Let q_i be the quality values of c_i. If c_i is a gap symbol, then q_i is quality value of the base immediately before c_i in the same row if such a base exists and quality value of the base immediately after c_i otherwise. Let q_s (d) denote the average quality for substitution involving base type d of the read, let q_d denote the average quality value of deletion, and let q_n denote the average quality for insertion and d is a regular base. Next it is formulas of quality.

$$q_s(d) = \left[\left(\sum_{1 \le i \le k \text{ and } c_i = d} q_i \right) - \left(\sum_{1 \le i \le k \text{ and } c_i \ne d} q_i \right) \Big/ k \right]$$

$$q_d = \left(\sum_{1 \le i \le k \text{ and } c_i \ne - } q_i \right) \Big/ k$$

$$q_n = \left(\sum_{1 \le i \le k} q_i \right) \Big/ k$$

$q_s(N) = - q_n$

It should be noted that the quality values of the gap symbols in the column are excluded from the calculation of the average quality value for deletion. Any substitution involving base N is considered as a mismatch. The score alignment of substitution, insertion and deletion are defined as follows. Consider a substitution involving the column of the block and a base d of the read with quality value q_r. If $q_s(d)>0$ then the substitution is considered as a match and its score is m* min $[q_s(d), q_r]$ where the positive integer m is a match score factor. If $q_s(d) \le 0$, then the substitution is consider as mismatch and its score is n*min$[-q_s(d), q_r]$ where n is a mismatch score factor.

The score of a gap is the gap open score plus the gap extension score of each element in the gap. Let a positive number g be gap extension penalty factor. The quality value q_d in deletion gap is $-g*\min[q_d, q_r]$ where q_r is the quality of a base of the read immediately before or after the gap. The extension score of a base of the read with quality value q_r in an insertion gap is $-g*\min[q_n, q_r]$, where q_n is the average insertion quality of a column of the block immediately before or after the gap.

4.3.6 Contig alignment with clonemate information

The output contig are then finally linked by clonemate information. The flow chart for contig alignment is shown in Fig. 4.22.

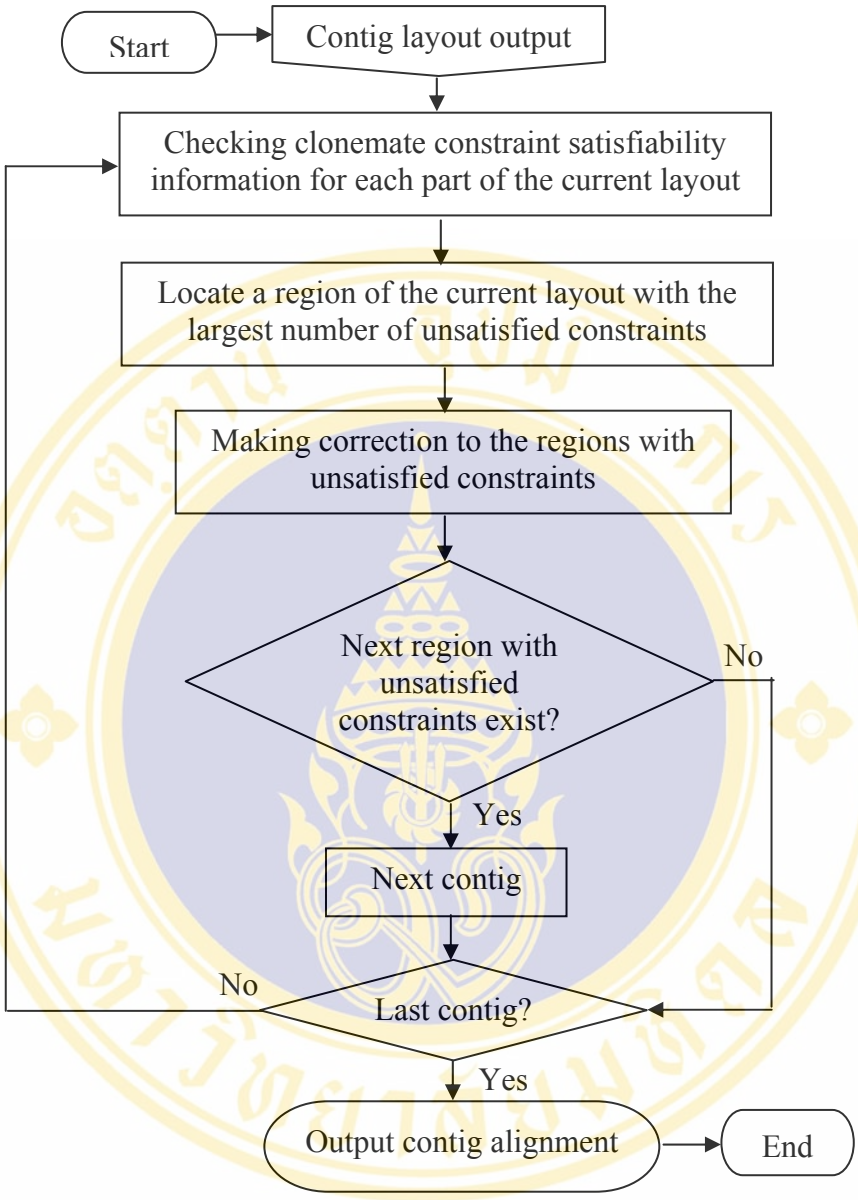

Fig. 4.22 The contig alignment module

The procedure for using clonemate constraints in linking of contigs consists of three steps. First, the quality of the current layout is assessed by checking constraints. The constraint satisfiability information for each part of the current layout is collected. In the second step, a region of the current layout with the largest number of unsatisfied constraints is located such that the unsatisfied constraints are satisfiable by making corrections to the region. If such a region exists, corrections to the region are made and steps 1 and 2 are repeated. Otherwise, the correction procedure is terminated. In final step, contigs are linked with constraints and each list of linked

contigs is reported. A forward–reverse constraint consists of two reads and two integers that specify a range for the distance between the reads. The constraint is satisfied by a layout if the two reads occur in the same contig, the upstream read is in forward orientation, the downstream read in reverse orientation, and the distance between the two reads is within the given range. Otherwise, the constraint is unsatisfied. In addition, an overlap is unused if the overlap is not used in the current layout.

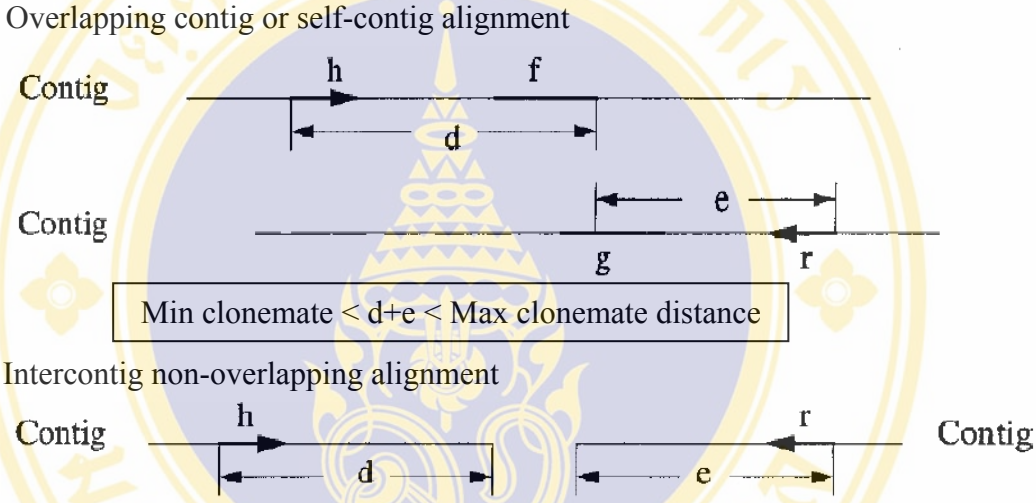

Fig. 4.23 The concept of contig alignment with clonemate constraints.

The concept of contig alignment with clonemate constraints is illustrated in Fig. 4.23. There are two cases of contig alignment, overlapping contig or self-contig alignment and intercontig non-overlapping alignment. In the first case, let f $\rightarrow$ g denote an overlap from read f to read g. An unsatisfied constraint involving reads h and r is satisfiable by an unused overlap f $\rightarrow$ g if read f occurs downstream of read h (r) in forward orientation in a contig, read g occurs upstream of read r (h) in reverse orientation in a contig, and the sum of the distance between read h (r) in forward orientation and read f and the distance between read g and read r (h) in reverse orientation is within the distance range of the constraint. In the second case, an unsatisfied constraint involving reads h and r is a link between two contigs if read h (r) in the forward orientation occurs in one contig, read r (h) in the reverse orientation occurs in the other contig, and the sum of the distance between read h (r) and the 3' end of the contig and the distance between the 5' end of the other contig and read r (h)

is less than the maximum distance of the constraint. Let u be the minimum number of unsatisfied constraints required to indicate a problem in the layout. The value for u is changeable by the user.

CHAPTER V
MODULES EVALUATION AND DISCUSSION

The SEQASS assembly program is experimented with the raw read data of large to medium-size mammal organisms including *Canis familiaris*, *Carollia perspicillata*, *Dasypus novemcinctus*, *Equus caballus*, *Felis catus*, and *Gallus gallus*. All data were obtained from the Genbank database. The assembly results and all detailed statistics will be digested with the most details for *Canis familiaris*, the first organism to be presented, while only summary of assembly results and statistics will be provided for other organisms because the details are similar for all organisms.

5.1 Information of raw data sets

Table 5.1 General information of the raw data sets

Bio-name	Description	Primary code	Accession number	Date
Canis familiaris	A dog	GI:22128664	AC110665	14-02-02
Carollia perspicillata	Seba's short-tailed bat	GI:27777582	AC138555	10-01-03
Dasypus novemcinctus	A nine-banded armadillo	GI:57222439	AC145418	07-01-05
Equus caballus	A horse	GI:32441298	AC124157	03-07-03
Felis catus	A cat	GI:23505548	AC091542	05-10-02
Gallus gallus	A chicken	GI:21553214	AC092081	28-01-03

The general information of six raw data sets are listed in Table 5.1. The data are DNA of six well-known mammal organisms as described. The primary code is the Genbank code associated with each organism and the accession number references the clone of this organism from Genbank. The date indicates the last updated time of the

data set. It should be noted that each organism consists of several clones and each data set is not the whole genome data of the organism.

Table 5.2 Data statistics of raw read data and whole clone sequence

Statistical Data	*Canis familiaris*	*Carollia perspicillata*	*Dasypus novemcinctus*	*Equus caballus*	*Felis catus*	*Gallus gallus*
Total number of reads	953	630	624	650	653	657
Total number of mate pair	466	124	252	250	195	208
The length of shortest read	393	635	607	495	507	552
The length of longest read	786	1663	878	893	890	871
The average length of reads	709	1054	729	728	732	750
Combined length of all reads	676,396	664,256	455,138	473,426	478,574	493,110
Total number of each base character of all reads						
A	198,364	182,418	139,150	135,410	117,962	130,767
C	137,190	156,011	84,509	100,593	114,664	110,802
G	136,797	141,296	85,766	98,009	113,544	114,135
T	201,255	181,362	143,959	137,521	125,089	132,887
N	2,790	3,169	1,754	1,893	7,315	4,519
The whole clone length	152,896	132,060	117,124	193,986	142,451	144,092
Total number of each base character of whole clone						
A	46,389	39,880	37,358	55,360	34,799	40,356
C	28,283	28,615	22,186	35,568	38,750	31,286
G	29,855	28,707	22,152	39,462	36,646	32,354
T	48167	34,658	35,428	63,596	32,256	40,096
N	202	200	0	0	0	0
Oversampling ratio	4.42	5.03	3.89	2.44	3.36	3.42

The statistics of raw read data obtained upon data preprocessing are listed in Table 5.2. It can be seen that they are medium-size clones with length in the range from 117,124 to 193,986 base pairs (bps). In addition, data of all organisms except *Felis catus* are A-T rich clones, containing more than 53% A-T base pairs. While most read data except *Felis catus* are good representatives of the corresponding whole clone data because they contain approximately the same proportion of A-T base pairs. From the data statistics, it can be implied that the quality of *Felis catus* is poorer than that of other organisms. The effect of poor quality data will be shown in the assembly results. Furthermore, most read data are of the same average length (709-750 bps) except *Carollia perspicillata* whose the average length is considerably longer.

The oversampling ratio is defined as the ratio of total combined length of all reads to the total number of base characters of whole genome. It represents the multiple read data that are oversampled with respect to the actual required data, which are the total number of base characters of whole genome sequence. The oversampling ratio of all data is in the range from 2.44 to 5.03, which indicates that the read data, especially for the *Equus caballus*, are not sufficient because generally oversampling ratio of more than 8 is needed to obtain more than 90% read coverage of the whole genome. The coverage problem will be worse if poor quality data are included and this could adversely affect the accuracy of the assembly.

5.2 SEQASS program parameters

All read data are assembled by SEQASS with optimum assembly (default) parameters as listed in Table 5.3. The minimum percent of overlap length and percent identity are used for input selection before 3'-5' clipping of read module, while other parameters are used for dynamic programming in final overlap determination module. The assembly results based on these parameters are digested for every main module in the following sections.

Table 5.3 Optimum assembly parameters of the SEQASS assembly program

Minimum overlapping percentage	20
Minimum identity percentage	90

Minimum similarity score	40
Match score	2
Heavy mismatch penalty score	-2
Light mismatch penalty score	-1
Gap open penalty	0
Gap extension penalty	2
Light gap extension penalty	1
Beginning position for heavy penalty at 5' end: POS5	20
Ending position for heavy penalty at 3' end: POS3	450
Maximum difference of base quality	40

5.3 Results and statistics from quick overlap determination

Table 5.4 Overall statistics of the initial overlap, satisfied overlap, and unsatisfied overlap

Statistics	*Canis familiaris*	*Carollia perspicillata*	*Dasypus novemcinctus*	*Equus caballus*	*Felis catus*	*Gallus gallus*
Total number of initial overlaps	18665	11638	9705	9136	7776	10634
Total number of satisfied overlaps	8958 (47.99%)	6340 (54.48%)	5007 (51.59%)	5066 (55.45%)	5153 (66.27%)	5841 (54.93%)
Total number of unsatisfied overlaps	9707 (52.01%)	5298 (45.52%)	4698 (48.41%)	4070 (44.55%)	2623 (33.73%)	4793 (45.07%)
Number of reads with initial overlaps	953	628	623	650	651	657
Number of reads with satisfied overlaps	950 (99.69%)	622 (99.04%)	619 (99.36%)	645 (99.23%)	629 (96.62%)	655 (99.70%)
Number of reads with no initial overlaps	0	2	1	0	2	0
Number of reads with no satisfied overlaps	3	8	5	5	24	2

The overlaps between pairs of reads of all read data are computed by BLAT tool function in which a default configuration utilizes 11 tuples for quick pattern search. The initial overlaps whose length satisfies than the minimum overlapping

percentage and whose identity percentage satisfies minimum identity percentage are stored in a temporary tab-delimited file, namely flterp.tsv. The remaining overlaps are stored in another temporary tab-delimited file, namely extra.tsv. Table 5.4 shows the overall statistic of the initial overlaps, the satisfied and unsatisfied overlaps. It can be observed that the total number of initial overlaps, that of satisfied overlaps, and that of unsatisfied overlaps for the *Canis familiaris* are highest. This implies that the total coverage of *Canis familiaris* should be higher than those of other organisms. In addition, it may be seen that the data of most organisms except *Felis catus* have similar total number of satisfied and unsatisfied overlaps (*Felis catus* has much more number of satisfied overlaps than that of unsatisfied overlaps).

Furthermore, it can be seen that all read data have very few reads with no initial overlaps and the number of read with no satisfied overlaps is always more than that with no initial overlaps. This means that most reads has local similarity to each other. It should also be noticed that the data of *Felis catus* again has distinct characteristics from other cases that the number of reads with no satisfied overlaps is much higher than other data. This should be another indication of poor quality data.

To see the results from quick overlap determination in more details, the statistics, including minimum, maximum, and the average values of various output data for all initial and satisfied overlaps of *Canis familiaris*, *Carollia perspicillata*, *Dasypus novemcinctus*, *Equus caballus*, *Felis catus*, and *Gallus gallus* are calculated and listed in Table 5.5 to Table 5.10, respectively. For *Canis familiaris*, all reads contain initial overlaps with a minimum of 6 initial overlaps in a read and a maximum of 116 initial overlaps in a read and reads consist of ~39 initial overlaps on average. While the number of satisfied overlap per read is varied from 2 to 32 with the average value of ~8, which are much less than the corresponding values in the initial overlaps. This indicates that there are large amount of poor-quality initial overlaps that do not satisfied the minimum requirements. The initial and satisfied overlaps are varied in length from 30 to 1284 and from 138 to 771 bases with the average length of ~316 and ~409.5 bases, respectively. This implies that these reads have large number of homologous regions to each other.

Table 5.5 Detailed statistics of initial and satisfied overlaps of *Canis familiaris*

Field	Initial overlaps			Satisfied overlaps		
	Minimum	Maximum	Average	Minimum	Maximum	Average
Number of overlaps per single read	6	116	39.16	2	32	8.23
Match	30	776	277.22	130	728	403.74
Mismatch	0	48	1.05	0	31	1.37
Rep.match	0	0	0	0	0	0.00
N's	0	278	0.26	0	16	0.26
Q gap count	0	14	0.52	0	14	0.57
Q gap bases	0	661	17.18	0	69	2.10
T gap count	0	14	0.58	0	14	0.58
T gap bases	0	679	20.05	0	73	2.03
Q size	393	786	710.65	573	786	711.09
Q start	0	724	120.04	0	596	166.18
Q end	30	786	415.76	160	775	573.65
T size	393	786	710.63	573	786	709.69
T start	0	724	117.99	0	598	161.60
T end	30	786	416.57	160	774	568.99
block count	1	16	1.86	1	15	1.93
Overlap length	30	1284	315.76	138	771	409.50
Percent ID	3.97	100	93.02	90.01	100	98.63
Percent overlap length	3.83	100	43.78	20.03	100	56.70

The corresponding matches, mismatches, N characters, and indels of initial and satisfied overlaps are varied from 30 to 776 bases and 130 to 728 bases with the average matches of ~277 and ~403 bases, 0 to 48 bases and 0 to 31 bases with the average mismatches of ~1 and ~1.3 base, 0 to 278 and 0 to 16 bases with the same average Ns of 0.26 base, 0 to 661 and 0 to 69 bases with the average indels on query side of ~17 and ~2 bases, and 0 to 679 and 0 to 71 bases with the average indels on reference side of ~20 and ~2 bases.

Table 5.6 Detailed statistics of initial and satisfied overlaps of *Carollia perspicillata*

Field	Initial overlaps			Satisfied overlaps		
	Minimum	Maximum	Average	Minimum	Maximum	Average

Number of overlaps per single read	1	75	37.06	1	45	10.13
Match	30	1165	433.84	140	1147	545.48
Mismatch	0	74	6.43	0	64	6.03
Rep.match	0	0	0.00	0	0	0.00
N's	0	15	0.60	0	12	0.64
Q gap count	0	37	2.01	0	17	1.58
Q gap bases	0	799	22.78	0	82	4.06
T gap count	0	32	2.58	0	26	2.04
T gap bases	0	909	20.39	0	77	4.47
Q size	635	1620	1094.73	635	1400	1131.96
Q start	0	1144	229.03	0	926	210.42
Q end	30	1387	692.68	171	1268	766.63
T size	635	1620	989.68	635	1400	980.75
T start	0	1028	154.88	0	913	132.09
T end	30	1341	616.14	174	1249	688.71
block count	1	38	4.74	1	31	4.12
Overlap length	30	1575	484.03	142	1209	560.68
Percent ID	6.78	100	91.68	90.03	100	97.40
Percent overlap length	2.23	100	40.81	20.03	100	46.76

While related starting and ending position on query and reference of initial overlaps are both in the range of 0 to 724 and 30 to 786 with the average starting/ending position on query and reference side of ~120/~416 and ~118/~417, respectively. Similarly, related starting and ending position on query and reference of satisfied overlaps are both in the range of 0 to 598 and 160 to 775 with the average starting/ending position on query and reference side of ~166/~574 and ~162/~569, respectively. The related block count, percent identity, and percent of overlap length of initial overlaps are varied from 1 to 16, 3.97 to 100, and 3.83 to 100, with the average values of ~2, 93.02, 43.78, respectively. While the block count, percent identity, and percent of overlap length of satisfied overlaps are varied from 1 to 15, ~90 to 100, and ~20 to 100, with the average values of ~2, 98.63, 56.7, respectively.

