|  |  |
| --- | --- |
|  | .NET Bio Comparative Assembly Technical Guide  Version 1.0 June 2011 |

Abstract

Comparative genome assembly is a process in which an already sequenced close relative genome is used as a *reference* against which the reads from the *target* genome can be matched, providing a framework that can greatly speed up the assembly of a genome. Comparative Assembly allows for the efficient assembly of large complex genomes where a close relative species genome is already available.

This document describes the **ComparativeGenomeAssember** class and the **ComparativeUtil** utility. **ComparativeGenomeAssember** and Padena are the two assembly algorithms in the .NET Bio library.

.NET Bio Framework software and documentation are available at: <http://bio.codeplex.com>

Disclaimer: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2011 The Outercurve Foundation.

Distributed under Creative Commons Attribution 3.0 Unported License

Microsoft, Visual Basic, Visual Studio, and Windows are trademarks of the Microsoft group of companies.  All other trademarks are property of their respective owners.

Contents

[Introduction 4](#_Toc294528518)

[Scenarios 5](#_Toc294528519)

[Large Plant Genome and Close Relative Reference 5](#_Toc294528520)

[Use Comparative Assembly to get more insight into mutations and SNP’s 5](#_Toc294528521)

[Assembling sequences of different strains 6](#_Toc294528522)

[A Comparative Assembly Overview 6](#_Toc294528523)

[Comparative Genome Assembly Design 7](#_Toc294528524)

[Comparative Assembly Atomic Processes 9](#_Toc294528525)

[ComparativeGenomeAssembler Assemble Method 10](#_Toc294528526)

[Step 1 - Align reads 12](#_Toc294528527)

[Step 2 - Repeat Resolution: 16](#_Toc294528528)

[Step 3 - Layout Refinement 16](#_Toc294528529)

[Step 4 Consensus Generation 21](#_Toc294528530)

[Step 5 Scaffold Generation 22](#_Toc294528531)

[Delta Alignment 24](#_Toc294528532)

[Comparative Assembly Command Line Utilities 25](#_Toc294528533)

[ComparativeUtil 26](#_Toc294528534)

[NUCmer 27](#_Toc294528535)

[RepeatResolutionUtil 28](#_Toc294528536)

[LayoutRefinementUtil 29](#_Toc294528537)

[ConsensusGenerationUtil 29](#_Toc294528538)

[ScaffoldUtil 30](#_Toc294528539)

[Glossary 30](#_Toc294528540)

# Introduction

With the increasing number of genomes with a published reference sequence, more and more species have close relatives with their reference genome sequence known. Therefore, based on common evolutionary origin, it is expected that large portions of two closely related genomes will share a lot of similarity. This presumption can provide us a framework which will greatly enhance the quality of assembly and speed up the assembly process. De-novo assemblies are orders of magnitude slower and more memory intensive than mapping assemblies. To take advantage of mapping assemblies, the .NET Bio Framework library provides the Comparative Assembly add-in. A comparative assembler can be used to assemble reads from one genome (called the target) using as a template the sequence of a related genome (called the reference). This technique allows for the efficient assembly of large complex genomes where a close relative species genome is already available. This process provides the framework which greatly enhances the quality of assembly and speeding up assembly process. De-novo assemblies, using many overlapping sequences (high coverage) to infer ordering directly from the sequences themselves, are orders of magnitude slower and more memory intensive than mapping assemblies.

In sequence assembly, there are three basic types of assembly:

* **de-novo assembly**: assembles reads together using only the reads to piece the final genomic sequence together. De Novo sequencing is the initial sequencing that results in the primary genetic sequence of organisms.
* **re-assembly**: assembles reads together using an existing reference genome for that specie. This results in building an individual speciman’s sequence that is similar but not identical to the reference sequence. In this case the reference genome is a representative example of the same species genetic code. The reference provides a good approximation of the DNA of any single individual.
* **comparative assembly**: assembles reads usingan existing close relative reference genome to initially cluster neighboring reads together and building a sequence that is expected to have large portions that will be nearly identical to the reference sequence. The reference provides a good guide for the assembly.

.NET Bio Framework provides algorithms for de novo and comparative assembly.

**Note**: There is a command line utility provided, PadenaUtil, that defines the scaffolding.

For more information on the .NET Bio Framework Parallel de Novo Assembler (Padena) see the Bio Parallel de Novo Assembler Technical Guide.docx on [Codeplex](http://bio.codeplex.com/documentation) or at $\**..\Bio\Doc** in the source tree.

# Scenarios

## Large Plant Genome and Close Relative Reference

You can use the genomes of two plants that are genetically very similar to each other, one very much large than the other.

The information from the smaller genome can be used to reconstruct the larger genome when the smaller one is already sequenced and assembled using de novo methods. In terms of complexity and time requirements, de-Novo assemblies tend to be orders of magnitude slower and more memory intensive than mapping assemblies. This is mostly due to the fact that the assembly algorithm needs to compare every read with every other read (an operation that is has a complexity of O(*n*2) but can be reduced to O(*n* log(*n*)).

## Use Re-Assembly to get more insight into mutations and SNP’s

Use comparative assembly for assembling genomes of already sequenced organisms to get more insight into mutations and SNP’s. Draft human genome was available in 2001. Since the human genome sequencing many unanswered questions are still present which cannot be answered with single copy of human genome. This led to projects like 1000 genomes, by which scientist like to decipher the phenotypic variations caused due to mutations and SNPs. In these cases doing sequencing using denovo techniques is costly exercise. So we can use already sequenced human genome as reference and align reads to it and find variations among different copies of human genome.

## Assembling sequences of different strains

Multiple strains of Mycobacterium tuberculosis, Streptococcus pneumoniae and Staphylococcus aureus are sequenced in order to understand virulence, drug resistance and other phenotypic differences between strains. If one strain is sequence is available, comparative genome assembly can be used to further assemble sequence of other strains.

# A Comparative Assembly Overview

Alignment is a methodology for arranging the sequences of DNA, RNA, and proteins to identify the regions of similarity that may be a consequence of functional, structural or evolutionary relationships between the sequences.

Use the **ComparativeGenomeAssembler** to use an already sequenced close relative genome as a *reference* against which the reads from the *target* genome are matched.

1. Read Alignment
   1. Invokes NUCmer
   2. NUCmer invokes MUMmer
2. Repeat Resolution
3. Layout Refinement
4. Consensus Generation
5. Scaffold Generation

The comparative assembly process begins by aligning reads to a close relative reference genome using NUCmer. NUCmer uses MUMmer to align each shotgun read to the reference genome. The delta alignments are returned after the read alignment is complete.

