CoCoNUT

Computational Comparative GeNomics Utilities Toolkit

Windows Version 1.0

User Manual and Tutorial

Mohamed I. Abouelhoda

March 29, 2009

Contents

Chapter 1

Introduction

1.1 *CoCoNUT*

The Windows version of *CoCoNUT* is a software system for finding regions of high similarity (candidate regions of conserved synteny) among two genomes.

To cope with the large genomic sequences, *CoCoNUT* is based on the *anchor-based strategy* that is composed of three phases:

1. Computation of fragments (similar regions among genomic sequences).
2. Computation of highest-scoring chains of colinear fragments. Each of these highest-scoring chains corresponds to a region of similarity. The fragments in each of such chains are *the anchors*.
3. Post-processing of the regions between the anchors of a chain, by visualizing them or computing detailed alignment on the characetr (nucelotide) level using standard dynamic programming.

The fragments we use in the current Windows version are computed using the `genometools` package, which is based on an efficient data structure called the *enhanced suffix array*. Our program *CHAINER* carries out the chaining step in our system.

1.2 Installation and system requirements

Detailed installation on Windows server is descried in the accomanying document "CoCoNUTOVerWindowsHPC2008.doc". Here we layout the main requirements.

1.3 System requirements

- Windows HPC server 2008 operating system.
- Perl (at least version v5.8.1).

- cygwin.
- Gnuplot (at least version 3.7), optional for producing images of comparison results.
- The `genometools` package (`www.genometools.org`). We distribute a pre-compiled Windows version of it with *CoCoNUT*.

1.4 Installation

1.4.1 Pre-compiled version

The distributed version of *CoCoNUT* is pre compiled for Windows HPC server 2008. It includes also pre-compiled the `genometools` package.

To have *CoCoNUT* running, first decompress the compressed distributed file.

The destination directory of the system is called `CoCoNUT.distrib`. It is recommended to download the test data from *CoCoNUT* website and put it in the *CoCoNUT* directory to test your installation.

For successful running, we recommend that you do not change the directory structure of the system. Moreover, *CoCoNUT* must be called while being in its directory. The main program interface `coconut.pl` exists in the `coconut` directory. In addition, you will find the `config` file, where you can specify the location of the `genometools` package, as will be explained in Section 1.5.

1.5 The config file

The config file contains the paths for the programs needed for the system. Below we show the default config file. The lines separated by # are comment lines. The line starting with “FRAGMENT=” specifies the path to the `genometools` package. Each specified directory should contain the associated programs distributed with multimat and ramaco. The line starting with “CHAINING=” specifies the path to the programs *CHAINER*, *chainer2permutation.x*, and other programs for format transformation.

```
# PATHS

#fragment generation  directory for genometools
FRAGMENT=bin/genometools-current/bin

#chaining directory
CHAINING=bin/i686-pc-linux-gnu-32-bit/chainer
```

1.6 Files and directory structure

The system has the following directory structure:

- `bin` : This directory contains subdirectories for `genometools` and *CHAINER*

- `finalscripts`: This directory contains Perl's scripts for performing deferent the comparison. These scripts encapsulate the programs in the `bin` directory. This directory includes the sub-directory: `comp_finished`, which includes scripts for comparing finished genomes.
- `src`: This directory contains source code for *CHAINER* and the post-processing modules

1.7 Test data

Test data can be downloaded from the system web-page. We have supplied some data to demonstrate our system. In this manual we will assume that the test data directory exists in the `CoCoNUT.distrib` directory. The test data directory contains the following directories:

- `ecoli_shig`: contains the three fasta files `NC_000913.fasta`, `NC_007384.fasta`, and `NC_007613.fasta` containing the three bacterial genomes *Escherichia coli Shigella sonnei Ss046*, and *Shigella boydii Sb227*, respectively.
- `chlamd`: contains the three files `AE001273.fasta`, `AE001363.fasta`, and `AE002160.fasta` containing the three bacterial genomes *Chlamydia trachomatis*, *Chlamydia pneumoniae*, and *Chlamydia muridarum*, respectively.

1.8 Basic algorithms in *CoCoNUT*

To completely understand how our system work, we present here some details about the algorithms in our system and how they are used to solve the previously mentioned comparative genomic tasks.

1.9 Basic concepts and definitions

For $1 \leq i \leq 2$, let S_i denote a string of length $|S_i|$. The string $S_i[1..n]$ is a DNA sequence or a complete genome of n characters (nucleotides). $S_i[l_i \dots h_i]$ is the substring of S_i starting at position l_i and ending at position h_i. A fragment is a similar region occurring in the given genomes. This region is specified by the substrings $S_1[l_1 \dots h_1], S_2[l_2 \dots h_2]$. A fragment is called *exact* if $S_1[l_1 \dots h_1] = S_2[l_2 \dots h_2]$, i.e., the substrings composing it are identical. In this case, one speaks of fragments of the type *exact match*. Such a match is called *left maximal*, if $S_i[l_i - 1] \neq S_j[l_j - 1]$, for some $i \neq j$, and it is called *right maximal* if $S_i[h_i + 1] \neq S_j[h_j + 1]$, for some $i \neq j$. A *maximal exact match*, denoted by *MEM*, is left and right maximal, i.e., the substrings cannot be extended to the left and to the right in all S_i, $1 \leq i \leq k$, simultaneously.

