Hierarchical Alignment (HAL) Format API (v1.4)

HAL is a structure to efficiently store and index multiple genome alignments and ancestral reconstructions. HAL is a graph-based representation which provides several advantages over matrix/block-based formats such as MAF, such as improved scalability and the ability to perform queries with respect to an arbitrary reference or subtree.

This package includes the HAL API and several analysis and conversion tools which are described below. HAL files are presently stored in HDF5 format, but we note that the tools and most of the API are format-independent, so other databases could be implemented in the future.

Citing

Glenn Hickey, Benedict Paten, Dent Earl, Daniel Zerbino, and David Haussler. HAL: A Hierarchical Format for Storing and Analyzing Multiple Genome Alignments. Bioinformatics. 2013. Advance Online Access

Installation

Requirements

- gcc 4.2 or newer
- git

Downloading HAL

From the parent directory of where you want HAL installed:

```
git clone git://github.com/glennhickey/hal.git
```

Progressive Cactus Package

Note that HAL can also be downloaded and installed (automatically along with all its dependencies) as part of the Progressive Cactus installation package

Installing Dependencies

HDF5 1.8 with C++ API enabled

- Using MacPorts:

sudo port install hdf5-18 +cxx

- From Source:

```
wget http://www.hdfgroup.org/ftp/HDF5/current/src/hdf5-1.8.9.tar.gz
tar xzf hdf5-1.8.9.tar.gz
cd hdf5-1.8.9
./configure --enable-cxx
make && make install
```

- Local install from source into DIR (do not need root password)

```
mkdir DIR/hdf5
wget http://www.hdfgroup.org/ftp/HDF5/current/src/hdf5-1.8.9.tar.gz
tar xzf hdf5-1.8.9.tar.gz
cd hdf5-1.8.9
./configure --enable-cxx --prefix DIR/hdf5
make && make install
```

Before building HAL, update the following environment variables:

```
export PATH=DIR/hdf5/bin:${PATH}
export h5prefix=-prefix=DIR/hdf5
```

sonLib

From the same parent directory where you downloaded HAL:

```
git clone git://github.com/benedictpaten/sonLib.git
pushd sonLib && make && popd
```

If sonLib and HAL are not sister directories, update hal/include and change

```
sonLibRootPath=${rootPath}/../sonLib
```

to reflect the directory where you installed sonLib

Optional support of reading HAL files over HTTP via UCSC's URL Data Cache (UDC)

Define ENABLE_UDC before making, and specify the path of the Kent source tree using KENTSRC. When built with this enabled, all HAL files opened read-only will be accessed using UDC which supports both local files and URLs.

```
export  ENABLE_UDC=1
export  KENTSRC=<path to top level of Kent source tree>
```

Those without the UCSC genome browser already installed locally will probably find it simpler to first mount URLs with HTTPFS before opening with HAL.

Optional support of PhyloP evolutionary constraint annotation

PhyloP is part of the Phast Package, and can be used to test for genomic positions that are under selective

pressure. We are working on prototype support for running PhyloP on HAL files. In order to enable this support, Phast must be installed. We recommend downloading the latest source using Subversion. Note that PhyloP support and UDC support cannot be currently enabled at the same time.

From the same parent directory where you downloaded HAL:

- First install CLAPACK (Linux only)

```
wget http://www.netlib.org/clapack/clapack.tgz
tar -xvzf clapack.tgz
mv CLAPACK-3.2.1 clapack
cd clapack
cp make.inc.example make.inc && make f2clib && make blaslib && make lib
export CLAPACKPATH=`pwd`
cd ..
```

- Install Phast (Mac or Linux)

```
svn co http://compgen.bscb.cornell.edu/svnrepo/phast/trunk phast/
cd phast
export PHAST=`pwd`
cd src && make
cd ../..
```

- Before building HAL

```
export ENABLE_PHYLOP=1
```

Special thanks to Melissa Jane Hubiz and Adam Siepel from Cornell University for their work on extending their tools to work with HAL.

Building HAL

From the hal/ directory:

```
make
```

Before using HAL, add it to your path:

```
export PATH=<path to hal>/bin:${PATH}
```

The parent directory of hal/ should be in your PYTHONPATH in order to use any of the Python functionality. This includes running `make test`

```
export PYTHONPATH=<parent of hal>:${PYTHONPATH}
```

HAL Tools

General Options

Detailed command line options can be obtained by running each tool with the `--help` option.

