pdp Documentation

Leighton Pritchard

Jun 07, 2019

Contents:

CHAPTER 1

Build Information

CHAPTER 2

PyPI version and Licensing

CHAPTER 3

`conda` version

If you're feeling impatient, please head over to the *QuickStart Guide*

CHAPTER 4

Description

`pdp` is a program and Python package (`diagnostic_primers`) that provides support for design and identification of diagnostic qPCR and metabarcoding primers and marker sequences. It performs automated finding of discriminatory (real-time) PCR or qPCR primers that distinguish among genomes or other biological sequences of interest. It is also useful for the identification of metabarcoding marker sequences that can discriminate within a subset of bacterial genomes.

Where available, `pdp` natively takes advantage of multicore systems, and can integrate with SGE or OGE-compatible job schedulers to manage the computationally-heavy sequence comparisons.

This package can be used with either of two command-line interfaces: `find_differential_primers/find_differential_primers.py`, which maintains backwards compatibility with the old `find_differential_primers` package, and `pdp`, which is a new step-wise interface for scripting and performing specific stages of the design in isolation. The ability to separate out individual stages allows for scripting of branching designs, where alternative design filters and parameters may be used.

Attention: The `find_differential_primers/find_differential_primers.py` interface may be deprecated in a future release.

CHAPTER 5

Reporting problems and requesting improvements

If you encounter bugs or errors, or would like to suggest ways in which `pdp` can be improved, please raise a new issue at the `pdp` GitHub issues page.

If you'd like to fix a bug or make an improvement yourself, contributions are welcomed, and guidelines on how to do this can be found at the *Contributing to pdp* documentation page.

5.1 QuickStart Guide

5.1.1 Installation

To use `pdp` you will need to install it on your machine (laptop, desktop, server, or cluster). Installation is easiest when using one of the two popular Python package managers:

1. `conda`

`pdp` is available through the `bioconda` channel of Anaconda:

```
conda install -c bioconda pdp
```

2. `PyPI`

`pdp` is available *via* the PyPI package manager for Python:

```
pip install pdp
```

Tip: `pdp` can also be installed directly from source. More detailed installation instructions can be found on the *Installation Guide* page.

5.1.2 pdp Walkthrough for qPCR diagnostic design

An example qPCR primer set design using `pdp` is provided as a walkthrough below. The general procedure for any `pdp` analysis is:

1. Create a configuration file that describes which input sequences will be used for design, the paths to the sequence files, and labels describing which diagnostic classes or groups the sequences belong to
2. Fix the input sequences (if necessary) by stitching multiple contigs together and removing ambiguity symbols
3. Design primers to each input sequence
4. Deduplicate primer designs (this reduces computation time)
5. Screen primers against a `BLAST` or `DIAMOND` database of off-target sequences
6. Perform *in silico* cross-hybridisation of primers against each input sequence
7. Classify each primer set by specificity to the input sequence group labels, to generate candidate diagnostic primer sets

Tip: To see options available for the `pdp` program, or any subcommand, use the `-h` or `--help` option, e.g.:

```
pdp -help
pdp config -h
```

1. Create configuration file

We will work with three bacterial genomes in the `tests/walkthrough/sequences` subdirectory of the `find_differential_primers` repository. These genomes represent three different *Pectobacterium* species, and are provided as `.fasta` files containing complete genome sequences. We can see these files using the `ls` command:

```
$ ls tests/walkthrough/sequences/
GCF_000011605.1.fasta       GCF_000291725.1.fasta   GCF_000749845.1.fasta
```

To use these files in our analysis, we need to construct a configuration file. In this quick walkthrough will use a pre-prepared configuration file at the location `tests/walkthrough/pectoconf.tab`. The contents are shown below (note the extensive commenting, with lines preceded by a hash/octothorpe).

```
$ cat tests/walkthrough/pectoconf.tab
# Pectobacterium genomes downloaded from GenBank/NCBI; genomovars inferred from ANIm
# Annotated Pba: genomovar 1
Pba_SCRI1043        Pectobacterium,atrosepticum_NCBI,gv1    tests/walkthrough/
↪sequences/GCF_000011605.1.fasta       -
# Annotated Pwa: genomovars 2, 3
Pwa_CFBP_3304       Pectobacterium,wasabiae_NCBI,gv2        tests/walkthrough/
↪sequences/GCF_000291725.1.fasta       -
# Annotated Pb       : genomovar 7
Pbe_NCPPB_2795      Pectobacterium,betavasculorum_NCBI,gv7  tests/walkthrough/
↪sequences/GCF_000749845.1.fasta       -
```

The first line describing an input sequence tells us that its name is `Pba_SCRI1043`, that it belongs to classes/groups `Pectobacterium`, `atrosepticum_NCBI`, and `gv1`, and that the sequence's FASTA file can be found at `tests/walkthrough/sequences/GCF_000011605.1.fasta`. There are no features associated with the sequence.

