WiseScaffolder v1.1 – User's Handbook

WiseScaffolder is a stand-alone semi-automatic application for genome scaffolding of pre-assembled contigs using mate-pair data. It produces editable scaffold maps, usable either to build gapped scaffolds or as a common thread for the manual improvement of scaffolding.

All python scripts proposed in this handbook are available on the website http://abims.sb-roscoff.fr/wisescaffolder/.

1. Installation

1.1. Requirements

WiseScaffolder has been tested with Python 2.7. It requires the BioPython package available at http://www.biopython.org.

1.2. Installation

WiseScaffolder comes packaged as a Python module. Its installation is done like any other Python module, either by the system administrator (section 1.2.1) or by any user (section 1.2.2). Detailed information about installation parameters and procedures can be found at: https://docs.python.org/2/install/.

1.2.1. System-wide installation

This installation needs system administration privileges and makes the wisca.py script available to all users of the system where it is installed.

Actual installation is done using the following command:

```
python setup.py install
```

1.2.2. User installation

The installation of WiseScaffolder in user mode is done in a directory belonging to the user performing the installation and does not require any administration privilege.

Actual installation is done using the following command:

```
python setup.py install --user
```

To determine the location of the wisca.py script, if needed, the following command can be used:

```
python -m site --user-base
```

It displays the base directory of the user's Python installation. The wisca.py script will then be located in the bin subdirectory.

For example, if the above command displays:

```
      /home/jdoe/.local
```

The script can be run using:

```
      /home/jdoe/.local/bin/wisca.py
```

This installation mode does not need any adjustment to the PYTHON_PATH environment variable, as the Python interpreter will automatically look for the package components in directories located below the user-base directory.

2. Prior to WiseScaffolder

This section is dedicated to the construction of the input files of WiseScaffolder.

2.1. Contig Renaming

Assemblers may produce contigs with various naming styles. This step allows a homogenization of contig identifiers to avoid any potential issue within WiseScaffolder.

```
   python contigs_renamer.py -i contigs.fasta -o contigs_renamed.fasta
```

2.2. Construction of the contig_info_file

Some genome assemblers (*e.g.* CLC AssemblyCell©) propose tools to remap the reads used for contig assembly and produce a report describing contig length mean coverage. Alternatively, a SAM file containing mapping of reads onto contigs, generated for instance using bowtie2, may be used along with the 'contig_info_builder.py' script to build the contig_info_file used by WiseScaffolder:

```
   bowtie2-build contigs_renamed.fasta contigs_renamed_DB

   bowtie2   --end-to-end   -x   contigs_renamed_DB   -q   reads.fastq   -S
reads_vs_contigs_renamed.sam

   python     contig_info_builder.py     -f     contigs_renamed.fasta     -s
reads_vs_contigs_renamed.sam -l 101 -o contigs.info
```

(-l: length of reads)

2.3. Mapping Mate-Pair reads to contigs

WiseScaffolder uses a SAM mapping of mate-pairs reads (MPs) against contigs. Using the bowtie2 database built in 2.2:

```
   bowtie2   --end-to-end   -x   contigs_renamed_DB   -1   reads_MP1.fastq   -2
reads_MP2.fastq -X 3500 -S MPs_vs_contigs.sam -p 15
```

(-X: MP library insert size; -p: number of processors)

Alternatively, the mapping may be realized using a variety of sequence alignment tools. WiseScaffolder can use a custom-designed tabulated file with columns set as in Figure 1 below (any other organization is possible but should be stated in the `MATEPAIRFILEINFO` section of the configuration file, see 3.2).

NB: All linking information, including multiple matches, will be taken into account in all WiseScaffolder tables describing links between and within contigs.