Our system can basically use any kind of fragments, provided that they are output in the adopted *CoCoNUT* format.

Geometrically, a fragment f of k genomes can be represented by a rectangle in $\mathbb{R}^2$ with the two extreme corner points $beg(f)$ and $end(f)$. $beg(f)= (l_1, l_2)$, where the fragment starts at positions l_1, l_2 in S_1 qnd S_2 respectively, and $end(f)= (h_1, h_2)$, where it ends at positions h_1 and h_2 in S_1 and S_2 respectively; see Figure 1.1. With every fragment f, we associate a positive weight $f.weight \in \mathbb{R}$. This weight can, for example, be the length of the fragment (in case of exact fragments) or its statistical significance. In our system, we use the fragment length as the defualt fragment weight.

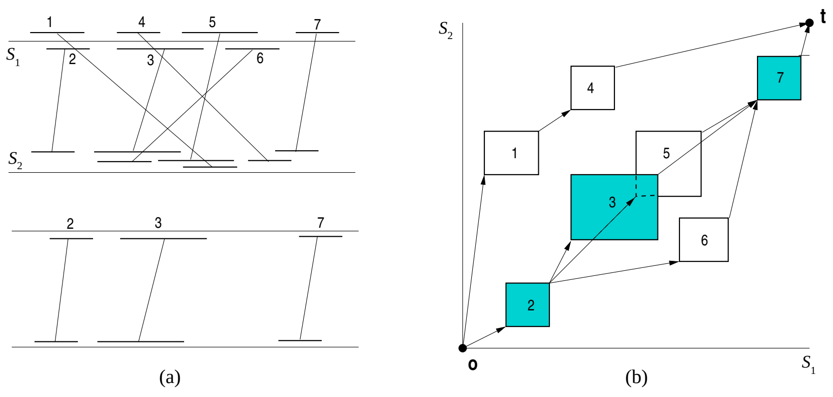

Figure 1.1: The fragments in (a) can be represented by (hyper-) rectangles in a space with dimension equals the number of genomes, and each axis corresponds to one genome, as shown in (b). Given a set of fragments, an optimal global chain of colinear non-overlapping fragments (left figure) starts and ends with two imaginary fragments of weight equals one: **O** and **t**.

We also define two imaginary points $\mathtt{0} = (0, 0)$ (the origin) and $\mathtt{t} = (|S_1|, |S_2|)$ (the terminus) as imaginary fragments with weight 1. In some output files in *CoCoNUT*, the origin point might be reported; it is done for ease of computations.

The program *CHAINER* is used on our system to compute significant chains of fragments. In case of comparing genomic sequences, each chain corresponds to a region of similarity among the given genomes.

1.10 The basic chaining problem

Definition 1.10.1 We define a binary relation $\ll$ on the set of fragments by $f \ll f'$ if and only if $end(f).x_i < beg(f').x_i$ for all $1 \leq i \leq k$. If $f \ll f'$, we say that f *precedes* f'.

Definition 1.10.2 A *chain* of colinear non-overlapping fragments (or chain for short) is a sequence of fragments $f_1, f_2, \ldots, f_\ell$ such that $f_i \ll f_{i+1}$ for all $1 \leq i < \ell$. The $score$ of C is

$$score(C) = \sum_{i=1}^{\ell} f_i.weight - \sum_{i=1}^{\ell-1} g(f_{i+1}, f_i)$$

where $g(f_{i+1}, f_i)$ is the cost of connecting fragment f_i to f_{i+1} in the chain. We will call this cost *gap cost*. The gap cost implemented in the current version of *CHAINER* is defined as follows. For two fragments $f \ll f'$,

$$g(f', f) = \sum_{i=1}^{2} |beg(f').x_i - end(f).x_i|$$

That is, the gap cost between two fragments is the distance between the end and start point of the two fragments in the L_1 (rectilinear) metric.