The second step of the comparative assembly attempts to resolve the ambiguity for reads that have been ambiguously placed because of the reference reads. This step requires mate pair information to resolve the placement of repeated sequences.

When performing comparative assembly the reads from the target genome will only partially match the close relative reference genome. This is a result of the genomic divergence that has led to the two species. It is expected that the process will have to account for new SNPs, indels (insertions and deletions,) translocations, chromosomal duplication and other rearrangements between the two genomes. During layout refinement the reference layout between the target genome and reference genome is refined by analyzing the indels and rearrangements and applying sequencing error mitigation including correction of indels considered to be spurious. Mate-pair information is utilized in this process.

For each group of overlapping reads in the refined layout, a *multi-alignment* is computed to generate a consensus sequence for the genomic region covered by those reads. In this comparative assembler an alignment-consensus algorithm is used that computes the consensus sequence of the new genome. The algorithm loops through all deltas at each index of a contig with a delta and finds the consensus alignment. In the final stage of consensus formation, the assembler builds an alignment of all the reads covering the genome and infers, as a consensus of the aligned reads, the original sequence of the genome that it is assembling.

The output of the consensus generation process results in a collection of contiguous DNA sequences (contigs) whose relative placement along the genome is not defined. A scaffolding procedure is used to order and orient these contigs using paired read information. After building contigs, the comparative assembler uses mate-pair information to order and orient the contigs and place them into larger structures called scaffolds or supercontigs.

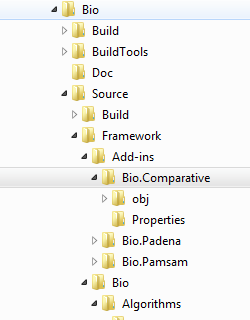
Contigs are linked by using the following:

* overlaps
* clone mates
* alignments to reference genome
* alignments to physical maps
* conservation of gene synteny

# Comparative Genome Assembly Design

The **ComparativeGenomeAssembler** class implements a comparative genome assembler for the assembly of DNA sequences. This class calls the five steps of the comparative assembler in order. It also exposes the individual steps so that they can be independently consumed enabling users to manipulate the data before using it as an input for the next step in the chain.

The **ComparativeGenomeAssember** class is provided as an add-in.

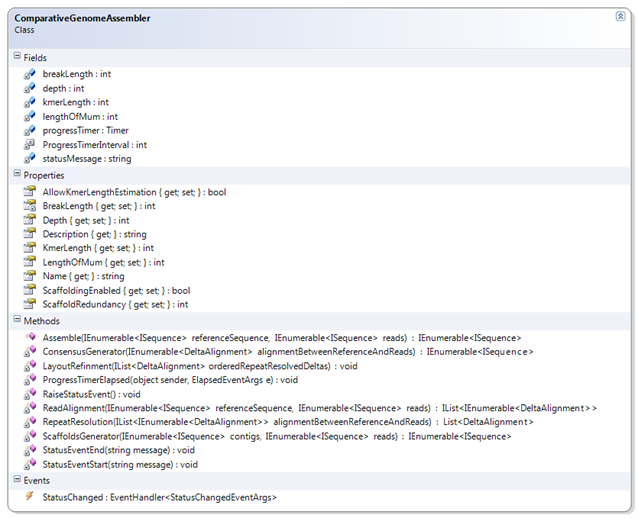


The following figure shows the Comparative Assembly steps.

**Note**: You can perform comparative assembly from the command line by using the utility, ComparativeUtil, which is described in the [ComparativeUtil Command Line Utility](#_ComparativeUtil_Command_Line) section.

### Class Diagram ComparativeGenomeAssembler

**ComparativeGenomeAssembler** implements a comparative assembler for the assembly of DNA sequences using a reference genome to help determine the sequence order.



## Comparative Assembly Processes

To perform comparative genome assembly, the following processes are required and must be performed in order:

ComparativeGenomeAssembler Processes

|  |  |
| --- | --- |
| Process | Description |
| ReadAlignment | Comparative genome assembly step 1. Used for aligning reads to a reference genome using NUCmer. Read Alignment may find multiple locations where a specific read ‘fits’ against the reference. These ambiguous locations are resolved in the next step. Users can manipulate the data before using it as an input for the next step in the chain by using the command line utility NucmerUtil. |
| RepeatResolution | Comparative genome assembly step 2. Used to resolve the ambiguity for ambiguously placed reads. Users can manipulate the data before using it as an input for the next step in the chain by using the command line utility RepeatResolutionUtil. |
| LayoutRefinement | Comparative genome assembly step 3. Used to refine the reference layout between the target genome and reference genome by addressing indels and rearrangements. Users can manipulate the data before using it as an input for the next step in the chain by using the command line utility LayoutRefinementUtil. |
| ConsensusGeneration | Comparative genome assembly step 4. Used for groups of overlapping reads to generate a consensus sequence for the genomic region covered by those reads. Users can manipulate the data before using it as an input for the next step in the chain by using the command line utility ConsensusUtil.exe. |
| ScaffoldGeneration | Comparative genome assembly step 5. Used to build scaffolds from contigs. Users can manipulate the data before using it as an input for the next step in the chain by using the command line utility ScaffoldUtil. |

For more information on the command line utilities go to the [Comparative Assembly Command Line Utilities](#_Comparative_Assembly_Command) section at the end of this document.

## ComparativeGenomeAssembler Assemble Method

In addition to these atomic operational components, the **ComparativeGenomeAssembler.Assemble()** method can be called to perform the operations in the proper order to produce an assembled genome. The **ComparativeGenomeAssembler.Assemble()** method implements a comparative genome assembly which assembles the input sequences into the largest possible contigs.

int KmerLength = 11;

int MumLength = 20;

ComparativeGenomeAssembler asmblr = new ComparativeGenomeAssembler();

asmblr.ScaffoldingEnabled = false;

asmblr.KmerLength = KmerLength;

asmblr.LengthOfMum = MumLength;

IEnumerable<ISequence> assemblerResult = asmblr.Assemble(referenceSequences, sequences);

Where

* **referenceSequence** is the sequence used as a *reference* against which the reads from the *target* genome can be matched.
* **sequence** is the reads from the *target* genome to assemble.
* and returns a FastA file which contains the scaffolds and generated unaligned contig sequences.

This will return an **IComparativeAssembly** instance which contains a list of assembled sequences.