Table 5.7 Detailed statistics of initial and satisfied overlaps of *Dasypus novemcinctus*

Field	Initial overlaps			Satisfied overlaps		
	Minimum	Maximum	Average	Minimum	Maximum	Average

Number of overlaps per single read	1	79	31.11	1	34	16.05
Match	30	744	316.72	140	744	415.42
Mismatch	0	46	1.34	0	46	1.31
Rep.match	0	0	0.00	0	0	0.00
N's	0	342	0.21	0	13	0.18
Q gap count	0	11	0.57	0	11	0.44
Q gap bases	0	655	29.23	0	72	2.02
T gap count	0	13	0.60	0	12	0.45
T gap bases	0	679	26.27	0	70	1.52
Q size	607	878	728.14	607	878	729.29
Q start	0	729	133.63	0	620	163.08
Q end	30	796	481.13	151	796	582.01
T size	607	878	727.95	635	878	728.32
T start	0	742	137.61	0	659	169.86
T end	30	826	482.15	162	826	588.30
block count	1	15	1.97	1	13	1.77
Overlap length	30	1090	373.77	144	758	420.45
Percent ID	5.88	100	90.89	90.04	100	98.92
Percent overlap length	3.55	100	50.48	20.03	100	56.78

These statistics indicate that initial overlaps are generally long overlaps with low number of mismatches, indels, N-characters, and block count, and high percent of identity except some odd records that cause large variation in overall statistical data. While satisfied overlaps are shorter but are of better quality than initial overlaps. In addition, the variation of statistics of satisfied overlaps are much less than initial overlaps with better average values indicating that some odd records are removed as expected.

Table 5.8 Detailed statistics of initial and satisfied overlaps of *Equus caballus*

Field	Initial overlaps			Satisfied overlaps		
	Minimum	Maximum	Average	Minimum	Maximum	Average
Number of overlaps per single read	4	79	28.11	1	34	15.59
Match	30	735	305.71	131	735	413.42

Mismatch	0	49	2.53	0	47	2.96
Rep.match	0	0	0.00	0	0	0.00
N's	0	35	0.36	0	17	0.42
Q gap count	0	13	0.72	0	11	0.69
Q gap bases	0	683	17.63	0	62	1.90
T gap count	0	14	0.77	0	11	0.67
T gap bases	0	666	20.20	0	70	1.96
Q size	495	856	729.97	495	856	728.26
Q start	0	754	143.76	0	640	172.55
Q end	30	799	470.00	163	799	591.25
T size	495	893	726.36	495	893	724.60
T start	0	738	144.16	0	625	168.08
T end	30	788	472.97	147	788	586.84
block count	1	16	2.16	1	13	2.10
Overlap length	30	1258	346.44	141	762	420.66
Percent ID	7.69	100	92.73	90.01	100	98.25
Percent overlap length	3.74	100	46.56	20.03	100	56.61

From initial and satisfied overlaps of other organisms (Tables 5.6-5.10), it can be seen that initial and satisfied overlaps of these organisms share similar statistics as those of *Canis familiaris*. However, it can be noticed that initial and satisfied overlaps of *Carollia perspicillata* has notably different statistics from other cases. The initial and satisfied overlaps of *Carollia perspicillata* have lower average percent identity and average percent overlap length than other cases. This characteristic suggests that read data *Carollia perspicillata* may contain poorer quality overlaps than other data.

Table 5.9 Detailed statistics of initial and satisfied overlaps of *Felis catus*

Field	Initial overlaps			Satisfied overlaps		
	Minimum	Maximum	Average	Minimum	Maximum	Average
Number of overlaps per single read	1	49	23.82	1	31	15.78
Match	30	724	327.74	140	724	405.23
Mismatch	0	60	2.76	0	50	2.47
Rep.match	0	0	0.00	0	0	0.00
N's	0	94	1.27	0	22	0.85
Q gap count	0	20	0.76	0	20	0.69
Q gap bases	0	548	11.87	0	69	1.62

T gap count	0	22	0.83	0	18	0.71
T gap bases	0	567	12.25	0	66	1.59
Q size	507	890	731.84	589	874	731.36
Q start	0	750	193.03	0	624	172.05
Q end	32	806	536.67	162	791	582.21
T size	507	874	730.73	507	874	730.31
T start	0	750	198.15	0	629	176.32
T end	32	821	542.16	164	821	586.46
block count	1	23	2.29	1	22	2.21
Overlap length	30	1032	355.89	142	743	411.76
Percent ID	6.34	100	93.65	90.03	100	98.36
Percent overlap length	3.72	100	47.83	20.03	100	55.34

Table 5.10 Detailed statistics of initial and satisfied overlaps of *Gallus gallus*

Field	Initial overlaps			Satisfied overlaps		
	Minimum	Maximum	Average	Minimum	Maximum	Average
Number of overlaps per single read	4	88	32.37	1	41	17.78
Match	30	759	296.30	136	759	424.46
Mismatch	0	48	1.11	0	48	1.00
Rep.match	0	0	0.00	0	0	0.00
N's	0	35	0.60	0	27	0.62
Q gap count	0	13	0.47	0	8	0.46
Q gap bases	0	681	15.15	0	70	1.28
T gap count	0	12	0.53	0	8	0.50
T gap bases	0	677	14.17	0	71	1.12
Q size	552	871	748.61	552	871	753.99
Q start	0	799	151.09	0	681	181.84
Q end	31	857	464.25	176	857	609.20
T size	552	871	747.54	552	871	751.51
T start	0	781	150.49	0	647	186.55
T end	31	862	462.68	165	862	613.75
block count	1	20	1.87	1	13	1.88
Overlap length	30	933	327.33	143	769	428.48
Percent ID	5.29	100	94.08	90.07	100	99.08
Percent overlap length	3.59	100	42.68	20.02	100	55.61

5.4 Results and statistics from the read-ends clipping

The read-ends clipping is performed to cut off poor quality region of reads using satisfied overlaps by local alignment. The overall statistical results of 3' and 5' clipping of reads for all data are listed as shown in Table 5.11. It should be noted that the percentage of the number of clipped read and the total number of clipped base characters are calculated with reference to the total number of reads and the total number of combined length of read (listed in Table 5.2) of the organism, respectively.

Table 5.11 Overall statistics of clipping of the 5' and the 3' of reads of all data

Organism	*Canis familiaris*			*Carollia perspicillata*			*Dasypus novemcinctus*		
Total number of clipped reads	747(78.4%)			579(91.1%)			436(69.9%)		
Total number of reads with no clipping	206			51			188		
Total number of clipped base characters	25,769 (3.8%)			102,445(15.4%)			23,242(5.1%)		
Field	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
5' clipping position	0	260	17.37	0	786	27.95	0	420	17.64
3' clipping position	212	785	700.08	212	1663	919.72	215	867	709.78
5' clipping length	0	260	17.37	0	786	27.95	0	420	17.64
3'clipping length	0	491	9.67	0	908	134.66	0	564	19.61
Length of clipped read	178	781	682.71	184	1663	891.76	203	867	692.14
Organism	*Equus caballus*			*Felis catus*			*Gallus gallus*		
Total number of clipped reads	531 (81.7%)			596(91.3%)			607(92.4%)		
Total number of reads with no clipping	119			57			50		
Total number of clipped base characters	26,535(5.6%)			40,575(8.5%)			28,261(5.7%)		
Field	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
5' clipping position	0	378	27.23	0	431	35.52	0	343	32.68
3' clipping position	262	846	714.76	218	890	706.27	226	862	740.21
5' clipping length	0	378	27.23	0	431	35.52	0	343	32.68
3'clipping length	0	534	13.59	0	567	26.61	0	547	10.34
Length of clipped read	225	846	687.52	195	890	670.75	225	790	707.53

It can be seen that three organisms including *Canis familiaris*, *Dasypus novemcinctus*, and *Equus caballus* have low percentage (70-80%) of total number of clipped reads while the other three organisms have much higher percentage (>90%) of

total number of clipped reads. This implies that *Canis familiaris*, *Dasypus novemcinctus*, and *Equus caballus* have high number poor quality reads that are needed to be clipped. In addition, *Carollia perspicillata* and *Felis catus* have relatively higher the percentage of the total number of clipped base characters than other organisms, whose percentage of clipped characters are only 5% or less. This suggests that *Carollia perspicillata* and *Felis catus* data have reads with a number of long poor quality on either or both ends. Thus, considerable amount of base pair of reads of both organisms have been removed by 3'-5' clipping process and this could notably reduce the total percent coverage of the final assembly. However, this process is necessary for producing higher quality assembly, which comes at some cost of total coverage. While quite small amount of base characters of reads for other organisms have been removed by 3'-5' clipping process and this should not significantly affect the total percent coverage of final assembly of these organisms.

In details, most organisms except *Carollia perspicillata* share similar statistics of the 5' and the 3' clipping position, the 5' and the 3' clipping length, and the clipped length. It can clearly be seen that *Carollia perspicillata* has significantly larger the average 3' clipping length than other cases.

5.5 Results and statistics from final overlap determination

Overlap determination is finalized by performing global alignment between clipped reads in possible overlaps and checking with minimum requirements of the overlap length, percent identity, and maximum global similarity score values as described in section 4.3.4. For each final overlap, one segment is denoted as Host, in which the overlap is attached to in the overlap linklist ***segment****.list* of the host segment and the other is called Pair.

There are generally three distinct sets of final overlaps, containment, positive overlap, and negative overlap. Containment is the overlap that host is contained in its pair. Positive overlap is the overlap that host and pair have the same orientation, while negative overlap is the overlap that host and its pair have the opposite orientation. In addition, host is always 5' segment of the overlap. The final overlap data were stored in structure ***OVERLAP*** previously listed in Table 4.4. Three typical records of three

kinds of final overlaps taken from *Canis familiaris* are shown in Table 5.12. From the table, it can be seen that the final containment has *kind* value of 0, *Host start* position of 1, and *Host end* position equal to its segment's length. In addition, the final overlap has *kind* value of 1, *Host end* position equal to its segment's length (requirement of 5' segment), and *Pair start* position of 1 (requirement of 3' segment).

Table 5.12 Details of three typical final overlaps

Field	Containment	Positive overlap	Negative overlap
Host name	avt07f05.x1	avt05c02.x1	avt16h12.x1
Pair name	avt02f09.y1	avt12g03.y1	avt16d09.y1
Host ID	1	0	6
Pair ID	2	8	13
Overlap kind	0	1	1
Similarity score	1287	309	626
Overlap length	654	162	344
Match	647	157	324
Mismatch	2	5	13
Insertion/Deletion	5	0	7
Host orientation	1	1	0
Host start	1	439	248
Host end	653	600	578
Pair orientation	1	1	1
Pair start	26	1	1
Pair end	678	157	344
Percent identity	99.24	100	97.97

To view the overall results, the overall statistics, including minimum, maximum, and the average values of various final overlap data of all organisms are listed in Table 5.13. It can be seen that most organisms except *Canis familiarlis* have similar number of final overlaps and number of reads that host no final overlap. However, it can be seen that *Felis catus* has considerably high number of read that host no final overlap. High number of read that host no final overlap can leads to higher number of unsequencable reads, called *singlets*. It should be noted that read that host no final overlap is the read that does not overlap with other read on their 5' end but it may overlap with other read on the other end. In addition, all organisms appear to share similar statistics of the similarity score, overlap length, match,

mismatch, indel, host start, host end, pair start, and pair end. Furthermore, it can be noticed that the variation of statistics of final overlaps are much less than those of satisfied overlaps.

Table 5.13 Overall statistics of final overlaps of all data

Organism	*Canis familiaris*			*Carollia perspicillata*			*Dasypus novemcinctus*		
Number of final overlaps	10734			6668			6180		
Number of host reads with final overlaps	947			623			614		
Number of reads that are not host	6			7			10		
Field	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Similarity score	162	1418	834.40	167	2288	1101.3	149	1466	853.46
Overlap length	84	767	443.40	91	1219	597.09	94	764	451.19
Match	82	732	427.01	86	1176	566.97	81	748	435.64
Mismatch	0	115	6.50	0	164	13.79	0	89	6.10
Insertion/Deletion	0	126	9.88	0	235	16.33	0	70	9.45
Host start	1	603	247.06	1	1094	289.27	1	639	249.69
Host end	178	766	685.63	184	1620	876.99	203	764	696.44
Pair start	1	496	3.06	1	882	41.90	1	488	5.94
Pair end	84	750	439.40	88	1613	630.02	83	762	450.12
Percent ID	91.22	100	98.38	90.1	100	94.78	90.2	100	96.21
Number of overlaps per read	1	37	11.26	1	34	13.48	1	30	10.57
Organism	*Equus caballus*			*Felis catus*			*Gallus gallus*		
Number of final overlaps	5683			5410			6160		
Number of host reads with final overlaps	639			633			653		
Number of reads that are not host	11			20			4		
Field	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Similarity score	107	1449	850.14	148	1444	805.22	191	1515	880.69
Overlap length	64	781	452.45	86	762	430.99	106	776	464.28
Match	58	736	435.33	78	733	413.12	99	760	449.02
Mismatch	0	164	7.85	0	186	8.79	0	80	5.85
Insertion/Deletion	0	117	9.26	0	116	9.08	0	46	9.41
Host start	1	597	243.20	1	648	242.74	1	632	248.44
Host end	225	846	691.01	195	834	669.20	225	790	708.07
Pair start	1	472	4.91	1	470	7.98	1	489	4.63
Pair end	64	828	450.73	83	758	432.42	106	789	462.15
Percent ID	90.1	100	95.84	90.32	100	95.36	90.2	100	96.37

Number of overlaps per read	1	30	9.29	1	29	9.00	1	34	9.93

5.6 Results and statistics from multiple alignments by greedy-tree algorithm

Greedy graph and tree algorithm is used to merge correct overlaps successively and form contigs. As described in section 4.3, the multiple alignments consist of five main modules: Initial assembly, assembly repair, tree construction, graph creation, and consensus output generation. The details through steps in the five main modules are only illustrated for *Canis familiaris* data to demonstrate the concept of greedy tree algorithm. Only the consensus contig output results at the end of multiple alignments are then shown for other organisms.

Initial assembly

From the initial assembly, contig layout and overlaps for the initial layout are created for all read segments. There are four main kinds of contigs, containment, conflicted containment overlap, single-sided overlap, and double-sided overlap. The typical records of three kinds of contigs are shown in Table 5.14. The listed data fields from structure ***CONTIGS*** are previously described in Table 4.5.

From Table 5.14, there are 316 containment contigs and it can be seen that the containment contig, whose kind is 0, has all values of its *isfive[0]*, *isfive[1]*, *isthree[0]*, and *isthree[1]* equal to 1 (true). This means that both ends of contig and its corresponding reverse complement are free or not overlapped to other contig, which is the necessary requirement of proper containment contig. In addition, its must have a *father* contig and *other* contig on its end is itself. Moreover, next[0] and next [1], pointer to 3' adjacant segment of contig, must be 0 (NULL). Containment contigs may have *childs*, *brothers*, and *node[0]/node[1]* (overlapping edges at 3' end). The containment illustrated in the table has no child and brother but it hosts one overlapping edges at 3' end of forword contig and orientation of 3' segment in both direction (*orient[0]* and *orient[1]* are reverse (0)).

Table 5.14 Typical records of initial contigs

Field	Containment contig	Conflicted containment-overlap	Single-sided overlap contig	Double-sided overlap contig
Number of contigs of this kind	316	8	6	623
Contig ID number	4	25	2	3
Kind	0	1	1	1
isfive[0]	1	1	0	0
isfive[1]	1	1	1	0
isthree[0]	1	1	1	0
isthree[1]	1	1	0	0
orient[0]	0	0	0	0
orient[1]	0	0	1	1
group	-1	-1	-1	-1
Next[0]	0	0	0	2
Next[1]	0	0	3	5
Other	4	25	443	16
Child	-1	-1	1	-1
Brother	-1	-1	-1	-1
Father	5	-1	-1	-1
No. of node[0]	0	0	0	1
Node[0]	-	-	-	2,3,1,1087,563,550,9,1,145,707,1,1,559
No. of node[1]	1	0	1	2
Node[1]	4,5,0,1302,676,662,7,1,1,673,1,1,672	-	2,3,1,1087,563,550,9,1,145,707,1,1,559	3,5,1,1194,623,607,10,1,64,682,1,1,621
				3,4,1,1191,624,607,10,1,64,682,1,1,622

Note: 1. Format for each node of overlap is: host, number, kind, similarity score, overlap length, match, mismatch, host kind, host start position, host end position, number kind, number start position, number end position.

2. [0] and [1] correspond to reverse complement and contig itself, respectively.

3. *isfive[0]* and *isthree[1]* or *isfive[1]* and *isthree[0]*, must have the same value because 5' end of forward contig and 3' end of reverse complement or 3' end of forward contig and 5' end of reverse complement are the same edges.

In another case, conflicted containment-overlap has no *father* and no *next* contig with no overlapping edges, making it improper containment and overlap. While a single-sided overlap has true (1) value of its *isfive[0]* and *isthree[1]* but has

false value of its *isthree[0]* and *isfive[1]*, or vice versal. On the other hand, a double-sided overlap has false (0) value of its *isfive[0]*, *isthree[1]*, *isthree[0]* and *isfive[1]*. These mean that single-sided overlaps are those overlapped with other contig only at one end while double-sided overlaps are those overlapped with other contig at both end. In addition, either *next[0]* or *next [1]* and corresponding *node[0]* or *node[1]* must be 0 (NULL) for single-side contig while both *next[0]* and *next [1]* and related *node[0]* and *node[1]* must not be NULL. Both kinds of overlap must not have *father* and *brother* contigs but may have *child* contig and *other* contig on its end is not itself.

Table 5.15 Overall statistics of overlaps in the initial layout of *Canis familiaris*

Total number of overlaps in initial contig layout		4718	
Number of contig with overlaps in initial contig layout		945	
Number of contig with no overlap in initial layout		8	
Field	Minimum	Maximum	Average
Similarity score	215	1376	835.91
Overlap length	128	716	441.94
Match	118	697	426.85
Mismatch	0	111	5.99
Insertion/Deletion	0	117	9.09
Host start	2	559	257.42
Host end	306	760	694.95
Pair start	1	1	1.00
Pair end	127	712	436.26
Percent ID	92.98	100	98.51
Number of overlaps at 5’ end (Node[0])	1	27	5.61
Number of overlaps at 3’ end (Node[1])	1	27	7.80

To discuss the overall results, the statistics of overlaps in the initial layout are listed in Table 5.15. There are only 4718 final overlaps that are contained in 945 initial contigs and 8 initial contigs host no final overlap. The related number of final overlap at 5’ and 3’ end are varied from 1 to 27 with the average value of ~5.6 overlaps at 5’end and ~7.8 overlaps at 3’ end. The final overlaps contained in the initial contig layout are varied in length from 128 to 716 bases with the average length of ~442 bases.

The corresponding matches, mismatches, and indels are varied from 118 to 697 bases with the average matches of ~427 bases, 0 to 111 bases with the average mismatches of ~6 bases, and 0 to 126 bases with the average indels of ~9 bases. The related starting and ending positions on host are in the range of 2 to 559 and 306 to 760 with the average starting/ending position of ~257 and ~695, respectively. While the starting position on pair are all 1 and ending positions on pair are in the range of 127 to 712 with the average position of ~436. The similarity score is varied from 215 to 1376 with the average value of ~836 while the percent identity is varied from ~93 to 100 with the average values of 98.5. It can be noticed that the variation of statistics of overlaps in initial contig layout are less than those of all final overlaps.