Validate the configuration file

To confirm that the configuration file can be used in the rest of the design process, use the command `pdp config --validate` on that file:

```
$ pdp config --validate tests/walkthrough/pectoconf.tab
WARNING: Validation problems
    Pbe_NCPPB_2795 requires stitch (tests/walkthrough/sequences/GCF_000749845.1.fasta)
    Pwa_CFBP_3304 requires stitch (tests/walkthrough/sequences/GCF_000291725.1.fasta)
    Pwa_CFBP_3304 has non-N ambiguities (tests/walkthrough/sequences/GCF_000291725.1.
↪fasta)
```

2. Prepare the input sequences

Attention: To generate diagnostic primers and metabarcoding markers, the input sequences must each be "stitched" so that there is only a single contiguous sequence corresponding to each input file. Also, any IUPAC ambiguity symbols (e.g. *W*, *Y*, etc.) must be replaced with *N*.

```
pdp config --fix_sequences tests/walkthrough/fixed.json tests/walkthrough/pectoconf.
↪tab
```

3. Design primers to each input sequence

Now we can design primer sets against each input sequence, using the EMBOSS package `ePrimer3`.

```
pdp eprimer3 --outdir tests/walkthrough/eprimer3 \
    tests/walkthrough/fixed.json \
    tests/walkthrough/with_primers.json
```

The new `tests/walkthrough/eprimer3` directory now contains files describing primers designed to each input sequence, and corresponding `JSON` files describing the primer sets.

```
$ tree tests/walkthrough/eprimer3/
tests/walkthrough/eprimer3/
├── GCF_000011605.1.eprimer3
├── GCF_000011605.1_named.eprimer3
├── GCF_000011605.1_named.json
├── GCF_000291725.1_concat_noambig.eprimer3
├── GCF_000291725.1_concat_noambig_named.eprimer3
├── GCF_000291725.1_concat_noambig_named.json
├── GCF_000749845.1_concat.eprimer3
├── GCF_000749845.1_concat_named.eprimer3
└── GCF_000749845.1_concat_named.json
```

4. Deduplicate primer sets (optional)

Attention: This step is recommended, but not necessary, when designing diagnostic primer sets

When designing primers to groups of closely-related genomes, it is usual to have a large number of identical primer sets that originate from different genomes. We only need to test one of these redundant primer sets to know whether it may be diagnostically useful, so we can remove duplicates with the `pdp dedupe` command:

```
pdp dedupe --dedupedir tests/walkthrough/deduped \
    tests/walkthrough/with_primers.json \
    tests/walkthrough/deduped_primers.json
```

The complete set of nonredundant primers is written to `tests/walkthrough/deduped`, and a new `JSON` configuration file recording only the deduplicated primers for each input sequence is written to `deduped_primers.json`.

5. Screen primers against a local sequence database (optional)

Attention: This step is recommended, but not necessary, when designing diagnostic primer sets

Prescreening the primers we have just designed against a local database of off-target sequences allows us to remove primer sets that do not specifically amplify our input sequences without having to perform computationally costly *in silico* cross-hybridisation.

```
pdp blastscreen --db tests/walkthrough/blastdb/e_coli_screen.fna \
    --outdir tests/walkthrough/blastn \
    tests/walkthrough/deduped_primers.json \
    tests/walkthrough/screened.json
```

6. Perform *in silico* cross-hybridisation

This is the critical step in determining the predicted diagnostic specificity of the candidate primer sets. Each candidate primer set is tested in turn against all the input sequences to determine whether it has the potential to amplify that sequence. This is the most computationally-demanding step of the analysis.

```
pdp primersearch \
    --outdir tests/walkthrough/primersearch \
    tests/walkthrough/screened.json \
    tests/walkthrough/primersearch.json
```

7. Classify primer sets by specificity

The final step in determining qPCR primer set specificity is to analyse the *in silico* hybridisation results to determing which primer sets amplify exactly the members of each class/group defined in the initial configuration file.

```
pdp classify \
    tests/walkthrough/primersearch.json \
    tests/walkthrough/classify
```

