CVTree 3.0
User's Manual

(Standalone Version)

Guanghong Zuo
ghzuo@fudan.edu.cn

November 19, 2020

Contents

1 Introduction

CVTree stands for **Composition Vector Tree** which is the implementation of a cluster alignment-free algorithms to generate dissimilarity matrices from a comparatively large collection of DNA or Amino Acid sequences, preferably genome data, for phylogenetic studies. It is first developed to infer the evolutionary relatedness of Bacteria and Archaea (Qi *et al.*, 2004*b*), and then successfully applied to viruses, chloroplasts, and fungi.

There are two available ways to use the algorithms:

1. **CVTree Web Server** which has been published twice in the Web Server Issues of *Nucleic Acids Research*, (Qi *et al.*, 2004*a*) and (Xu and Hao, 2009). The latest released CVTree Web Server, CVTree3 Web Server (Zuo and Hao, 2015), have two identical but independent installations:

   - CVTree3 Web Server on Fudan University, Shanghai:
     `http://tlife.fudan.edu.cn/cvtree/`
   - CVTree3 Web Server on Beijing Institute of Genomics, Beijing:
     `http://bigd.big.ac.cn/cvtree`

   There are 3000+ inbuilt whole genomes in the CVTree3 web servers which covered 1700+ prokaryotic species. A much more powerful web server, CVTree4 Web Server, which included 100000+ genomes and covered 8000+ prokaryotic species was in testing on Aliyun.

   - Testing Web Server on Aliyun:
     `http://cvtree.online`

2. **CVTree Standalone Version** which is provided to those who are interested in the intermediate results, e.g., the collection of all CVs, or deal with extremely huge datasets of their own, as well as bioinformatics developers. We provide also a few options and tools that were not available in the Web Server versions.

This manual is for the CVTree Standalone Version.

2 The Installation and Testing

CVTree is distributed via source code. After downloading the source codes. There are two ways to compile the source codes of CVTree: the classical way, compiled and performed in normal like other Unix-like programs; the docker way.

2.1 Normal Unix-like Mode

All the programs were implemented in C++, some compile tools and library is required, and other libraries optional for special purposes.

2.1.1 Preparation

- cmake $\geq$ 3.0
- g++ $\geq$ 4.8 or other compiler supporting C++11 standard
- require library: libz
- compiler with support OpenMP for parallel (*option*)
- Library (*option*): netcdf, netcdf_cpp, libhdf5 for c++ [1]

2.1.2 Compile by Cmake

1. unzip the package file and change into it
2. mkdir build and change into it
3. cmake .. and some options you wanted
4. make
5. make install (*option*)

2.1.3 Testing with Example

If this is the first time you use the CVTree package, please go to the “example” folder. Edit “list” to include the genome names, and run the cvtree command to get the phylogeny tree by:

```
../build/cvtree -G faa
```

More detail of the command usage can be obtained by ‘-h’ option or read the following sections.

2.2 Run CVTree in Container

The containers allow users to run programs on both Windows and Linux/MacOS, and transfer the programs easily. To employ the container with CVTree, you should install `docker` at first. You can download docker free and reference from `https://docs.docker.com/install/` to how to install it. After install docker, basic usages for CVTree in the container are shown below:

1. Obtain image: You can build the cvtree docker image based on `Dockerfile` in the source code by command

   ```
   docker build -t="cvtree" .
   ```

[1] It seem that the hdf5 libraries in Anaconda was not working in our testing

Here option '-t' sets the image name. After building the image, you can delete the dangling images for build by `docker image prune`. This will save much storage space. You can also download a prebuilt cvtree image from the internet by command:

```
docker pull ghzuo/cvtree .
```

In this step, an image with cvtree programs will be obtained.

2. Start container from the image: run the following command in the CVTree directory, i.e. the directory which include the 'example' directory of the CVTree

   ```
   docker run --rm -it -v $PWD/example:/root/data cvtree
   ```

   In this step, you will enter the cvtree container, and the "example" folder of this project will be found in the "data" folder. Change the path to the data folder, and run

   ```
   cvtree -G faa
   ```

   You will get the result for eight genomes in the `list` file. You can change the path `'$PWD/example'` to your own data directory.