Assembly repair

The assembly must be repaired the conflicted segments including conflicted containment-overlap and single-sided overlap due to false overlaps. After conflicted segments are identified, local and global alignments among them are performed to produce correct overlap information, which will be used to reassemble these contigs. The information for correction of conflicted segments are listed in Table 5.16. There are 14 conflicted contigs whose kind are all overlap (1). 6 of them are single-sided overlap contigs (contig 2, 443, 447, 948, 949, and 950) while the remaining 8 contigs are the conflicted containment overlap contigs. After realignment calculation, 3 of them remain overlap (kind =1) while the remaining 11 contigs may become containment (kind =0). The conflicted contigs are then reassembled and the information of these contigs are corrected using information from the Table 5.16.

The information of all reassembled contigs before and after correction are detailed in Table 5.17. After the actual reassembly, five kinds of correction occur. For the first kind, there are four single-sided contig (contig 2, 443, 948, and 950) that are corrected to be containment (kind =0 with all isfive and isthree are free). Second, single-sided contig (contig 447) is corrected to be double-sided contig. Third, single-sided contig (contig 949) remains single-sided contig but free end and associated relating contigs/overlaps information are modified. For the fourth kind, there are four conflicted containment contigs (contig 25, 110, 626, and 946) that are modified to be double-sided contigs (kind =1 with all isfive and isthree are not free). Last, there are

four conflicted containment contigs (contig 158, 687, 740, and 855) that are modified to be single-sided contigs (kind =1 with either isfive and isthree is free).

Table 5.16 The information for correction of conflicted segments

Number of conflicted segments							14
Number of conflicted single-sided overlap segments							6
Number of conflicted containment-overlap							8
ID	Original kind	Conflicted fields				Corrected kind	No. of overlaps (after realignment)
		Isfive [0]	Isfive [1]	Isthree [0]	Isthree [1]		
2	1	0	1	1	0	0	11
25	1	1	1	1	1	0	4
110	1	1	1	1	1	1	9
158	1	1	1	1	1	0	12
443	1	1	0	0	1	0	10
447	1	1	0	0	1	0	9
626	1	1	1	1	1	0	13
687	1	1	1	1	1	1	17
740	1	1	1	1	1	0	15
855	1	1	1	1	1	0	12
946	1	1	1	1	1	0	14
948	1	0	1	1	0	0	20
949	1	1	0	0	1	1	16
950	1	0	1	1	0	0	20

Table 5.17 The information of all reassembled contigs before and after correction

ID	Kind	Isfive 0	Isfive 1	Isthree 0	Isthree 1	Orient 0	Orient 1	Next 0	Next 1	Other	Child	Brother	Father	Node 0	Node 1
2	1	0	1	1	0	0	1	0	3	443	1	-1	-1	0	1
2	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**3**	**2**	**1**	**951**	**949**	**0**	**8**
25	1	1	1	1	1	0	0	0	0	25	-1	-1	-1	0	0
25	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**626**	**855**	**158**	**-1**	**-1**	**-1**	**3**	**2**
110	1	1	1	1	1	0	0	0	0	110	-1	-1	-1	0	0
110	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**447**	**946**	**949**	**-1**	**-1**	**-1**	**2**	**4**
158	1	1	1	1	1	0	0	0	0	158	-1	-1	-1	0	0
158	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**626**	**855**	**-1**	**-1**	**-1**	**0**	**4**
443	1	1	0	0	1	0	0	442	0	2	444	-1	-1	1	0
443	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**442**	**0**	**443**	**444**	**-1**	**687**	**1**	**9**
447	1	1	0	0	1	0	0	448	0	948	-1	-1	-1	1	0

447	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**448**	**110**	**945**	**-1**	**-1**	**-1**	**1**	**2**
626	1	1	1	1	1	0	0	0	0	626	-1	-1	-1	0	0
626	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**25**	**158**	**158**	**-1**	**-1**	**-1**	**3**	**4**
687	1	1	1	1	1	0	0	0	0	687	-1	-1	-1	0	0
687	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**740**	**740**	**950**	**-1**	**-1**	**0**	**10**
740	1	1	1	1	1	0	0	0	0	740	-1	-1	-1	0	0
740	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**687**	**687**	**-1**	**-1**	**-1**	**0**	**10**
855	1	1	1	1	1	0	0	0	0	855	-1	-1	-1	0	0
855	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**25**	**158**	**-1**	**-1**	**-1**	**0**	**2**
946	1	1	1	1	1	0	0	0	0	946	-1	-1	-1	0	0
946	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**949**	**110**	**949**	**-1**	**-1**	**-1**	**16**	**4**
948	1	0	1	1	0	0	1	0	945	447	947	-1	-1	0	2
948	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**945**	**948**	**947**	**2**	**949**	**0**	**19**
949	1	1	0	0	1	0	0	950	0	950	951	-1	-1	5	0
949	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**950**	**946**	**945**	**948**	**-1**	**-1**	**5**	**16**
950	1	0	1	1	0	0	1	0	949	949	952	-1	-1	0	5
950	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**949**	**950**	**952**	**443**	**687**	**0**	**11**

Note: Bold letters indicate contig information after correction.

Tree construction

The final tree structure of the whole assembly is created from corrected contig layout information of every read segment as described in the last chapter. Each tree of assembled contig consists of a root (head) segment and main trunk of overlapping segment that are split and expanded into several branches. Each branch must consist of three kinds of tree segments, head, middle, and end segments. Typical records of three kinds of trees are shown in Table 5.18. The listed data fields of each tree segment are previously described in Table 4.6.

Table 5.18 Typical records of three kinds of tree segments

Field	Head segment	Middle segment in a branch	End segment of a branch
Number of segments of this kind	4 (3, 158, 687,945)	633	316
ID	3	4	6
Head	1	0	0

Orient	1	1	0
Begin	0	64	96
Next	5	7	-1
Child	-1	4	-1
Brother	-1	-1	-1
Script size	0	622	594
Alignment Script		0,0,1,0,0,0,–1,0,0,1,0,0,0,0,0,0,0,0,0,2,0,0,0, 0,0,0,0,0,–1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,	0,0,0,-1,0,-1,0,0,-1,0,0,-1,0,0,0,0,-1,0,0,0,0, 0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,0,- 1,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,- 2,0,0,-4,0,0,0,0,0,-2,0,-3,0,0,

There are 4 head segments (with ID 3, 158, 687, and 945) that expands to 316 ends of branches through 633 middle segments. For each head segment, the value of *head* field must be true, *begin* field, which is the starting position of previous segment, must be 0 (because there is no previous segment), and there is no alignment script because alignment script is the overlapping part between current segment and its previous segment. In addition, head segment should contain *next* segment even though *next* segment may be NULL (-1) in the case that head segment is also end segment and hence the contig is a singlet. Group number (which were initially set as –1) of corresponding contigs will be set incrementally from 1 for each head segment found.

Middle and end segments are non-head segments (*head*=0) and they are always followed other segments. The only difference between middle and end segments is that *next* of end segment is always NULL (-1) while that of middle

segment must be a valid ID. In general, head, middle, and end segment may consist of brother and child. Group number of contigs corresponding to middle and end segments connected to each head will be set to the same value as the group of its head segment. For both middle and end segments, their *begin* field must be greater than 0 (because there must be begin point from the previous segment), and there must be non-empty alignment script. Begin and alignment script are the parameters used for graph and final output assembly construction. It can be seen that alignment script mostly consists of 0 which corresponds to matches or substitution, some positive integers that are the number of insertion, and some negative integers that indicate the number of deletion.

Graph creation

Graphs of all group contigs are then generated and the details and statistics of all graphs are shown in Table 5.19. Four graphs corresponding to 4 assembled contigs are generated from 4 head segments with head ID of 3, 158, 687, and 945. The four tree-graphs with length of 16918, 754, 874, and 26769 consists of 157, 1, 5, and 153 branches including the main trunk, respectively. The corresponding main paths are composed of 282, 4, 2, and 353 main overlapping segments, in which 148, 2, 1, and 172 segments are negative (reverse complement).

Table 5.19 Details and statistics of all graphs

Head ID	Number of branches	Graph length (bases)	Number of main overlaps	Main overlap list
3	157	16918	282	3,5,-7,8,-10,-11,13,-14,-16,-17,18,-19,20,21,-23,-24,27,28,30,32,-33,35,-36,-37,38,-40,-42,44,-45,-46,-49,52,-53,54,-56,57,-59,58,62,-64,-67,66, 68,-70,-71,-74,-75,-76,77,78,79,-82,83,-84,85,87,88,90,91,-93,-95,-96, 99,-100,-101,103,-105,-107,108,111,113,116,118,-120,119,-121,-123,-124,-125,127,128,129,-131,132,-133,-134,-135,137,139,142,-144,-143, 145,-146,-149,151,152,153,154,155,159,-162,161,-163,-164,-167,168, 169,170,-173,175,-176,-178,181,182,-183,-185,-186,187,190,-191,-193, 194,195,-197,-198,-199,202,-203,-205,204,-206,207,-210,-213,215,-216,-217,220,-221,-222,223,-224,-226,225,228,-229,-230,231,-233,232,-234,-236,237,-238,-241,244,247,249,-251,250,-252,255,-256,257,-258,259,-260,261,-263,264,-265,266,267,268,-270,-271,-272,273,274,275,-277, 278,-279,-280,285,286,287,288,290,291,292,296,299,300,302,-305,306, 308,-310,-314,-316,317,318,320,324,-327,330,-332,-333,-334,-338,340, 341,-342,-343,-346,-348,-349,-351,-354,357,359,-361,-363,-364,366,-368,370,-372,-375,374,-376,377,-378,-379,-380,-381,-382,-383,-386,-384,387,-388,-390,391,393,-394, 396,-397,399,403,404,-405,-406,-407,-408,-409,-410,412,413,415,-416,417,418,420,422,-423,424,-425,-426,-428,-427,-430,-433,-434, 435,-436,-437,-438,440,442,
158	1	754	4	158,-626,25,-855,
687	5	874	2	687,-740,
945	153	26729	353	945,944,943,942,941,-939,937,-936,-935,933,932,930,929,-928,-927,-925,926,924,-923,921,-920,919,918,-917,-916,-915,-913,-912,-910, 909,-907,-906,903,-900,901,-899,897,896,-895,-892,893,891,-889,-888,887, 885,883,882,-881,880,-878,-875,870,-868,867,866,865,864,-863,861, 859,857,856,-853,854,-852,-851,-849,-850,-846,-845,844,-843,-841,840, 839,838,-837,-836,-834,-832,827,824,823,-822,-821, 820,-819,-818,-815, -814,813,812,811,810,-809,806,805,804,-803,-801, -800,799,-797,796, 795,-794,793,792,-791,-789,790,-787,-785,-786, 784,783,781,782,780, 779,778,777,-773,-772,770,769,767,765,-764, 763,-760,-759,756,757, 755,754,-753,751,-749,748,746,-745,-744,-743,742,739,-738,737,735, 734,-729,728,727,726,723,721,-717,716, 715,711,712,710,708,704,-703, 702,700,-699,697,-696,-695,-693,-692, -690,-689,-688,685,-684,682,-679,-678,-676,674,-671,-669,-668,-666, -665,-664,662,-661,-659,-658, 657,-656,-655,-654,653,-652,-651,-650,-647,644,-643,641,-640,638,-637, -635, -634,-632,630,-628,629,627,-625,624,-623,622,621,620,618,616,-615, 614,-613,612,610,608,-607,-606,605,-604,603,602,601,-600,597, 596,-594,-592,-591,593,-588,-587, 586,585,-584,-582,583,-581,579,577, 576, 574,-571,567,566,565,-561,-560,-559,-558,-555,556,-553,552,551,-549,-548,547,546,-542,-541,-539,-537,-536,535,534,532,531,-530,-526,-525,-523,-522,520,519,-518,517,516,515,514,-513,-510,508,507,506, 505,-504,502,503,501,-500,499,-498,497,495,494,493,-491,490,-489, 488,-487,-486,485,-483, 482,-481,-480,-479,477,-476,-475,474,-472,-470,-471, -469,-468,467,-466,465,-463,464,-462,-461,460,-459,458,-457, -455,453, 452,-451, 449,448,447,110,-946,-949,

Note: negative segment IDs denote the corresponding reverse complement and negative orientation.

Consensus output generation

The output contigs are constructed from the corresponding graphs with tree information. There are four output contigs corresponding to the four listed graphs. A

typical construction of consensus output of a contig is shown in Fig. 5.1. This contig is the shortest contig (contig 3) so it is most convenient to illustrate, while other contigs are too long to be displayed. The construction of output is reported by splitting the whole output sequence into 60-character group lines separated by semicolon dotted line. The consensus output is listed in italic font after aligned columns of overlapping input segments in each 60-base region.

```
vt14a02.y1+
TTCAGAGCTGAGCTCAATGTGGTGGAATTCGACTTTGGCCAGAGGGGCTAAGCAGTTGGT
consensus
TTCAGAGCTGAGCTCAATGTGGTGGAATTCGACTTTGGCCAGAGGGGCTAAGCAGTTGGT
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+
GGTACAGGNAAGNGNCNNANNTTTNNNNTTTNGNNNNNNNNNNNNNCNNNNNNTTTTTT
consensus
GGTACAGGNAAGNGNCNNANNTTTNNNNTTTNGNNNNNNNNNNNNNCNNNNNNTTTTTT
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+
TTNNNNNNNNNNNNNNNNNNAAAATTTTTTTTTTTTTNNNNNNNNNTNNNNNTTTTT
consensus
TTNNNNNNNNNNNNNNNNNNAAAATTTTTTTTTTTTTNNNNNNNNNTNNNNNTTTTT
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ TTTTTTTTTTCCCCCCCCCCCCNTTTTTTTTTCNNNNNTTGGAAAAAGAAAGT-
TGT-C
vt13g10.y1-                                          AAGGC-G--
GGT--
vt06c09.y1-                                           AGCA--
TCTCTAC
vt18d07.x1-                                           AGCA--
TCTTTAC
consensus   TTTTTTTTTTCCCCCCCCCCCCNTTTTTTTTTCNNNNNTTGGAAAAAGCA-
GTCTGTAC
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ ATTACNTNNTNNTT-C-ANTGNAA-T--TNTC--AACCAA-G---GCC-
AATTCCACCAA
vt13g10.y1- A--ACCTG-TT-TTAC-A-T-AAA-----GGC--ATTTA--G---GGA-AAGTG-
AGCTA
vt06c09.y1- ATC-CCTG-TGTTTGCCA-T-AAACTAATTTCTGAACAAATGTTAGTTGAATTA-
ATGAA
vt18d07.x1- ATA-CCTG-TGTTTGGCA-T-AAACTAATTTTTGACCAAATGTTAGTTGAATTA-
ATGAA
consensus   ATAACCTG-TGTTTGCCA-T-AAACTAATTTCTGAACAAATGTTAGTTGAATTA-
ATCAA
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ -AAAA-CAATGGGGGNGGGNCA---A-TTCA-AGNCA--AGA-A-A--
AAGGGGGNCAAG
vt13g10.y1- GAAAA-GATTC----C---TCA---CCTTCC-TTTCT--TTT-T-T--TATTT--
TCT-G
vt06c09.y1- -AAAAACCATAGACTT--TCCATGTA-
TTGATAGGTACCAGTCATAGCAATTTTGCCAAT
vt18d07.x1- -AAAAACCATAGCCTT--TGCATGTA-
TTGATAGGTACCAGTCATAGCAATTTTGCCAAT
```

```
consensus   -AAAAACAATAGACTT--TCCATGTA-
TTCATAGGCACCAGTCATAGCAATTTTGCCAAG
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ A--A-GAAT------G-AATG----AACCNTCNNNTTG-G-CCCCTTGNC----T-
CC-T
vt13g10.y1- A--A--A-T------G---T-----A----TCAG-T-G-G-TACCTGGAC----T-
GG--
vt06c09.y1- TTTATGATTCTTTCTGTAATGTGGGAA--AT-GTGTTGTGTCCCCCTGGAAAGAT-
CCCT
vt18d07.x1- TTTATGATTCTTTCTGTAATGTGGGAA--AT-
GTGTTGTGTCCCCCTGGAATGATGCCC-
consensus   ATTATGATTCTTTCTGTAATGTGGGAA--ATCGTGTTGTGTCCCCCTGGAAAGAT-
CCCT
              .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ ---TTCAAG--GG--GA-ACCC-----CCCAGG--G-T-CC-T-T--T----CT-T-
-CC
vt13g10.y1- ----TCAAA--GTA-GA-ACGATTATTCCCATG--AATGCCCTGTAATGATGCT-T-
-CC
vt06c09.y1- CAATTCCTTTTGGAAGAGACAG-----CT-ATTTTGATACCCT-TG-T-----
TATGACC
vt18d07.x1- CAATTCCTTTTGGAAGAGACAG-----CT-ATTTTGATAGCAT-TG-T-----
TATGACC
consensus   CAATTCAATTTGGAAGAGACAG-----CCCATGTTGATACCCT-TG-T----
CTATGACC
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ C--AATG--GGG------GT---G-GGNAAAAAAGAATCC---C--CCAA-TT-
GAA--T
vt13g10.y1- CCTAATGAAGGGCA----GT---TTGGTAAAA--G---CC---C--CCAA-AT-
CAAACT
vt06c09.y1- C--C-TG--GCCCCCCCAGTTTTG-GGGGTTAACCT-TCCAGGCAACCAGCTTTG--
--T
vt18d07.x1- C--C-TG--GCCCCCCCAGCTTTG-GGGATTAACCT-TCCAGGCAACCAGCTTTG--
--T
vt01e12.y1+                                                   TG-
GAA--T
vt16b09.x1+                                                   TA-
CAA--C
consensus   C--AATG--GCCCCCCCAGTTTTG-GGGAAAAACCT-TCCAGGCAACCAACTT-
GAA--T
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ -TA-ACAA---A-TAAAAATA-T-A----C-T---TT--C-A--GCCCA-CC--
CCAAGT
vt13g10.y1- -TT-ACA------TCAAAATCCTGATGTGCATG--TTTTCTAAAGCC-
ATCCTTCCCACT
vt06c09.y1- GTA-G-AAGGGAGTTAAC-TG-T------C-TCCCTT--G-T--GTCCA-TC--C-
A-GC
vt18d07.x1- GTA-G-AAGGGAGTTAAC-TG-T------T-TCCCTT--G-T--GTCCA-TC--C-
A-GC
vt01e12.y1+ -TATACC------TGCCTCTT-T-A----C-T---AT--G-N--GCTGA-TG-----
TGT
vt16b09.x1+ -T-T-CC------TGC-TCTT-T-A----C-T---AT--G----GCTGA-TG-----
TGT
consensus   -TA-ACAA---A-TGAAAATG-T-A----C-TC--TT--G-A--GCCCA-TC--C-
AAGT
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ -ATTCCAC-CC-CCA-TTT-TT---A-A-A--A---A-ATT-TG----G--
GAAAGGAAT
vt13g10.y1- GATAACAT-CTTCCAGTTTCTTTTGATACAGTAGGTA-ATT-
TGCCATGTTGGGAGGCAT
vt06c09.y1- --T-CAAAACC-CC--TTT-TT---A-G-G--G---C-CCT-TC----G--GTTT--
AAT
```

```
vt18d07.x1- --T-CAAAACC-CC--TCT-TT---A-G-G--G---C-CCT-TG----G--GTTT--
AAT
vt01e12.y1+ TNTTG-AC-GC---A-GGT-C----A-A-A--AG--ACAGCCTG----G--
TCAACGACT
vt16b09.x1+ T-TTG-AC-GC---A-GGT-C----A-A-A--AG--ACAGCCTG----G--
TCAACGACT
consensus   TATTCAAC-CC-CCA-TTT-TT---A-A-A--AG--A-ACT-TG----G--
GCAACGAAT
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ T------T-T-T---C-CA--CCT-CC-C-T--TGGGAAAAAAAA-AAGAAAA-A--
-A-
vt13g10.y1- T------T-TCTTTGCTCA--GTT-CATC-TGATCTGAAACAAGG---GTAAT-G--
---
vt06c09.y1- T------T-T----------C-T------T--TAAAAAAAAAAA-AAAAAAA-A--
-A-
vt18d07.x1- T------T-T----------C-C------T--AAAAAAAAAAAA-AAAAAAA-A--
-A-
vt01e12.y1+ TCCACCATGT-T---C-CATTCCTTCC-CAT--
TCTGAGAGTGAATAATAATTCACTCAC
vt16b09.x1+ TCCACCATGT-T---C-CATTCCTTCC-CAT--
TCTGAGAGTGAATAATAATTCACTCAC
consensus   T------T-T-T---C-CA--CCT-CC-C-T--TCTGAAAAAAAA-AAAAAAA-A--
-A-
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ TG-GGGGCTT---TT--T-TTCCCC-C-CT-TGG-A----AG-GAAACAAAA---
TT-TT
vt13g10.y1- TGTGGGG-TGGAATTGCT-TTCAGGACACTGTGGCATTTTAGTGAAAAAAAA---
AA-AA
vt06c09.y1- AA-AAAAAAA---AA--A-AAAAAA-A-AA-AAA-A----AA-AAAAAAAAA-
GCTTATT
vt18d07.x1- AA-AAAAAAA---AA--A-AAAAAA-A-AA-AAA-A----AA-
AAAAAAAAAAGCTTATT
vt01e12.y1+ TC-TGAA-TA---AT--TATTCACT-C--T-----C----AG--AATCCA-------
--T
vt16b09.x1+ TC-TGAA-TA---AT--TATTCACT-C--T-----C----AG--AATCCA-------
--T
consensus   TA-AGAAATA---AT--T-TTCACA-C-AT-AAA-A----AG-AAAAAAAAA---
TT-TT
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ C--CC--C-CGG-GTTT-T--TTCA-AAN-T-TT----T-TTAA-A--A---ATG--
-GG
vt13g10.y1- AGACCAGC-CTGTGTGGCTCATTCT-AA--T-TT----A-TCAA-A--A-----G--
-CT
vt06c09.y1- T--GC--CAC----
TTTCTAGTTCATAAAATGTTAGAGTATTAATATCAGGGATGCCTGG
vt18d07.x1- T--GC--CAC----
TTTCTAGTTCATAAAATGTTAGAGTATTAATATCAGGGATGCCTGG
vt01e12.y1+ C--CT--T-CGA-ATTTCTG-TTCA-A---T-TT----T-T-------C----T---
--G
vt16b09.x1+ C--CT--T-CGA-ATTTCTG-TTCA-A---T-TT----T-T-------C----T---
--G
consensus   C--CC--C-CGA-ATTTCTAGTTCA-AAA-T-TT----T-TTAA-A--A---ATG--
-GG
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ GCCC-CC----C-A-CAA---A-A---A--AAT---T---TTCC-CCAAGGG-G-G-
---
vt13g10.y1- GCCAGCCTTGGCTA-CACTGCAGATCTAGCAAT---C---TACT-TCAATG------
---
vt06c09.y1- GTGG-CT----C-AGCGG---T-T---T--
AGCGCCTGCCTTCCGCCCAGGGCGTGATCC
```

```
vt18d07.x1- GTGG-CT----C-AGCGG---T-T---T--
AGCGCCTGCCTTCCGCCCAGGGCGTGATCC
vt01e12.y1+ -CTC-CT----C-TTCAT---C-A---A--AAT---T---TTCT-TCA-GTG-T-T-
---
vt16b09.x1+ -CTC-CT----C-TTCAT---C-A---A--AAT---T---TTCT-TCA-GTG-T-T-
---
consensus   GCCC-CT----C-AGCAG---A-A---A--AAT---T---TTCC-CCAAGGG-G-G-
---

                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ -GG-G----GGGAA--AT--T-CCTT--T--T-C---T-T-----TTAAAAC-CCC-
--C
vt13g10.y1- --T-A-------AAC-AT--T---TTAAT--TAC---TAT-----TTCTAAC-
ATTTGGC
vt06c09.y1- TGGAGTCCCGGGATGGAG--T-CCCA--GGAT-CGAGT-TCCGCGTTGGG-
CTCCCA--C
vt18d07.x1- TGGAGTCCCGGGATGGAG--T-CCCA--GGAT-CGAGT-TCCGCGTTGGG-
CTCCCA--C
vt01e12.y1+ -AT-C----TAGAG--TTGCTGCCTT--T--A-C---T-T-----TTT---C-T---
---
vt16b09.x1+ -AT-C----TAGAG--TTGCTGCCTT--T--A-C---T-T-----TTT---C-T---
---
consensus   -GG-G----GGGAAG-AT--T-CCTT--T--T-C---T-T-----TTGGA-C-
CCCA--C
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ TT--T--T-T-----T-TA-A-AAAA-C-A--TTTTTTTTCCCCCCCTTT-TTT-T-
---
vt13g10.y1- TT--TCAT-TAAAAGT-TAGAGAAAAACTAAATTTTTTTTCATAAAGTAA-
ATACTGGAA
vt06c09.y1- TGCCTA-TGTC----TCTG-C------C----TCTCTTT-CTCCC--TGT-GTC-
TC---
vt18d07.x1- TGCCTA-TGTC----TCTG-C------C----TCTCTTT-CTCCC--TGT-GTC-
TC---
vt01e12.y1+ TT--TC-T-T-----T-T--------------TTTTTTTTAAGA--TTTTATTTAT-
---
vt16b09.x1+ TT--TC-T-T-----T-T--------------TTTTTTTTAAGA--TTTTATTTAT-
---
consensus   TT--TC-T-TC----T-TA-A------C----TTTTTTTTCACAC-TTTT-
TTTATC---
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ ---T--T---T---TAAA-AGGGGCC-ACAAAAA-AA-A--A-AGAG----GGT--
GTT-
vt13g10.y1- AACTCCT---T---TAAA-AAGC-CC-ACCCCCA-AC-A--C-ACAA----T-T--
GCA-
vt06c09.y1- ---TCATGAATAAATAAATAAAATCTTAAAAAAAGAATATTA-ATATCAAAGGCA-
GTTG
vt18d07.x1- ---TCATGAATAAATAAATAAAATCTTAGGAAAAGAATATTA-
ATATCAAAGCCTCGTTT
vt01e12.y1+ ---T--TA--T---TCATGAGAG----ACAGAGA-GA-G--AGAGAGNNCNG
vt16b09.x1+ ---T--TA--T---TC-TGA-A-----AC
consensus   ---TCATA--T---TAAAGAAAATCC-ACAAAAA-AA-A--A-AGAGCAAAGGCA-
GTTG
                .    :    .    :    .    :    .    :    .    :    .
:
vt14a02.y1+ CTTT-T-----TTTT--TGG----GGGGAAAAAA-A-A-A
vt13g10.y1- CACA-C-----TCTC---GG----G---AACACACACACACGGTTCTACGTAGA
vt06c09.y1- CAGCATCAAGATTTTCCTGGAACTGG
vt18d07.x1- CT
consensus   CACA-T-----TTTT--TGG----GGGGAAAAAACACACACGGTTCTACGTAGA
```