READ_ID	CONTIG HIT LEFT PAIR	IDENTITY	START	END	CONTIG HIT RIGHT PAIR	IDENTITY	START	END
M1:A2GRG:1:1101:10000:18521	contig_19	100.00	31124	31274	contig_8	100.00	24876	24735
M1:A2GRG:1:1101:10001:18495	contig_1	100.00	129525	129675	contig_1	100.00	123796	123670
M1:A2GRG:1:1101:10001:23187	contig_15	100.00	234540	234390	contig_7	100.00	240139	240284
M1:A2GRG:1:1101:10001:26304	contig_1	100.00	266541	266686	contig_1	99.34	260732	260582
M1:A2GRG:1:1101:10001:6387	contig_9	99.34	256636	256786	contig_9	100.00	251452	251316
M1:A2GRG:1:1101:10001:9774	contig_19	100.00	245762	245612	contig_19	100.00	247653	247803
M1:A2GRG:1:1101:10002:13017	contig_15	100.00	208345	208195	contig_15	100.00	213959	214109
M1:A2GRG:1:1101:10002:13653	contig_1	100.00	7866	7995	contig_1	100.00	2905	2755
M1:A2GRG:1:1101:10002:26810	contig_4	100.00	44607	44757	contig_4	100.00	45001	44851
M1:A2GRG:1:1101:10002:5745	contig_19	100.00	5008	4858	contig_19	100.00	4738	4885

Figure 1: Example of the custom 7-columns tabulated MP mapping file required as input to run WiseScaffolder. This file specifies the MP read identifiers and the contig best matches as well as the matching positions and percentage of identity for both pairs.

2.4. Subsampling MP mappings

In order to lower the memory impact of WiseScaffolder, the amount of MP mappings may be reduced by subsampling:

```
    python   sam_subsampler.py   -s   MP_mappings.sam   -n   100000   -l   101   -o
MP_mappings_subsampling_100000.sam
```

(-n: number of sequences to be extracted; -l: read length (here 101 bp))

NB: WiseScaffolder requires the SAM header, constituted of lines starting with '@', to recognize a SAM formatted mapping file from a custom-designed tabulated format (see 2.3).

3. Running WiseScaffolder

This section describes the subcommands of WiseScaffolder.

3.1. WiseScaffolder help module

The help module of WiseScaffolder (option -h or --help) provides adaptive information if a subcommand is stated:

`wisca.py -p -h`	➔ lists subcommands and general options
`wisca.py -p dumpconfig -h`	➔ lists options for the *dumpconfig* subcommand
`wisca.py -p preprocess -h`	➔ lists options for the *preprocess* subcommand
`wisca.py -p scaffold -h`	➔ lists options for the *scaffold* subcommand
`wisca.py -p buildfasta -h`	➔ lists options for the *buildfasta* subcommand

NB: the -p option of WiseScaffolder enables the generation of valuable progression messages and should be used systematically.

An overview of the options is shown in Table 1.

Table 1: Parameters to be used with the different WiseScaffolder subcommands

			SubCommands			
Parameters	**Short Name**	**Long Name**	***dumpconfig***	***preprocess***	***scaffold***	***buildfasta***
Configuration file output name		--configout	X			
Maximal insert size for the mate-pair library	-i	--insertsize	(X)	(X)	(X)	
Minimal length for 'big contigs' (generally equals the insert size)	-b	--bigcontigminimalsize	(X)	(X)	(X)	
Configuration file input name		--configin		X	X	
Contig info table	-c	--contig		X	X	
Mate-pair mapping table	-m	--matepairs		X	X	
Enable generation of manual scaffolding output files		--dumpfiles		(X)		
Chimera resolution file	-k	--kimera		(X)	(X)	(X)
Contig copy number file	-v	--coverage		(X)	(X)	
Scaffold description output file		--scaffoldout			X	
Contigs multifasta	-f	--fastafile				X
Scaffold description input file	-s	--scaffoldin				X
Directory name for the final fasta files of scaffolds and unscaffolded contigs	-r	--resultdir				X
Enable verbose mode	-p	--progress	(X)	(X)	(X)	(X)
Enable debug mode	-d	--debug	(X)	(X)	(X)	(X)
Display the help message, depending on the subcommand	-h	--help	(X)	(X)	(X)	(X)

X: parameter required to run a given subcommand
(X): optional parameter. In the case of "insertsize" and "bigcontigminimalsize", it will take priority over the corresponding parameter in the configuration file