The output directory contains `.json` and `.ePrimer3` format files for each set of candidate primers that were determined to be specific to a class/group named in the initial configuration file, and two summary files (`results.json` and `summary.tab`):

```
$ tree tests/walkthrough/classify/
tests/walkthrough/classify/
├── Pectobacterium_primers.ePrimer3
├── Pectobacterium_primers.json
├── atrosepticum_NCBI_primers.ePrimer3
├── atrosepticum_NCBI_primers.json
├── betavasculorum_NCBI_primers.ePrimer3
├── betavasculorum_NCBI_primers.json
├── gv1_primers.ePrimer3
├── gv1_primers.json
├── gv2_primers.ePrimer3
├── gv2_primers.json
├── gv7_primers.ePrimer3
├── gv7_primers.json
├── results.json
├── summary.tab
├── wasabiae_NCBI_primers.ePrimer3
└── wasabiae_NCBI_primers.json
```

The `summary.tab` file is a tab-separated plain text file that describes how many primer sets were determined to potentially be diagnostic for each input class, and describes a path to the `JSON` file describing their results:

```
$ cat tests/walkthrough/classify/summary.tab
Group   NumPrimers      Primers
Pectobacterium  4       tests/walkthrough/classify/Pectobacterium_primers.json
atrosepticum_NCBI       1       tests/walkthrough/classify/atrosepticum_NCBI_primers.
↪json
betavasculorum_NCBI     2       tests/walkthrough/classify/betavasculorum_NCBI_
↪primers.json
gv1     1       tests/walkthrough/classify/gv1_primers.json
gv2     2       tests/walkthrough/classify/gv2_primers.json
gv7     2       tests/walkthrough/classify/gv7_primers.json
wasabiae_NCBI   2       tests/walkthrough/classify/wasabiae_NCBI_primers.json
```

5.2 Installation Guide

5.3 `find_differential_primers/find_differential_primers.py` command-line interface

5.4 `pdp` command-line interface

The `pdp` command-line interface is intended to split individual stages of the complete primer/metabarcoding marker design process into logically-sensible "chunks". Each stage is invoked using the formula:

```
pdp <SUBCOMMAND> <OPTIONS> <ARGUMENTS>
```

The stages are described and summarised below, in the usual order of implementation for design of qPCR primers or metabarcoding markers.

5.4.1 0. Create configuration file

The pdp software takes as initial input a configuration file. This is a plain text, tab-separated file containing columns describing the following data:

- The name of the sequence (for use during analysis)
- A comma-separated list of labels describing classes/groups to which the sequence belongs
- The path to the sequence's FASTA file
- (Optionally) the path to a set of features (e.g. gene annotations); if no features are supplied, this can be replaced with a hyphen: -.

Tip: Any line in the configuration file that starts with a hash/octothorpe (#) is ignored. This is useful for commenting and/or ignoring specific sequences.

An example configuration file is included at the location `tests/walkthrough/pectoconf.tab`. The contents are shown below (note the extensive commenting, with lines preceded by a hash/octothorpe).

```
$ cat tests/walkthrough/pectoconf.tab
# Pectobacterium genomes downloaded from GenBank/NCBI; genomovars inferred from ANIm
# Annotated Pba: genomovar 1
Pba_SCRI1043        Pectobacterium,atrosepticum_NCBI,gv1    tests/walkthrough/
↪sequences/GCF_000011605.1.fasta       -
# Annotated Pwa: genomovars 2, 3
Pwa_CFBP_3304       Pectobacterium,wasabiae_NCBI,gv2        tests/walkthrough/
↪sequences/GCF_000291725.1.fasta       -
# Annotated Pb       : genomovar 7
Pbe_NCPPB_2795      Pectobacterium,betavasculorum_NCBI,gv7  tests/walkthrough/
↪sequences/GCF_000749845.1.fasta       -
```

The first line describing an input sequence tells us that its name is `Pba_SCRI1043`, that it belongs to classes/groups `Pectobacterium`, `atrosepticum_NCBI`, and `gv1`, and that the sequence's FASTA file can be found at `tests/walkthrough/sequences/GCF_000011605.1.fasta`. There are no features associated with the sequence.

Tip: It can be easiest to prepare your configuration files in a spreadsheet program such as Microsoft Excel, then convert the file to `.json` format with `pdp config`.