3. Exit and stop container: `exit` in docker terminal.

4. Run cvtree in a temporary container by one command without enter the container:

   ```
   docker run --rm -v $PWD:/data -w /data cvtree cvtree -G faa
   ```

5. More usage for `docker` can reference `https://docs.docker.com/`.

3 Programs and Command-Line Options

3.1 Scheme of CVTree

Figure 1shows the scheme of CVTree. In CVTree, the genome sequences were cut into many small segments with k length. The composition vector (CV) of the genome was obtained based on the amount of these K-strings with or without statistical models. Here the genome sequence can be recorded in protein, RNA, and DNA. In this way, we converted the genome sequence into a vector, which possesses many good attributions in mathematics, e.g. alignment of vectors is independent with the comparing order. Then a dissimilarity matrix was obtained based on these composition vectors. And the phylogenetic tree was inferred based on the dissimilarity matrix. To sum up, there are three steps for CVTree:

1. Convert a genome sequence to a composition vector. And two methods, Hao and Count are provided in CVTree.

2. Calculate the dissimilarity matrix based on the composition vectors. And Three methods, Cosine, InterList, and InterSet provide in CVTree.

3. Infer the phylogenetic tree by dissimilarity matrix. And the Neighbor-Joint is provided in CVTree.

The classical CVTree algorithm was based on the Markov model and first released by Prof. Bailin Hao in 2004 (Qi *et al.*, 2004*b*). Recently we added some new algorithms into CVTree software. The detail of algorithms is described in the subsequent section. We noted that the CVTree is an open frame for the alignment-free phylogeny, and new methods will be added in the feature.

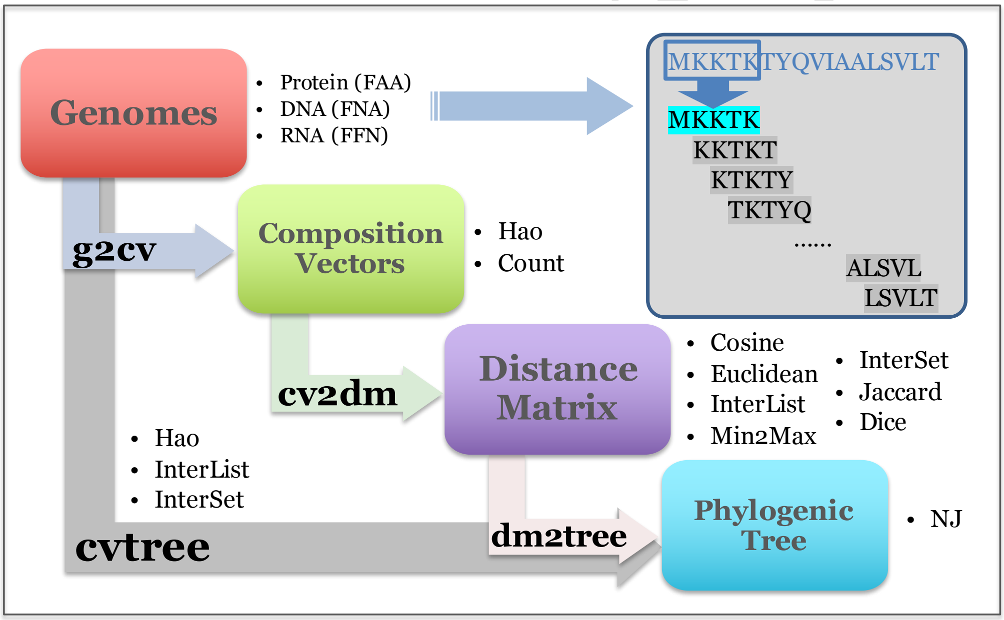

Figure 1: The Scheme of CVTree

In the standalone version CVTree, there are two ways to implement the CVTree algorithms to obtain the phylogenetic tree from the fasta files:

- all in one command: cvtree
- step by step commands: g2cv, cv2dm, dm2tree

We suggest you use the 'all in one command' as the way in testing.