Fig. 5.1 Typical consensus construction of the output contig

```
TTCAGAGCTGAGCTCAATGTGGTGGAATTCGACTTTGGCCAGAGGGGCTAAGCAGTTGGT
GGTACAGGNAAGNGNCNNANNTTTNNNNTTTNGNNNNNNNNNNNNNCNNNNNNTTTTTT
TTNNNNNNNNNNNNNNNNNNAAAATTTTTTTTTTTTTTNNNNNNNNNTNNNNNTTTTT
TTTTTTTTTTCCCCCCCCCCCCNTTTTTTTTTCNNNNNNTTGGAAAAAGCAGTCTGTAC
ATAACCTGTGTTTGCCATAAACTAATTTCTGAACAAATGTTAGTTGAATTAATCAA
AAAAACAATAGACTTTCCATGTATTCATAGGCACCAGTCATAGCAATTTTGCCAAG
ATTATGATTCTTTCTGTAATGTGGGAAATCGTGTTGTGTCCCCCTGGAAAGATCCCT
CAATTCAATTTGGAAGAGACAGCCCATGTTGATACCCTTGTCTATGACC
CAATGGCCCCCCCAGTTTTGGGGAAAAACCTTCCAGGCAACCAACTTGAAT
TAACAAATGAAAATGTACTCTTGAGCCCATCCAAGT
TATTCAACCCCCATTTTTAAAAGAACTTGGGCAACGAAT
TTTTCCACCTCCCTTCTGAAAAAAAAAAAAAAAAA
TAAGAAATAATTTTCACACATAAAAAGAAAAAAAAATTTT
CCCCCGAATTTCTAGTTCAAAATTTTTTAAAAATGGG
GCCCCTCAGCAGAAAAATTTTCCCCAAGGGGG
GGGGGGAAGATTCCTTTTCTTTTGGACCCCAC
TTTCTTCTTAACTTTTTTTTCACACTTTTTTTATC
TCATATTAAAGAAAATCCACAAAAAAAAAAAGAGCAAAGGCAGTTG
CACATTTTTTGGGGGGAAAAAACACACACGGTTCTACGTAGA
```

Fig. 5.2 Final consensus output (corresponding to Fig. 5.1)

From Fig. 5.1, columns of consensus output can be classified into two basic kinds. The first kind is unanimous consensus output base. It is the consensus output column that is contributed from all identical aligned input characters. In other word, there is no mismatch among aligned input characters and there is no contradiction in producing consensus output base. The output column that consists of only one aligned input character is automatically the unanimous base column. Second, non-unanimous consensus base is the consensus base that is produced when all aligned input column are not the same. The consensus base is computed from quality score of all input bases in the aligned column as described at the end of section 4.3.5. It can be seen that final consensus base is mostly the base that appears the most times in the input column. If there are more than one bases that appear equally the most times in the column, the quality score will decide which base will be the final result. The special kind of output column is indel consensus output, which is normally produced when indels (-) are mostly produced at the aligned column. The indel output bases are removed from the final output as shown in Fig. 5.2.

Table 5.20 Statistics of all output contigs of *Canis familiaris*

Field	Contig1	Contig2	Contig3	Contig4
Root ID	3	158	687	945
Number of segments	282:156	4:0	2:4	353:152

overlaps: containments				
Length (with no indels)	17443	929	881	27431
Total number of consensus bases (with indels)	19054	1086	1134	29634
Number of unanimous consensus bases	13154	293	409	21738
Number of non-uanimous bases (excluding indels)	4289	636	472	5693
Number of indels consensus base	1611	157	253	2203

To see all results, the statistics of all output contigs of *Canis familiaris* are listed in Table 5.20. It should be noted that the length of final contig output is equal to total number of consensus bases minuses the number of indels. It can be seen that there are two large contigs and two small contigs. Two large contigs (contig 1 and 4) are produced from a large number of overlaps and containments (282:156 and 353:152, respectively). In contig 1 that consists of 19054 total consensus bases, 13154 bases are unanimous, while 4289 bases are not and 1611 bases are indels. Similarly, contig 4 contains 29634 total consensus bases, 21738 unanimous bases, 5693 non-unanimous bases, and 2203 indels bases. On the other hand, two small contigs (contig 2 and 3) are created from a few numbers of overlaps and containments (4:0 and 2:4, respectively). The contig 2 and 3 consists of 1086 and 1134 total consensus bases, 293 and 409 unanimous bases, 636 and 472 non-unanimous bases, and 157 and 253 indels bases, respectively. It can be noticed that these small contigs contain relatively high percentage of nonunanimous and indel consensus base compared to large contigs. The relatively high percentage of nonunanimous and indel can be an indication of possible misassembly.

Table 5.21 Statistics of all output contigs of *Carollia perspicillata*

Field	Contig1	Contig2	Contig3
Root ID	3	70	346
Number of segments overlaps: containments	117:738	3:0	87:655
Length (with no	17802	1721	12912

indels)			
Total number of consensus bases (with indels)	19533	1982	15391
Number of unanimous consensus bases	12113	745	7182
Number of non-uanimous bases (excluding indels)	5689	976	5730
Number of indels consensus base	1731	261	2479

The statistics of all consensus output contigs for *Carollia perspicillata* are shown in Table 5.21. It can be seen that there are two large contigs and one small contigs. Two large contigs (contig 1 and 3) are produced from a large number of overlaps and containments (117:738 and 87:655, respectively). In contig 1 that consists of 19533 total consensus bases, 12113 bases are unanimous, while 5689 bases are not and 1731 bases are indels. Similarly, contig 3 contains 15391 total consensus bases, 7182 unanimous bases, 5730 non-unanimous bases, and 2479 indels bases. On the other hand, the small contig (contig 2) is created from a few numbers of overlaps and containments (3:0). The contig 2 consists of 1982 total consensus bases, 745 unanimous bases, 976 non-unanimous bases, and 261 indels bases, respectively. It can be noticed that this small contig contains very high percentage of nonunanimous and indel consensus base compared to large contigs. The high percentage of nonunanimous and indel can be a strong sign of possible misassembly.

Table 5.22 Statistics of all output contigs of *Dasypus novemcinctus*

Field	Contig1	Contig2	Contig3	Contig4
Root ID	2	6	104	478
Number of segments overlaps: containments	315:353	3:0	87:150	4:8
Length (with no indels)	21779	1165	7798	1992
Total number of consensus bases (with indels)	23561	1247	8715	2016
Number of unanimous	17325	632	5905	1958

consensus bases				
Number of non-uanimous bases (excluding indels)	4454	533	1893	34
Number of indels consensus base	1782	82	917	24

The statistics of all consensus output contigs of *Dasypus novemcinctus* are shown in Table 5.22. It can be seen that there are one large (>20000 bps), one medium (>2500 bps), and two small contigs. The large and medium contigs (contig 1 and 3) are produced from a number of overlaps and containments (315:353 and 87:150, respectively). In the large contig that consists of 23561 total consensus bases, 17325 bases are unanimous, while 4454 bases are not and 1782 bases are indels. Similarly, the medium contig contains 8715 total consensus bases, 5905 unanimous bases, 1893 non-unanimous bases, and 917 indels bases. On the other hand, the small contigs (contig 2 and 4) are created from a few numbers of overlaps and containments (3:0 and 4:8). The contig 2 consists of 1247 total consensus bases, 632 unanimous bases, 533 non-unanimous bases, and 82 indels bases, respectively. Similarly, the contig 4 consists of 2016 total consensus bases, 1958 unanimous bases, 34 non-unanimous bases, and 24 indels bases, respectively. It can be noticed that the contig 2 contains very high percentage of nonunanimous and indel consensus base compared to other contigs. The high percentage of nonunanimous and indel suggests that contig 2 is possibly a misassembly.

Table 5.23 Statistics of all output contigs of *Equus caballus*

Field	Contig1	Contig2	Contig3	Contig4
Root ID	3	20	179	250
Number of segments overlaps: containments	317:296	120:133	2:9	15:28
Length (with no indels)	25196	10201	944	2177
Total number of consensus bases (with indels)	26849	11129	1079	2446
Number of unanimous consensus bases	19972	7897	391	1534
Number of non-	5224	2304	553	643

uanimous bases (excluding indels)				
Number of indels consensus base	1653	928	135	269

The statistics of all consensus output contigs of *Equus caballus* are shown in Table 5.23. It can be seen that there are one large (>20000 bps), one medium (>2500 bps), and two small contigs. The large and medium contigs (contig 1 and 2) are produced from a number of overlaps and containments (317:296 and 120:133, respectively). In the large contig that consists of 26849 total consensus bases, 19972 bases are unanimous, while 5224 bases are not and 1653 bases are indels. Similarly, the medium contig contains 11129 total consensus bases, 7897 unanimous bases, 2304 non-unanimous bases, and 928 indels bases. On the other hand, the small contigs (contig 3 and 4) are created from a few numbers of overlaps and containments (2:9 and 15:28). The contig 3 consists of 1079 total consensus bases, 391 unanimous bases, 553 non-unanimous bases, and 135 indels bases, respectively. Similarly, the contig 4 consists of 2446 total consensus bases, 1534 unanimous bases, 643 non-unanimous bases, and 269 indels bases, respectively. It can be noticed that the contig 3 and 4 contains very high percentage of nonunanimous and indel consensus base compared to other contigs. The high percentage of nonunanimous and indel suggests that contig 3 and 4 are possible misassemblies.

Table 5.24 Statistics of all output contigs of *Felis catus*

Field	Ct1	Ct2	Ct3	Ct4	Ct5	Ct6	Ct7	Ct8	Ct9	Ct10	Ct11	Ct12
Root ID	3	22	113	114	158	355	387	426	434	500	557	587
No. of segments overlaps: containments	263: 328	7: 10	4: 7	2: 18	3: 12	2: 14	5: 14	1: 0	44: 31	42: 53	12: 21	51: 20
Length (with no indels)	20507	2199	1513	850	1267	831	949	690	6565	4659	2079	4555
Total number of consensus bases (with indels)	21776	2577	1734	1065	1570	1045	1497	690	6765	5465	2123	4693
No. of unanimous consensus bases	16388	1175	873	465	628	334	258	690	5775	3172	1860	3999

No. of non-uanimous bases	4119	1024	640	385	639	497	664	0	790	1487	219	556
No. of indels consensus base	1269	378	221	215	303	214	548	0	200	806	44	138

The statistics of all consensus output contigs of *Felis catus* are shown in Table 5.24. For *Felis catus*, SEQASS produces 12 contigs, which is considerably more than other data. The possible reason for many-contigs problem could be due to poor quality of input data as previously noted. It can be seen that there are one large (>20000 bps), three medium (>2500 bps), and eight small contigs. The large and medium contigs (contig 1, 9, 10, and 12) are produced from a number of overlaps and containments (263:328, 44:31, 42:53, and 51:23, respectively). In the large contig that consists of 21776 total consensus bases, 16388 bases are unanimous, while 4119 bases are not and 1269 bases are indels. Similarly, three medium contigs (contig 9, 10, 12) contain 6765, 5465, 4693 total consensus bases, 5775, 3172, 3999 unanimous bases, 790, 1487, 556 non-unanimous bases, and 200, 806, 138 indels bases.

On the other hand, the small contigs (contigs 2-8 and 11) are created from a few number of overlaps and containments. These small contigs have the total number of consensus bases in the range from 2577 to 690, the number of unanimous bases in the range from 1860 to 258, the number of non-unanimous bases in the range from 1024 to 219, and the number of indels bases in the range from 548 to 0, respectively. It should be noted that the contig 8, which contain one overlap, no containment, and no indels, is a singlet (unsequenchable read). It can also be noticed that most small contigs (except the singlet) contain very high percentage of nonunanimous and indel consensus base compared to larger contigs (except contig 10). The high percentage of nonunanimous and indel suggests that these small contigs are possible misassemblies.

Table 5.25 Statistics of all output contigs of *Gallus gallus*

Field	Contig1	Contig2	Contig3
Root ID	3	343	387
Number of segments overlaps: containments	397:411	49:95	2:0
Length (with no indels)	33144	3315	906
Total number of	34680	3665	906

consensus bases (with indels)			
Number of unanimous consensus bases	27657	2409	521
Number of non-uanimous bases (excluding indels)	5487	906	385
Number of indels consensus base	1536	350	0

The statistics of all consensus output contigs of *Gallus gallus* are shown in Table 5.25. It can be seen that there are one large, one medium, and one small contigs. The large and medium contigs (contig 1 and 2) are produced from a number of overlaps and containments (397:411 and 49:95, respectively). In contig 1 that consists of 34680 total consensus bases, 27657 bases are unanimous, while 5487 bases are not and 1536 bases are indels. Similarly, contig 2 contains 3665 total consensus bases, 2409 unanimous bases, 906 non-unanimous bases, and 350 indels bases. On the other hand, the small contig (contig 3) is created from a few numbers of overlaps and containments (2:0). The contig 2 consists of 906 total consensus bases, 521 unanimous bases, 385 non-unanimous bases, and no indels base, respectively. It can be noticed that the medium and small contigs contain high percentage of nonunanimous and indel consensus base compared to large contigs. The high percentage of nonunanimous and indel can be a strong sign of possible misassembly.

5.7 Results and statistics from contig alignment with clonemate information

The output contig are then finally linked by clonemate information as described in Section 4.3.6. The clonemate distance constraint between a pairmate (d+e in Fig. 4.22) was determined by finding positions of all reads in whole genome by BLAT comparison and calculate distance between far ends of the pair, which was done by our c program included in the contig alignment stage. It should be noted that the valid clonemate pair is identified when they are found in the whole genome with opposite orientations. The maximum and minimum clonemate distant constraints are

estimated to be 3693 and 1068 bases, respectively. The details and statistics of contig alignment with clone mate information of *Canis familiaris* are shown in Table 5.26.