//Comparative Assembly Steps

//1) Read Alignment (Calling NUCmer for aligning reads to reference sequence)

StatusEventStart(Properties.Resources.ReadAlignmentStarted);

IList<IEnumerable<DeltaAlignment>> alignmentBetweenReferenceAndReads = this.ReadAlignment(referenceSequence, reads.Where( a => a.Count >= LengthOfMum));

StatusEventEnd(Properties.Resources.ReadAlignmentEnded);

// 2) Repeat Resolution

StatusEventStart(Properties.Resources.RepeatResolutionStarted);

IList<DeltaAlignment> repeatResolvedDeltas = this.RepeatResolution(alignmentBetweenReferenceAndReads);

StatusEventEnd(Properties.Resources.RepeatResolutionEnded);

StatusEventStart(Properties.Resources.SortingResolvedDeltasStarted);

List<DeltaAlignment> orderedRepeatResolvedDeltas = repeatResolvedDeltas.OrderBy(a => a.FirstSequenceStart).ToList();

StatusEventEnd(Properties.Resources.SortingResolvedDeltasEnded);

// 3) Layout Refinement

StatusEventStart(Properties.Resources.LayoutRefinementStarted);

LayoutRefinment(orderedRepeatResolvedDeltas);

StatusEventEnd(Properties.Resources.LayoutRefinementEnded);

// 4) Consensus Generation

StatusEventStart(Properties.Resources.ConsensusGenerationStarted);

IEnumerable<ISequence> contigs = this.ConsensusGenerator(orderedRepeatResolvedDeltas.OrderBy(a => a.FirstSequenceStart));

StatusEventEnd(Properties.Resources.ConsensusGenerationEnded);

if (ScaffoldingEnabled)

{

// 5) Scaffold Generation

StatusEventStart(Properties.Resources.ScaffoldGenerationStarted);

IEnumerable<ISequence> scaffolds = ScaffoldsGenerator(contigs, reads);

StatusEventEnd(Properties.Resources.ScaffoldGenerationEnded);

return scaffolds;

}

else

{

return contigs;

}

## Step 1 - Align reads

This step calls **ReadAlignment** for aligning reads to a reference genome using NUCmer. It returns the delta alignments after the read alignment is complete.

**Note:** Comparative Assembly throws an exception when ambiguous reads are found.

Each read is compared to the reference genome using MUMmer to find common substrings greater than a specific length. This gives a list of candidate locations where the read may be clustered. Evolutionary changes between the reference and target genomes may cause MUMmer to have one or more complete contiguous matches, multiple partial matches, or no match at all. Repetitive sequences and polymorphisms between the target and the reference cause some reads to align in a non-contiguous fashion. A modified version of the Longest Increasing Subsequence (LIS) algorithm is used in order to generate chains of mutually consistent matches between each read and the reference. In addition to the longest consistent chain, a set of near-optimal chains is also computed in order to identify reads anchored in repeats. Those reads that are ambiguously placed in the genome (one or more chains are within 2 per cent identity from the best placement) are classified as repetitive and resolved later (in some cases) by using mate-pair information.

IList<IEnumerable<DeltaAlignment>> alignmentBetweenReferenceAndReads = this.ReadAlignment(referenceSequence, reads.Where( a => a.Count >= LengthOfMum));

Where the reads are in FastA or FastQ format.

Which in turn uses MUMmer with the reference sequence. NUCmer calls MUMmer to leverage its high performance maximum exact match algorithm.

List<IEnumerable<DeltaAlignment>> deltaAlignments = new

List<IEnumerable<DeltaAlignment>>();

Parallel.ForEach(referenceSequence, sequence =>

{

NUCmer nucmer = new NUCmer((Sequence)sequence);

…

foreach (ISequence qrySequence in reads)

{

deltaAlignments.Add(nucmer.GetDeltaAlignments(

qrySequence, false));

}

}

Which in turn uses MUMmer with the reference sequence. NUCmer calls MUMer to leverage its maximum identical match.

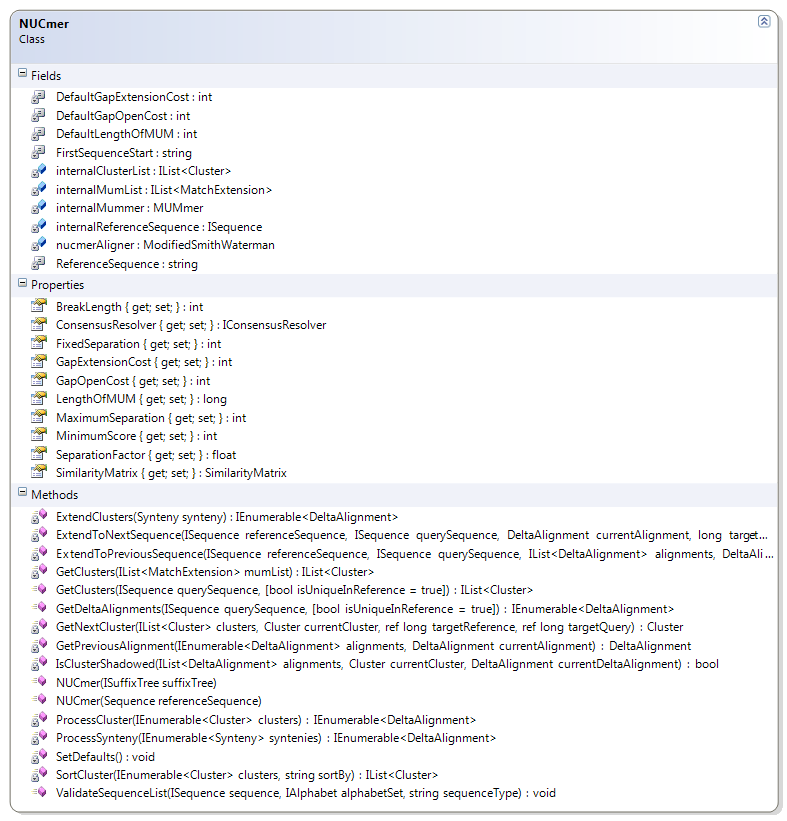
internalMummer = new MUMmer.MUMmer(referenceSequence);

NUCmer (NUCleotide MUMmer) allows for multiple references and multiple query sequences to be aligned in a many vs. many fashion. It is a three step process:

1. maximal exact matching
2. match clustering
3. alignment extension

It begins by using MUMmer to find all of the maximal unique matches of a given length between the two input sequences. Following the matching phase, individual matches are clustered into closely grouped sets with **mgaps**. Finally, the non-exact sequence between matches is aligned via a modified Smith-Waterman algorithm, and the clusters themselves are extended outwards in order to increase the overall coverage of the alignments.