Given n weighted fragments from two or more genomes, the following problems can be defined:

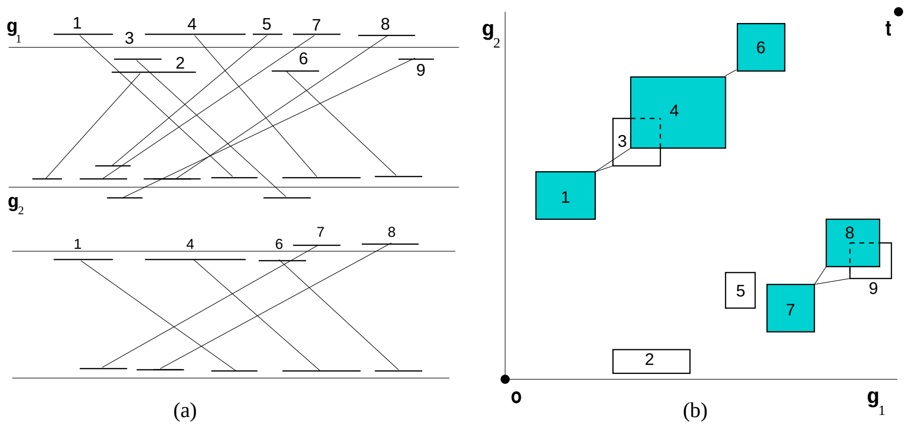

Figure 1.2: Computation of an optimal local chain of colinear non-overlapping fragments. The optimal local chain is composed of the fragments 1, 4, and 6. The local chains $\{1, 3, 6\}$ and $\{1, 4, 6\}$ share the fragment 1 and make up a cluster of the fragments $\{1, 3, 4, 6\}$. The representative chain of this cluster is the chain $\{1, 4, 6\}$. The chain $\{7, 8\}$ is a representative chain of the cluster $\{7, 8, 9\}$.

- The *global chaining problem* is to determine a chain of maximum score starting at the origin 0 and ending at terminus t.

  Such a chain will be called *optimal global chain*. Figure 1.1 shows a set of fragments and an optimal global chain.

- The *local chaining problem* is to determine a chain of maximum score ≥ 0. Such a chain will be called *optimal local chain*. It is not necessary that this chain starts with the origin or ends with the terminus. Figure 1.2 shows a set of fragments and an optimal local chain.

- Given a threshold T, the *all significant local chains problem* is to determine all chains of score $\geq T$. It is easy to see that the all significant local chains problem is the generalization of the local chaining problem.

In local chaining, some chains can share one or more fragments composing a cluster of fragments. In the example of Figure 1.2, the local chains $\{1, 3, 6\}$ and $\{1, 4, 6\}$ share the fragment 1 and make up a cluster of the fragments $\{1, 3, 4, 6\}$. The cluster $\{7, 8, 9\}$ contains two local chains $\{7, 8\}$ and $\{7, 9\}$. To reduce the output size, we report the clusters and from each cluster we report a local chain of highest score as a representative chain of this cluster. This representative chain is a significant local chain. In the example of this figure two chains are reported: $\{1, 4, 6\}$ and $\{7, 8\}$. The fragments $\{2\}$ and $\{5\}$ in the figure are chains of one fragment. They would be reported if their score is $\geq T$.

CHAINER uses techniques from computational geometry to solve the chaining problems. These techniques are based on *orthogonal range search for maximum*, which is implemented in *CHAINER* using an optimized version of kd-tree [4]. For more algorithmic aspects of *CHAINER* and these techniques, see [1–3].

The user can constrain the gap length between the fragments, which is achieved by limiting the region of the range queries. In other words, no two fragments can be connected in a chain if the number of characters separating them exceeds a user-defined threshold. This option prevents unrelated fragments from extending the chain.

Chapter 2

CoCoNUT in a nutshell: Exploring the main functions

The objective of this chapter is to test your installation and to explore the main functionalities of *CoCoNUT*. We will briefly investigate some of the output files to ascertain the correct installation. We will use the program default estimated parameters, which might not be the best. More on parameter tuning is addressed in detail in the following sections. (Briefly this is done by re-editing the parameter file and passing it to *CoCoNUT*.)
By calling *CoCoNUT* without parameters, you will obtain the following.

```
> coconut.pl
Usage: coconut.pl -pairwise
```

This means that you have to specify the task you want to accomplish and pass in addition the arguments and input data.

2.1 Comparing finished genomes

.

2.1.1 Test data

We use the two bacterial genomes *Chlamydia trachomatis* (`AE001273.fasta`), *Chlamydia pneumoniae* (`AE001363.fasta`). (These are also in the test data distributed with *CoCoNUT*). We assume that these data are in the `testdata/chlamd` directory within the *CoCoNUT* directory.