In the four contigs, there are 195, 0, 0, and 228 clonemate pairs in contigs 1 to 4, respectively. Among these clonemate pairs, 182, 0, 0, and 215 pairs satisfy the clonemate constraints. Thus, more than 90% of clonemate pair are good, which indicates that the assembly of reads within the contig is mainly valid. For intercontigs, there are 0, 0, 12, 0, 0, and 4 clonemate pairs linked between 1-2, 1-3, 1-4, 2-3, 2-4, and 3-4 contigs, respectively. The corresponding 0, 0, 11, 0, 0, and 3 pairs satisfy the constraints. Thus, there should be two valid links between 1-4 and 3-4 contigs. The details of crosslink information between these contigs are then detailed to determine final linking description.

Table 5.26 Details and statistics of contig alignment of *Canis familiaris*

Clonemate distant constraint between a pair mate		Max	3693	Min	1068
Field	Contig pair	Contig1	Contig2	Contig3	Contig4
Root ID		3	158	687	945
Number of clonemate pair linked across to other contigs: Note link to its own is clonemate within a contig	1	195	0	0	12
	2	0	0	0	0
	3	0	0	0	4
	4	12	1	4	228
Number of clonemate pair linked across to other contigs that satisfies constraints	1	182	0	0	11
	2	0	0	0	0
	3	0	0	0	3
	4	11	0	3	215

Cross linked output contig	Cross link information	Linking parameters		
		First contig	Second contig	Intercontig distance
Contig 1 vs. Contig 4	(418[14769],446[25966]):2912 (422[14808],447[25894]):2945 (429[15344],449[25598]):2705 (432[15565],448[25861]):2221 (435[15604],460[25079]):2964 (440[15935],454[25573]):2139 (439[16040],464[24925]):2682 (441[15964],453[25468]):2215 (445[15946],458[25304]):2397 (442[16264],456[25501]):1882	3’ end: forward	3’ end: reverse	1811 bps
Contig 3 vs. Contig 4	(-443[106],467[24758]):2739	5’end forward	3’end forward	954 bps
	(950[30],-923[1871]):2900 (952[11],-935[1330]):2322	5’ end: forward	5’end: forward	Invalid

Note: Each line item in the crosslink information is the pair of reads linked between contigs. The format is (read ID in first contig [starting position in this contig], read ID in second contig [starting position in this contig]): pairmate distance.

From the crosslink information of 1-4 contigs, it can be seen that read IDs in contig 1 and contig 4 are in the same orientation (both positive or negative). This indicates that contig 1 and 4 are connected in opposite directions because the clonemate pair will be connected along the same line when they are in opposite directions. Using starting position of linked reads in contigs along with graph information (in Table 5.14), it can be identified that 3' end of Contig 1 (in forward direction) connects to 3' end of contig 4 (in reverse direction) within the intercontig distance of 1811 bases. The layout of contig linking is shown in Fig. 5.3 (a). It should be noted that the range of intercontig distance is estimated from the difference between the maximum allowed clonemate distance and the minimum pairmate distance of all linked pairs. In addition, the other three cases of valid contig linking are shown in Fig. 5.3 (b)-(d).

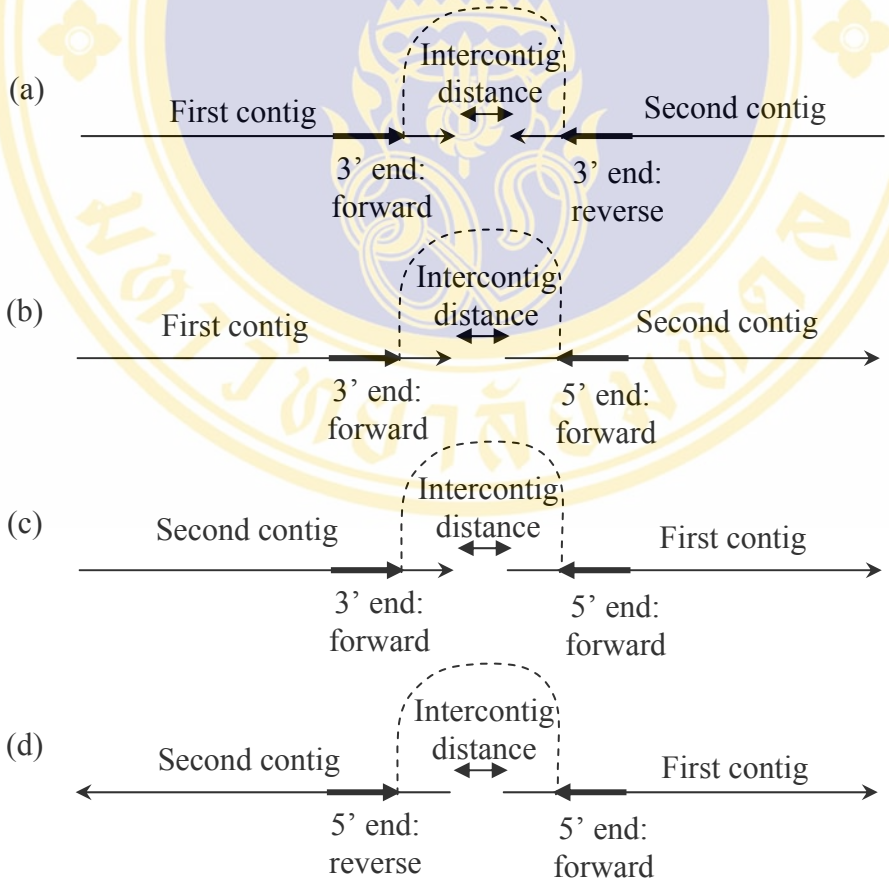

Fig. 5.3 Layout of valid intercontig linking

For the crosslink information of 3-4 contigs, there are two different linking cases. In the first case, it can be determined that 5' end of Contig 3 (in forward

direction) connects to 3’ end of contig 4 (in forward direction) within the intercontig distance of 954 bases (see Fig. 5.3 (c)). In the second case, two clonemate linking information suggests that 5’ end of Contig 3 (in forward direction) connects to 5’ end of contig 4 (in forward direction). This is invalid case. Thus, the 3-4 contig linking contains conflict information and it should be ignored.

It can be seen that the output contig linking information provide relative orientation between each contig pair and approximate distances between them because there is uncertainty of clonemate distance constraints that allow a variation of clonemate distance. Thus, this information cannot produce exact final contig assembly. However, it is useful for additional reassembly in which more reads are acquired around intercontig regions in the final finishing process.

The details and statistics of contig alignment with clonemate information of *Carollia perspicillata* are shown in Table 5.27. The maximum and minimum clonemate distant constraints for this data are estimated to be 5863 and 2720 bases, respectively. In the three contigs, there are 119, 0, and 105 clonemate pairs in contigs 1 to 3, respectively. Among these clonemate pairs, 103, 0, and 90 pairs satisfy the clonemate constraints. Thus, more than 90% of clonemate pair are good, which indicates that the assembly of reads within the contig is mainly valid. For intercontigs, there are 0, 1, and 2 clonemate pairs linked between 1-2, 1-3, and 2-3 contigs, respectively. The corresponding 0, 1, and 2 pairs satisfy the constraints. Thus, there should be two links between 1-3 and 2-3 contigs.

From the crosslink information of 1-3 contigs, it can be seen that read IDs in contig 1 and contig 3 are in the same orientation (both positive or negative). This indicates that contig 1 and 3 are connected in opposite directions because the clonemate pair will be connected along the same line when they are in opposite directions. Thus, it can be identified that 5’ end of Contig 1 (in forward direction) connects to 3’ end of contig 3 (in reverse direction). This is an invalid case and this contigs linking must be disregarded.

Table 5.27 Details and statistics of contig alignment of *Carollia perspicillata*

Clonemate distant constraint between a pair mate		Max	5863	Min	2720
Field	Contig pair	Contig1	Contig2	Contig3	
Root ID		3	158	687	
Number of clonemate pair linked across to other contigs: Note link to its own is clonemate within a contig	1	119	0	1	
	2	0	0	2	
	3	1	2	105	
Number of clonemate pair linked across to other contigs that satisfies constraints	1	103	0	1	
	2	0	0	2	
	3	1	2	90	

Cross linked output contig	Cross link information	Linking parameters		
		First contig	Second contig	Intercontig distance
Contig 1 vs. Contig 3	(59[1828],122[11040]):4254	5’ end: forward	3’ end: reverse	Invalid
Contig 2 vs. Contig 3	(627[3],600[10181]):4158	5’ end: forward	3’ end: reverse	Invalid
	(620[305],-626[11169]):2868	5’ end: forward	3’ end: forward	2995

Note: Each line item in the cross link information is the pair of reads linked between contigs. The format is (read ID in first contig [starting position in this contig], read ID in second contig [starting position in this contig]): pairmate distance.

For the crosslink information of 2-3 contigs, there are two different linking cases. In the first case, the clonemate linking information suggests that 5’ end of Contig 2 (in forward direction) connects to 3’ end of contig 3 (in reverse direction). This is also an invalid case. In the second case, it can be determined that 5’ end of Contig 2 (in forward direction) connects to 3’ end of contig 3 (in forward direction) within the intercontig distance of 2995 bases (refer to Fig. 5.3 (c)). Thus, the 2-3 contig linking contains conflict information and it should be ignored.

The details and statistics of contig alignment with clone mate information of *Dasypus novemcinctus* are shown in Table 5.28. The maximum and minimum clonemate distant constraints for this data are estimated to be 5451 and 2542 bases, respectively. In the four contigs, there are 204, 0, 56, and 0 clonemate pairs within

contigs 1 to 4, respectively. Among these clonemate pairs, 193, 0, 53, and 0 pairs satisfy the clonemate constraints. Thus, more than 90% of clonemate pair are good, which indicates that the assembly of reads within the contig is mainly valid. For intercontigs, there is only one clonemate pair linked between 3-4 contig. However, this pair does not satisfy the constraints. Thus, there is no valid link between these contigs and output of clonemate linking is none for this data.

Table 5.28 Details and statistics of contig alignment of *Dasypus novemcinctus*

Clonemate distant constraint between a pair mate		Max	5451	Min	2542
Field	Contig pair	Contig1	Contig2	Contig3	Contig4
Root ID		3	158	687	945
Number of clonemate pair linked across to other contigs: Note link to its own is clonemate within a contig	1	204	0	0	0
	2	0	0	0	0
	3	0	0	56	1
	4	0	0	1	0
Number of clonemate pair linked across to other contigs that satisfies constraints	1	193	0	0	0
	2	0	0	0	0
	3	0	0	53	0
	4	0	0	0	0
Cross linked output contig	Cross link information			Linking parameters	
None	None			None	

Note: Each line item in the crosslink information is the pair of reads linked between contigs. The format is (read ID in first contig [starting position in this contig], read ID in second contig [starting position in this contig]): pairmate distance.

The details and statistics of contig alignment with clone mate information of *Equus caballus* are shown in Table 5.29. The maximum and minimum clonemate distant constraints for this data are estimated to be 5621 and 2886 bases, respectively. In the four contigs, there are 186, 68, 0, and 0 clonemate pairs in contigs 1 to 4, respectively. Among these clonemate pairs, 176, 67, 0, and 0 pairs satisfy the

clonemate constraints. Thus, more than 90% of clonemate pair are good, which indicates that the assembly of reads within the contig is mainly valid. For intercontigs, there is no clonemate pair linked between these contigs. Thus, there is no possible link between these contigs.

Table 5.29 Details and statistics of contig alignment of *Equus caballus*

Clonemate distant constraint between a pair mate		Max	5621	Min	2886
Field	Contig pair	Contig1	Contig2	Contig3	Contig4
Root ID		3	20	179	250
Number of clonemate pair linked across to other contigs: Note link to its own is clonemate within a contig	1	186	0	0	0
	2	0	68	0	0
	3	0	0	0	0
	4	0	0	0	0
Number of clonemate pair linked across to other contigs that satisfies constraints	1	176	0	0	0
	2	0	67	0	0
	3	0	0	0	0
	4	0	0	0	0
Cross linked output contig	Cross link information			Linking parameters	
None	None			None	

Note: Each line item in the crosslink information is the pair of reads linked between contigs. The format is (read ID in first contig [starting position in this contig], read ID in second contig [starting position in this contig]): pairmate distance.

The details and statistics of contig alignment with clone mate information of *Felis catus* are shown in Table 5.30. The maximum and minimum clonemate distant constraints for this data are estimated to be 3334 and 1391 bases, respectively. It can be seen that there are 155, 19, 14, and 19 clonemate pairs within large and medium contigs 1, 9, 10, and 12, respectively but there is no clonemate pairs within any small contigs. Among these clonemate pairs, 148, 19, 13, and 18 pairs satisfy the clonemate constraints. Thus, more than 90% of clonemate pair are good, which again indicates that the assembly of reads within the contigs are mainly valid.

Table 5.30 Details and statistics of contig alignment of *Felis catus*

Clonemate distant constraint between a pair mate			Max			3334			Min			1391	
Field	Contig pair	Ct1	Ct2	Ct3	Ct4	Ct5	Ct6	Ct7	Ct8	Ct9	Ct10	Ct11	Ct12
Root ID		3	22	113	114	158	355	387	426	434	500	557	587
Number of clonemate pair linked across to other contigs: Note link to its own is clonemate within a contig	1	155	0	0	0	0	0	0	0	0	1	0	0
	2	0	0	0	0	0	0	0	0	0	1	0	0
	3	0	0	0	0	0	0	0	0	2	2	0	0
	4	0	0	0	0	1	0	0	0	0	0	2	1
	5	0	0	0	1	0	0	0	0	0	1	2	0
	6	0	0	0	0	0	0	0	0	0	0	0	4
	7	0	0	0	0	0	0	0	0	0	1	1	0
	8	0	0	0	0	0	0	0	0	1	0	0	0
	9	0	0	2	0	0	0	0	1	19	1	0	0
	10	1	1	2	0	1	0	1	0	1	14	0	0
	11	0	0	0	2	2	0	1	0	0	0	0	0
	12	0	0	0	1	0	4	0	0	0	0	0	19
Number of clonemate pair linked across to other contigs that satisfies constraints	1	148	0	0	0	0	0	0	0	0	1	0	0
	2	0	0	0	0	0	0	0	0	0	1	0	0
	3	0	0	0	0	0	0	0	2	2	0	0	0
	4	0	0	0	0	1	0	0	0	0	0	1	1
	5	0	0	0	1	0	0	0	0	0	1	2	0
	6	0	0	0	0	0	0	0	0	0	0	0	4
	7	0	0	0	0	0	0	0	0	0	1	0	0
	8	0	0	0	0	0	0	0	0	1	0	0	0
	9	0	0	2	0	0	0	0	1	19	1	0	0
	10	1	1	2	0	1	0	1	0	1	13	0	0
	11	0	0	0	1	2	0	0	0	0	0	0	0
	12	0	0	0	1	0	4	0	0	0	0	0	18

Table 5.31 Contig alignment output of *Felis catus*

Cross linked output contig	Cross link information	Linking output		
		First contig	Second contig	Intercontig distance
Ct 1 vs. Ct 10	(-30[850],+0[3631]):2594	5’ end: forward	3’ end: forward	740 bps.
Ct 2 vs. Ct 10	(580[955],551[3320]):2343	5’ end: forward	3’end: reverse	Invalid

Ct 3 vs. Ct 9	(-499[312],486[4929]):2441	5' end: forward	3' end: forward	893 bps.
	(-452[549],442[404]):1978	5' end: forward	5' end: forward	Invalid.
Ct 3 vs. Ct 10	(-450[548],431[4061]):1473	5' end: forward	3' end: forward	1861 bps.
	(-569[628],-579[3844]):1610	5' end: forward	3' end: reverse	Invalid.
Ct 4 vs. Ct 5	(114[1],145[342]):1635	5' end: forward	5' end: reverse	1699 bps.
Ct 4 vs. Ct 11	(-584[67],561[328]):1526	5' end: forward	5' end: forward	Invalid.
Ct 4 vs. Ct 12	(586[312],-610[1805]):3101	5' end: forward	3' end: forward	233 bps
Ct 5 vs. Ct 10	(-498[653],-514[688]):1967	5' end: forward	5' end: reverse	1367 bps.
Ct 5 vs. Ct 11	(577[343],564[713]):2210 (578[67],563[691]):2072	5' end: forward	5' end: reverse	1262 bps.
Ct 6 vs. Ct 12	(650[312],605[1553]):2725 (652[1],623[2741]):2325 (649[67],611[1846]):3307 (651[1],632[2797]):2383	5' end: forward	5' end: reverse	1009 bps.
Ct 7 vs. Ct 10	(-581[228],550[3631]):1740	5' end: forward	3' end: forward	1594 bps.
Ct 8 vs. Ct 9	(426[1],-437[450]):1826	3' end: forward	5' end: forward	1594 bps.
Ct 9 vs. Ct 10	(-436[83],-427[3751]):1685	5' end: forward	3' end: reverse	Invalid

Note: Each line item in the crosslink information is the pair of reads linked between contigs. The format is (read ID in first contig [starting position in this contig], read ID in second contig [starting position in this contig]): pairmate distance.

For intercontigs, there are a few (from 1 to 4) clonemate pairs linked between 1-10, 2-10, 3-9, 3-10, 4-5, 4-11, 4-12, 5-10, 5-11, 6-12, 7-10, 7-11, 8-9, and 9-10 contigs, respectively. The most of corresponding pairs satisfy the constraints except the link of contig 7-11. Thus, there should be 13 links between these contigs. The output of contig alignment is shown in Table 5.31. The contig linking output is divided into three groups. First, this group contains all valid links. There are eight valid linked output contigs including contig 1-10, 4-5, 4-12, 5-10, 5-11, 6-12, 7-10, and 8-9. Second, this group contains all invalid links. Contig 2-10, 4-11, and 9-10 fall within this group. Last, this group contains conflicted links. Contig 3-9 and 3-10 contains both invalid and valid links and thus are classified to be in this group.

Table 5.32 Details and statistics of contig alignment of *Gallus gallus*

Clonemate distant constraint between a pair mate		Max	3208	Min	1519
Field	Contig pair	Contig1	Contig2	Contig3	
Root ID		3	158	687	
Number of clonemate pair linked across to other contigs: Note link to its own is clonemate within a contig	1	268	0	0	
	2	0	27	0	
	3	0	0	0	
Number of clonemate pair linked across to other contigs that satisfies constraints	1	249	0	0	
	2	0	26	0	
	3	0	0	0	
Cross linked output contig	Cross link information			Linking description	
None	None			None	

The details and statistics of contig alignment with clone mate information of *Gallus gallus* are shown in Table 5.32. The maximum and minimum clonemate distant constraints for this data are estimated to be 3208 and 1519 bases, respectively. In the three contigs, there are 268, 27, and 0 clonemate pairs in contigs 1 to 3, respectively. Among these clonemate pairs, 249, 26, and 0 pairs satisfy the clonemate constraints. Thus, more than 90% of clonemate pairs are good, which indicates that the assembly of reads within the contig is mainly valid. For intercontigs, there are no clonemate pairs linked between these contigs, respectively. Thus, there are no possible contig linking output between them.

CHAPTER VI
QUALITATIVE ANALYSIS OF SEQUENCE ASSEMBLY PROGRAMS

6.1 Assembly quality measurement

In this chapter, we provide experimental results on the assembly quality of the four sequence assembly programs: SEQASS, CAP1 [38], CAP3 [18], and TIGR [22]. We have developed and evaluation program to quantitatively compare the assembly results from different programs with different aspects. The assembly quality is measured from five aspects [15]: The number and size of contigs, Percent identity of each contig, the overall percent of identity, the percent of genome covered, and the number of misassembly.