IMPORTANT: You need prepare the fasta files of genomes, one file for one genome, and a list file, which includes the genome names you want calculating.

3.2 File Format

The pipeline of CVTree algorithm inputs the fasta file, one file for one genome, and output the phylogenetic tree in Newick format. That is, the initial input and final output files are in a standard format. However, to save space, the intermediate data are saved in compressed binary files.

3.2.1 Composition Vector File Format

The CV file was compressed by the gunzip algorithm. And when the k is large, the composition vector is a sparse vector. Thus on only the non-zero items are save in the CV file. Table 1 shows the format of the uncompressed CV file. The two items at the header are the L^2 norm and the size of the composition vector. And the rest is the composition vector, which saved as the key-value pairs. And the keys of the vector were encoded from string to unsigned long integer. It can not be viewed directly. A program, named `cvdump`, is included in the software to view the CV files.

← 8 bytes →	←8 bytes→
L2 Norm	Size of Vector
Key01	Value01
Key02	Value02
Key03	Value03
...	...

Table 1: Format of CV file

3.2.2 Dissimilarity Matrix File Format

When all optional library is prepared in compiled, there are three formats for the dissimilarity Matrix, i.e. the PHYLIP format and two compressed formats (HDF5 and netCDF). When the optional library is not supported in installation, only the standard PHYLIP format will be supported. The default format is netCDF when all formats are supported. However, it can changed by the option and suffix of the dissimilarity matrix file, e.g. ".txt" for PHYLIP format, ".h5" for HDF5, and ".nc" for netCDF. The compressed files can be viewed by `h5dump` and `ncdump` respectively. It is obvious that the dissimilarity matrix is symmetrical with zero diagonal elements. Thus only the triangle matrix is saved in the compressed files. A command, named `getdist`, is included in the software to obtain the dissimilarity of genomes from the compress files. A command, named `mconv`, is included in the software to convert the formats of dissimilarity matrix. And a command, named `diffmtx`, is included in the software to compare two dissimilarity matrices. Please don't compare the compress format matrix with the text (PHYLIP format) matrix. Due to the round-off errors, save the matrix in to text format will lead the change of the matrix.

3.3 All in One Command

3.3.1 The Workflow of cvtree command

The `cvtree` command is the main command for this software. It inbuilt an automatic workflow to obtain the phylogenetic tree from the fasta files directly. Figure 2 shows the flowchart of the `cvtree` command. It builds the dissimilarity matrix based on the reference dissimilarity matrices and fills the missing dissimilarity between two genomes automatically. You need only preparing the fasta files of genomes and a genome list, and setting the options according to the requirements of your project. We noted that in method selection of `cvtree`, you can set the selection for cv method, distance method, and tree method by a string split by a colon, e.g. `cvtree -m Counted:Cosine:NJ` · · · . And we provided three shortcut tags (see figure 1): `Hao` for “Hao:Cosine:NJ”; `InterList` for “Count:InterList:NJ”; and `InterSet` for “Counted:InterSet:NJ”.