All HAL tools compiled with HDF5 support expose some caching parameters. Tools that create HAL files also include chunking and compression parameters. In most cases, the default values of these options will suffice.

`--cacheBytes <value>`: The maximum size of each array cache. 3 such caches can be allocated per genome in the alignment.

`--cacheRDC <value>`: The number of slots in each cache. This number should be set to a prime number that is roughly 50 x [cacheBytes / chunk].

`--cacheMDC <value>`: Size of the metadata cache. There is presently no reason to touch this.

`--chunk <value>`: The chunk size for the hdf5 arrays. Unreasonable chunk sizes can adversely affect cache performance. Larger chunks can lead to better compression. [default = 1000]

`--deflate <value>`: Compression level. Higher levels tend to not significantly decrease file sizes but do increase run time. [0:none - 9:max] [default = 2]

`--inMemory`: Load all data in memory (and disable hdf5 cache). [default = False]

Importing from other formats

MAF Import

The (MAF)[http://genome.ucsc.edu/FAQ/FAQformat.html#format5] is a text format used at UCSC to store genome alignments. MAFs are typically stored with respect to a reference genome. MAFs can be imported into HAL as subtrees using the `maf2hal` command.

To import primates.maf as a star tree where the first alignment row specifies the root, and all others the leaves:

```
maf2hal primates.maf primates.hal
```

To import primates.maf using "chimp" has the root

```
maf2hal primates.maf primates.hal --refGenome chimp
```

The more the underlying tree looks like a star tree, the less efficient HAL is as all genomes will be fragmented with respect to each other. If ancestral (or multiple reference) sequences are available, or if it is acceptable to use a non-reference species as a reference proxy, then trees of arbitrary typologies can be constructed using the `--append` option.

```
maf2hal mammals.maf mammals.hal --refGenome mouse --targetGenomes human,rat,chimp,dog
 maf2hal mammals.maf mammals.hal --append --refGenome human --targetGenomes chimp,goril
 maf2hal mammals.maf mammals.hal --append --refGenome dog --targetGenomes cow,horse
```

This will create a tree that looks like

```
((chimp, gorilla,orang)human, rat,(cow,horse)dog)mouse;
```

Progressive Cactus Import

HAL is most beneficial when consensus reference or ancestral sequences are available at the internal nodes of the tree. This is the type of information generated by progressive alignment pipelines. Progressive Cactus (manuscript in preparation) is our implementation of such a pipeline. A beta version is presently available on GitHub. A tool to convert from Cactus graphs to HAL graphs, cactus2hal, can be downloaded as well.

```
cactus2hal.py mammals_cactusProject.xml mammals.hal
```

Exporting to other formats

MAF Export

MAF files can be generated from HAL alignments or sub-alignments. The reference genome and alignment scope (subsequence of the reference and/or phylogenetic distance) are chosen through command-line options.

Export the HAL alignment as a MAF referenced at the root

```
hal2maf mammals.hal mammals.maf
```

Export a MAF with referenced at sequence *chr6* in the human genome

```
hal2maf mammals.hal mammals.maf --refGenome human --refSequence chr6
```

Export a MAF consisting of the alignment of human with respect to *chr2* in chimp

```
hal2maf mammals.hal mammals.maf --refGenome chimp --refSequence chr2 --targetGenome
```

Export a MAF consisting of the alignment of all apes referenced on gorilla

```
hal2maf mammals.hal mammals.maf --rootGenome ape_ancestor --refGenome gorilla
```

By default, no gaps are written to the reference sequence. The `--maxRefGap` can be specified to allow gaps up to a certain size in the reference. This is achieved by recursively following indels in the graph that could correspond to reference gaps.

Mafs can be generated in parallel using the hal2mafMP.py wrapper

```
hal2mafMP.py mammals.hal mammals.maf --numProc 10
```

FASTA Export

DNA sequences (without any alignment information) can be extracted from HAL files in FASTA format using `hal2fasta`.

Displaying in the UCSC Genome Browser

HAL alignments can be displayed as Assembly Hubs in the Genome Browser. To create an assembly

hub, run

```
hal2assemblyHub.py mammals.hal outputDirectory
```

Larger alignments require the use of the `--lod` option to generate precomputed levels of detail.