3.2. Generate a configuration file

```
wisca.py -p dumpconfig --configout wisca.conf -i 5000 -b 5000
```

This command line generates an editable ‘wisca.conf’ configuration file stating the insert size of the MP library used (here 5 kbp) as well as the minimal size for ‘big’ contigs, which is usually set as the MP library insert size. In principle, a contig which size exceeds the MP library insert size (a.k.a. ‘big contig’) should not be linkable to more than two other big contigs and the resulting assembly is thus unambiguous. An example of the configuration file, in Python format, is shown in Figure 2.

```
{'BIGCONTIGMINIMALSIZE': 3000.0,
 'BORDERFACTOR': 1.5,
 'CHIMERAHISTOGRAMSIZE': 50,
 'CHIMERALINKTHRESHOLD': 0.5,
 'CLEANEDLINKSFILE': 'cleanedlinks.csv',
 'CLEANEDNEIGHBORHOODLINKSFILE': 'cleanedneighborhoodlinks.csv',
 'CONTIGFILEINFO': {'contig ID': 0, 'coverage': 1, 'length': 2},
 'HISTOGRAMBINSIZE': 50,
 'INSERTSIZE': 3000.0,
 'INSERTSIZEDISTRIBUTIONFILE': 'insertsizedistribution.csv',
 'LINKSFILE': 'links.csv',
 'LOGGERS': {'wisescaffolder': {'file': 'wisca.log', 'level': 20}},
 'MATEPAIRCONTAMINATIONINSERTSIZE': 1000,
 'MATEPAIRFILEINFO': {'left contig ID': 1,
                      'left first': 3,
                      'left identity': 2,
                      'left last': 4,
                      'read ID': 0,
                      'right contig ID': 5,
                      'right first': 7,
                      'right identity': 6,
                      'right last': 8},
 'MATEPAIRFILTERS': {'identityfilter': {'active': True, 'args': {'minimumidentity': 99.0}},
                     'samecontigfilter': {'active': False}},
 'NEIGHBORHOODLINKSFILE': 'neighborhoodlinks.csv',
 'NEIGHBORHOODLINKSLOCATIONFILE': 'neighborhoodlinklocation.csv',
 'ORIENTATIONLINKRATIO': 0.9,
 'SCAFFOLDINGLINKTHRESHOLD': 0.2,
 'SCAFFOLDINGMODULES': ['iterativescaffoldextender', 'smallcontiginserter']}
```

Figure 2: Example of WiseScaffolder configuration file.

Description of the different options:

BIGCONTIGMINIMALSIZE: sets the minimal size for ‘big’ contigs, which may then be scaffolded unambiguously, this parameter is generally set at the same value as the MP library insert size

INSERTSIZE: sets the insert size of the MP library. Corresponds to the rightmost value of the Gaussian distribution curve of insert sizes (see also the `'insertsizedistribution.csv’` output file in section 3.3)

MATEPAIRCONTAMINATIONINSERTSIZE: sets the estimated insert size of the short insert contamination of the MP library. Corresponds to the rightmost value of the Gaussian distribution curve of insert sizes associated with short insert reads, if present. Default is 1kb (see also the `'insertsizedistribution.csv’` output file in section 3.3)

CHIMERAHISTOGRAMSIZE and HISTOGRAMBINSIZE: sets the bin size for histogram outputs ‘link_location_graphs’ and `'insertsizedistribution.csv'`, respectively (default: 50)

BORDERFACTOR: after chimera resolution, this factor is used to discard links more distant than 1.5 times (default value) the MP library insert size from the big contig borders. This step eliminates some of the noise associated with MP library construction.

CHIMERALINKTHRESHOLD and SCAFFOLDINGLINKTHRESHOLD: set the minimal ratio of $\frac{Number\ of\ links\ between\ two\ contigs}{Number\ of\ links\ for\ the\ contig\ with\ the\ maximum\ number\ of\ links}$ necessary to consider as significant the contiguity of two contigs in the context of chimera detection (default: 0.5) and genome scaffolding (default: 0.2), respectively.

ORIENTATIONLINKRATIO: sets the minimal ratio (default: 0.9) of $\frac{Number\ of\ MP\ links\ indicating\ an\ opposite\ orientation\ of\ two\ consecutive\ contigs}{Number\ of\ MP\ links\ indicating\ a\ same\ orientation\ of\ two\ consecutive\ contigs}$ necessary to consider as significant the opposite orientation of the two contigs (Illumina MPs are expected to exhibit reverse forward orientation).

