AGORA How To

Written by Alexandra LOUIS (alouis {at} biologie {dot} ens {dot} fr), Matthieu MUFFATO (muffato {at} ebi {dot} ac {dot} uk), and Hugues ROEST CROLLIUS (hrc {at} ens {dot} fr).
DYOGEN Laboratory, Institut de Biologie de l'École Normale Supérieure (IBENS)
46 rue d'Ulm, 75005 Paris

Table of contents:

History

AGORA stands for "Algorithm for Gene Order Reconstruction in Ancestors" and was developed by Matthieu Muffato during his PhD (2007-2010) in the DYOGEN Laboratory at the École normale supérieure in Paris. Since then it has been constantly used in the group, especially to generate ancestral genomes for the Genomicus online server for comparative genomics. Many algorithms used in AGORA are described in details in Matthieu's thesis, available only in French (Muffato 2010) and the core algorithm used to build and linearise the adjacency graph is described in a separate study (Berthelot et al 2015).

What AGORA does and does not do

AGORA takes as input a set of extant gene lists, ordered by chromosome (or scaffolds), a species tree linking the genomes, and phylogenetic gene trees reconciled with the species tree. It will produce linear ancestral gene orders (with transcriptional orientation) at all the nodes of the species tree. This may result in very long successive ancestral adjacencies or CARs (Contiguous Ancestral Regions) if the data allows it (e.g. closely related extant genomes with contiguous sequence assemblies) or very short ones if the data does not allow it (e.g. extant genes distributed in short scaffolds, or very rearranged extant genomes).

AGORA does not:

- Reconstruct ancestral nucleotide or protein sequences.
- Reconstruct circular chromosomes (*)

(*) The only exception is the mitochondrial genome, which has a canonical way of being represented in a linear fashion.

AGORA can be run in two modes. The first and simplest uses all possible adjacencies found in extant genomes to reconstruct ancestral adjacencies, eventually leading to contiguous ancestral regions. In principle this should work fine if the genomes are perfectly sequenced and annotated, but they rarely are. Also, gene duplications are difficult to resolve accurately in gene phylogenies, and AGORA is sensitive to errors in gene trees. A second, more complex version first identifies "robust" gene familes, on the basis of a user-defined criterion. Typically this can be a requirement that there are as many genes on a tree as there are species, thus limiting the chances that duplications have occurred. AGORA will first build a temporary ancestral genome with these genes (ignoring all other families) as a robust backbone. Then, it will use remaining gene families to fill in the space between robust genes, but without breaking a chain of robust genes.

In this HowTo, all the paths are relative to the root of the repository.

Input file formats

To reconstruct ancestral gene orders, AGORA needs 3 kinds of files (see example/data/):

- A species tree, e.g. example/data/Species.conf
- A set of extant gene trees reconciled with the species tree, e.g. example/data/GeneTreeForest.phylTree.bz2. Extant genes that are not in a tree will not be used for gene order reconstruction.
- The order of extant genes in each extant genomes, e.g. example/data/genes/genes.M1.list.bz2

Species tree

The species tree must be in *PhylTree* format. The PhylTree format is a human readable format of trees developed specifically for AGORA, based on tabulations. See the example species tree:

- example/data/Species.conf – PhylTree format
- example/data/Species.nwk – Newick format
- example/data/Species.pdf – Graphical representation

To convert a tree from NHX format to PhylTree format, use the script `newickSpeciesTree2phylTreeSpeciesTree.py`:

```
src/preprocessing/newickSpeciesTree2phylTreeSpeciesTree.py Species.nwk
> Species.conf
```

Warning: Internal labels (e.g. "Amniota" or "Anc659123") have to be unique !

The forest of gene trees

The forest of gene trees has to be in PhylTree format as well, in a single file. See an example family:

- example/data/Family1.phylTree – PhylTree format
- example/data/Family1.nhx – Newick format
- example/data/Family1.pdf – Graphical representation

To convert trees from NHX format to PhylTree format, use the script `nhxGeneTrees2phylTreeGeneTrees.py`:

```
src/preprocessing/nhxGeneTrees2phylTreeGeneTrees.py Family1.nhx >
Family1.phylTree
```

The forest file is merely the concatenation of all the families. See the example forest:

- example/data/GeneTreeForest.phylTree.bz2 – PhylTree format
- example/data/GeneTreeForest.nhx.bz2 – Newick format

The genes files

The genes files used by AGORA contain the list of genes on each extant genome. The format is tab-separated values, in 5 mandatory columns (the 6^th is optional). One file must be provided per extant genome.