2.1.2 Main options of *CoCoNUT*

To see the main options of this task, run the following command.

```
> coconut.pl -pairwise

Usage: perl coconut.pl -pairwise  <Options> seq_1 seq_2
```

```
Options:
  -pr        : parameter file (optional), if not given defaults are computed
  -v         : verbose mode, i.e., deisplay of the program steps
  -forward   : run the comparison for forward strands only
  -indexname : specify the index, if constructed
  -useindex  : do not construct index again
  -usematch  : do not compute matches again, this construct no index
  -prefix    : specify a prefix name for the output files
```

2.1.3 Calling *CoCoNUT*

To find similar regions in the two aforementioned genomes, run *CoCoNUT* as follows.

```
> coconut.pl -pairwise -v -plot \\
testdata/chlamd/AE001273.fasta testdata/chlamd/AE002160.fasta
```

The argument `-pairwise` specifies the task of comparing two genomes; the other arguments and options are specified as follows:

- The option `-v` (verbose mode) shows the intermediate steps of *CoCoNUT*.
- The option `-plot` produces Postscript 2D plots of the similar regions (chains). The produced plots are projections of the chains with respect to all pairwise genomes.

The parameters estimated (e.g., minimum fragment length, and maximum gap between fragments) for comparing these genomes are stored in the automatically generated file `parameters.auto`. You can re-edit the paramters and pass this file to *CoCoNUT* and run the system again, starting from the phase with the changed paramters using the option `usematch`. For more details about the format of the parameter file, see Section 3.2. The other options are handled below.

2.1.4 Ouput files

The output files have the prefix `fragment` and they are stored in the directory of the first genome (the prefix and the destination directory can be changed with the option `-prefix`, as explained in Section 3.1). As a result of the above options, the following set of files are generated.

- fragment.ppp, fragment.ppm, fragment.pmp, and fragment.pmm, containing fragments in *CHAINER* format. The letter "p" in the extension corresponds to the positive (+) strand, and the letter "m" corresponds to the negative (−) strand. The file "fragment.pp" contains fragments from the positive strands of the two genomes. The file "fragment.pm" contains fragments from the positive strand of genome 1 and from the negative strand of genome 2.
- *.pp.chn and .pm.chn, chain files, storing the resulting chains from the *.pp and *.pm fragment files.
- *.pp.ccn and *.pm.ccn files, containing the resulting chains for the respective fragment files, but in compact form for plotting. I.e., just the chain boundaries are stored, not the fragments of the chains.

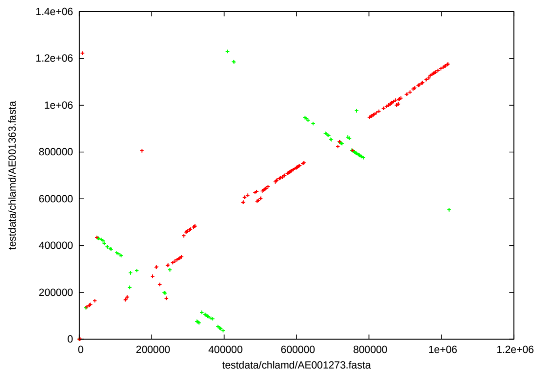

Figure 2.1: Comparison of the two bacterial genomes *Chlamydia trachomatis* (`AE001273.fasta`) and *Chlamydia pneumoniae* (`AE001363.fasta`). Red lines are chains between strands with the same orientation and green lines are chains between strands with different orientations (inversion).

- *.dat and *.gp files, used for generating plots using `gnuplot`.
- *.1x2.gp.ps, postscript files containing the 2D plot of the chain.

To visualize the output chain with respect to the first and second genome, run any postscript viewer over the file `fragment.mm.ccn.1x2.gp.ps` in the `testdatachlamd` directory. Figure 2.1 shows the postscript files for this run.

Chapter 3

Comparison of finished genomes

In this chapter, we discuss the comparison of finished genomes. For single-chromosomal genomes, each genome must be given as a single fasta file. Multi-chromosomal genomes should be compared by all pairwise comparison of chromosomes, where each chromosome is given as a single fasta file.

For repeating some parts of the comparison with different parameters, the user can re-start the comparison at two points: (1) after the index generation, (2) after the fragment generation. For example, if the user already computed the fragments, and computed the chains, then he could run the alignment program later using the already computed fragments and chains. He can also repeat this step with different parameters.

3.1 Calling *CoCoNUT*

The program *CoCoNUT* is called as follows:

`> coconut.pl` -pairwise -fp multimat|ramaco [*options*] $genome_1 \ldots genome_k$

Here is a description of the options:

`-pairwise`
: Specifies the task of comparing two finished genomes.

`-pr` parameter file
: Specifies the parameter file containing the parameters of the system. This file is generated automatically, if no file is specified. All the options, except for `-v` and `-plot`, are functionless if a parameter is specified. That is, the options in the parameter file dominate. The format of the parameter file is given in Section 3.2.

`-forward`
: run the comparison for forward strands only. This option is functionless if a parameter file is given. Restricting the processing to the forward strand only is achieved by deleting the option `-p` from the fragment line in the parameter file.

`-plot`

produce Postscript 2D plots of the chains. For multiple genomes, the plots are projections of the chains w.r.t. each pair of genomes.