- *The number and size of contigs*: Smaller number of final set of contigs and larger size of contigs implies higher quality of shotgun sequence assembly. These measures can directly be obtained from the output.
- *Percent identity of contigs*: Ratio of the number of matched bases to the total number of bases of overlapping region between the contig and the whole genome. The higher percent identity of contig indicates higher assembly quality. To determine this quantity, matching between the output contig and whole genome data will be found by searching and comparing using BLAT Tool. If several matches between a contig and whole genome data are found, the correct match is identify from maximum length and maximum percent ID of matched portion of contig The maximum length must also be greater than a minimum requirement for example percent ID must be greater than 95% and length of matched portion must be at least 90% of the length of contig. It should be noted that percent ID of contig could be directly calculated from the alignment output of BLAT Tool.
- *Total percent identity*: Ratio of the number of all matched bases to the total number of combined bases of all contigs. Total percent identity indicates

overall accuracy of the assembly and the higher total percent identity directly designates higher assembly quality. The total percent identity is computed from cumulative number of matches for all good contigs and combined size of all good contigs using the formula: *Totalmatch**100/*TotalQsize*, where *Totalmatch* and *TotalQsize* is the combined number of matches and combined contig size, respectively. In the case that contigs are overlap, number of matches and contig size of all overlapping are combined by union method similar to the method of total coverage computation.

- *Percent genome covered*: Ratio of the number of bases in assembly to the number of based of whole genome. The higher percent genome covered suggests higher assembly quality. It is calculated from total base position covered by contigs with a portion of the whole genome to the total number of bases of the whole genome. The percent coverage of a contig is estimated from the formula: (*Tend-Tst*)*100/*Tsize*, where *Tend*, *Tst,* and *Tsize* are starting position, ending position, and size of the contig found within the genome. Since an output contig may overlap with or contain in another contig, *Tend* and *Tstart* of every contig are stored in array variables, *Tst* and *Tend*, for subsequent union operation. The total percent coverage is computed by adding percent coverage of all contigs with the consideration that an output contig may overlap with or contain in another contig. The *Tst* and *Tend* of all contigs are compared with each other to determine overlap and containment. For contigs that are containment, their percent coverage will not be added to the total percent coverage. For overlapping contigs, the percent coverage are combined by union operation with *Tst* of the whole union is the minimum *Tst* of overlapping contigs and Tend of of the whole union is the maximum Tend of overlapping contigs.
- *The number of misassemblies*: The number of genome portions that are found to be incorrectly assembled: Smaller number missassemblies signifies higher assembly quality. Misassemblies can be found from contigs that were not determined to match with the whole genome based on the requirement of minimum percent ID and length.

6.2 Implementation of evaluation program

Our evaluation program has been developed by using c language. The program takes into account a set of contigs that obtained from each sequence assembly program. Then, the evaluation programs compare the contigs to the whole genome input obtained from Genbank. The blat tool is used for comparison. Finally, the five assembly quality measures are calculated. Fig. 6.1 shows the algorithm of the evaluation program. In implementation, *blat* function is called with the whole genome input as the database file, the consensus output contig as the query file, and the blat output file name is fixed with the name "fullctg.psl". The PSL-format output record, previously described in Chapter IV, is temporarily loaded into memory in structure ***pslinfo***, also explained in Chapter IV, and following processing is performed.

- Filter out overlaps with overlap length less than minimum overlapping percentage. The minimum overlapping percentage in this stage is different from the one in the quick overlap determination stage, which is fixed at 90% to properly select the correct contig-whole genome matching. The minimum overlapping length (*pslovlngmin*) is again calculated from the percentage of *Qsize* or *Tsize*, whichever is greater. The overlap length (*ovlng*:field) is then estimated from the PSL data from the relationship: *ovlng = Tend-Tstart+ Qgapbase*. If *ovlng* is less than *pslovlngmin*, the overlap will be considered as the incorrect match between the contig and whole genome input and discarded. Recall that T and Q correspond database and query fields, respectively.

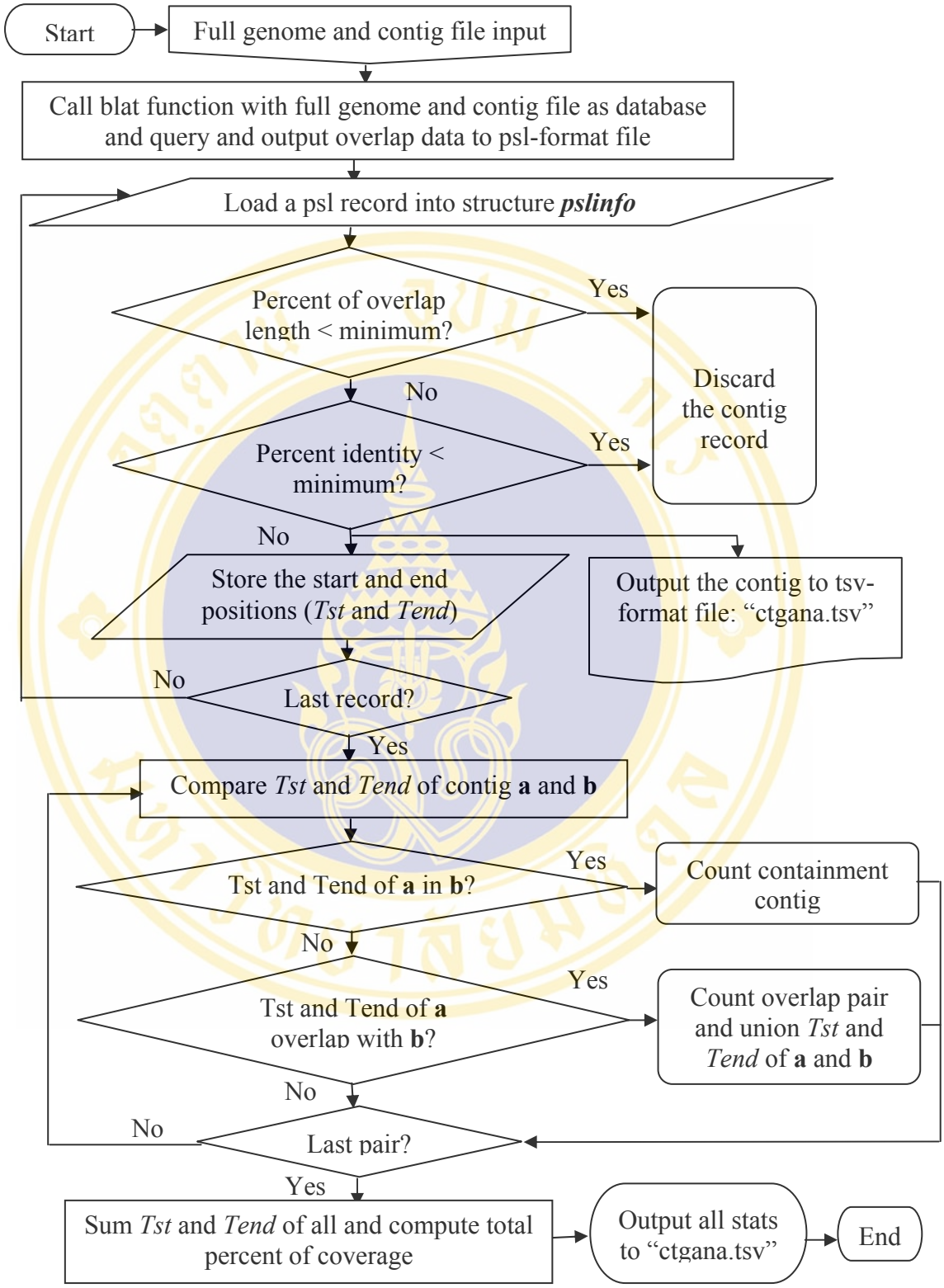

Fig. 6.1 The evaluation module.

- Filter out overlaps with percent identity less than minimum percent identity (*pcidmin*). The minimum overlapping percentage is the same as the one in quick overlap determination stage and the default value is 90%. The percent identity (*pcid*) is calculated from the percentage of *match* to *ovlng*. If *pcid* is less than *pcidmin*, the overlap will also be discarded. The number of good contig is counted as the number overlaps that pass both requirement and the number of misassembled is determined by substracting the number of good contig from the number of all contigs.
- Calculate the five assembly quality measures as explained in section 6.1. All correct matching data and calculated statistics are finally stored in the output tab-delimited file (anactg.tsv).

6.3 Evaluation results of SEQASS

The assembly results of all data, including *Canis familiaris*, *Carollia perspicillata*, *Dasypus novemcinctus*, *Equus caballus*, *Felis catus*, and *Gallus gallus*, are evaluated by the c program just described. The program is compiled and run under Linux TLE 2.0 operating system on 3.0 GHz Intel Pentium 4 processor with 256 MB RAM. The evaluation is first used to determine the optimum parameters of SEQASS by varying the parameters to obtain the best assembly quality. The evaluation results and all detailed statistics with the optimum parameters are then digested with the most details for *Canis familiaris*, while only the summary of evaluation results and statistics will be provided for other organisms because the details are mostly similar.

6.3.1 Determination of optimum parameters of SEQASS

The optimum parameters of SEQASS, which were previously listed in Table 5.2, have been determined experimentally by varying the parameters and evaluating the results to obtain the best assembly quality. The determination of the optimum value of minimum percent of overlap length is only illustrated in this section because this parameter is specific to SEQASS program, while other parameters are common among SEQASS, CAP1 and CAP3.

The optimum value of minimum percent of overlap length is determined for all organisms in this study. The evaluated performance parameters including the

number of good contigs, number of misassembled contigs, total percent coverage, and total percent identity, are plotted as a function of minimum percent of overlap length of SEQASS program as shown in Figs. 6.2-6.5, respectively. It should be noted that the remaining parameters are fixed as formerly shown in Table 5.2. From Fig. 6.2, it is evident that the optimum percent of overlap length, which yields maximum total percent of coverage of most organisms, is approximately *20%*. In addition, it can be seen that the total percent of coverage of three organisms including *Canis familiaris*, *Carollia perspicillata*, and *Dasypus novemcinctus*, falls rapidly as the percent of overlap length decrease below or increase above the optimum percent of overlap length, while the total percents of coverage of the remaining organisms do not change significantly as the percent of overlap length deviates from the optimum value.

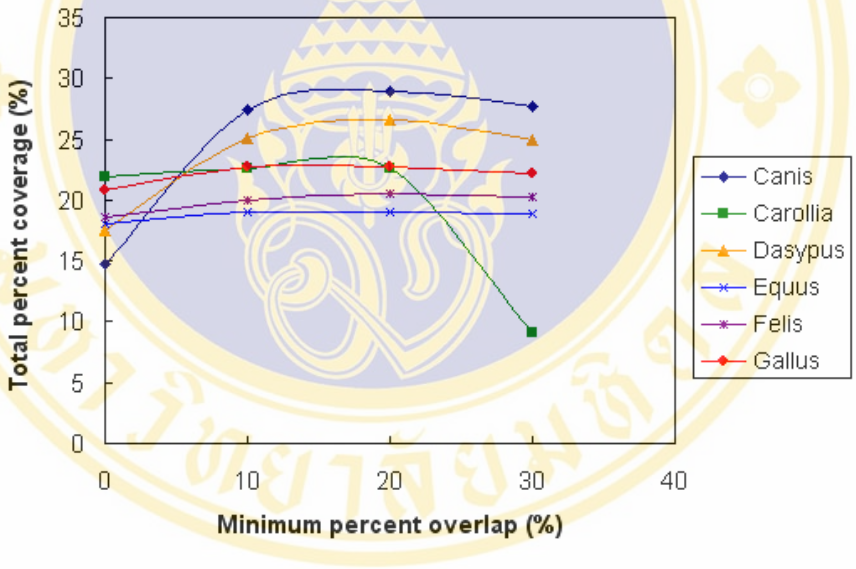

Fig. 6.2 Total percent coverage vs. Minimum percent overlap for various organisms

Fig. 6.3 shows the relationship between the total percent identity and the minimum percent overlap. We can see that the minimum percent overlap that yields the best total percent identity is varying widely for various organisms from 0% to 30 %. However, the variation of total percent identity is quite very small for all organisms. From Fig. 6.4, it can be seen that the number of good contigs does not vary considerably with minimum percent overlap. The minimum percent overlap that yields the best number of good contigs are not clearly seen because it is a constant

function for several organisms. Thus, the total percent identity and number of good contig can not be used to decide the optimum value of percent of overlap length.

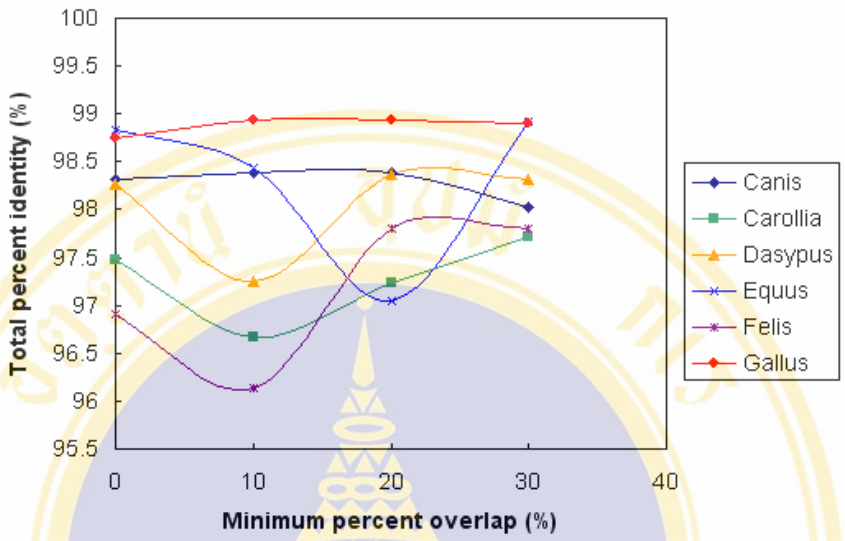

Fig. 6.3 Total percent identity vs. Minimum percent overlap for various organisms

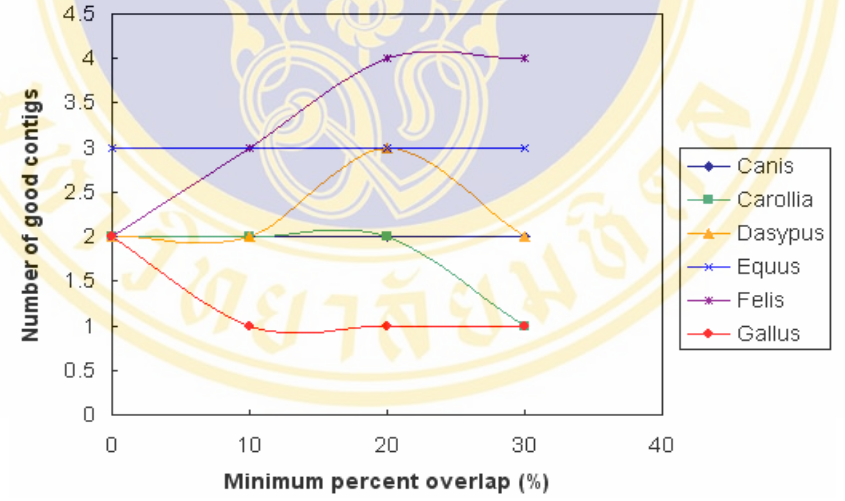

Fig. 6.4 Number of good contigs vs. Minimum percent overlap for various organisms

From Fig. 6.5, it is clear that the optimum percent of overlap length, which yields minimum number of misassembled contigs of most organisms, is in the range of 10-20%. From all evaluated results, it can be concluded that the optimum minimum percent overlap should be *20%* because it yields maximum total percent coverage and minimum number of misassemble contigs. Furthermore, it may be conceptually explained that the optimum minimum percent overlap exists because there should be a suitable minimum percent overlap that can properly select satisfied overlaps from initial overlaps for the 3' and 5' clipping. Too small minimum percent overlap would

results in poor satisfied overlap selection because too short overlaps should be false overlaps. While too high minimum percent overlap would results in too low number of satisfied overlaps that can participate in the 3' and 5' clipping.

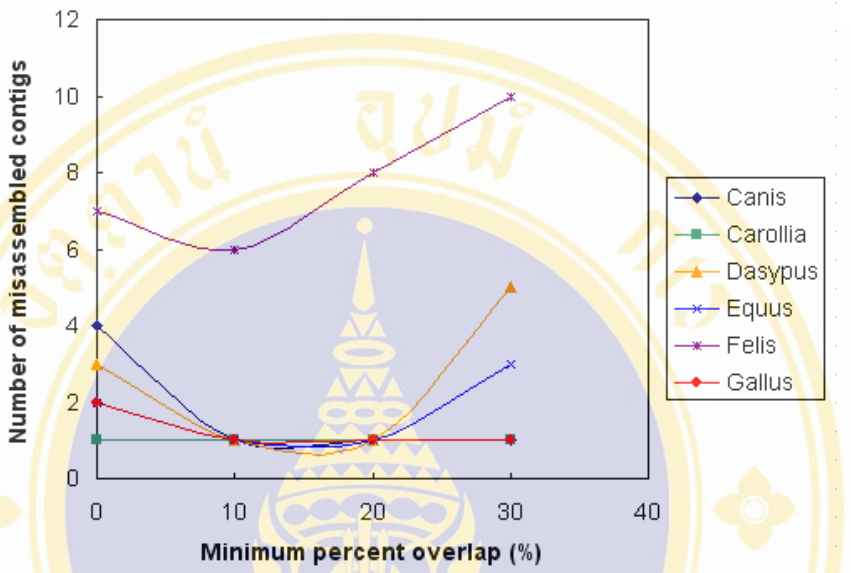

Fig. 6.5 Number of misassembled contigs vs. Minimum percent overlap for various organisms

6.3.2 Performance of SEQASS

All organisms were assembled by SEQASS with an optimum assembly parameters as determined from section 6.3.1 and the details of assembly results in various stages were digested in the chapter 5. Output contigs from assembly are analyzed by the written c program using the condition of minimum 90% overlap length and 90% identity. The output performance parameters of evaluated contigs of for all organisms are summarized in Table 6.1.

Table 6.1 Output performance parameters of the SEQASS assembly program for all organisms

Organism	*Canis familiaris*	*Carollia perspicillata*	*Dasypus novemcinctus*	*Equus caballus*	*Felis catus*	*Gallus gallus*
Number of good contigs	2	2	3	3	3	1

Average size of good contigs	22437	15357	10523	12525	9717	33144
Number of misassembled contigs	2	1	1	1	9	2
Number of containment contigs	0	0	0	0	0	0
Number of singlets	0	0	0	0	1	0
Number of overlap contig pairs	1	0	0	0	0	0
Total percent coverage	28.89	22.63	26.55	19.07	20.03	22.76
Total percent identity	97.47	97.23	98.36	97.05	97.81	98.93
Total run time (min)	12.96	14.93	7.40	8.0	6.63	8.83

It can be seen that from the output contigs produced by SEQASS for all organisms except for the *Felis catus*, few contigs were found to be good and the remaining few contigs were determined to be misassembled because they did not pass the requirement of minimum percent overlap length and percent identity. The average size of good contigs is satisfactorily large (between 9000 and 33000 bps). It should be noted that considerably high number of misassembled contigs is produced for *Felis catus* case. In addition, it can be noticed that there is no or only one singlet (unassembled read), thus most reads are participated in the output contigs. For all cases, no containment contig is produced. This means that there is no good contig that is contained in another good contig. It can also be seen that good contigs are not overlapped with each other for all cases except *Canis familiaris*. Few good and misassembled contigs with very few singlets, containment contigs, and overlap contig pairs, are primary indications of high DNA assembly performance.

Table 6.2 Typical details of good contig information by SEQASS taken from *Canis familiaris*

	Contig 1	Contig 4
match	17159	26846
mismatch	0	5

Rep.match	0	0
N's	1	22
Q gap count	1	5
Q gap bases	18	305
T gap count	2	7
T gap bases	16	405
strand	-	+
Q name	1	4
Q size	17443	27431
Q start	7	10
Q end	17185	27188
T size	152896	152896
T start	107908	80908
T end	125084	108186
block count	3	8
blockSizes	17097,5,58,	5,9,209,26580,12,8,13,37,
qStarts	258,17355,17378,	10,15,29,248,26829,26841,27132,27151,
tStarts	107908,125006,125026,	80908,80914,80930,81147,107729,107742,108132,108149,
Overlap length	17194	27583
Percent ID	99.8	97.33
Percent Coverage	11.23	17.84

Furthermore, it can be observed that the total percent coverage spanned by the good contigs for all cases are varied in the range from 19% to 29% and the corresponding total percent identity is in the range from 97% to 99%. The total percent coverage appears to be quite low but total percent identity is satisfactory for all cases. Insufficient oversampling of input data as previously noted in Section 5.1 and poor quality of data as previously noted in Section 5.4 should be the reason for the fairly low percent coverage results. However, the reasonably good total percent coverage and high percent identity with the given input data suggests satisfactory assembly performance of SEQASS program. Moreover, SEQASS requires moderate run time in the range from 6 to 15 minutes with these fairly large DNA data and is operable on a personal computer system.