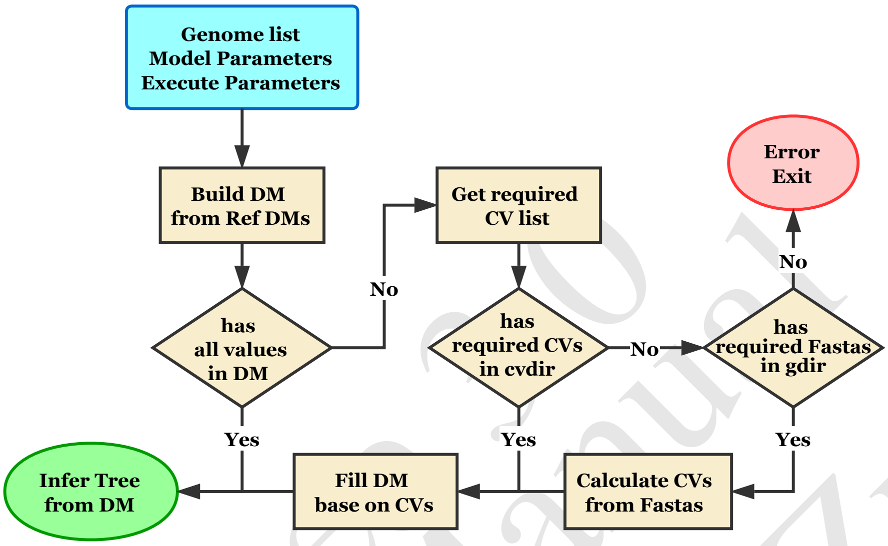

Figure 2: The Flowchart of the cvtree command

3.3.2 The command usage

- cvtree – Generate Newick tree from input genomes

```
cvtree
  [ -d <dm> ]         Output distance matrix name,
                      default: <Method><Suffix><K>
  [ -t <nwk> ]        Output newick file name,
                      default: <Method><Suffix><K>.nwk
  [ -G <gdir> ]       Super directory of Input genome file,
                      default: <current directory>
  [ -g faa ]          the type of genome file,
                      default: faa
  [ -V <cvdir> ]      Super directory of cv files
  [ -i list ]         Genome list for distance matrix,
                      default: list
  [ -k '5 6 7' ]      values of k,
                      default: K = 5 6 7
  [ -r <matrix> ]     Reference distance matrices, split with ','
  [ -M <N> ]          Running memory size as G roughly,
                      default 80% of physical memory
```

```
  [ -m Hao ]          Method for cvtree Hao/InterList/InterSet,
                      default: Hao
  [ -R ]              Refer the output distance matrix
  [ -q ]              Run command in quiet mode
  [ -h ]              Display this information
```

3.4 Step by Step Commands

You can also implement your project step by step by three programs, i.e. `g2cv, cv2dm, dm2tree`. The function and usage of these three programs are shown below:

- g2cv – Generate CV files from input data

```
g2cv
 [ -G <gdir> ]       input genome file directory
 [ -V <cvdir> ]      output cv directory
 [ -i list ]         input species list, default: list
 [ -f <Fasta> ]      get cv for only one fasta
 [ -k '5 6 7' ]      values of k, default: K = 5 6 7
 [ -g faa ]          the type of genome file, default: faa
 [ -m Hao/Count ]    the method for cvtree, default: Hao
 [ -q ]              Run command in quiet mode
 [ -h ]              Display this information
```

- cv2dm – Generate distance matrix based on CV files

```
cv2dm
 [ -o <dm> ]         Output distance matrix,
                     default: <Method><Suffix>.h5
 [ -V <cvdir> ]      Super directory of extend cv files
 [ -i list ]         Genome list for distance matrix,
                     default: list
 [ -s <Suffix> ]     Suffix of the cvfile, default: .faa.cv6
 [ -r <matrix> ]     Reference distance matrices, split with ','
 [ -M <N> ]          Running memory size as G roughly,
                     default 80% of physical memory
 [ -m Cosine ]       Method for distance:
                     Cosine/Euclidean based on vector;
                     InterList/Min2Max based on count of kmers;
                     InterSet/Jaccard/Dice based set of kmers.
                     default: Cosine
 [ -q ]              Run command in quiet mode
 [ -h ]              Display this information
```

- dm2tree – Generate Newick tree from distance matrix by using the Neighbor-Joint method

```
dm2tree
  [ -d infile ]      Input distance matrix,
                     default: dist.matrix
  [ -o Tree.nwk ]    Output Newick tree,
                     default: Tree.nwk
  [ -i <list> ]      index list of selection genomes
                     of the distance matrix, if no defined,
                     whole distance matrix are used
  [ -C ]             Use the netCDF input format,
                     default false
  [ -q ]             Run command in quiet mode
  [ -h ]             Display this information
```

3.5 Tools

In cvtree software, the final output phylogenetic tree is in Newick format. And in the implementation of cvtree, the intermediate results are also saved to avoid the reduction calculation. These intermediate results are in binary format in default, and can not be examined directly. We prepared some tools to examine these binary files. The function and usage of these tools are shown below:

- cvdump – convect the binary cv file to acsii file

```
cvdump  -i <cvfile>  input file name
  [ -g faa|ffn ]      the type of genome file, default: faa
  [ -n ]              output the number code,
                      default: the letters
  [ -h ]              display this information
```