SCAFFOLDINGMODULES: lists the scaffolding modules to be used sequentially. The 'iterativescaffoldextender' scaffolds big contigs and may use single copy small contigs when no more big contig can be added. The 'smallcontiginserter' attempts to insert small contigs within the scaffolds built in the previous step. More scaffolding modules may be scripted and added to the chain of action.

LINKSFILE, CLEANEDLINKSFILE, CLEANEDNEIGHBORHOODLINKSFILE, NEIGHBORHOODLINKSFILE, INSERTSIZEDISTRIBUTIONFILE and NEIGHBORHOODLINKSLOCATIONFILE: set default names for outputs generated by the ***preprocess*** subcommand (see section 3.3).

LOGGERS: sets the default name for WiseScaffolder logs and level of verbosity (default: 20 corresponding to 'INFO'). The -d option allows to raise this level to 'DEBUG'.

CONTIGFILEINFO: sets the column index corresponding to a given descriptor in the 'contig_info_file'.

MATEPAIRFILEINFO: sets the column index in the custom-designed tabulated format of MP mappings.

MATEPAIRFILTERS: enable filters for the MP mapping file. The 'identityfilter' discards links with identity lower than the given value (default: enabled and at 99 % identity). The 'samecontigfilter' discards auto-links (i.e., pairs of reads mapping on the same contig) and lowers the amount of MP information stored in memory (default: disabled).

3.3. Preprocess mapping information, chimera detection, contig copy number estimation

```
    wisca.py   -p   preprocess   --configin   wisca.conf   -c   contigs.info   -m
reads_mapping.sam --dumpfiles
```

This command generates two important files: the chimerae resolution file 'chimera.csv' and the contig coverage/copy number file 'coverage.csv' as well as a folder containing for each contig a histogram showing the location of links associating this contig to all others. The chimerae resolution file contains one line per contig and states potentially chimeric (or multicopy) contigs, i.e. those displaying a significant number of links with more than two other contigs.

If the `--dumpfiles` option is enabled, additional files dedicated to manual scaffolding are generated, namely the *links.csv* and *cleanlinks.csv*, providing the amount of MP links associating each pair of contigs. The ‘*cleanlinks.csv*’ file ignores links far from the edges of big contigs and those too close to the edges, potentially corresponding to the MP library short-insert contamination. ‘*neighborhoodlinks.csv*’ and ‘*cleanneighborhoodlinks.csv*’, also provide number of links between contigs but make the distinction between links directed toward contigs located at the right or left of each contig, always set in forward orientation. The *neighborhoodlinklocation.csv* displays the mean position of MP links associating pairs of contigs and may be used to order small contigs at the vicinity of a large one. Finally, the *insertsizedistribution.csv* contains a histogram of the MP library insert sizes calculated using MPs for which both pairs fall on the same contig, allowing to determine the insert size of the MP library (and subsequently the minimal size for big contigs) and the maximal size of the short insert reads contaminating MP libraries, if any.

NB: WiseScaffolder was originally designed to work with Illumina© mate-pair libraries, the orientation of which is expected to be REVERSE-FORWARD. If the library orientation is set otherwise, libraries should be reverse-complemented accordingly.

3.4. Scaffold

```
   wisca.py -p scaffold --configin wisca.conf -c contigs.info -m
reads_mapping.sam --scaffoldout scaffolds_maps.txt -k chimera.csv -v
coverage.csv
```

This step generates an editable ‘scaffold_maps.txt’ describing the contig composition of each scaffold as well as a list of unscaffolded contigs. Contigs may remain unscaffolded when they belong to contaminants and share no links with the main contigs or when their insertion between big contigs is ambiguous.

At this stage, users may choose either to retrieve sequence information from the map using the buildfasta subcommand (section 3.5) or to manually improve the scaffold map (section 4) using MP-linking information provided in the different files generated by the preprocess subcommand (cf. 3.3).

