# kWIP Documentation

***Release 0.2.0-rc***

**Kevin Murray**

October 13, 2016

Contents

Contents:

CHAPTER 1

# kWIP

The $k$-mer Weighted Inner Product

## 1.1 Overview

kWIP is a method for calculating genetic similarity between samples. Unlike similar alternatives, e.g. SNP-based distance calculation, kWIP operates directly upon next-gen sequencing reads. kWIP works by decomposing sequencing reads to short $k$-mers, hashing these $k$-mers and performing pairwise distance calculation between these sample $k$-mer hashes. We use khmer from the DIB lab, UC Davis to hash sequencing reads. kWIP calculates the distance between samples in a computationally efficient manner, and generates a distance matrix which may be used by downstream tools. The power of kWIP comes from the weighting applied across different hash values, which decreases the effect of erroneous, rare or over-abundant $k$-mers while focusing on $k$-mers which give the most insight into the similarity of samples.

## 1.2 kWIP CLI Usage

```
USAGE: kwip [options] sample1 sample2 ... sampleN

OPTIONS:
-t, --threads       Number of threads to utilise. [default N_CPUS]
-k, --kernel        Output file for the kernel matrix. [default None]
-d, --distance      Output file for the distance matrix. [default stdout]
-U, --unweighted    Use the unweighted inner proudct kernel. [default off]
-w, --weights       Bin weight vector file (input, or output w/ -C).
-C, --calc-weights  Calculate only the bin weight vector, not kernel matrix.
-h, --help          Print this help message.
-V, --version       Print the version string.
-v, --verbose       Increase verbosity. May or may not acutally do anything.
-q, --quiet         Execute silently but for errors.
```

The kwip executable is the core of kWIP; its help statement is reproduced above. This program operates on the saved Countgraphs of khmer. One can run with or without the entropy weighting, using the -U parameter to disable weighting.

An example command could be:

```
kwip \
    -t 4 \                          # Use 4 threads
    -k rice.kern \                  # Output kernel matrix to ./rice.kern
```

```
    -d rice.dist \                # Output distance matrix to ./rice.dist
    ./hashes/rice_sample_*.ct.gz  # Path to sample hashes, with wildcard
```

Note that this is purely illustrative and won't run as-is due to the in-line comments. Were it to run, it would calculate the Weighted Innner Product (WIP) kernel pairwise between all samples given as arguments, utilising four threads and saving the raw kernel matrix to rice.kern and the normalised distance matrix to rice.dist.

## 1.3 The Concepts Behind `kWIP`

The inner product between two vectors is directly related to the distance between the vectors in Euclidean space. This has been utilised several times in bioinformatics to implement measures of genetic similarity between two sequences, including the $D2$ statistic. Traditionally, the software which implement these and similar algorithms operate on known genetic sequences, e.g. those taken from a reference genome. `kWIP`'s innovation is to weight the inner product operation by a weight vector, and to derive weights in a way which minimises the noise inherent in next-gen sequencing datasets while maximising the signal of genetic distance between samples.

CHAPTER 2

# Installing kWIP

## 2.1 The full way

Dependencies:

- zlib
- cmake>=2.8
- Optionally, liboxli and Eigen3 are required. These libraries are bundled with kWIP, and the internal copy will be used if system copies are not.
- A C++11 compiler that supports OpenMP (i.e. gcc >=4.8)

On Debian (or Debian derivatives) the dependencies of kWIP can be installed with:

```
sudo apt-get install zlib1g-dev cmake build-essential git
```

Then, to compile kWIP:

```
git clone https://github.com/kdmurray91/kWIP.git
cd kWIP
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=${HOME}
make
make test
make install
```

The commands above assume you want to install kWIP to your home directory. This is probably required on clusters, and necessary without root privileges. To install to, e.g, /usr/local/, replace $HOME with your preferred installation prefix.