To see more details of evaluated output, the typical information of good contigs taken from *Canis familiaris* produced by our evaluation program that utilized BLAT are detailed in Table 6.2. The description of data field is previously listed in

Table 4.2. In this case, queries (Q) are good contigs and reference (T) is the whole genome. Two good contigs, contig 1 and 4 of *Canis familiaris*, are 17433 and 27431 base-long, respectively. They were found to be overlap containment in whole genome with negative and positive orientations, respectively. In the 17159-base overlap of contig 1 wherein the query range is from 7 to 17185 base position and the reference range is from 107908 to 125084 base position, there are 17159 matches, no mismatch, no repeat match, 1 N character, 18 indels (gap) on the query side, 16 indels on reference side, and 3 blocks. Three blocks occur due to 2 separated gaps. The 17097-base, 5-base, and 58-base blocks start at 258,17355,17378 base positions on query side and 107908, 125006, 125026 base positions on reference side, respectively. It is a long overlap with 11.23 % coverage and very high homologous similarity with 99.8% identity.

In the 26846-base overlap of contig 4 wherein the query range is from 10 to 27188 base position and the reference range is from 80908 to 108186 base position, there are 26846 matches, 5 mismatches, no repeat match, 22 N character, 305 indels (gap) on the query side, 405 indels on reference side, and 8 blocks. Eight blocks occur from 7 separated gaps. The 5-base, 9-base, 209-base, 26580-base, 12-base, 8-base, 13-base, and 37-base blocks start at 10, 15, 29, 248, 26829, 26841, 27132, 27151 base positions on query side and 80908, 80914, 80930, 81147, 107729, 107742, 108132, 108149 base positions on reference side, respectively. It is a long overlap with 17.84 % coverage and high homologous similarity with 97.33 % identity. In addition, it can be seen that contig 1 and 4 are overlap from 107908 to 108186 base position of reference. The overlap region results in slight reduction of total percent coverage of the two contigs.

6.4 Comparative performance between SEQASS, CAP3, CAP1, and TIGR

For comparison, all organisms are assembled by CAP3, CAP1, and TIGR with their default assembly parameters. The performance of SEQASS, CAP3, CAP1, and TIGR are summarized and plotted as a function of the total percent coverage, total percent identity, number of good contigs, average size of good contig, number of

misassembled contigs, and total run time for comparison as shown in Figs. 6.6-6.11, respectively.

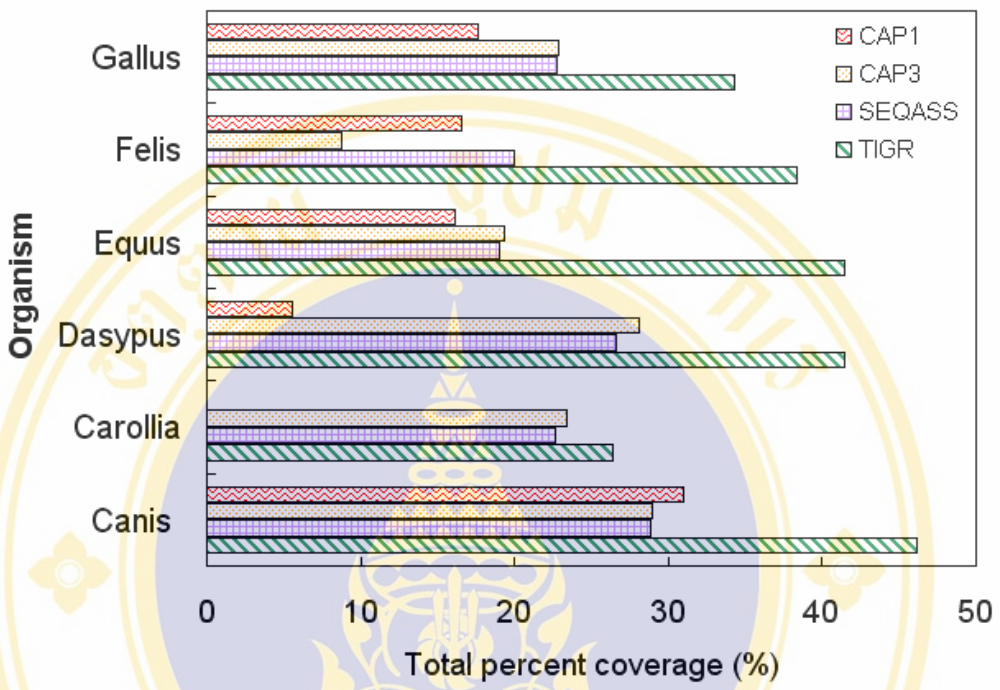

Fig. 6.6 Comparative chart of total percent coverage of all organisms for SEQASS, CAP3, CAP1, and TIGR

From Fig. 6.6, SEQASS produces assemblies with comparable (slightly lower) total percent coverage to CAP3 for all organisms, except *Felis catus*, that SEQASS performs better with more than twice of total percent coverage. From our initial analysis, SEQASS may perform better than CAP3 in this case because SEQASS use different and probably better method in the selection of satistied overlap for the 3' and 5' clipping by using the optimum minimum percent of coverage as formerly presented in Section 6.3.1. In addition, SEQASS produces assemblies with better total percent coverage than CAP1 for all organisms except *Canis familaris* that CAP1 performs slightly higher total percent coverage. In particular, SEQASS can perform well with *Carollia perspicillata* while CAP1 fails to correctly assemble this organism and yield 0% of coverage. Moreover, SEQASS produces assemblies with less total percent coverage than TIGR for all organisms.

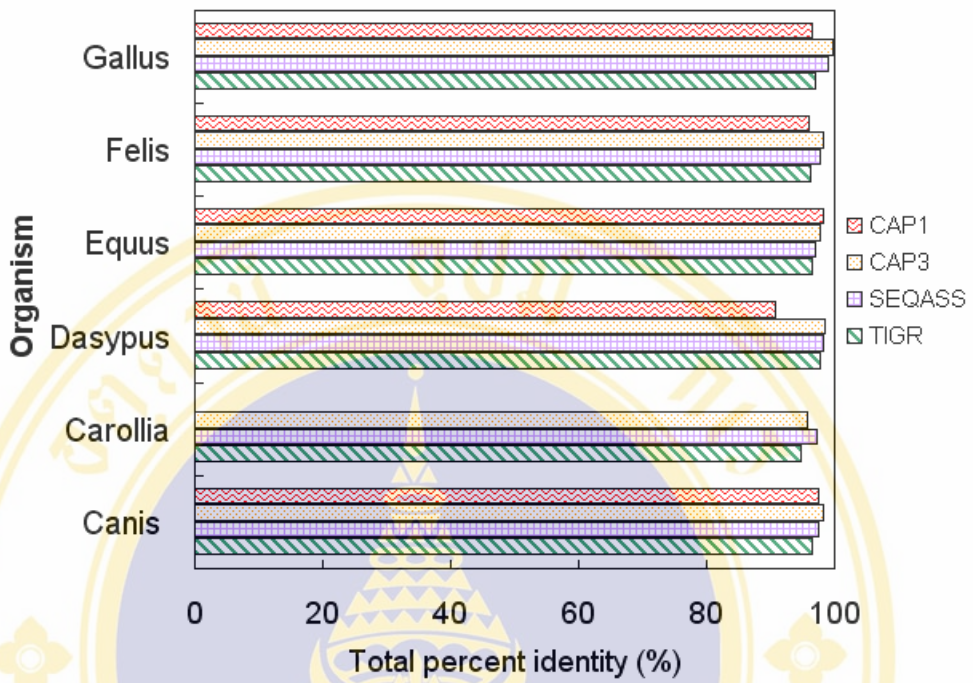

Fig. 6.7 Comparative chart of total percent identity of all organisms for SEQASS, CAP3, CAP1, and TIGR

From Fig. 6.7, SEQASS produces assemblies with comparable total percent identity to CAP3 for all organisms and SEQASS performs better for *Carollia perspicillata* case. In addition, SEQASS produces assemblies with better total percent identity than CAP1 for most of organisms except Canis familaris and Equus caballus that CAP1 performs slightly higher total percent identity. Furthermore, SEQASS produces assemblies with better total percent identity than TIGR for all organisms.

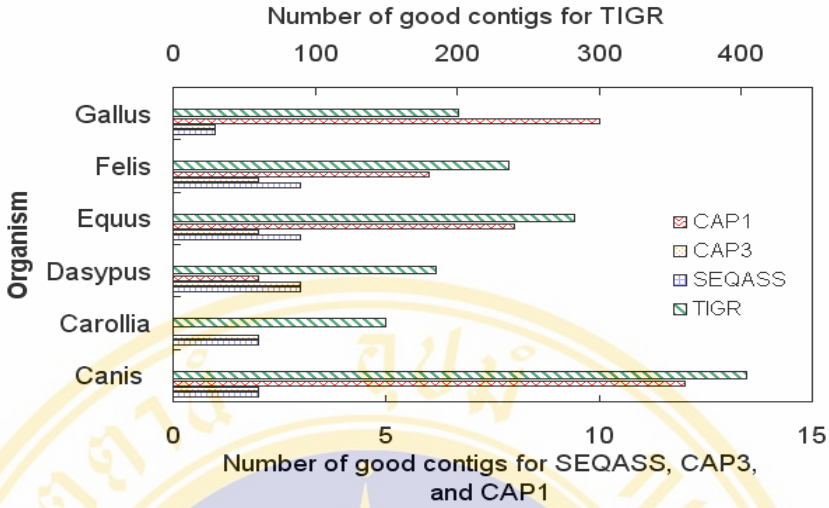

Fig. 6.8 Comparative chart of number of good contigs of all organisms for SEQASS, CAP3, CAP1, and TIGR

From Fig. 6.8, SEQASS produces assemblies with comparable (the same or only one more) number of good contigs to CAP3 for all organisms. In addition, SEQASS produces assemblies with less number of good contigs than CAP1 for most of organisms except Carollia perspicillata that CAP1 fails to produce any good contig. Thus, SEQASS always performs better than CAP1 in term of the number of good contigs. Moreover, SEQASS produces assemblies with much less number of good contigs than TIGR for all organisms. Therefore, SEQASS is much better than TIGR in term of the number of good contigs and TIGR produces too many good output contigs.

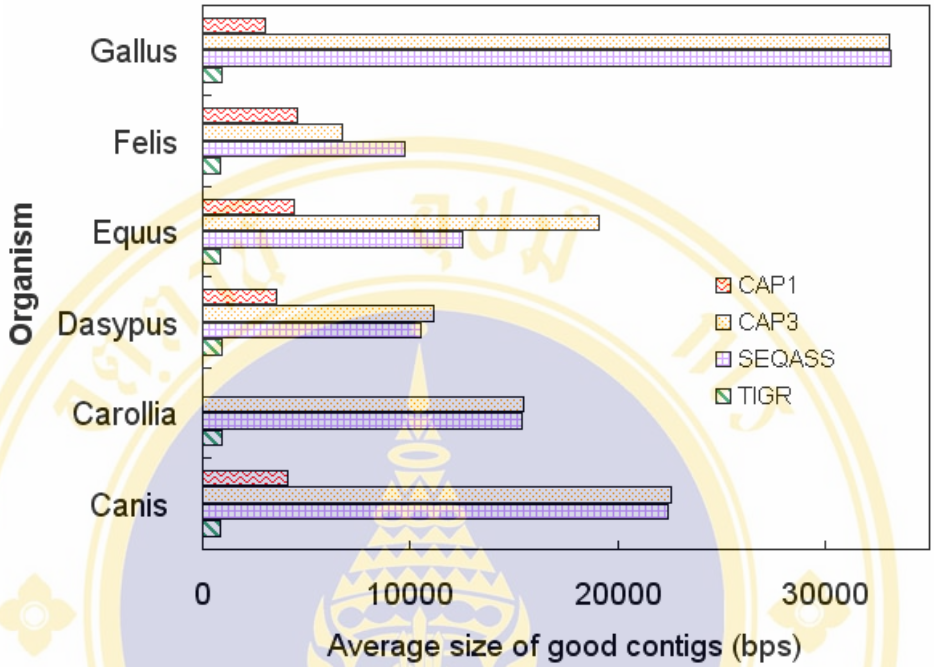

Fig. 6.9 Comparative chart of average size of good contigs of all organisms for SEQASS, CAP3, CAP1, and TIGR

From Fig. 6.9, SEQASS produces assemblies with comparable (slightly less) average size of good contigs to CAP3 for most of organisms except *Equus caballus* and *Felis catus*. SEQASS performs better with larger size of good contigs for *Felis catus* while CAP3 performs better with larger size of good contigs for *Equus caballus*. In addition, SEQASS produces assemblies with larger average size of good contigs than CAP1 for most of organisms except *Carollia perspicillata* that CAP1 fails to produce any good contig and hence the average size of good contigs is zero. Thus, SEQASS always performs better than CAP1 in term of the average size of contigs. Furthermore, SEQASS produces assemblies with much larger average size of good contigs than TIGR for all organisms. Therefore, SEQASS performs much better than TIGR that produces too small good output contigs.

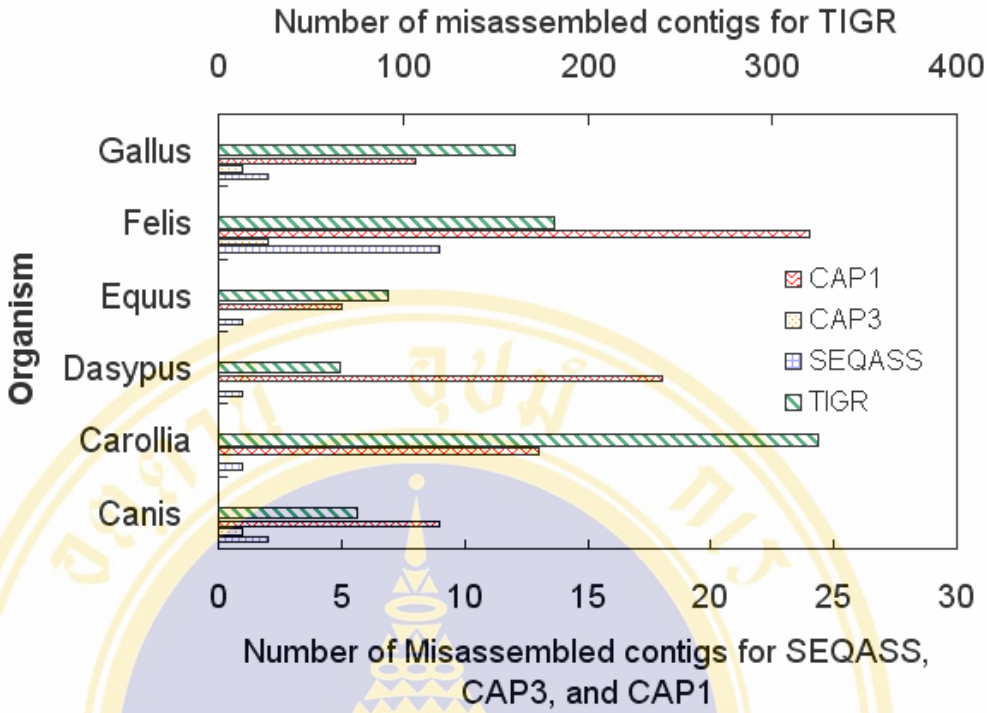

Fig. 6.10 Comparative chart of number of misassembled contigs of all organisms for SEQASS, CAP3, CAP1, and TIGR

From Fig. 6.10, SEQASS produces assemblies with comparable (only one more) number of misassembled contigs to CAP3 for all organisms except *Felis catus*. SEQASS produces seven more number of misassembled contigs for *Felis catus*. In addition, SEQASS produces assemblies with less number of misassembled contigs than CAP1 for all organisms. Thus, SEQASS always performs better than CAP1 in term of the number of misassembled contigs. Moreover, SEQASS produces assemblies with much less number of misassembled contigs than TIGR for all organisms. Therefore, SEQASS performs much better than TIGR that produces too many misassembled contigs. From Fig. 6.11, SEQASS requires longer run time than CAP3 and TIGR but shorter run time than CAP1 for all cases. CAP3 is the faster program and CAP1 is the slowest programs. Thus, the inferior run time performance is the main deficiency of SEQASS program.

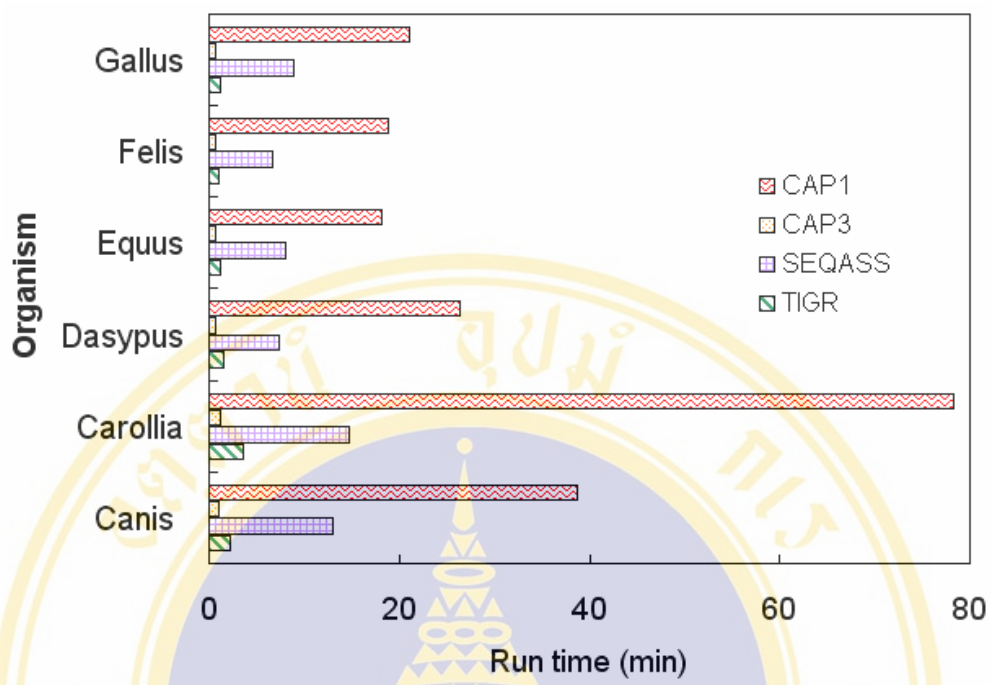

Fig. 6.11 Comparative chart of total run time of all organisms for SEQASS, CAP3, CAP1, and TIGR

Based on the experimental results, our SEQASS program has comparable overall performance to all tested assembly programs, particularly to CAP3, in term of the total percent coverage, the total percent identity, the number of good contigs, the average size of good contigs, and the number of misassembled contigs. However, SEQASS requires longer run time than CAP3 and TIGR. From our analysis, there are two main reasons of relatively slow run time of SEQASS. First, SEQASS made use of BLAT tool library, while CAP3 and TIGR make direct use of optimized blast technique within c source code. Second, CAP3 and TIGR utilize more optimized dynamic programming code than SEQASS. In addition, SEQASS shows the best performance for *Felis catus*. SEQASS produces much less unsequenchable segment compared to assembly by CAP3. It is clear that CAP1 shows poor assembly performance compared to other programs. CAP1 produces more number of good contigs, shorter length of good contigs, and higher number of misassembled contigs with lower percent coverage and identity in most cases. This is expected because CAP1 does not utilizes BLAT tool and does not performs the read-ends clipping. While TIGR produces many more number of short good/misassembled contigs compared to other program. However TIGR yields highest total percent of coverage

compared to other programs. The higher total percent of coverage by TIGR seems to be the advantage of TIGR, nevertheless this large percent coverage is not really useful since TIGR produces many more number of short good/misassembled contigs and they must be further assemble together.