The fields are:

1. Name of the chromosome (text).
2. Start position of the gene (integer).
3. End position of the gene (integer).
4. Gene orientation (1 or -1)
5. Gene identifier (text)
6. Transcript identifier (text, optional)

Warning: The gene identifiers have to be consistent with the ones used in the gene trees. The genes files must be named consistently with the names of the species in the species tree, using the format `prefix.species_name.suffix`.

For example, if the species in the species tree are: `HUMAN`, `MOUSE`, `DOG`, genes files have to be named:

- `prefix.HUMAN.suffix`, e.g. `genes.HUMAN.list`
- `prefix.MOUSE.suffix`
- `prefix.DOG.suffix`

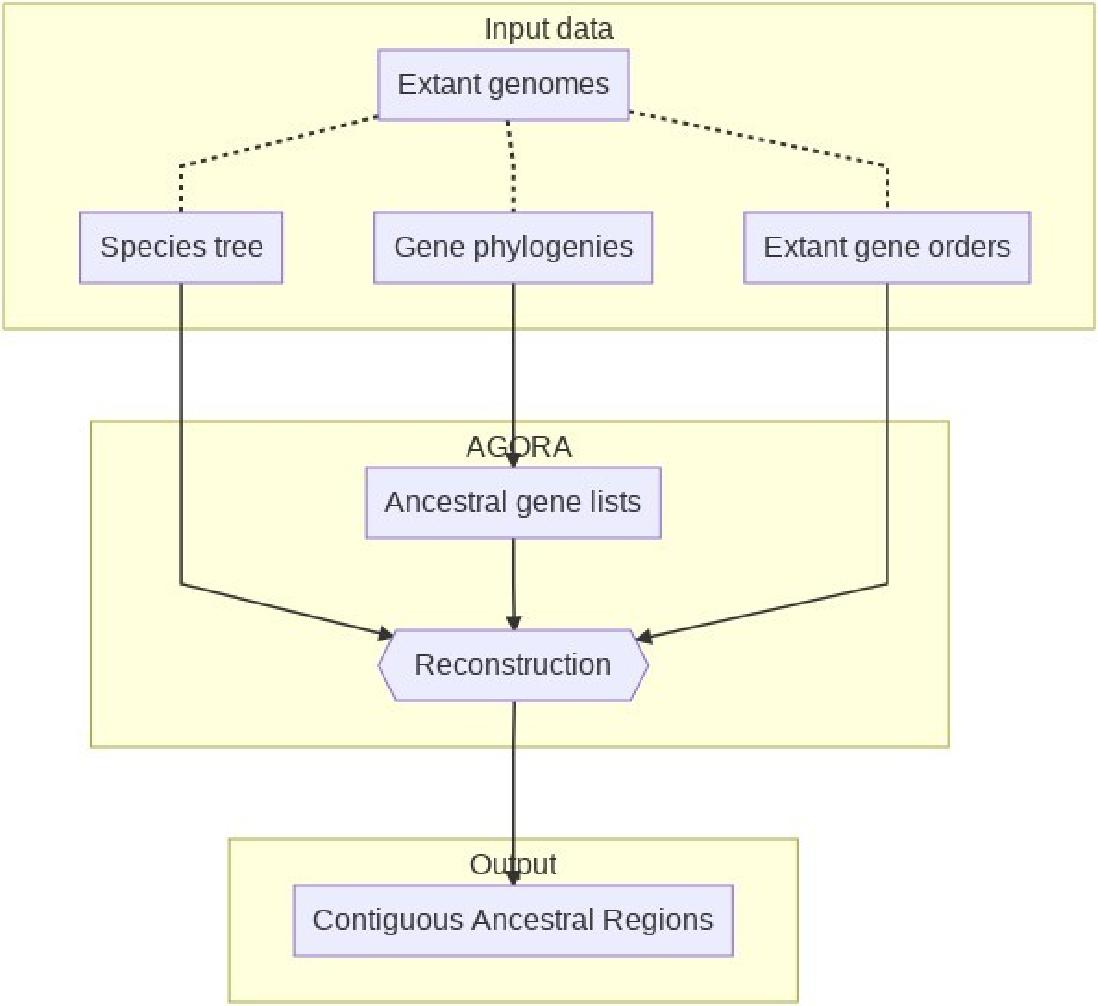
If the species are `Homo.sapiens`, `Mus.musculus` and `Canis.familiaris`, genes files have to be named:

- `prefix.Homo.sapiens.suffix`, e.g. `genes.Homo.sapiens.list`
- `prefix.Mus.musculus.suffix`
- `prefix.Canis.familiaris.suffix`

In example/data, the five species named in the species-tree are `M1`, `M2`, `M3`, `M4`,, and `M5`, and the genes files are named genes.M1.list.bz2, etc.

Running AGORA

General AGORA workflow



The reconstruction itself can be perfomed with different approaches, explained below, and the output is a set of CARs. The scripts can be run individually, or through the wrapper `agora.py` (with an appropriate `agora.ini` configuration file).

All AGORA scripts automatically creates the necessary output directories given to them as command line arguments. This excludes standard output / error shell redirections, which should still point at valid paths.

AGORA supports several compression formats for input and output files:

- gzip – `.gz` extension
- bzip2 – `.bz2` extension
- LZMA – `.lzma` and `.xz` extensions

Compression / decompression costs extra CPU time, but decreases file transfer times and storage footprint (typically 6x with bzip2). Compression is also supported on the standard output by adding `+gz`, `+bz2`, `+lzma`, or `+xz` on the command-line.

Extraction of ancestral gene content

The first step in AGORA is to identify all ancestral genes for all ancestral genomes, and print them in one file per target ancestral genome. The `ALL.extractGeneFamilies.py` script takes as input the species tree, the forest of gene trees and a template to name the output files.

```
mkdir -p example/results
src/ALL.extractGeneFamilies.py \
  example/data/Species.conf \
  example/data/GeneTreeForest.phylTree.bz2 \
  -OUT.ancGenesFiles=example/results/ancGenes/all/ancGenes.%s.list.bz2
\
  +bz2 \
  > example/results/GeneTreeForests.withAncGenes.phylTree.bz2 \
  2> example/results/log.ancGenes
```

Be careful to provide the correct path to write the *ancGenes* files (`ancGenes/all/ancGenes.%s.list.bz2`), it will be important if you use AGORA on *robust* family in a second step (see article). The `%s` will be automatically replaced by the extant and ancestral species name, as indicated in the species tree.

ancGenes files are tab-separated files, with the following two fields:

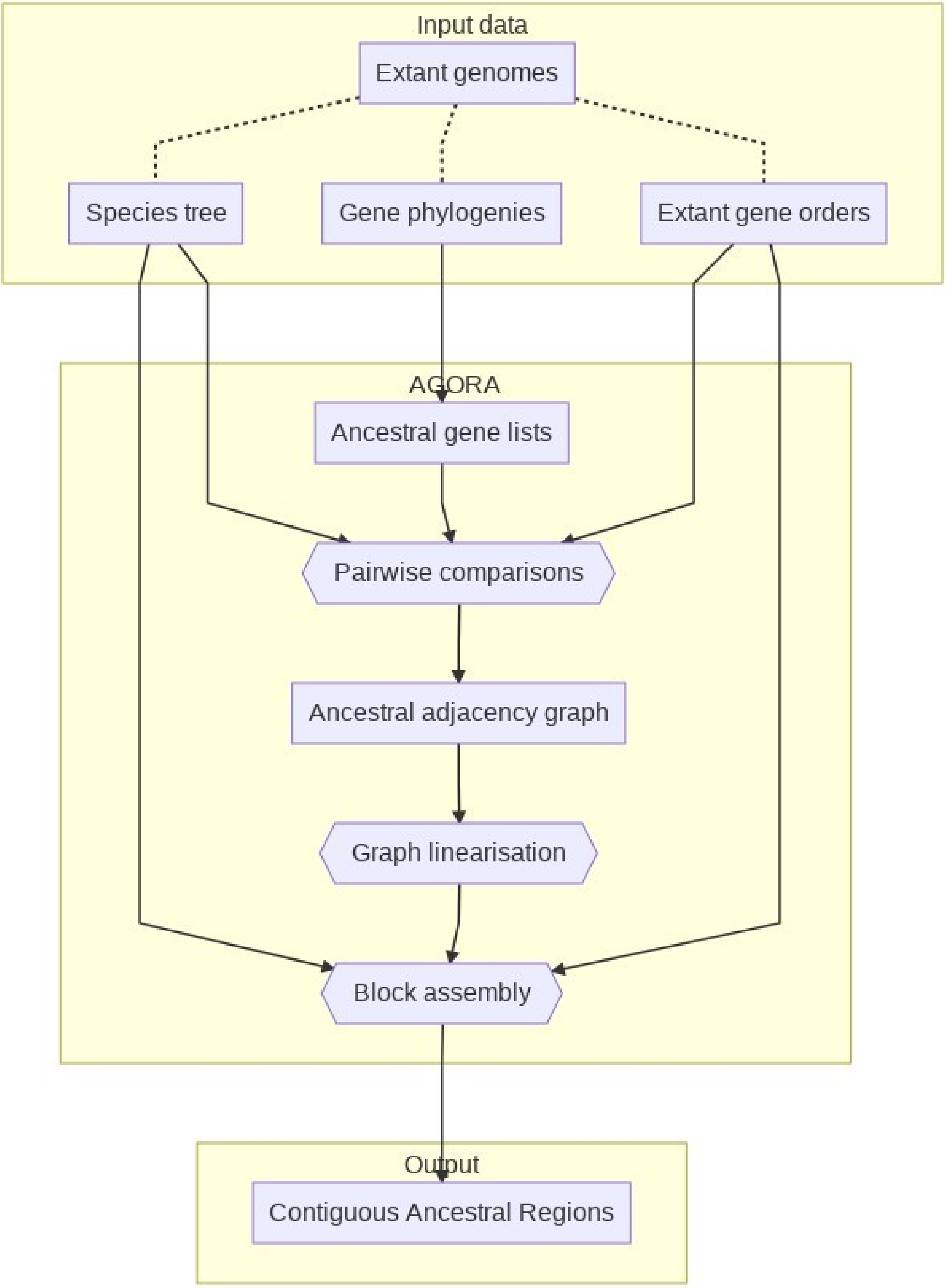
1. Ancestral gene names (generated by AGORA)
2. A space separated list of extant copies of this ancestral gene, in the genome of extant species.

On the standard output, the script produces the forest of gene trees, rewritten with the ancestral gene names at each node, in PhylTree format. Note that the rest of the scripts will use these ancGenes files rather than the forest of gene trees.

AGORA with no selection of robust families

This is the simplest way of running AGORA. It will compare all extant genomes pairwise to extract conserved adjacencies, generate the ancestral adjacency graphs and linearise them to produce CARs.

AGORA workflow with no selection of robust families



In this mode, the reconstruction is composed of three steps akin to a genome assembly:

- The pairwise comparisons provide “reads” of the ancestral genomes

- The reads are assembled into contigs (“Graph linearisation” step)
- The contigs are assembled into scaffolds (“Block assembly” step)

The ancestral gene lists have been generated in the previous section. The scripts can be run step by step or through the wrapper `agora.py`. In the following command lines, `A0` represents the name of the target ancestor for the reconstructions (which must exist in the species tree). This ancestor and all its descendants will be reconstructed.

Step by step

Pairwise comparisons

This step will compare extant genomes in all possible pairwise combinations to identify conserved adjacencies.

```
mkdir -p example/results/pairwise/pairs-all/
src/buildSynteny.pairwise-conservedPairs.py \
  example/data/Species.conf \
  A0 \
  -OUT.pairwise=example/results/pairwise/pairs-all/%s.list.bz2 \
  -genesFiles=example/data/genes/genes.%s.list.bz2 \
  -ancGenesFiles=example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  2> example/results/pairwise/pairs-all/log
```