Note that this process is presently dependent on having UCSC's faToTwoBit installed. The `outputDirectory` must be accessible as a URL in order to load the hub.

Summary Information

halValidate

It is a good idea to check if a hal file is valid after creating it.

```
halValidate mammals.hal
```

halStats

Some global information from a HAL file can be quickly obtained using `halStats`. It will return the number of genomes, their phylogenetic tree, and the size of each array in each genome.

```
  halStats mammals.hal
```

The `--tree`, `--sequences`, and `--genomes` options can be used to print out only specific information to simplify iterating over the alignment in shell or Python scripts.

halSummarizeMtuations

A count of each type of mutation (Insertions, Deletions, Inversions, Duplications, Transpositions, Gap Insertions, Gap Deletions) in each branch of the alignment can be printed out in a table.

```
 halSummarizeMutations mammals.hal
```

Subtrees can be specified using the `--targetGenomes` or `--rootGenome` option. The `--maxGap` option is used to distinguish from small, 'gap' indels and larger indels. This distinction is somewhat arbitrary (but conventional). HAL allows gap indels to be nested within larger rearrangements: ex. an inversion with a gap deletion inside would be counted as a single inversion, but an inversion containing a non-gap event would be identified as multiple independent inversions.

```
 halSummarizeMutations mammals.maf --maxNFraction 0
```

will prevent rearrangements with missing data as being identified as such. More generally, if an insertion of length 50 contains c N-characters, it will be labeled as missing data (rather than an insertion) if c/N > `maxNFraction`.

Levels of Detail

Some applications such as genome browsers my need to quickly access high-level information about the

alignment without scanning every segment. We provide tools to resample a HAL graph to compute a coarser-grained levels of detail to speed up subsequent analysis at different scales. To generate an output hal file based on a sampling of every `100` bases:

```
halLodExtract mammals.hal mammals_100.hal 100
```

To generate a series of levels of details, such that each level of detail is 5x coarser than the previous, and that there are at most (approx.) 100 segments at the lowest level, use the following script:

```
halLodInterpolate.py mammals.hal lod_summary.txt --scale 5 --maxBlock 100
```

Note that both tools have a `--keepSequences` option to specify whether or not the DNA sequences are stored in the output files.

Analysis

Liftover

Annotations in BED, ie tab-delimited files whose first three columns are

```
SequenceName     StartPosition    LastPosition+1
```

can be lifted over between genomes using `halLiftover`. halLiftover does a base-by-base mapping between any two sequences in the alignment (following paralogy relations as well).

```
halLiftover mammals.hal human human_annotation.bed dog dog_annotation.bed
```

will map all annotations in human_annotation.bed, which must refer to sequences in the human genome, to their corresponding locations in dog (if they exist), outputting the resulting annotations in dog_annotation.bed

halLiftover attempts to autodetect the BED version of the input. This can be overried with the `--inVedVersion` option. Columns that are not described in the official BED specs can be optionally mapped as-is using the `--keepExtra` option.

Alignability

The number of distinct genomes different bases of a set of target genomes align to can be computed using the `halAlignability` tool. The output is in `.wig` format.

Mutation Annotation

SNPs

To compute the point mutations (SNPs) between a given pair of genomes in the HAL graph, `halSnps` can be used:

```
halSnps mammals.hal human duck --bed human_duck_snps.bed
```

will produce a BED files listing the SNPs in human coordinates between human and duck. A count of the number of snps and the total aligned columns are printed to stdout.

General mutations along branches

Annotation files, as described above, can be generated from the alignment to provide the locations of substitutions and rearrangements. Annotations are done on a branch-by-branch basis, but can be mapped back to arbitrary references using `halLiftover` if so desired. The produced annotation files have the format

```
SequenceName     StartPosition    LastPosition+1  MutationID  ParentGenome ChildGenome
```

The ID's refer to the types of mutations described above, and are explained in the header of each generated file. To generate tables of rearrangement mutations between human and its most recent ancestor in the alignment, run

```
halBranchMutations mammals.hal human --refFile ins.bed --parFile del.bed
```

Two bed files must be specified because the coordinates of inserted (and by convention inverted and transposed) segments are with respect to bases in the human genome (reference), where as deleted bases are in ancestral coordinates (parent).

Point mutations can optionally be written using the `--snpFile <file>` option. The '--maxGap' and '--maxNFraction' options can specify the gap indel threshold and missing data threshold, respectively, as described above in the *halSummarizeMtuations* section.