`-useindex`
: do not construct the index again. This option is useful if one repeats the comparison with different fragment parameters. Also, with the program `ramaco`, one can compare the indexed genome to another genomes without re-constructing the index.

`-indexname`
: specify the index, if constructed. This option is useful to use indices that are already constructed, and stored somewhere in your system.

`-usematch`
: do not compute the fragments again. With this option the constructed index is used again.

`-prefix` prefix name
: specify a prefix name for the output files. This prefix name should include the destination path, otherwise the resulting files will be in the *CoCoNUT* directory. If this option is not set, then the default prefix for the index is `Index` and for the fragments and post-processing is `fragment`. The resulting files will be stored in the directory in which the first input genome resides.

`-v`
: verbose mode, i.e., display of the program steps.

3.2 The parameter file

The parameter file contains parameters for each step in the comparison. The parameter file, if not specified, is automatically generated with parameters estimated statistically. Below is an example of a parameter file. The lines separated by # are comment lines. The line starting with “FRAGMENT=” specifies the parameters passed to the fragment generation program. The line starting “CHAINING=” specifies the parameters passed to the chaining program *CHAINER*. In the remaining part of this chapter, we handle each of these parameters in detail along with the respective program implementing it.

```
FRAGMENT= -p -d -v -l 20
CHAINING= -l -chainerformat -lw 7 -gc 100 -length 40
```

3.3 The fragment generation phase

In the parameter file, the line starting with “FRAGMENT=” specifies the parameters passed to the fragment generation program. The parameters are

`-l` ℓ
: to specify the minimum fragment length, i.e., generate fragments with length at least ℓ,

`-d`
: to generate fragments for the positive strands only,

`-p`

to generate fragments between all the positive and negative strands.

`-v`

verbose mode, where some more details are stored in the output match file.

To restrict the processing to the forward strand only, one should delete the option `-p` from the fragment line.

3.3.1 The fragment generation parameters

Now, we discuss some issues about setting the parameters of the fragment generation phase.

The minimum fragment length The default minimum fragment length is estimated using a statistical model, which is suitable in average for all comparisons. However, for closely related sequences, the minimum fragment length can be increased, to avoid generating redundant fragments and unnecessarily increase the comparison time. For distantly related sequences, the minimum fragment length should be decreased as much as possible to achieve high sensitivity. Thus, we suggest you first run the system with the default parameters, unless you know that the sequences are closely related. Then you can reduce the minimum fragment length in the generated parameter file, which is called `parameters.auto`. Afterwards, you run the system again using the option "`-pr parameters.auto`", which sets the parameter file option. This procedure can be repeated until no improvement is observed.

3.4 The chaining parameters, and the program *CHAINER*

The chaining step in the *CoCoNUT* system is performed using the program *CHAINER*. In this chapter we discuss the parameters related to the comparison of multiple genomic sequences. In the above example of the parameter file, the line starting with "CHAINING=" specifies the parameters passed to this program. These parameters are

`-l` local chaining

to solve the local chaining program. The gap cost between the fragments in a local chain is the L_1 distance between the start and end points of the fragments.

`-chainerformat`

to report the resulting chains in *CHAINER* format. The resulting chain files have extension ".chn". For chains computed w.r.t. the reverse complement of any sequence, the coordinates are given w.r.t. the reverse complement sequences. *CHAINER* can report output in another table-wise format with extension `*.dbf`. However, in the *CoCoNUT* system we use the *CHAINER* format for the following phases of the system. In other words, if it is required to produce only chains without visualization and post-processing, one can use the default format or report to the stdout with the option "-stdout"; see the *CHAINER* manual for more details about the other formats.

`-gc` gap constraint

specifies the maximum *gap* length allowed between two fragments in the chain. For example, `-gc 300` imposes that the start and end point of two fragments f_i and f_{i+1} in a local chain are not separated by a distance larger than 300 bp in any genome, i.e., $beg(f_{i+1}).x_r - end(f_i).x_r \leq 300$ for every genome $0 < r \leq k$.

`-lw` multiplying factor

multiplies the weight of every fragment by a numerical value ℓ and uses the result as the weight of the fragment. The value ℓ can be a non-integer value. The default value for ℓ is 1 for global chains without gap costs and for cDNA/EST mapping, while it is 10 otherwise. Using this option is essential for finding homologous regions of reduced sequence conservation. In other words, to connect fragments with larger gap distances.

`-length` average chain length

specifies the minimum average chain length. For k genomes and a chain starting with fragment f and ending with fragment f', the average chain length is

$$\frac{1}{k}\sum_{i=1}^{k}(end(f').x_i - start(f).x_i)$$

For example, `-length 50` yields all representative local chains whose average length is larger than or equal to 50.

Other options of *CHAINER* that can be used with option “-l” include the following:

`-s` score

Specify the minimum *score* of the reported chains; the default value is zero. For example, `-s 500` yields all representative local chains whose score is larger than or equal to 500.