- getdist – get the distance from the distance matrix

```
getdist
  [ -d infile ]      distance matrix files list separate by ","
                     default: infile
  [ -i selfile ]     the select genomes, default: none
  [ -H ]             whether output html table, default: false
  [ -l ]             get the list of species of distance matrix
  [ -h ]             display this information
```

- diffmtx – convert the dissimilarity matrix format

```
mconv
 [ -i input ]     Input matrix file, format determined by suffix
 [ -o output ]    output matrix file, format determined by suffix
 [ -h ]           Display this information
```

- diffmtx – whether two matrixes is equal

```
diffmtx <dm1> <dm2>   input the two distance matrices
```

4 Algorithm

4.1 The Classical CVTree Method: Hao Method

As the description above, there are three steps to obtain the phylogenetic tree based on the genomes by CVTree, i.e., modeling composition vector from genome, calculating dissimilarity matrix from composition vectors, and inferring phylogenetic tree from dissimilarity matrix. Here we describe the classical algorithm of CVTree briefly. For more detailed description of the algorithm please consult reference (Qi *et al.*, 2004*b*).

1. Fix a string length k ($k \in [3, 14]$ for Amino Acid sequences and $k \in [3, 30]$ for nucleotide sequences). Read in the sequence collection of each species separately. Count the number of all k, $k-1$ and $k-2$ tuples for a species. And the counts of the K-tuples are recorded as $f(a_1a_2\cdots a_k)$ with a_i indicates the letter of the residue of protein or nucleic acid. Thus the probability of a k-tuple in the genome sequence is:

$$p(a_1a_2\cdots a_k) = \frac{f(a_1a_2\cdots a_k)}{N_k}, \tag{1}$$

here N_k is the total number of k-tuples. It is obvious that a k-tuple $a_1a_2\cdots a_k$ can be obtained by adding a a_k after $a_1a_2\cdots a_{k-1}$. As the Markov model, the probability of a k-tuple $a_1a_2\cdots a_k$ can be predicted by the probability of its subsequence $a_1a_2\cdots a_{k-1}$ by a conditional probability, i.e.:

$$\tilde{p}(a_1a_2\cdots a_k) = p(a_k|a_1a_2\cdots a_{k-1})p(a_1a_2\cdots a_{k-1}) \tag{2}$$

we supported that the conditional probability is independent with the first letter, then the conditional probability can be calculated by the probability of $k-1$-tuple and $k-2$-tuple, i.e.:

$$\begin{aligned}\tilde{p}(a_1a_2\cdots a_k) &\approx p(a_k|a_2\cdots a_{k-1})p(a_1a_2\cdots a_{k-1}) \\ &= \frac{p(a_2\cdots a_{k-1}a_k)p(a_1a_2\cdots a_{k-1})}{p(a_2\cdots a_{k-1})}\end{aligned} \tag{3}$$

Then we define the composition vector $\vec{V}$ as the relative difference between the predicted value and the real value

$$v_i(a_1a_2\cdots a_k) \equiv \frac{p(a_1a_2\cdots a_k) - \tilde{p}(a_1a_2\cdots a_k)}{\tilde{p}(a_1a_2\cdots a_k)}, \tag{4}$$

And $v_i(a_1a_2\cdots a_k) = 0$ if $\tilde{p}(a_1a_2\cdots a_k) = 0$.

2. For a project with N genomes, the dissimilarity matrix is a $N \times N$ matrix with its diagonal equal 0. The non-diagonal items are calculated by the cosine of two composition vectors in the classical CVTree algorithm, i.e.

$$d_{ij} = \frac{1}{2}\left(1 - \frac{\vec{V}_i \cdot \vec{V}_j}{||\vec{V}_i||_2 \cdot ||\vec{V}_j||_2}\right) \tag{5}$$

   Here $||\cdot||_p$ is the L^p norm of the composition vector. In the CVTree algorithms, $p = 0, 1, 2$ are used.