### Class Diagram NUCmer



NUCmer is a system for rapidly aligning entire genomes or very large DNA sequences. It allows alignment of multiple reference sequences to multiple query sequences. This is commonly used to identify the position and orientation of set of sequence contigs in relation to the assembled sequence. NUCmer defines the skeleton of the NUCmer algorithm, deferring some steps to the derived class.

**Note**: Large binary alignment files are only supported on 64 bit machines – 32 bit machines will throw an out of memory exception.

### MatePair

**MatePair** class store read pairs with library information and **MapPairMapper** class converts input list of reads into paired reads using information available in FASTA header.

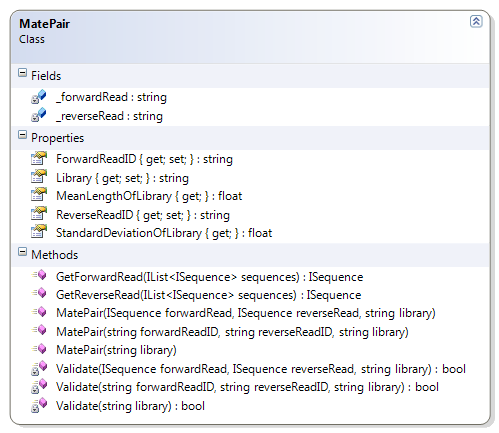
Supported mate pair formats

|  |  |
| --- | --- |
| Format | Description |
| chrI0.X1:abc ATGC | forward reads |
| chrI0.Y1:abc TACG | reverse reads |
| chrI0.F:abc ATGC | forward reads |
| chrI0.R:abc TACG | reverse reads |
| chrI0.1:abc ATGC | forward reads |
| chrI0.2:abc TACG | reverse reads |

Where:

* X1,F,1 denotes forward reads
* Y1,R,2 denotes reverse reads
* abc denotes library name
* chrI0 is the sequence id

### Class Diagram MatePair



## Step 2 - Repeat Resolution:

This step resolves the ambiguity with a read that has been ambiguously placed because of the reference genomic reads. This step requires mate pair information to resolve the ambiguity about placements of repeated sequences. The mate-pair information is used as follows:

1. If the paired-end sequence is uniquely anchored in the genome, the repetitive read is placed in the location that satisfies the constraints imposed by mate-pair information.
2. If both mates are ambiguously placed, an attempt is made to find whether the mate-pair information allows us to place them both in the assembly. In some cases, there exists only one placement of both a read and its mate that satisfies the mate-pair constraints on distance.
3. If there is more than one possible coherent placement, one of them is chosen at random.

private List<DeltaAlignment> RepeatResolution(IList<IEnumerable<DeltaAlignment>> alignmentBetweenReferenceAndReads)

{

return RepeatResolver.ResolveAmbiguity(alignmentBetweenReferenceAndReads);

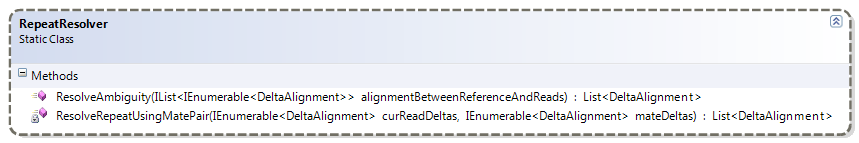
}

Where:

* **alignmentBetweenReferenceAndReads** represents the alignment between the reference genome and the reads.
* **referenceSequence** represents the sequence of the reference genome.

**reads** represents the list of sequence reads which are from either a FastA or FastQ file.

* .



## Step 3 - Layout Refinement

Since reads from the target genome only partially match the reference genome, indels (insertions and deletions) and rearrangements between the two genomes must be addressed. This step refines the reference layout between the target genome and reference genome by taking care of indels and rearrangements based on the alignment information contained in the delta file. Mate-pair information is utilized to place repetitive sequences, or random placement to simulate even coverage.

Layout refinement uses steps from denovo assembly to merge unaligned reads. The previous steps help to determine which are overlapping reads which can be assembled by using denovo steps to generate the missing piece and the later scaffold step used to aid in merging the contig with contigs. This step is parallelized for each set of evolutionary events like indels and rearrarangements. Part of the difficulty arises because reads from the target genome only partially match the reference, or adjacent sections of the reads may match non-adjacent portions of the reference genome. Both of these events result in the breaking of the alignment algorithm to align reads to a reference genome.

Step 3 contains the **LayoutRefinment** method:

Signature:

LayoutRefinment(IList<DeltaAlignment> orderedRepeatResolvedDeltas)

Which calls the **LayoutRefiner.RefineLayout()** method.

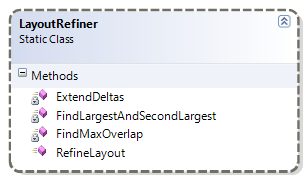
private void LayoutRefinment(IList<DeltaAlignment> orderedRepeatResolvedDeltas)

{

LayoutRefiner.RefineLayout(orderedRepeatResolvedDeltas);

}

### Class Diagram LayoutRefiner



**RefineLayout** Signature:

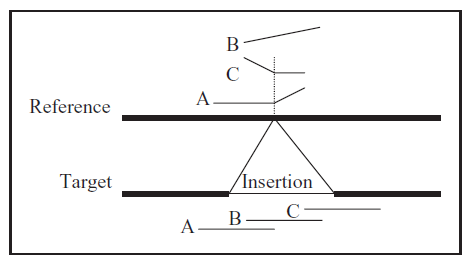
public static void RefineLayout(IList<DeltaAlignment> orderedDeltas)

The following indels and rearrangements are addressed in the LayoutRefinement process:

* [Insertions in the target](#_Insertions_in_the)
* [Insertions in the reference (deletions from the target)](#_Insertions_in_the_1)
* [Rearrangements](#_Rearrangements)
* [Divergent DNA](#_Divergent_DNA)
* [Errors vs. polymorphisms](#_Errors_vs._polymorphisms)

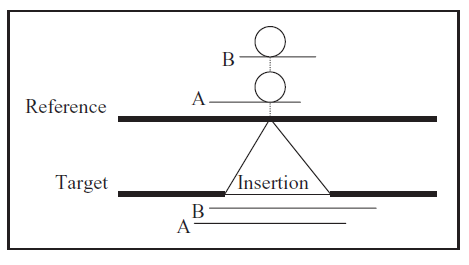
### Insertions in the target

One type of insertion is when portions of the reads do not match the reference genome. The result is depicted in the following figure. In such cases, the MUMmer alignment does not provide any information about the relationship between reads A, B and C, but since we know their respective positions in the layout their relationship can be inferred and the problem solved by using a typical *de novo* assembly method.