Graph linearisation

This step will integrate all the pairwise comparisons identified above for each ancestor and combine them into adjacency graphs, from which a first set of CARs are derived.

```
mkdir -p example/results/ancGenomes/denovo-all/
src/buildSynteny.integr-denovo.py \
  example/data/Species.conf \
  A0 \
  example/results/pairwise/pairs-all/%s.list.bz2 \
  +searchLoops \
  -OUT.ancDiags=example/results/ancGenomes/denovo-all/diags.
%s.list.bz2 \
  -LOG.ancGraph=example/results/ancGenomes/denovo-all/graph.%s.log.bz2
\
  -ancGenesFiles=example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  2> example/results/ancGenomes/denovo-all/log
```

Block assembly

In this step, we basically reiterate the same process (pairwise comparisons and integration into an adjacency graph) but on the previous CARs, which allows finding higher-level adjacencies. The result is a set of CARs made of CARs, that are much longer than in the previous steps.

> **Warning**: The name of the ancestor has to be repeated !

```
mkdir -p example/results/ancGenomes/denovo-all.groups/
src/buildSynteny.integr-groups.py \
  example/data/Species.conf \
  A0 \
  A0 \
  -IN.ancDiags=example/results/ancGenomes/denovo-all/diags.%s.list.bz2
\
  -OUT.ancDiags=example/results/ancGenomes/denovo-all.groups/diags.
%s.list.bz2 \
  -LOG.ancGraph=example/results/ancGenomes/denovo-all.groups/graph.
%s.log.bz2 \
  -genesFiles=example/data/genes/genes.%s.list.bz2 \
  -ancGenesFiles=example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  2> example/results/ancGenomes/denovo-all.groups/log
```

All in one: agora.py

`agora.py` is a script that encapsulates the 3 previous scripts and runs them automatically. The parameters, paths, etc, are defined in a configuration file agora.ini. The example one contains documentation of all the options.

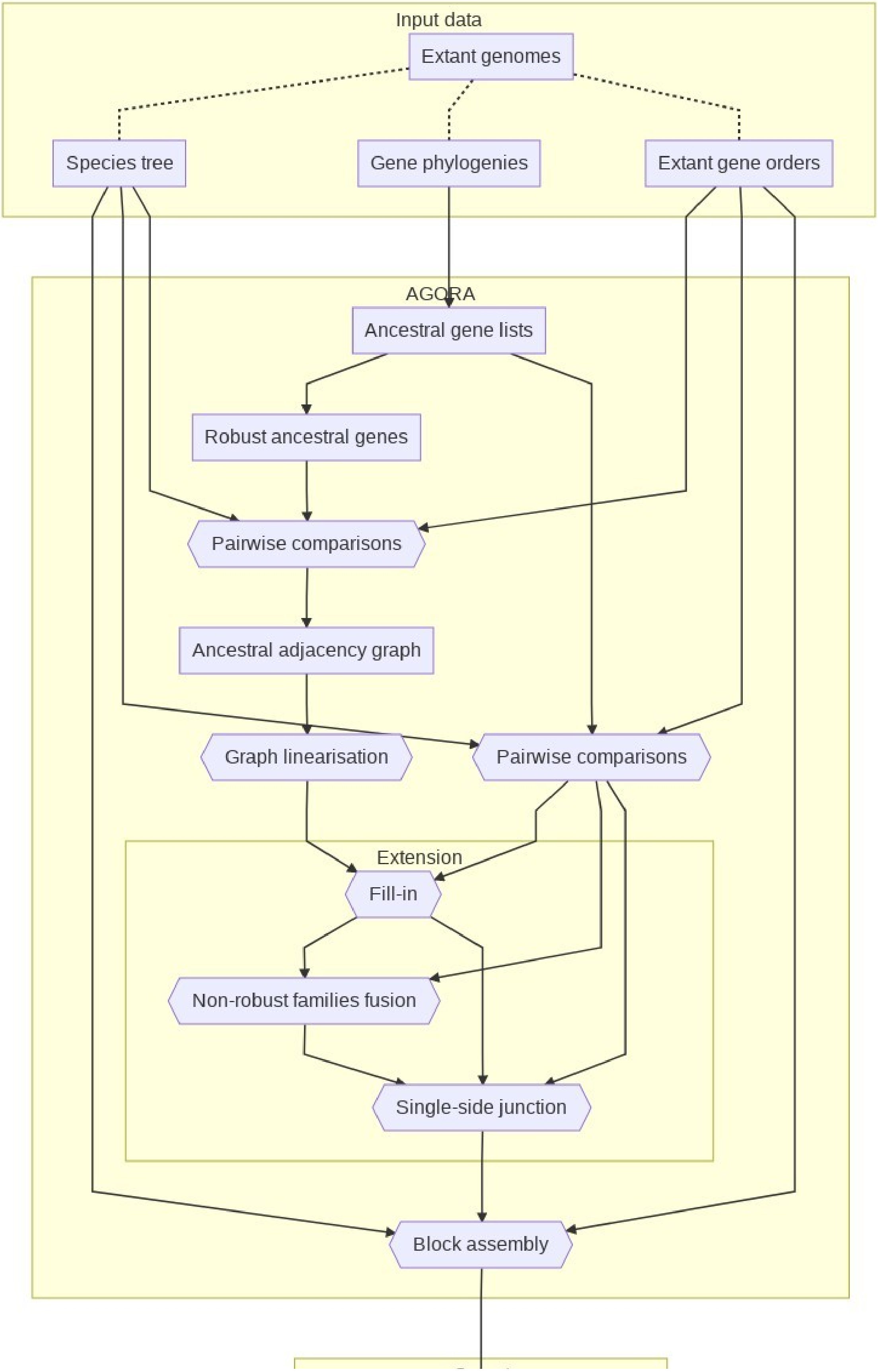
```
src/agora.py conf/agora.ini -workingDir=example/results
```