## 2.2 The easy way

Pre-compiled static binaries for 64-bit GNU/Linux are provided on the GitHub releases page. The following commands will obtain and install kWIP.

```
# If ~/bin/ is not in $PATH, you won't be able to use kwip.
# Perform the command below to ensure PATH is set correctly.
echo "PATH=\"${HOME}/bin:\${PATH}\"" >> ~/.bashrc
. ~/.bashrc

cd $HOME
```

```
wget https://github.com/kdmurray91/kWIP/releases/download/0.2.0/kwip-binaries_0.2.0.tar.gz
wget https://github.com/kdmurray91/kWIP/releases/download/0.2.0/kwip-binaries_0.2.0.tar.gz.sha256sums
sha256sum -c kwip-binaries_0.2.0.tar.gz.sha256sums

# Extract the precompiled binaries
tar xvf kwip-binaries*.tar.gz

# Check your installation by typing
kwip --help
```

Please note that as these binaries are compiled to be as widely compatible as possible, the compiler will use few modern optimisations. Therefore it is possible that these static binaries will be slower on modern processors than binaries compiled from source.

CHAPTER 3

# Example kWIP Analysis Protocols

These protocols should work on any computer running a unix-like OS with sufficient resources (8GB RAM, two cores, 100GB disk). If any step fails, please re-try that step. If you are unable to complete a protocol, this is probably a bug in either kWIP or this protocol, so please let me know by creating an issue on GitHub.

## 3.1 *Oryza sativa* grouping

This example uses data from the 3000 Rice Genomes Project. The rice genome isn't tiny, so you'll want a machine with at least 8GB RAM, and a least 2 cores.

### 3.1.1 Requirements

- virutalenvwrapper
- git
- the SRA toolkit
- kWIP, installed such that the kwip binary is in $PATH
- We also need the img.R script which comes with kWIP, but isn't installed. Hereafter I assume that you have this script in the analysis directory, ~/rice_kwip. Copy it with cp util/img.R ~/rice_kwip from the directory you git clone-d or unziped kWIP into.

### 3.1.2 Setup

Create a working directory

```
mkdir ~/rice_kwip
cd ~/rice_kwip
```

Install SRApy and khmer using pip. We will require the master branch of khmer until the 2.0 release, as bugs exist in the latest release uploaded to PyPI.

```
mkvirtualenv rice-kwip
pip install srapy
pip install -e git+https://github.com/dib-lab/khmer.git
```

Create a list of SRA identifiers in file sra_run_ids.txt using the following command:

```
cat >sra_run_ids.txt <<EOF
ERR626208
ERR626209
ERR626210
ERR626211
ERR619511
ERR619512
ERR619513
ERR619514
EOF
```

Download the above sra files:

```
mkdir sra_runs
get-run.py -s -f sra_run_ids.txt -d sra_files
```

And export them to FASTQ files:

```
mkdir fastqs
for srr in $(cat sra_run_ids.txt)
do
    fastq-dump                              \
        --split-spot                        \
        --skip-technical                    \
        --stdout                            \
        --readids                           \
        --defline-seq '@$sn/$ri'            \
        --defline-qual '+'                  \
        sra_files/${srr}.sra |              \
        gzip -1 > fastqs/${srr}.fastq.gz
done
```

Don't worry about the full details of this `fastq-dump`, but it produces a gzipped interleaved fastq file.

### 3.1.3 Hashing

We directly utilise `khmer`'s `load-into-counting.py` to hash reads to a hash (Countgraph).

```
mkdir hashes
for srr in $(cat sra_run_ids.txt)
do
    load-into-counting.py       \
        -N 1                    \
        -x 1e9                  \
        -k 20                   \
        -b                      \
        -f                      \
        -s tsv                  \
        hashes/${srr}.ct.gz     \
        fastqs/${srr}.fastq.gz
done
```

This creates a hash with a single table and a billion bins for each run. Hashes are saved, with gzip compression, to the `*.ct.gz` files under `./hashes`. These hashes are the direct input to `kwip`. Note that this hash is probably a bit small for this dataset, but we will go ahead anyway so this works on most modern laptops.

### 3.1.4 Distance Calculation

So here's the core of the protocol: calculating the pairwise distances between these samples, which are from the two major groups of rice, Indica and Japonica.

```
kwip                            \
    -t 2                        \
    -k rice.kern                \
    -d rice.dist                \
    hashes/*.ct.gz
```

This should calculate the weighted distance matrix between these samples, using two threads.

Now, we plot these results using the R script `img.R`. This creates plots of the distance and kernel matrices, as well as a cluster dendrogram and multi-dimensional scaling plot.

```
Rscript img.R rice
```

This should create `rice.pdf`. Inspect, and you should see two large groupings corresponding to the two rice families.

CHAPTER 4

# Experiments Around `kWIP`

Jupyter notebooks will appear here as they are published.

CHAPTER 5

# Indices and tables

- genindex