Constrained Element Prediction

(Under development)

PhyloP is part of the Phast Package, and can be used to test for genomic positions that are under selective pressure. We are working on prototype support for running PhyloP on HAL files.

- Train a neutral model

  See `halPhyloPTrain.py`

- Detect constrained elements

  See `halPhyloPMP.py`

Special thanks to Melissa Jane Hubiz and Adam Siepel from Cornell University for their work on extending their tools to work with HAL.

Example of HAL Genome Representation

The following is obtained by running h5ls -v -r (included with hdf5) on an ancestral genome, in this case a small simulated human-chimp ancestor named sHuman-sChimp. The genome itself is stored as a group. It contains four important 1-dimensional arrays:

- BOTTOM_ARRAY: The bottom segments of the genome (containing alignment mapping to the descendants). The size of each entry is dependent on the number of descendants.
- DNA_ARRAY: The DNA bases, stored as two bases / byte
- SEQUENCE_ARRAY: The names and lengths of subsequences (ie chromosomes or scaffolds in the genome)
- TOP_ARRAY: The top segments in the genome (containing alignment mapping to the parent). Paralogous top segments are presently stored in a circular linked list.

More information can be found in the manuscript:

Glenn Hickey, Benedict Paten, Dent Earl, Daniel Zerbino, and David Haussler. HAL: A Hierarchical Format for Storing and Analyzing Multiple Genome Alignments. Bioinformatics. 2013. Advance Online Access

and API manual.

```
/sHuman-sChimp           Group
   Location:  1:204059420
   Links:     1
/sHuman-sChimp/BOTTOM_ARRAY Dataset {1595768/1595768}
   Location:  1:365340961
   Links:     1
   Chunks:    {1000} 42000 bytes
   Storage:   67022256 logical bytes, 12879397 allocated bytes, 520.38% utilization
   Filter-0:  deflate-1 OPT {2}
   Filter-1:  deflate-1 OPT {2}
   Type:      struct {
           "genomeIdx"        +0    native long
           "length"           +8    native unsigned long
           "topIdx"           +16   native long
           "childIdx0"        +24   native long
           "reverseFlag0"     +32   native signed char
           "childIdx1"        +33   native long
           "reverseFlag1"     +41   native signed char
   } 42 bytes
/sHuman-sChimp/DNA_ARRAY Dataset {92368315/92368315}
   Location:  1:253117606
   Links:     1
   Chunks:    {1000} 1000 bytes
   Storage:   92368315 logical bytes, 55478173 allocated bytes, 166.49% utilization
   Filter-0:  deflate-1 OPT {2}
   Filter-1:  deflate-1 OPT {2}
   Type:      native 8-bit field
/sHuman-sChimp/Meta      Group
   Location:  1:204060452
   Links:     1
/sHuman-sChimp/Rup       Group
   Attribute: Rup scalar
           Type:      variable-length null-terminated ASCII string
           Data:  "0"
   Location:  1:204061484
   Links:     1
/sHuman-sChimp/SEQUENCE_ARRAY Dataset {1/1}
   Location:  1:253117878
   Links:     1
   Storage:   96 logical bytes, 96 allocated bytes, 100.00% utilization
   Type:      struct {
```

```
           "start"            +0    native unsigned long
           "length"           +8    native unsigned long
            "numSequences"     +16   native unsigned long
           "numBottomSegments" +24   native unsigned long
           "topSegmentArrayIndexOffset" +32   native unsigned long
           "bottomSegmentArrayIndexOffset" +40   native unsigned long
           "name"             +48   384-bit little-endian integer
           (8 bits of precision beginning at bit 0)
           (376 zero bits at bit 8)
   } 96 bytes
/sHuman-sChimp/TOP_ARRAY Dataset {2273166/2273166}
   Location:  1:253118446
   Links:     1
   Chunks:    {1000} 33000 bytes
   Storage:   75014478 logical bytes, 13067609 allocated bytes, 574.05% utilization
   Filter-0:  deflate-1 OPT {2}
   Filter-1:  deflate-1 OPT {2}
   Type:      struct {
           "genomeIdx"        +0    native long
           "bottomIdx"        +8    native long
           "paralogyIdx"      +16   native long
           "parentIdx"        +24   native long
           "reverseFlag"      +32   native signed char
   } 33 bytes
```