`-perc` percentage

reports chains whose minimum score is higher than a percentage of the highest score of a local chain. For example, `-perc 80` reports all chains whose score is at least 80% of the highest score of a local chain. Note that we take the maximum of the two parameters `-perc` and `-s` if both are set to different values.

`-d` chain size

specifies the *chain size*, i.e., the minimum number of fragments in a reported chain.

`-cluster` cluster file

reports cluster of chains in a cluster file; see *CHAINER* manual for details.

`-stdout` standard output

Output the chains to the standard output. The chains are reported in table format; see the *CHAINER* manual for more details. The default output in *CHAINER* is to store the chains in table-format in a file with extension `.dbf`. But for *CoCoNUT*, we adopt the *CHAINER* format.

See the *CHAINER* manual (distibuted with *CoCoNUT* manual) for more details about these and other options.

3.4.1 The input and output files for the chaining step

The fragment file produced by the fragment generation program is first transformed to the *CHAINER* format. Below is an example of a fragment file for three genomes in *CHAINER* format. The first line is a header line containing the keyword `>CHA 2` and specifying the number of genomes. Each fragment starts with the character #, which is followed by the fragment weight. As default in our system, the fragment weight is its length. The i^{th} bracket encloses the start and end positions of the fragment in the i^{th} genome. For example, the first fragment in this file starts at position 225137 and ends at position 225150 in the second genome.

```
>CHA 3
#14
[938654,938667] [225137,225150]
#14
[862792,862805] [437801,437814]
#14
[1041684,1041697] [949290,949303]
```

The format transformation step includes also a classification of the fragments w.r.t. the DNA strand. Assume the fragment file generated by the fragment generation program is called `fragment`, then the resulting files after the transformation have the extensions pp and pm, i.e., we have the files `fragment.pp` and `fragment.pm`. In the transformation step, it is also assured for a fragment from some negative strands that the coordinates in the negative strands are given w.r.t. the negative strands. (Note that the coordinates output from the fragment generation program are w.r.t. the positive strand only.)

The chaining step is done for each of the files output from the transformation program separately. That is, for the two genomes in the example given above, the chaining step is performed for each file separately. The output of the chaining step includes, for each combination, the following set of files:

- *chain files* with extension `*.chn`: These files contain the computed chains. For the three genomes in the example given above, we have as output the files `fragment.pp.chn` and `fragment.pm.chn`. Below is an example of a chain file for two genomes containing two chains:

  ```
  >CHA 3
  #4527.000000
  [1018162,1018192] [1175914,1175944] 1800
  [1018109,1018158] [1175861,1175910] 2378
  #2172.000000
  [1016423,1016436] [1174342,1174355] 971
  [1016318,1016331] [1174237,1174250] 1871
  [1015908,1015927] [1173827,1173846] 2034
  [1015863,1015880] [1173782,1173799] 1744
  ```

  The first line is a header line containing the keyword `>CHA 2`, specifying the number of genomes. Each chain starts with the character #, which is followed by the chain score. The fragments composing the chain are written in the following lines in a descending order. At the end of the line defining the fragment, the fragment id. (number) in the fragment file is given.

- *compact chain files* with extension `*.ccn`: These files contain chain files but in a compact form. That is, the chain boundaries are given without the fragments of each chain. For the above chain file, the compact chain file is

  ```
  >CHA 3
  #4527.000000
  [1018109,1018192] [1175861,1175944]
  #2172.000000
  [1015863,1016436] [1173782,1174355]
  ```

  For the two genomes example given above, we have as output the files `fragment.pp.ccn` and `fragment.pm.ccn`.

- *statistics files* with extension `*.stc`: This file contains statistics about the chaining task. Here is an example of a statistics file

  ```
  2 3041 262 14
  1041888 1229966
  # no. of chains: 112
  ```

  The numbers from left to right in the first line are number of genomes, number of fragments, maximum fragment length, and minimum fragment length. The numbers in the second line are the maximum end positions of the fragments in each genome. The third line stores the number of computed chains. For the three genomes example given above, we have as output the files `fragment.pp.stc` and `fragment.pm.stc`.

3.4.2 The chaining parameters

The most important parameters that affect the sensitivity and specificity of the procedure are (1) the multiplying factor l_w and (2) the gap constraint gc.