Insertion in the target genome. The bottom shows the true layout of reads A, B and C, and slanted lines indicate non-matching portions of the read

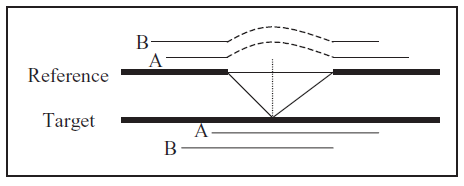
If the insertion is small enough as to be comprised in a single read, the middle of this read will not match the reference but both ends will, as shown in the following figure. An alignment that breaks and continues at the same position in the reference, but at distant points in the read, is evidence of this kind of occurrence.



Short insertion in the target genome. Bubbles are portions that did not aligned to the reference

### Insertions in the reference (deletions from the target)

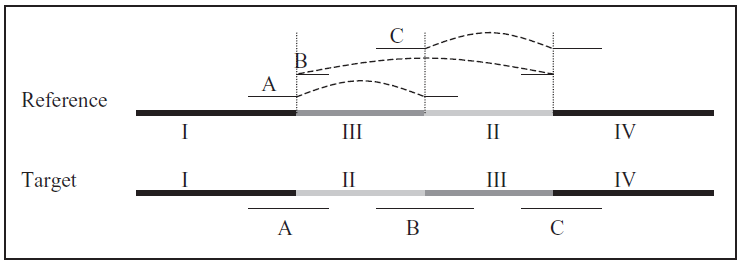
In the case of an insertion in the reference, the reads that span the insertion point match two disjoint regions of the reference, as shown in the following figure. The relative placement of the reads spanning the insertion point, and those in the surrounding areas, can be easily determined.



Deletion from the target. Dashed lines indicate the ‘stretch’ of the reads needed to align to the reference

### Rearrangements

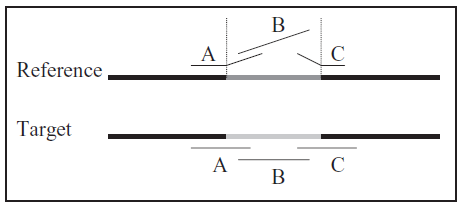
Translocations (reciprocal or not) and inversions are challenging situations for comparative assembly. The algorithm takes a conservative approach by restricting itself to resolving areas that have the *signature* of an insertion in the reference, as discussed in the preceding item. The following figure shows this type of situation, where sections II and III in the target appear in a different order than in the reference. Read A identifies section III as an insertion in the reference between sections I and II; likewise, read C identifies section II as an insertion between sections III and IV; read B does not have the signature of an insertion into the reference, so the algorithm results in two contigs (I + II and III + IV), which will be joined together during the scaffolding stage.



Rearrangement – regions II and III from the target appear in a different order in the reference. All reads match the reference in disjoint locations (connected by dashed lines), but only reads A and C have the insertion into the reference signature

### Divergent DNA

If both genomes have had enough time to mutate extensively, the sequences can no longer be aligned, because the difference between them is greater than the alignment threshold as shown in the following figure. This is the same as an insertion into the target, thus resulting in two contigs.

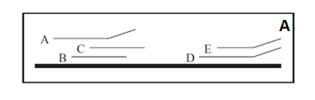


Significant divergence between the two genomes. Slanted lines are portions of the reads not matching the reference

### Errors vs. polymorphisms

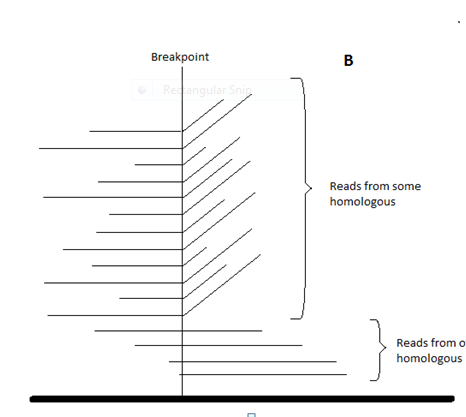
Errors introduced during the preparation or the sequencing stages have similar signatures to those caused by true polymorphisms between both genomes. The quality of each read must be assessed before doing any assembly.

After identifying reads with at least one breakpoint, the alignment of a read to the reference consists of more than one contiguous segment, other reads are identified that share the same breakpoint, or span the breakpoint. A straightforward majority rule can then be applied to decide if a breakpoint is an error or an actual rearrangement, see the following figure.



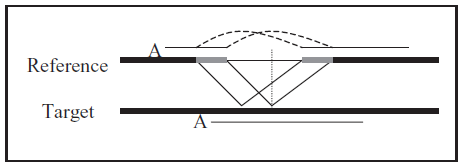
A) Error detection. Read A is probably incorrect, because B and C disagree with it; reads D and E probably indicate a true polymorphism.

For cases when there are more than two paired (homologous) sets of chromosomes (polyploidy), care is necessary while taking this decision, because one or more homologues can have a breakpoint, while others can span it as shown in the following figure.



B) Since we have lots of copies of the same chromosome, the decision between errors and polymorphism is not easy.

If a polymorphism is flanked by short repeats, as in the following figure, the LIS algorithm has to allow an overlap between adjacent alignments to the reference. In this case, the positions of the reads following the insertion are adjusted to allow for the presence of the short repeat.



Short flanking repeats on the alignment of a read to the reference. Dashed lines connect sections of read A that occur twice in the reference but once in A (and, thus, in the target genome)

## Step 4 Consensus Generation

This step generates a consensus sequence for the genomic region covered by the reads. For each group of overlapping reads in the refined layout, a *multi-alignment* is computed to generate a consensus sequence for the genomic region covered by those reads. For this, a pairwise alignment of each read to the current consensus sequence is computed and the resulting multi-alignment is used to generate a new consensus sequence; hence, this stage comprises a series of rounds of pairwise alignments using the Progressive Consensus Generation algorithm.