3. Then infer the phylogenetic tree (Newick Format) based on this dissimilarity matrix by Neighbor-Joint method.

4.2 Two Typical New CVTree Methods

In this version of CVTree, we added two new methods, named by `InterSet` and `InterList`. The `InterSet` method was proposed by Dr. Qiang Li, a student of Prof. Hao. It is one of the methods described in his Ph.D. thesis of Qiang Li. The `InterList` method proposed by Dr. Guanghong Zuo. All these methods have the same scheme. That is, there are three steps, calculating the composition vector, building a dissimilarity matrix, and inferring phylogenetic tree. In the following sections, we describe these two methods briefly.

4.2.1 The InterSet Method

In the `InterSet` method, the genome is represented by the K-strings set in theory. And the dissimilarity between genomes was defined based on the intersection of two represented sets.

1. In practice, the composition vector is sparse for most of k, and only the existed K-strings will be recorded in cvfile as key-value items. Thus the set of all keys of K-strings counts is the K-strings set. Considering of reusing of data, we use the counts of the K-strings as the composition vector of `InterSet` method. That is, the value of the composition vector for every dimension (K-strings) is:

$$v_i = f(a_1 a_2 \cdots a_k) \tag{6}$$

2. The dissimilarity between two genomes is calculated by

$$d_{ij} = 1.0 - \frac{||\vec{V}_i \odot \vec{V}_j||_0}{\sqrt{||\vec{V}_i||_0 \cdot ||\vec{V}_j||_0}}, \tag{7}$$

   Here $\odot$ is the component-wise product (Hadamard product).

3. The phylogenetic tree (Newick Format) is also inferred by the Neighbor Joint method.

4.2.2 The InterList Method

In the `InterList` method, the genome is represented by the vector of the histogram of K-strings. And the dissimilarity between two vectors is defined based on the number of overlapped K-strings of two genomes. The method is inspired by the `InterSet` method. It may be helpful to study the relationship between the dissimilarity of CVTree and the molecular clock.

1. The counts of the K-strings is used as the composition vectors in `InterList` method. That is, the value of the composition vector for every dimension (K-strings) is:

$$v_i = f(a_1 a_2 \cdots a_k) \tag{8}$$

2. The dissimilarity between two genomes is calculated by

$$d_{ij} = 1.0 - \frac{2.0 \cdot ||\min\circ(\vec{V}_i, \vec{V}_j)||_1}{||\vec{V}_i||_1 + ||\vec{V}_j||_1} \tag{9}$$

   where $\min\circ(\cdot,\cdot)$ is component-wise minimum of two vectors. We noted that here the d_{ij} can also be written as:

$$d_{ij} = \frac{||\vec{V}_i - \vec{V}_j||_1}{||\vec{V}_i||_1 + ||\vec{V}_j||_1} \tag{10}$$

   and $||\vec{V}_i - \vec{V}_j||_1$ is the Manhattan distance between two vectors.

3. The phylogenetic tree (Newick Format) is also inferred by the Neighbor Joint method.

4.3 Available Methods in Detail

The standalone CVTree is an open workflow for the alignment-free phylogenetic algorithm. It is easy to implement new methods by C++ coding (see detail for how to add method in the subsequent section). Thus there are many optional methods available in the standalone CVTree in every step. And we will add other methods in the program in the feature.

4.3.1 Available Composition Vector Methods

There are two composition methods in standalone CVTree:

- The Hao vector based on the difference between the counted probability and the predicted probability of the Markov model (see detail in section 4.1).
- The number of counted. We noted that the count vector is also used as the set of the K-strings in practice, because the composition vector is sparse for most of k, and only the non-zero dimension will be recorded in cvfile.