3.5. Build FASTA of contigs

```
   wisca.py -p buildfasta -f contigs.fasta –s scaffolds_map.txt –r
wisca_scaffolds -k chimera.csv
```

Based on the scaffold map file, this subcommand generates a folder (here named ‘wisca_scaffolds’) containing FASTA formatted files for the scaffolds and the unscaffolded contigs. Within scaffolds, contigs are arbitrarily separated by 50 Ns.

4. Manual improvement of scaffolds and genome finishing

This section proposes methods and scripts to guide the post-scaffolding process, expectantly till genome closure.

4.1. Improving scaffold maps

The scaffold map generated by the *scaffold* subcommand of WiseScaffolder is editable and allows the user to resolve multi-copy contigs, collapsed repeats, etc. that most often result in scaffold breakpoints.

By taking into consideration the information contained in the various files produced by WiseScaffolder (essentially the cleanneighborhoodlinks.csv and the per contig link location histograms (folder link_location_graphs), one may improve the global assembly contiguity, e.g. by locating the ribosomal operon (a highly conserved 5-Kbp contig with copy number often >1) and its right and left neighbors.

4.2. Extraction of complementary MPs

Once the scaffold map is satisfactory, the user can either directly generate the scaffold sequences using the *buildfasta* subcommand or may use the MPs to locally reassemble contigs potentially covering gaps. For each big contig edge, the python script 'contig_edge_extractor.py' will generate for each contig edge a multifasta file of reads, which complementary pairs map to the contig edges within a range corresponding to the insert size and in the expected orientation (i.e., toward the gap to close):

```
   contig_edge_extractor.py -c contigs_renamed.fasta -t 3000 -i 3000 -s
MPs_vs_contigs.sam -d output_directory (-v)
```

(-t: minimal contig length to attempt read extraction; -i: insert size of the MP library).

For each contig above the threshold length, the script will produce two multifasta files of reads (left and right).

4.3. Assembly of contig extensions and contig

The multifasta files generated in 4.2 may then be assembled into 'locally assembled contigs'. Taking into account the fact that the extraction is local (except when a multicopy area is present in the contig edge), the assembly should perform well and result in long contigs overlapping contig edges and leading to gap closure.

The resulting 'locally assembled contigs' may then be assembled together with the original contigs using any software allowing contig visualization (e.g., Geneious©) and the consistency of resulting scaffolds may be verified using the scaffold map generated by WiseScaffolder.

4.4. Final gap closing

The remaining gaps are generally associated with either collapsed repeats or low complexity regions resulting in low sequencing coverage. The closure of these remaining gaps may be attempted by extracting and aligning reads matching a ~20 bp nucleotide motif located at the edge of the contig (using for instance the 'grep' unix command on the MP fasta files). The extracted sequences may then be aligned and the consensus sequence used to extend the contigs until gap closure by reaching another contig edge.

4.5. Detection of coverage anomalies

Contig assembly errors may be detected by remapping sequencing reads, the coverage of which is expected to be rather homogeneous along the chromosome. A coverage curve will then allow one to detect areas with 2-fold coverage (or higher), indicating either a collapsed repeat or an integrative plasmid (episome).

4.6. Polymorphism correction

The final step of correction is performed by 'polymorphism_corrector.py' on a SAM file, preferentially generated by remapping MP reads with distance constraint corresponding to the library insert size.

```
   polymorphism_corrector.py -f chromosome.fasta -s MPmapping.sam –l 101 -m
position_weigth_matrix.tab -o corrected_chromosome.fasta
```

This script generates a polymorphism correction log, a position-weight matrix (indicating at each position of the chromosome the abundance of A, C, T & G in the mapped reads) and a fasta file of the corrected chromosome sequence.

5. Scripts availability

All scripts may be downloaded from the WiseScaffolder website at http://abims.sb-roscoff.fr/wisescaffolder:

- `WiseScaffolder (wisca.py)`
- `contigs_renamer.py`
- `contig_info_builder.py`
- `sam_subsampler.py`
- `contig_edge_extractor.py`
- `polymorphism_corrector.py`

Thanks for using WiseScaffolder.

Please cite: Farrant, G.K., Hoebeke, M., Partensky, F., Andres, G., Corre, E. and Garczarek L., 2015. *WiseScaffolder: an algorithm for the semi-automatic scaffolding of Next Generation Sequencing data*, in revision for BMC Bioinformatics.