Signature:

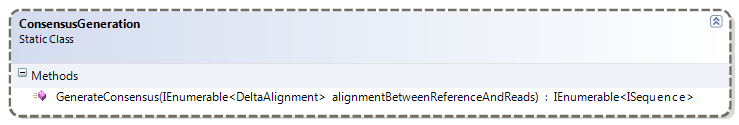
IEnumerable<ISequence> GenerateConsensus(IEnumerable<DeltaAlignment> alignmentBetweenReferenceAndReads)

Where:

* **alignmentBetweenReferenceAndReads** is the Input list of reads.
* returns a file containing a list of the generated contigs sequences. Each element in the List consisting of an alignment offset from the start and a sequence.

outputSequences.Add(currentAlignmentStartOffset, new Sequence(AmbiguousDnaAlphabet.Instance, currentContig.ToArray(), false));

### Class Diagram ConsensusGeneration



## Step 5 Scaffold Generation

Step 4, consensus generation, results in a set of contigs, and the scaffold generation step uses the mate-pair information to determine their relative order and orientation, so that the scaffolds can be built as is usually done by other assembly software.

A scaffold is composed of contigs and gaps representing a portion of the genome sequence reconstructed from end-sequenced whole-genome shotgun reads. Each contig represents a contiguous length of genomic sequence where the order of bases is known with a high degree of confidence. Gaps are the result of overlapping reads, that is, reads from the two sequenced ends of at least one fragment that overlap with other reads in two different contigs that may be adjacent.

Overlapping scaffolds can produce results that place a sequence in more than one location, on more than one scaffold, when it actually exists in only one location. This happens because it is common for a significant percentage of genes to be created from an inferred organization of the contigs into a gapped mosaic of assemblies called “scaffolds”. Ambiguities in the order and orientation of the contigs can greatly increases the number of possible placements for the sequence. This ambiguity can then obscure the actual gene structure.

Comparative genome assembly uses a reference genome to help define how the contigs should be assembled. The total coverage of the genome by the set of contigs directly affects the accuracy of the process of finding genes by their similarity to a given close relative reference segment. The greater the overall coverage and the closer it approaches 100% the higher the probability of properly placing the contigs in the assembly.

This step builds scaffolds from contigs and paired reads using Padena Step 6 for assembly. The following shows Padena Step 6:

PadenaAssembly assemblyResult = (PadenaAssembly)this.Assemble(inputSequences);

if (includeScaffolds)

{

// Step 6: Build \_scaffolds

this.BuildScaffoldsStarted();

IList<ISequence> scaffolds = this.BuildScaffolds(assemblyResult.ContigSequences);

this.BuildScaffoldsEnded();

if (scaffolds != null)

{

assemblyResult.AddScaffolds(scaffolds);

}

}

return assemblyResult;

**Note**: Comparative assembler requires Bio.Padena.dll for this step to work because this scaffold generation step in the comparative assembler calls the **GenerateScaffolds** method that is part of the Bio.Padena.dll.

In this step **ScaffoldsGenerator**() calls the **Bio.Algorithms.Assembly.Padena.Scaffold.GraphScaffoldBuilder** class **BuildScaffold()** method which actually does the work.

IEnumerable<ISequence> ScaffoldsGenerator(IEnumerable<ISequence> contigs, IEnumerable<ISequence> reads)

Where:

* **contigs** is a List of contigs.
* **reads** is a List of paired reads.
* and returns a FastA file which contains the scaffolds and the generated unaligned contig sequences.

public IList<ISequence> BuildScaffold(

IEnumerable<ISequence> reads,

IList<ISequence> contigs,

int lengthofKmer,

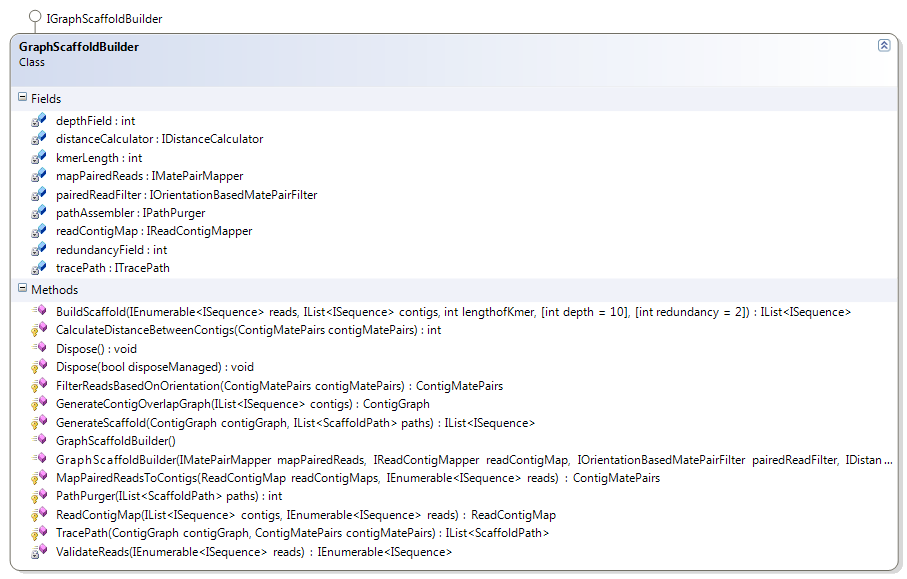
int depth = 10,

int redundancy = 2)

Where:

* **reads** is a List of reads
* **contigs** is a List of contigs
* **lengthofKmer** is theKmer Length
* **depth** is the depth for graph traversal
* **redundancy** is the number of mate pairs required to create a link between two contigs.
* and returns a List of scaffold sequences.

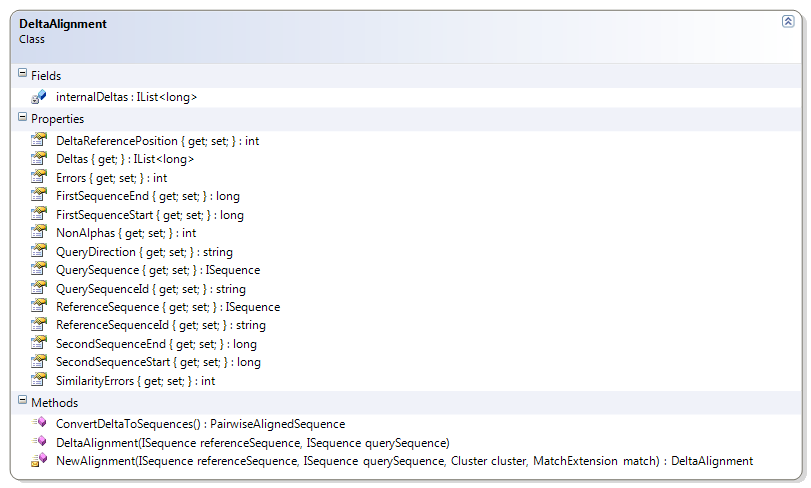
### Class Diagram GraphScaffoldBuilder



# Delta Alignment

Delta Alignment represents an alignment object in terms of the delta. Delta is an encoded representation of alignments between input sequences. It contains the start and end indices of alignment in reference and query sequence followed by error values and a list of integers in the following lines. Each integer represent an insertion (+ve) in the reference sequence and deletion (-ve) in the reference sequence.