4.3.2 Available Dissimilarity Methods

According to the major factor of these methods, they can be divided into three groups:

- Based on the dot-product/difference and L^2 norm composition vectors
  - Cosine:
    $$d_{ij} = \frac{1}{2}\left(1 - \frac{\vec{V}_i \cdot \vec{V}_j}{||\vec{V}_i||_2 \cdot ||\vec{V}_j||_2}\right)$$
  - Euclidean:
    $$d_{ij} = ||\vec{V}_i - \vec{V}_j||_2$$
- Based on component-wise minimum/maximum and L^1 norm of composition vectors, i.e. the number of overlapped K-strings.
  - InterList:
    $$d_{ij} = 1.0 - \frac{2.0 \cdot ||\min \circ(\vec{V}_i, \vec{V}_j)||_1}{||V_i||_1 + ||V_j||_1}$$
  - Min2Max:
    $$d_{ij} = 1.0 - \frac{||\min \circ(\vec{V}_i, \vec{V}_j)||_1}{||\max \circ(\vec{V}_i, \vec{V}_j)||_1}$$
- based on the component-wise product and L^0 norm of composition vectors, i.e. the intersection of K-strings.
  - InterSet:
    $$d_{ij} = 1.0 - \frac{||\vec{V}_i \odot \vec{V}_j||_0}{\sqrt{||\vec{V}_i||_0 \cdot ||\vec{V}_j||_0}}$$
  - Jaccard:
    $$d_{ij} = 1.0 - \frac{||\vec{V}_i \odot \vec{V}_j||_0}{||\vec{V}_i + \vec{V}_j||_0}$$
  - Dice:
    $$d_{ij} = 1.0 - \frac{2 \cdot ||\vec{V}_i \odot \vec{V}_j||_0}{||\vec{V}_i)||_0 + ||\vec{V}_j||_0}$$

4.3.3 Available Tree Methods

Presently, only the neighbor-joint (NJ) tree method is included in the CVTree standalone package.

5 Design Pattern of CVTree

The standalone CVTree is coded by C++ language, and designed in an object-oriented model. The main programs of this version is parallel, implemented by OpenMP.

5.1 Coding of K-string

In the CVTree, a class named `Kstr` is used to handle the K-string. The main member variable of this class is a long integer. By using a coding map, this class will encode the K-string into a long integer with base-m number system. Here, $m = 20 + 1$ for protein sequences and $m = 4 + 1$ for DNA/RNA sequences. For the zero represents vacancy and non-zero numbers represent the letters of the sequence according to the coding map. There are limits for the length of the K-string in CVTree due to the length of the long integer of the computer system, i.e. $K_{max} < 64/\log_2(m)$. That is, the longest K-string is 14-letter for protein and 27-letter for DNA/RNA. And as in our study, coding K-strings into integer may obviously reduce the memory cost and speed the calculation. And it is enough for the phylogeny studies.

5.2 Two Data Structures for Composition Vector

The number of the K-string is L^K, where L is the number of letters, i.e. 20 for protein sequence and 4 for DNA/RNA sequence, K is the length of the K-string. However, for a N-letter sequence, there are only $N - K + 1$ K-strings. It is obvious there are massive missing K-strings when K-string is long. That is, the composition vector of genomes is a sparse vector. Thus only the existing K-string and its value will be saved as a key-value pair in CVTree. And two different data structures, associative array, and sequential array were designed to handle the composition vectors in different situations, generation, and calculation. In the generation of the composition vectors, an associative array was used to generate the composition vector. And the complete composition vector will be converted into a sequential array with the order of K-strings and save to the hard device in gnuzip format (see table 1). Here the associative array can be a map or a hash (unordered map). As our testing, the costs of the two data structures are very close. The calculation between two composition vectors was handled by the sorted sequential array. For two composition vectors with length N_1 and N_2, the worst complexity for comparing two sorted sequential array is $W(N_1 + N_2)$. It is remarkably smaller than that of binary search $O(N_1 \log_2 N_2)$ when $N_1 \sim N_2$. In fact, the calculation for the distance between composition vectors is the most cost in CVTree, and there are some more cheap algorithms are in testing in our lab.

5.3 Three Base Classes for Methods

Figure 3 shows the basic design pattern of CVTree. There are three steps to obtain the phylogenetic tree based on the genomes by CVTree, i.e. calculating composition vector form genome, building dissimilarity matrix from composition vectors, and inferring phylogenetic tree from dissimilarity matrix. Thus we designed three interfaces (C++ virtual classes), i.e. `CVmeth`, `DistMeth`, and `TreeMath`, to implement these three steps. Despite member functions which perform basic events, there are a factory function, named `create(···)`, and virtual function, named `cv(···)`, `dist(···)`, and `tree(···)` respectively in these three classes. To add a new method in a step:

- derives a class from the base class and implements the calculating algorithm in the derived class to override the virtual function in the base class.