The multiplying factor should be adjusted in connection with the gap constraint, and the minimum fragment length parameter. As we mentioned before, the score of a chain C is

$$score(C) = \sum_{i=1}^{t} f_i.length - \sum_{i=1}^{\ell-1} g(f_{i+1}, f_i)$$

A significant chain should have score larger than zero. With the multiplication factor, the chain score is

$$score(C) = \sum_{i=1}^{t} l_w \times f_i.length - \sum_{i=1}^{\ell-1} g(f_{i+1}, f_i) > 0$$

Assume, in the worst case, that a chain has only two fragments f_1 and f_2 and assume that both fragments has the minimum fragment length ℓ. For the L_1 metric, $g(f_1, f_2) = \sum_{i=1}^{k}(f_2.x_i - f_1.x_i) = k \times gc$, and we have

$$2 \times l_w \times \ell - k \times gc > 0$$

For automatically generated files, we set $gc = 500$ for two genomes, and $gc = 1000$ for more than two genomes. We then use the above formula to compute l_w.

Note that this is the worst case for a chain in our program, because a similar region usually has a chain of multiple fragments.

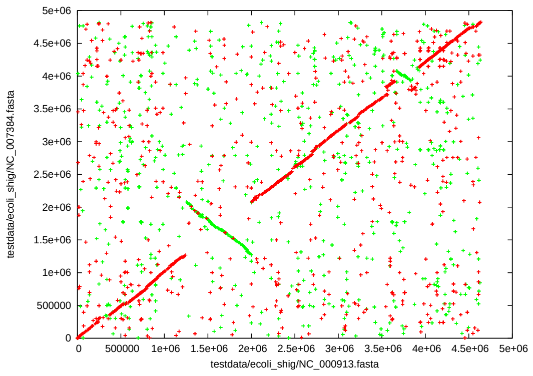

Figure 3.1: Comparison of the two genomes *E. coli* and *S. sonnei*. Red lines correspond to chains between positive strands and green lines correspond to chains between the +ve strand of the genome on the x-axis and the -ve strand of the genome on the y-axis.

3.5 The 2D plots

The 2D plots are generated for either the chains. The option `-plot` generates these plots. Choosing the option `-plot`. In the 2D plots, each chains is represented by a line connecting its start and end points in the genomes. We use red lines in the direction "north-east" to represent chains computed between forward strands, and use green lines in the direction "south-east" to represent chains computed between the positive strand of the genome on $x - axis$ and the negative strand of the genome on $y - axis$. Figure 3.1 shows an example for 2D plots from comparing tw0 genomes.

For 2D plots of chains, the input to the plotting step are the compact chain files with extension `*.ccn` computed by *CHAINER*. These files are processed to adjust the coordinates back to the positive strand, and to transform the format into the `gnuplot` format.

3.6 Summary of output files

At this point, we summarize the resulting output files after carrying out each step. The following table describes the output files for each step. Of course not all files are generated; only those that meet the users options. In this table, we assume that the user assigns the resulting files the prefix `fragment`, which is also the fragment file name.

Step	files
Fragment generation	`fragment`
Format transformation	`fragment.p{p|m}`$^+$ Ex., `fragment.pp, fragment.pm`
First chaining	1- Chain files with extension `.chn` Ex., `fragment.pp.chn, fragment.pm.chn` 2- Compact chain files with extension `.ccn` Ex., `fragment.pp.ccn, fragment.pm.ccn` 3- Statistics files with extension `.stc` Ex., `fragment.pp.stc, fragment.pm.stc`
2D plots	The plots are with the extension `.ps` 1- For first chaining, we have `fragment.mm.ccn.1x2.gp.ps`

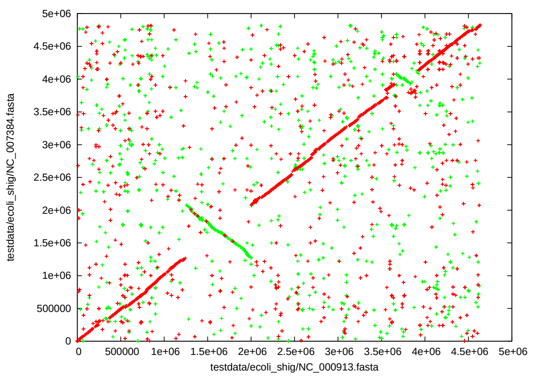

Figure 3.2: Projections of the comparison of the two bacterial genomes *E. coli* and *S. sonnei*.

3.7 Tutorial: Comparison of two genomes

To demonstrate the usage of *CoCoNUT*, we show step-by-step how to compare the two bacterial genomes *Escherichia coli* and *Shigella sonnei Ss046*. These genomes are in the directory ~`/CoCoNUT/testdata/e`

Running with default parameters: From *CoCoNUT* basic directory, you can start the comparison by using the default parameters. This is reasonable because it is assumed that we have no idea how similar the three genomes are. We would like also to plot the chains, to see how good the parameters are before computing the alignment. Therefore, we use the option `-plot`. Moreover, the verbose mode option `-v` is used to see the intermediate step of the program. It would also be more convenient to assign a prefix to the output files; we can choose the prefix `EcSsSb`. Because the genomes are of small size, we can directly use the program multimat. The command line for calling *CoCoNUT* is

```
> coconut.pl -pairwise testdata/ecoli_shig/NC_000913.fasta
testdata/ecoli_shig/NC_007384.fasta -v -plot -prefix testdata/ecoli_shig/EcSsSb
```