All data will be created in the `workingDir` directory.

AGORA with selection of robust families

This approach builds ancestral adjacencies considering a subset of the genes. The idea here is to build “robust” ancestral adjacency scaffolds, and to insert within these adjacencies the remaining ancestral genes.

AGORA workflow with selection of robust families


Input data
Extant genomes
Species tree
Gene phylogenies
Extant gene orders
AGORA
Ancestral gene lists
Robust ancestral genes
Pairwise comparisons
Ancestral adjacency graph
Graph linearisation
Pairwise comparisons
Extension
Fill-in
Non-robust families fusion
Single-side junction
Block assembly

From the complete list of ancestral genes, AGORA will identify a subset of robust genes according to a user-defined criterion. It will compare all extant genomes pairwise (considering all genes and robust genes separately), build the adjacency graphs on the comparisons of robust genes and linearise them to obtain robust contigs. It will then *fill in* the robust contigs with non-robust genes, build contigs of non-robust genes (*non-robust families fusion*) and insert these in the filled-in robust contigs (*single side junction*). Finally it will assemble the resulting contigs (block assembly) into Contiguous Ancestral Regions (CARs).

Step by step

Selection of robust genes

Let's consider the genes families that have exactly the same number of extant genes as extant species (i.e. *minSize* and *maxSize* parameters equal to 1). Having undergone fewer losses and duplications, the synteny signal of those families is less ambiguous and their adjacencies easier to compare and more conserved.

```
src/ALL.filterGeneFamilies-size.py \
  example/data/Species.conf \
  A0 \
  example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  example/results/ancGenes/size-%s-%s/ancGenes.%s.list.bz2 \
  1.0 \
  1.0 \
  2> example/results/ancGenes/size.log
```

Pairwise comparison

This step is run once for all ancestral genes, and once for the set of robust families.

For all ancestral genes:

```
src/buildSynteny.pairwise-conservedPairs.py \
  example/data/Species.conf \
  A0 \
  -OUT.pairwise=example/results/pairwise/pairs-all/%s.list.bz2 \
  -genesFiles=example/data/genes/genes.%s.list.bz2 \
  -ancGenesFiles=example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  2> example/results/pairwise/pairs-all/log
```

For the robust gene families:

```
src/buildSynteny.pairwise-conservedPairs.py \
  example/data/Species.conf \
  A0 \
  -OUT.pairwise=example/results/pairwise/pairs-size-1.0-
```

```
1.0/%s.list.bz2 \
  -genesFiles=example/data/genes/genes.%s.list.bz2 \
  -ancGenesFiles=example/results/ancGenes/size-1.0-1.0/ancGenes.
%s.list.bz2 \
  2> example/results/pairwise/pairs-size-1.0-1.0/log
```