- adds items in the factory function, named `create(···)`, of the base classes to provide the selection of methods.

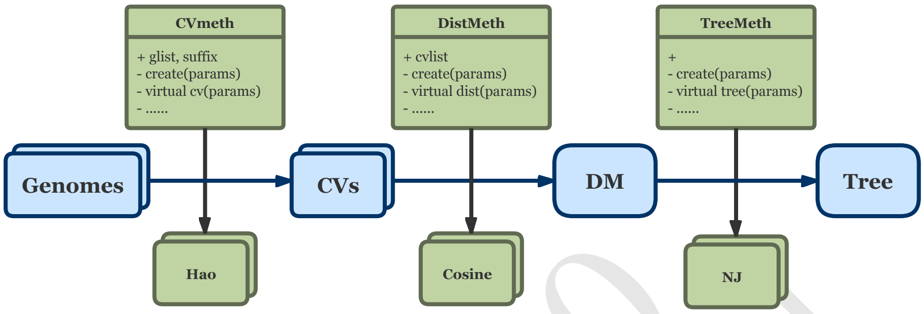

Figure 3: The Design Pattern of CVTree3

6 Version History and Contributors

Since the publication of the paper (Qi *et al.*, 2004*b*), many groups had implemented the classical CVTree algorithms. Here we list the major versions which implemented by our group, and the version numbers of the Standalone CVTree were reset by the version number of the Web Server CVTree as the standalone CVTree have never published:

- Most 0.x Standalone CVTree was written by Lei Gao; Ver. 0.9.6 was written by Ji Qi.
- Web Server CVTree 1.0 was written by Ji Qi, Hong Luo, and Bailin Hao
- Standalone CVTree 1.x was written by Zhao Xu
- Web Server CVTree 2.0 was written by Zhao Xu and Bailin Hao
- Standalone CVTree 2.x was written by Guanghong Zuo
- Web Server CVTree 3.0 was written by Guanghong Zuo and Bailin Hao
- Standalone CVTree 3.x was written by Guanghong Zuo
- Web Server CVTree 4.x was written by Guanghong Zuo and Bailin Hao

7 Citing CVTree in a Publication

Please cite:

1. Guanghong Zuo (2020) CVTree: A Parallel Alignment-free Phylogeny and Taxonomy Tool based on Composition Vectors of Genomes. *Genomics Proteomics & Bioinformatics,* `in submission`.
2. Guanghong Zuo and Bailin Hao (2015) CVTree3 Web Server for Whole-genome-based and Alignment-free Prokaryotic Phylogeny and Taxonomy. *Genomics Proteomics & Bioinformatics,* **13**: 321–331.

3. Ji Qi, Bin Wang, Bailin Hao (2004), Whole proteome prokaryote phylogeny without sequence alignment: a K-string composition approach. *Journal of Molecular Evolution*, **58**: 1–11.

References

Qi, J., Luo, H. and Hao, B. (2004*a*) Cvtree: a phylogenetic tree reconstruction tool based on whole genomes. *Nucleic acids research,* **32**, W45–7. PMID: 15215347.

Qi, J., Wang, B. and Hao, B. (2004*b*) Whole proteome prokaryote phylogeny without sequence alignment: a k-string composition approach. *Journal of Molecular Evolution,* **58**, 1–11.

Xu, Z. and Hao, B. (2009) Cvtree update: a newly designed phylogenetic study platform using composition vectors and whole genomes. *Nucleic Acids Research,* **37 Web Server Issue**, W174–W178.

Zuo, G. and Hao, B. (2015) CVTree3 Web Server for Whole-genome-based and Alignment-free Prokaryotic Phylogeny and Taxonomy. *Genomics Proteomics & Bioinformatics,* **13**, 321–331.

Zuo, G. (2020) CVTree: A Parallel Alignment-free Phylogeny and Taxonomy Tool based on Composition Vectors of Genomes. *Genomics Proteomics & Bioinformatics,* `in submission`.