This class represents such alignment with required properties and utility methods.



public DeltaAlignment(ISequence referenceSequence, ISequence querySequence)

# Comparative Assembly Command Line Utilities

This section describes the general usage of the command line utility ComparativeUtil to assemble genomes by using the comparative assembly process.

Comparative Assembly Utilities

|  |  |
| --- | --- |
| Utility | Description |
| [ComparativeUtil](#_ComparativeUtil) | A command line utility to initiate the comparative assembly process and runs all five steps. |
| [NucmerUtil](#_NUCmer) | Used for ComparativeUtil step 1. Users can manipulate the data before using it as an input for the next step in the chain. |
| [RepeatResolutionUtil](#_RepeatResolutionUtil) | Used for ComparativeUtil optional step 2. Users can manipulate the data before using it as an input for the next step in the chain. |
| [LayoutRefinementUtil](#_LayoutRefinementUtil) | Used for ComparativeUtil optional step 3. Users can manipulate the data before using it as an input for the next step in the chain. |
| [ConsensusUtil](#_ConsensusGenerationUtil) | Used for ComparativeUtil step 4. Users can manipulate the data before using it as an input for the next step in the chain. |
| [ScaffoldUtil](#_ScaffoldUtil) | Used for ComparativeUtil step 5. Users can manipulate the data before using it as an input for the next step in the chain. |

## ComparativeUtil

ComparativeUtil initiates the comparative assembly process to assemble a genome and determine the sequence order by using a reference genome. The following line illustrates the basic command line format:

ComparativeUtil.exe <options> <referencefilename> <readsfilename>

ComparativeUtil uses all of the underlying utilities described below to assemble the designated genome.

To execute the ComparativeUtil utility

Executing ComparativeUtil initiates the comparative assembly process which will run each of the underlying five steps to assemble a genome.

ComparativeUtil.exe /s referenceFile readsFile

|  |  |
| --- | --- |
| Argument | Description |
| ComparativeUtil | Command to run the algorithms to assemble the sequences. |
| ComparativeUtil /s | Command to execute all five steps in order and get the scaffolds. |
| referenceFile | FastA file containing the reference sequences. |
| readsFile | FastA file containing the reads. |

The following are the command line options:

Options

|  |  |
| --- | --- |
| Argument | Description |
| h | “help” - Print the help information. The default is **false**. |
| i | “meanlengthofinsert” - Mean Length of clone library. The default value is 0. |
| k | “kmerlength” - Set kmer length. The default value is 10. |
| m | “Mumlength” - Mum Length. The default value is 20. |
| n | “clonelibraryname” - Clone Library Name. The default string is empty. |
| o | “outputFile” - Output file |
| s | “scaffold” - Run scaffolding step after generating contigs. The default value is **false**. |
| sd | “standarddeviationofinsert” - Standard Deviation of Clone Library. The default value is 0. |
| v | “verbose” - Display verbose logging during processing. The default value is **false**. |

You also have the option of executing each of the following individual comparative assembly steps one at a time.

## NUCmer

To execute read Alignments

NUCmer (NUCleotide MUMmer) allows for multiple references and multiple query sequences to be aligned in a many vs. many fashion.

**Note**: While MUMUtil and NUCmerUtil both call into the code of MUMer, MUMUtil performs a Maximum Unique Match while NUCmerUtil does this as well as other operations such as cluster generation.

NucmerUtil.exe [options] ReferenceFile QueryFile

|  |  |
| --- | --- |
| Argument | Description |
| NucmerUtil | Command to align multiple reference and multiple query sequences in a many-vs-many fashion. |
| ReferenceFile | FastA file containing the reference sequences. |
| QueryFile | FastA file containing the query sequences. |

Options

|  |  |
| --- | --- |
| Argument | Description |
| b | “breaklength” - Distance an alignment extension will attempt to extend poor scoring region before giving up. The default value is 200. |
| c | “mincluster” - Minimum cluster length. The default value is 65. |
| d | “diagfactor” - Maximum diagonal difference factor for clustering.  For example, diagonal difference / match separation. The default value is 0.12. |
| e | “reverse” - Align only the reverse strand of the query sequence to the forward strand of the reference. The default value is **false**. |
| f | “forwardAndReverse” - Align only the forward and reverse strands of each sequence. Default value is **false**. |
| g | “maxgap” - Maximum gap between two adjacent matches in a cluster. The default value is 90. |
| h | “help” - Print the help information. The default is **false**. |
| l | “minmatch” - Minimum length of a maximal exact match. The default value is 20. |
| m | “mum” - Use anchor matches that are unique in both the reference and query. The default value is **false**. |
| n | “extend” - Toggle the outward extension of alignments from their anchoring clusters.  Set to not extend will prevent alignment extensions but still align the DNA between clustered matches and create the .delta file. The default value is to extend, **NotExtend** = **false**. |
| o | “outputFile” - Output file |
| r | “mumreference” - Use anchor matches that are unique in the reference but not necessarily unique in the query. The default value is **true**. |
| x | “maxmatch” - Use all anchor matches regardless of their uniqueness. The default value is **false**. |
| v | “verbose” - Display verbose logging during processing. The default value is **false**. |

## RepeatResolutionUtil

To execute Repeat Resolution

Position query sequences on a reference based on the alignment information contained in the delta file. This utility can utilize mate-pair information to place repetitive sequences, or random placement to simulate even coverage.

RepeatResolutionUtil.exe [options] InputDeltaAlignmentFile InputReadsFile

|  |  |
| --- | --- |
| Argument | Description |
| RepeatResolutionUtil | Command to resolve the repeats. |
| InputDeltaAlignmentFile | File containing delta alignments. |
| InputReadsFile | FastA file containing the reads. |

Options

|  |  |
| --- | --- |
| Argument | Description |
| -h | Print the help information |
| -o | Output file |
| -v | Display verbose logging during processing. |

## LayoutRefinementUtil

To execute Layout Refinement

LayoutRefinement.exe [options] InputdeltaAlignmentFile

|  |  |
| --- | --- |
| Argument | Description |
| LayoutRefinement | Command to refine the layout. |
| InputDeltaalignmentfile | File containing delta alignments. |

Options

|  |  |
| --- | --- |
| Argument | Description |
| -h | Print the help information |
| -o | Output file |
| -v | Display verbose logging during processing. |

## ConsensusGenerationUtil

To execute Consensus generation

Read layout information from the contig file describing positions of reads, and create multi alignments and/or consensus sequences for them.