Graph linearisation

This step will integrate all the pairwise comparisons of robust genes identified above for each ancestor and combine them into adjacency graphs, from which a first set of CARs are derived.

```
src/buildSynteny.integr-denovo.py \
  example/data/Species.conf \
  A0 \
  example/results/pairwise/pairs-size-1.0-1.0/%s.list.bz2 \
  -OUT.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0/diags.
%s.list.bz2 \
  -LOG.ancGraph=example/results/ancGenomes/denovo-size-1.0-1.0/graph.
%s.log.bz2 \
  -ancGenesFiles=example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  2> example/results/ancGenomes/denovo-size-1.0-1.0/log
```

Fill-in

This step will insert non-robust genes in each interval of the ancestral contigs, following paths in the complete ancestral adjacency graph.

```
src/buildSynteny.integr-refine.py \
  example/data/Species.conf \
  A0 \
  example/results/pairwise/pairs-all/%s.list.bz2 \
  -IN.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0/diags.
%s.list.bz2 \
  -OUT.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0.refine-
all/diags.%s.list.bz2 \
  -LOG.ancGraph=example/results/ancGenomes/denovo-size-1.0-1.0.refine-
all/graph.%s.log.bz2 \
  2> example/results/ancGenomes/denovo-size-1.0-1.0.refine-all/log
```

Non-robust families fusion

This step will take all the remaining singletons, which are mostly non-robust genes, and try to assemble them into contigs.

```
src/buildSynteny.integr-extend.py \
  example/data/Species.conf \
  A0 \
  example/results/pairwise/pairs-all/%s.list.bz2 \
  -IN.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0.refine-all/diags.%s.list.bz2 \
  -OUT.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-all/diags.%s.list.bz2 \
  -LOG.ancGraph=example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-all/graph.%s.log.bz2 \
  2> example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-all/log
```

Single-side junction

This step will insert the contigs of non-robust families created above and insert them in the CARs.

```
src/buildSynteny.integr-halfinsert.py \
  example/data/Species.conf \
  A0 \
  example/results/pairwise/pairs-all/%s.list.bz2 \
  -IN.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-all/diags.%s.list.bz2 \
  -REF.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0.refine-all/diags.%s.list.bz2 \
  -OUT.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-all.halfinsert-all/diags.%s.list.bz2 \
  -LOG.ancGraph=example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-all.halfinsert-all/graph.%s.log.bz2 \
  2> example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-all.halfinsert-all/log
```

Block assembly

Like in non-robust mode, this step will do pairwise comparisons and a graph linearisation of the CARs themselves, which allows finding higher-level adjacencies.

> **Warning**: The name of the ancestor has to be repeated !

```
src/buildSynteny.integr-groups.py \
  example/data/Species.conf \
  A0 \
  A0 \
  -IN.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-all.halfinsert-all/diags.%s.list.bz2 \
  -OUT.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-all.halfinsert-all.groups/diags.%s.list.bz2 \
  -LOG.ancGraph=example/results/ancGenomes/denovo-size-1.0-1.0.refine-
all.extend-all.halfinsert-all.groups/graph.%s.log.bz2 \
  -genesFiles=example/data/genes/genes.%s.list.bz2 \
  -ancGenesFiles=example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  2> example/results/ancGenomes/denovo-size-1.0-1.0.refine-all.extend-
all.halfinsert-all.groups/log
```

Copy of the results in the final repository

```
src/buildSynteny.integr-copy.py \
  example/data/Species.conf \
  A0 \
  -IN.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0.refine-
all.extend-all.halfinsert-all.groups/diags.%s.list.bz2 \
  -OUT.ancDiags=example/results/ancGenomes/final/diags.%s.list.bz2 \
  2> example/results/ancGenomes/final/log
```

All in one: agora.py

The whole process can be automated with `agora.py`and (agora-robust.ini.

```
src/agora.py conf/agora-robust.ini -workingDir=example/results
```

AGORA with multiple selection of robust families

The process can be further tuned to use multiple sets of robust genes for specific ancestors. Along the *1.0-1.0* robust families used above, we can define other, more relaxed, sets, like *0.9-1.1*, which tolerates a 10% deviation between the number of extant genes and extant species, and so forth.