CHAPTER VII
CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this thesis, we have addressed the problem of DNA sequence assembly. We have studied and provided the advanced knowledge of several state-of-the-art DNA sequence assembly programs, including CAP3, ARCHNE, CELERA, TIGR, and PHRAP. The algorithms of these programs have been investigated and compared extensively. Following the study, we have developed a new DNA sequence assembly program called *SEQASS*. It has implemented following the framework presented in CAP3, which has shown high assembly quality, moderate complexity and straightforward methodology. Unlike CAP3, however, we have used BLAT, a BLAST-like tool, to accelerate pair-wise comparisons and improve assembly quality. The SEQASS implementation was started with the free C source code program of BLAT tool and greedy-tree multiple alignment from their developers. We developed a new main program to include them as a library and internal function and then implemented the assembly in six modules. The six modules include 1) preprocessing, 2) quick overlap determination by BLAT, 3) the read-ends clipping, 4) final overlap determination, 5) greedy-tree multiple alignment, and 6) contig alignment with clonemate information.

SEQASS program has been tested with DNA data of large to medium-size mammal organisms obtained from Genbank, including *Canis familiaris*, *Carollia perspicillata*, *Dasypus novemcinctus*, *Equus caballus*, *Felis catus*, and *Gallus gallus*. The data and assembly results have been presented and studied in details through each program module. The program performance of SEQASS have been evaluated based on several key parameters including the total percent coverage, the number of good contigs, the average size of good contigs, the number of misassembled contigs, and the total run time. The evaluation of assembly results has been implemented with the C language. The output contig sequences are compared with the known whole

genome input by BLAT. SEQASS performances have been compared to other available programs including CAP3, CAP1, and TIGR.

The evaluation results show that SEQASS has comparable performance to CAP3 in term of the total percent coverage, the total percent identity, the number of good contigs, the average size of good contigs, and the number of misassembled contigs but SEQASS requires longer run time for most input data. However, SEQASS performs better than CAP3 in providing only sequenchable segments and total percent coverage for the sequence of *Felis catus*, which is the poorest-quality input data. Thus, SEQASS seems to be more tolerant to poor quality input data than CAP3, which is a notable achievement. Moreover, the assembly by SEQASS is considerably better than that by CAP1 for all data in terms of the number of good contigs, the average size of good contigs, the number of misassembled contigs, and the total run time. However, the total percent coverage of assembly by SEQASS is slightly less than that by CAP1. Furthermore, the assembly by SEQASS is considerably better than TIGR in term of the number of good contigs, the average size of good contigs, and the number misassembled contigs, but the total percent coverage by SEQASS is much less than that by TIGR and the total run time is considerably longer. From our initial analysis, there are two main reasons of relatively slow run time of SEQASS. First, SEQASS made use of BLAT tool library, which consumes extra time in quick overlap determination because of self-overlap/duplicate overlap comparisons and additional file I/O operations. Second, SEQASS utilizes less optimized dynamic programming code than other existing programs. The time consuming problem of SEQASS may be corrected in the future work.

7.2 Suggestion for future work

Our SEQASS has shown good assembly performance comparable to other state-of-the-art commercial softwares. However, it is still inferior in terms of the time complexity and the assembly quality. It is recommended that the program may be improved in the future as follows:

- BLAT tool library should be modified so that the sequence comparison by BLAT technique is performed with direct memory access.

- Dynamic programming routine should be amended so that the sequence comparison is performed in the shortest linear time and space.
- Repeat and chimeric reads detection and correction should be added to enhance the program capability.
- Overall code should be further optimized to reduce any unnecessary steps and operations.
- More input optional parameters may be added to enhance the program flexibility.
- Graphic user interface (GUI) may be incorporated to make the program more user friendly.

REFERENCES

1. J. D. Watson, and F. H. Crick, "Genetical implications of the structure of deoxyribonucleic acid," Nature, Vol. 171, p. 964, 1953.
2. M. Pop, S. L. Salzberg, and M. Shumway. "Genome sequence assembly: Algorithms and issues," BioInformatic, p. 47, 2002.
3. F. Sanger, "Nucleotide sequence of Bacteriophage Lambda DNA," J. Molecular Biology, Vol. 162, p.729, 1982.
4. T. Bankier, et al., "Genome sequence of Cytomegalovirus," DNA Seq., 2, p.1, 1995.
5. S. E. Celniker, et al., "Finishing a whole genome shotgun: the Drosophila melanogaster euchromatic genome sequence," Genome biology 3, p. 1,2002.
6. M. Maxam, and W. Gilbert, "A new method for sequencing DNA," Proc. Nat'l Academy of Science, Vol. 74, pp. 560-564, 1977.
7. L. M. Smith, et al., "Flouresence detection in automated DNA sequence analysis," Nature, Vol. 321, p. 674-679, 1986.
8. F. Sanger, S. Niclen, and A. R. Coulson, "DNA sequencing with Chain-Terminating Inhibitor," Proc. Nat'l Academy of Science, Vol. 74, p. 5463, 1977.
9. M. D. Adams, et al., "A model for high-throughput automated DNA sequencing and analysis core facilities," Nature, Vol. 368, p. 474-475, 1994.
10. T. J. Gleeson, and R. Staden, "An X windows and UNIX implementation of our sequence analysis package," Comput. Appl. Biosci. Vol. 7, p. 398, 1991.
11. R. Dolz, "GCG: fragment assembly programs," Meth. Mol. Bio. Vol. 24, p. 9,

1994.

12. Applied Biosystems, Foster City, CA.
13. H. Peltola, et al., "SEQAID: A DNA sequence assembling program based on a mathematical model," Nucleic Acids Res. Vol. 12, p. 307, 1984.
14. G. Gryan, "Faster sequence assembly software for megabase shotgun assemblies," Genome Sequencing and Analysis Conference VI, 1994.
15. M. J. Miller, and J. I. Powell, "A quantitative comparison of DNA sequence assembly programs," J. Comp. Bio. Vol. 1, p.257, 1994.
16. S.Smith, et al., "High throughput DNA sequencing using an automated electrophoresis analysis system and a novel sequence assembly program," BioTechniques Vol. 14, p. 1014, 1993.
17. R. Staden, "A new computer method for the storage and manipulation of DNA gel reading data," Nucleic Acids Res. Vol. 8, p. 3673, 1980.
18. X. Haung, and A. Madan, "CAP3: DNA Sequence Assembly Program," Genome Research, p. 868, 1999.
19. S. Batzoglou, et al., "ARACHNE: A Whole-Genome Shotgun Assembler," Genome Research, vol. 12, p. 177, 2002.
20. S. Batzoglou, et al., "Whole-Genome Shotgun Assembler for Mammalian Genomes: ARACHNE2," Genome Research, vol. 13, p. 91, 2003.
21. G. Myers, "Whole-Genome DNA Sequencing," Computing in Science & Engineering, p. 33, 1999.
22. G.G. Sutton, et al. "TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects," Genome Sci. & Technol., Vol. 1, p. 9, 1995.
23. P.Green, "Phrap documentation: Algorithms" Phred/Phrap/Consed System Home page; http://www.phrap.org.
24. M.S. Waterman, "Introduction to computational biology," 1st Ed. Chapman&Hall.
25. E.W. Myers, "A whole-genome assembly of Drosophila," Science, Vol.287, p. 2196, 2000.
26. J.D. Watson, N. Hopkins, J. Roberts, J.A. Steitz, and A. Weiner, "Molecular

biology of gene," 4th ed. Benjamin-Cummings, Menlo Park, CA, 1987.

27. T.F. Smith and M.S. Waterman, "Identification of common molecular sequences," J. Mol. Biol. Vol. 147, p. 195, 1995.
28. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, "Basic local alignment search tool," J. Mol. Biol. Vol.215, p. 403, 1990.
29. S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman, "Gapped BLAST and PSI-BLAST: A new generation of protein database search programs," Nucleic Acids Res. 25 p. 3389, 1997.
30. Z. Zhang, S.Schwartz, L. Wagner, and W. Miller, "A greedy algorithm for aligning DNA sequences," J. Comput. Biol. Vol.7, p. 203, 2000.
31. W. Gish, and D.J. States, "Identification of protein coding regions by database similarity search," Nat. Genet. Vol.3, p. 266, 1993.
32. D.J. States, and W. Gish, "Combined use of sequence similarity and codon bias for coding region identification," J. Comput. Biol. Vol.147, p. 195, 1994..
33. W. James Kent, "BLAT—The BLAST-Like Alignment Tool," Genome Research, Vol.12, p. 656, 2002.
34. X. Huang, "On global sequence alignment," Comput. Appl. Bio-sci. Vol.10, p. 227,1994.
35. Huang, X., and Miller, W. (1991). A time-efficient, linear-space local similarity algorithm. Adv. Appl. Math. Vol.12: 337–357.
36. Gleizes and A. Henaut, "A global approach for contig construction," Comput. Appl. Biosci. Vol. 10, p. 401, 1994.
37. M. Hirosawa, et al., "Comprehensive study on iterative algorithms of multiple sequence alignment," Comput. Appl. Biosci. Vol. 11, p. 13, 1995.
38. X. Haung, "An Improved Sequence Assembly Program," Genomics Vol.33, p. 21, 1996.
39. Kececioglu, J. D., and Myers, E. W. "Combinatorial algorithms for DNA sequence assembly," Algorithmica Vol.13, p. 7, 1995.
40. Lishan Li and Sami Khuri, "A Comparison of DNA Fragment Assembly

Algorithms," Algorithmica Vol.18, p. 145, 2000.

APPENDIX

A. BASICS OF MOLECULAR BIOLOGY

A.1 Overview of molecular biology for DNA

A main problem of biology is to understand inheritance of organisms. In 1865, Mendel proposed the first abstract model of inheritance that there is a basic unit called *gene* that defined inheritance of organisms. However, the actual nature of gene was not known until 1944 when gene was known to be made of Deoxyribonucleic acid (DNA) and James Watson and Francis Crick proposed the double helical structure for DNA that can be used to explain copying mechanism for genetic materials, in which specific pairing of DNA molecules induced self-replication [1].

The biological molecules are divided into two classes, large and small. The large molecules, called macromolecules, are of three types: DNA, RNA, and proteins. The small molecules are basic elements of these macromolecules [10]. DNA is the basis of heredity and it is a polymer of nucleic acid, consisted of small molecules called *nucleotides* that have only four basic types called four bases: Adenine (A), Ctyosine (C), Guanine (G), and Thymine (T). RNA (ribonucleic acid) is the polymer of another kind of nucleic acid (called ribonucleotide), made up of the four bases similar to DNA except thymine is replaced by Uracil (U). RNA's function in the cell is to produce proteins. Proteins are the polymers of 20 kinds of amino acids listed in Table A.1. Proteins are structural and enzymatic elements that perform various cell activities. Both RNA and proteins are made from instructions in the DNA, and new DNA molecules are made from copying existing DNA molecules.

A.2 Biological chemistry of DNA, RNA, and Protein molecules

It is essential to know basic molecular structures of DNA, RNA, and proteins in order to understand the whole DNA assembly problems. Nucleotides, the basic elements of DNA, compose of a phosphate group, a pentose (a 5-carbon sugar) and an organic base. The general molecular structure of a nucleotide is shown in Fig. A.1 [24].

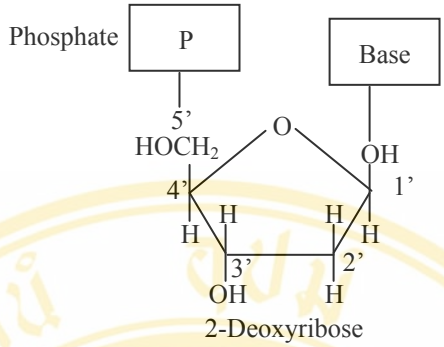

Fig. A.1 The general structure of a nucleotide.

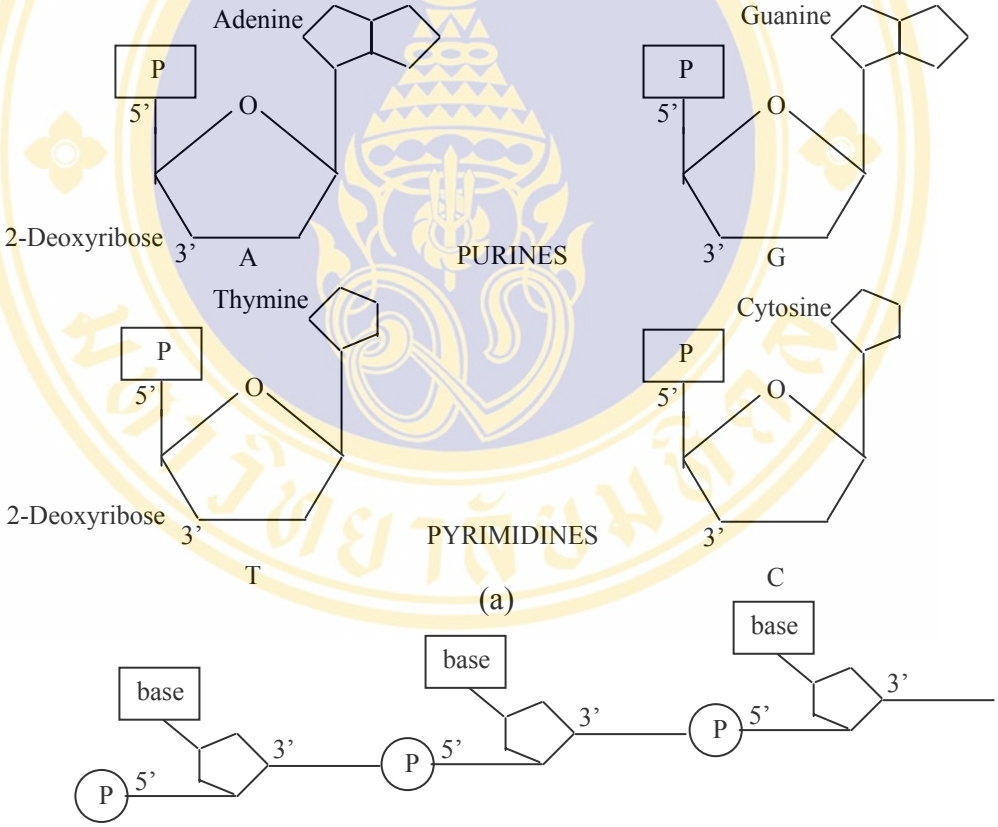

Fig. A.2 (a) Purines and pyrimidines and (b) a single strand of DNA molecule

It should be noted that the carbon atoms are often not shown and present where two lines intersect. The number 1' to 5' refer to carbon atom locations in the pentose (2-Deoxyribose). The four different bases determine the identity of a

nucleotide. The structure of bases in DNA is shown in Fig. A.2 where two types, purine and pyimidlines, are illustrated. Purines type (Adenine and Guanine) has two rings whereas pyrimidlines type (Cytosine and Thymine) has only one ring.

A single strand of the DNA molecule is formed by the sequence phosphate-sugar-phosphate-····-sugar, with 5' carbon of the sugar linked to the phosphate, 1' carbon linked to the base, and 3' carbon connected to the next phosphate as shown in Fig. A.2 (b). Thus, there is a definite direction to the chain and the carbon atoms 5' to 3' are used to define the forward orientation of the molecule. The reverse orientation is denoted by beginning with 3' signature.

A.3 Double helical structure for DNA

The complete physical structure of DNA is a complementary pair of two helical-shape strands of nucleotide. The complementary basepairs are the bases pair with A pairing T and G pairing C. This structure can be best illustrated by an example in Fig. A.3. In this example, a single strand of DNA: 5' TACGT 3' (written in 5' to 3' direction) is paired to a complementary strand running in the opposite direction. The A-T and G-C pairs are formed by hydrogen bonds indicated by dotted bars. DNA generally occurs double stranded and its length is measured by the number of base pairs. The opposite strands are always reverse complementary to each other. DNA can be completely defined by a single sequence of DNA base pair with associated direction where 5' denotes forward and 3' denotes reverse direction. The complementary structure of DNA is what enables the self-copying mechanism.

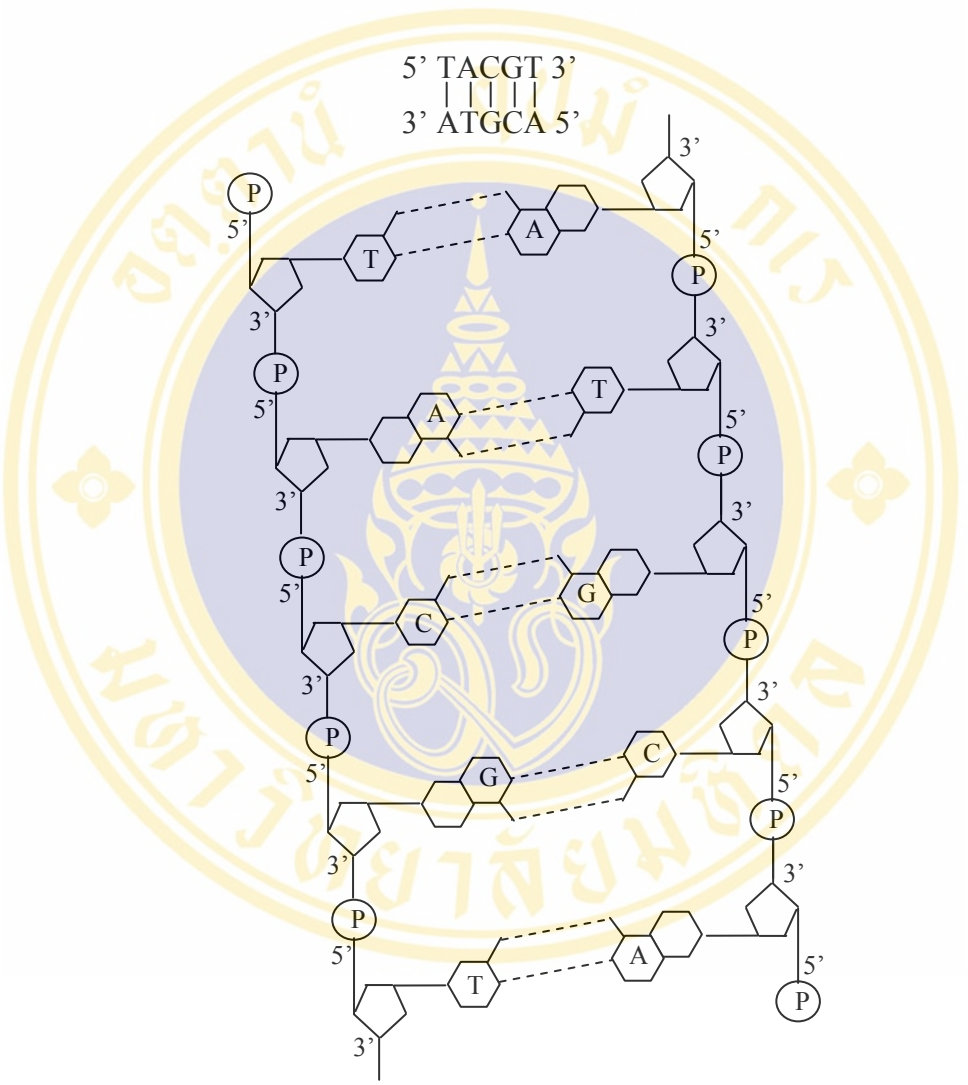

Fig. A.3 A complete double stranded DNA molecule

BIOGRAPHY

NAME	Mr.Anukoon Wisitsoraat
DATE OF BIRTH	September 15, 1968
PLACE OF BIRTH	Bangkok, Thailand
INSTITUTIONS ATTENDED	Mahidol University, 1990-1994: Bachelor of Electrical Engineering Mahidol University, 2000-2005: Master of Science (Computer Science)
POSITION&OFFICE	5/2005-3/1999 Thailand Post Co. Ltd., Bangkok, Thailand Position: Engineer 6/1995-5/1998 Radio Phone Co. Ltd., Bangkok, Thailand Position: Engineer 1/1995-5/1995 Communication Authority of Thailand, Bangkok, Thailand Position: Engineer