The following is the automatically generated parameter file, which is alway called `parameters.auto` in our system. It sets the minimum fragment length to 15 bp. There is only one chaining step, where the length of each fragment is multiplied by 102. The maximum gap between two fragments in a chain is set to 1000 bp. Chains with average length 30 bp are filtered out.

```
FRAGMENT= -p -d -v -l 15
CHAINING= -l -chainerformat -lw 102 -gc 1000 -length 30
```

Now, we can have a look at the resulting plot stored in the file `EcSsSb.ccn.1x2.gp.ps`, see Figure 3.2.

In the same directory, the index files have prefix `EcSsSb.index`. The fragment file generated by `genometools` is `EcSsSb`. The fragment files transformed to *CHAINER* format are `EcSsSb.pm` and `EcSsSb.ppp`. The chain files are `EcSsSb.pm.chn` and `EcSsSb.pp.chn`. The compact chain files are `EcSsSb.pm.ccn` and `EcSsSb.pp.ccn`. The statistics files for the chains are `EcSsSb.pm.stc` and `EcSsSb.pp.stc`.

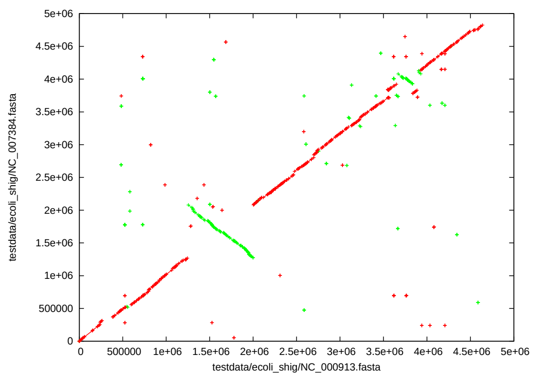

Figure 3.3: The comparison of the two bacterial genomes *E. coli* and *S. sonnei*, after filtering out chains with average length less than 1000 bp.

The coordinates in a chain file for a negative strand is given w.r.t. the negative strand. These coordinates are transformed back to the positive strand for plotting. The all compact chains for all combinations are stored in the file `EcSsSb.ccn.dat`. From this file, the projections for producing the plots are obtained.

Repeating some steps with better parameters: From the plots, we can directly observe that the three genomes are highly similar. We can also conclude that the chains with smaller average length might appear by chance. Therefore, we open the file `parameters.auto` and re-edit the option `-length 30` to be, say `-length 1000`. We then re-run *CoCoNUT* with the option `-usematch` so that the steps of index construction and the fragment generation are not repeated again. (Note that the changed option affects only the chaining step.) The command line is

```
> coconut.pl -pairwise testdata/ecoli_shig/NC_000913.fasta
  testdata/ecoli_shig/NC_007384.fasta -v
  -plot -prefix testdata/ecoli_shig/EcSsSb -pr  parameters.auto -usematch
```

The resulting plots are shown in Figure 3.3.

Bibliography

[1] M. I. Abouelhoda and E. Ohlebusch. A local chaining algorithm and its applications in comparative genomics. In *Proceedings of the 3rd Workshop on Algorithms in Bioinformatics, LNCS*, volume 2812, pages 1–16. Springer Verlag, 2003.

[2] M. I. Abouelhoda and E. Ohlebusch. Chaining algorithms and applications in comparative genomics. *Journal of Discrete Algorithms*, 3:321–341, 2005.

[3] M. I. Abouelhoda and E. Ohlebusch. CHAINER: Software for comparing genomes. In *Proceedings of the 12th International Conference on Intelligent Systems for Molecular Biology/3rd European Conference on Computational Biology*, ISMB/ECCB, 2004.

[4] J. L. Bently. K-d trees for semidynamic point sets. In *6th Annual ACM Symposium on Computational Geometry*, pages 187–197. ACM, 1990.

[5] Broad Institute. Sequencing and comparison of yeasts to identify genes and regulatory elements. `http://www.broad.mit.edu/annotation/fungi/comp_yeasts/downloads.html`.

[6] M. Höhl, S. Kurtz, and E. Ohlebusch. Efficient multiple genome alignment. *Bioinformatics*, 18:S312–S320, 2002.

[7] M. Kellis, N. Patterson, M. Endrizzi, B. Birren, and E.S. Lander. Sequencing and comparison of yeast species to identify genes and regulatory elements. *Nature*, 423:241–254, 2003.

[8] C. Wawra, M. I. Abouelhoda, and E. Ohlebusch. Efficient mapping of large cdna/est databases to genomes: A comparison of two different strategies. In *German Conference on Bioinformatics, LNI*, pages 29–43. GI, 2005.