ConsensusGenerationUtil.exe [options] InputDeltaAlignmentFile InputReadsFile

|  |  |
| --- | --- |
| Argument | Description |
| ConsensusGenerationUtil | Command to generate the contig sequences. |
| InputDeltaAlignmentFile | File containing delta alignments. |
| InputReadsFile | FastA file containing the reads. |

Options

|  |  |
| --- | --- |
| Argument | Description |
| -h | Print the help information |
| -o | Output file |
| -v | Display verbose logging during processing. |

## ScaffoldUtil

To execute Scaffolding

This step uses the output from the consensus generation as the input, ContigFile.

ScaffoldUtil.exe [options] ContigFile ReadsFile

|  |  |
| --- | --- |
| Argument | Description |
| ScaffoldUtil | Command to generate the scaffolds using mate pair information. |
| ContigFile | FastA file containing the contig sequences generated. |
| ReadsFile | FastA file containing the reads. |

Required Parameter

|  |  |
| --- | --- |
| Required Parameters | Description |
| -k:<int> | Length of k-mer. |

Options

|  |  |
| --- | --- |
| Argument | Description |
| -d | Depth for graph traversal.  Default is 10. |
| -h | Print this usage message.  Default is **false**. |
| -k | Length of k-mer.  Default is 10. |
| -o | Output file.  Default is **null**. |
| -r | Number of paired read required to connect two contigs. Default is 2. |
| -v | Display verbose logging during processing.  Default is **false**. |

# Glossary

This section defines some basic bioinformatics terminology that is relevant to the project. It contains only terms that are used later in this paper; it is not a complete list.

Assembler

Sequencer assembler algorithms used to merge short sequences or reads to reconstruct an original or base sequence.

Annotation

The process of attaching biological information to sequences. It encompasses identifying elements on the genome, a process called gene prediction, and attaching biological information to these elements.

BAM

A binary equivalent to SAM.

BED

Browser Extensible Display. A plain text file format for data that describes sequence ranges.

Bioinformatics

A discipline that uses mathematical, statistical, and computational approaches to analyze DNA and amino acid sequences and related information.

BLAST

The Basic Local Alignment Search Tool (BLAST) compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families.

Breakpoint

The situation where the alignment of a read to the reference consists of more than one contiguous segment, or a single segment that does not extend to the end of the read.

Comparative Assembly

The assembly of a genome using the sequence of a close relative as a reference.

Consensus

A reconstructed sequence of nucleotides or amino acids inferred from an alignment of multiple subsequences. It is also known as a contig.

Contig

A set of nucleotide or amino acid sequences—presumably part of a larger molecule—that have been aligned and overlap with each other. A non-redundant sequence formed by joining one or more smaller sequences, based on sequence overlap. The smaller sequences can be individual sequence reads (traces) or entire clone sequences. There should be no gaps in a contig (although there may be short runs of Ns due to ambiguous base calls). The number of contigs reported in the sequence data and their spectrum of sizes are important parameters in the analysis of genes.

DNA (deoxyribonucleic acid)

A molecule that consists of a double chain of nucleotides and codes the genetic information for all organisms.

EBI (European Bioinformatics Institute)

A bioinformatics research institute. It hosts one of the available BLAST services.

FASTA

FASTA format—also known as Pearson format—is a text-based data format for representing nucleotide or peptide sequences. It represents base pairs or amino acids with single-letter codes and allows the sequences to be preceded by sequence names and comments.

FASTQ

A plain text format for storing sequence data that combines a FASTA sequence with its quality data.

GFF (general feature format)

A plain text file format for describing DNA, RNA, and protein sequences.

Gap

A sub region within an object where there is no known sequence. Generally represented as a series of the letter ‘N’.

GenBank

The GenBank sequence database is an annotated open-access, collection of all publicly available nucleotide sequences and their protein translations. It is hosted by the NCBI as part of the International Nucleotide Sequence Database Collaboration (INSDC).

Genomics

The study of genetic sequences.

homologues

Same copy of DNA. Such as in the case of two copies for all autosomal chromosomes, one coming from the mother and the other coming from the father. The pair is called homologues.

k-mer

Identifies a region within molecules such as DNA.

N50 length

A statistic to indicate sequence contig length or scaffold length that is systematically larger than the simple average length—the N50 length (the largest length such that 50% of all base-pairs are contained in contigs of this length or larger).

NCBI

The National Center for Biotechnology Information.

nucleotide

The basic structural unit of DNA and RNA. They are usually referred to by their purine base. DNA uses four nucleotides: adenine, guanine, thymine, and cytosine, commonly abbreviated as A, G, T, and C. RNA also uses A, G, and C, but replaces T with uracil (U).

Phylogenetics

A phylogenetic tree describes evolutionary relationships between organisms that derive from a common ancestor.

Polymorphism

Natural variations in a gene, DNA sequence, or chromosome. The most common type of polymorphism involves variation at a single base pair.

Polyploidy

Occurs in cells and organisms when there are more than two paired (homologous) sets of chromosomes.

Protein

A molecule that consists of a chain of amino acids.

RNA (ribonucleic acids)

A single chain of nucleotides.

Sequence

Defines the structure of polymers such as DNA, RNA, and proteins.

SAM (sequence alignment map)

A plain text file format for data that describes nucleotide alignment.

Scaffold

(supercontig) A non-redundant sequence formed by joining one or more contig sequences. The distinction is that no sequence overlap is required to construct the larger sequence. Additional information, such as clone end analysis, can support the relationship. There can be, and typically there are, gaps in a scaffold.

Shotgun sequencing

Also known as shotgun cloning, a method used for sequencing long DNA strands. DNA is broken up randomly into numerous small segments, which are sequenced using the chain termination method to obtain reads. Multiple overlapping reads for the target DNA are obtained by performing several rounds of this fragmentation and sequencing.

SNP (single-nucleotide polymorphism)

Items represent sequence variations between species or paired chromosomes.

Synteny

The condition of two or more genes being on the same chromosome whether or not there is demonstrable linkage between them.