The most efficient way of extracting multiple sets is to do all at once, for instance:

```
src/ALL.filterGeneFamilies-size.py \
  example/data/Species.conf \
  A0 \
  example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  example/results/ancGenes/size-%s-%s/ancGenes.%s.list.bz2 \
  1.0,0.9,0.77 \
  1.0,1.1,1.33 \
  2> example/results/ancGenes/size.log
```

These different sets can be used to generate the first set of ancestral adjacencies for different ancestors, e.g.:

```
src/buildSynteny.integr-denovo.py \
  example/data/Species.conf \
  =A3 \
  example/results/pairwise/pairs-size-1.0-1.0/%s.list.bz2 \
  -OUT.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0/diags.
%s.list.bz2 \
  -LOG.ancGraph=example/results/ancGenomes/denovo-size-1.0-1.0/graph.
%s.log.bz2 \
  -ancGenesFiles=example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  2> example/results/ancGenomes/denovo-size-1.0-1.0/log
src/buildSynteny.integr-denovo.py \
  example/data/Species.conf \
  =A1,=A2 \
  example/results/pairwise/pairs-size-0.9-1.1/%s.list.bz2 \
  -OUT.ancDiags=example/results/ancGenomes/denovo-size-0.9-1.1/diags.
%s.list.bz2 \
  -LOG.ancGraph=example/results/ancGenomes/denovo-size-0.9-1.1/graph.
%s.log.bz2 \
  -ancGenesFiles=example/results/ancGenes/all/ancGenes.%s.list.bz2 \
  2> example/results/ancGenomes/denovo-size-0.9-1.1/log
```

These sets can be combined by running the copy script multiple times, like this:

```
src/buildSynteny.integr-copy.py \
  example/data/Species.conf \
  =A3 \
  -IN.ancDiags=example/results/ancGenomes/denovo-size-1.0-1.0/diags.
%s.list.bz2 \
  -OUT.ancDiags=example/results/ancGenomes/denovo-size-custom/diags.
%s.list.bz2 \
  2> example/results/ancGenomes/denovo-size-custom/log
```

However, the easiest is to use `agora.py` with a suitable configuration file.

```
src/agora.py conf/agora-multirobust.ini -workingDir=example/results
```

Output format and post-processing scripts

- The diags files

```
example/results/ancGenomes/final/diags.*.list.bz2
```

This directory contains a file for each ancestral reconstructed genome (e.g. `diags.A0.list.bz2`). There are five tab-separated fields, and values in each field are further separated by single spaces. The term ‘diag’ historically refers to the diagonal lines that appear in 2 dimensional matrices comparing 2 genomes and reflecting successive conserved adjacencies.

The fields are:

1. Name of the ancestral species.
2. Number of genes in the ancestral block.

3. List of gene IDs. Each ID corresponds to the line number in the corresponding ancGenes.Species.list (starting from 0).
4. Gene transcriptional orientation (strand) within the block.
5. A relative confidence index for each inter-block linkage. The values in parenthesis are the size of the initial blocks.

The values without parenthesis represent the number of time the two adjacent blocks are adjacent in extant species.

The sum of the lengths of the initial blocks (numbers in parenthesis) is thus equal to the size of the whole block (field number 2)

ex: A block of 8 genes in A0 made of 2 sub-blocks linked by an adjacency of score 6.

```
A0  8   4559 4179 10099 15638 1304 10998 5675 13765 -1 -1 -1 1 1 -1 -1
1    (5) 6 (3)
```

- Post processing scripts: transforming diags files to genome files

```
src/postprocessing/misc.convertContigsToGenome.py \
  example/results/ancGenomes/final/diags.A0.list.bz2 \
  tmp/ancGenes/all/ancGenes.A0.list.bz2 \
  > tmp/ancGenomes/ancGenome.A0.list
```

As an example:

```
tmp/ancGenomes/ancGenomes.\*.list
```

This directory contains a file for each ancestral reconstructed genome (e.g. `ancGenome.A0.list`).

There are five tab-separated fields, and values in each field are further separated by single spaces.

The fields are:

1. Name of the ancestral block.
2. Relative start position of the ancestral gene.
3. Relative end position of the ancestral gene.
4. Ancestral gene orientation within the block.
5. Ancestral gene names. The first name corresponds to the ancestral gene, subsequent ones are the list of extant copies of this ancestral gene, in the genome of extant species.