5.4.2 1. `pdp config`

The `pdp config` command provides functions for validation and handling of `pdp` configuration files. The key functions provided by this command are:

validation `pdp config` can check the formatting and content of a configuration file, to confirm that the appropriate fields are all populated for each input sequence, and that the specified files exist on the filesystem. The program will warn if input sequences require some repair.

```
$ pdp config --validate tests/walkthrough/pectoconf.tab
WARNING: Validation problems
    Pbe_NCPPB_2795 requires stitch (tests/walkthrough/sequences/GCF_000749845.1.fasta)
    Pwa_CFBP_3304 requires stitch (tests/walkthrough/sequences/GCF_000291725.1.fasta)
    Pwa_CFBP_3304 has non-N ambiguities (tests/walkthrough/sequences/GCF_000291725.1.
↪fasta)
```

(continues on next page)

(continued from previous page)

```
```

sequence repair Sequences used as input to `pdp` may not conform to requirements of some of the third party tools, or the pipeline's other assumptions. Some stages in the pipeline cannot accommodate genomes presented as multiple sequence files, and others cannot cope with nucleotide ambiguity symbols other than `N`. `pdp config` can repair input genome sequences by stitching them and replacing ambiguity codons, writing the "fixed" genome to a new file. Instead of modifying the input sequence directly, which would change the source data (a problem for reproducibility), `pdp` will "fix" these sequences by writing new, "stitched" and "cleaned" versions of the input sequences, then automatically update the configuration file to point to the modified files. This is done with the `pdp config --fix_sequences` command:

```
pdp config --fix_sequences tests/walkthrough/fixed.json tests/walkthrough/pectoconf.
↪tab
```

Attention: The `--fix_sequences` option takes as its argument the location to write the output configuration file; the final positional argument is the path to the input configuration file.

format conversion The `pdp config` subcommand can convert configuration files between `.tab` and JSON format. The `.tab` format is easier to read and manipulate in spreadsheet software, but all the tools in the `pdp` pipeline require input in `.json` format.

```
pdp config --to_json myconfig.json myconfig.tab
pdp config --to_tab myconfig.tab myconfig.json
```

`pdp config` output files are (except when converting to `.tab` format) written as JSON files. This is a machine-readable version of the configuration file, and all the other `pdp` tools accept `JSON` format configuration files, but not `.tab` files. As an example the `fixed.conf` configuration file produced in the example above is shown below. Here, the `Pba_SCRI1043` line is unmodified, the `Pbe_NCPPB_2795` is stitched (hence `concat` appears in the filename), and the `Pwa_CFBP_3304` is both stitched and has ambiguity symbols replaced (so has `concat_noambig` in the filename).

```
[
    {
        "features": null,
        "filestem": "GCF_000011605.1",
        "filtered_seqfile": null,
        "groups": [
            "Pectobacterium",
            "atrosepticum_NCBI",
            "gv1"
        ],
        "name": "Pba_SCRI1043",
        "primers": null,
        "primersearch": null,
        "seqfile": "tests/walkthrough/sequences/GCF_000011605.1.fasta"
    },
    {
        "features": null,
        "filestem": "GCF_000749845.1_concat",
        "filtered_seqfile": null,
        "groups": [
            "Pectobacterium",
            "betavasculorum_NCBI",
            "gv7"
```

(continues on next page)

(continued from previous page)

```
        ],
        "name": "Pbe_NCPPB_2795",
        "primers": null,
        "primersearch": null,
        "seqfile": "tests/walkthrough/sequences/GCF_000749845.1_concat.fas"
    },
    {
        "features": null,
        "filestem": "GCF_000291725.1_concat_noambig",
        "filtered_seqfile": null,
        "groups": [
            "Pectobacterium",
            "gv2",
            "wasabiae_NCBI"
        ],
        "name": "Pwa_CFBP_3304",
        "primers": null,
        "primersearch": null,
        "seqfile": "tests/walkthrough/sequences/GCF_000291725.1_concat_noambig.fas"
    }
]
```

As can be seen from this file, the modified sequences are written to the `tests/walkthrough/sequences` subdirectory, alongside the original:

```
$ tree tests/walkthrough/sequences/
tests/walkthrough/sequences/
├── GCF_000011605.1.fasta
├── GCF_000291725.1.fasta
├── GCF_000291725.1_concat.fas
├── GCF_000291725.1_concat_noambig.fas
├── GCF_000749845.1.fasta
└── GCF_000749845.1_concat.fas
```

5.4.3 2. `pdp filter`

As an optional step in the primer/marker design pipeline, input genomes may be filtered such that primer sets are only designed to restricted sections of the input. The `pdp filter` subcommand has options to automate this process for the following genomic regions:

predicted coding sequences The `pdp filter --prodigal` option carries out *a priori* CDS prediction on each input genome using the Prodigal software tool. It generates a `.gff` file, and includes that in the output configuration file. This is suggested as an approach to maximise conserved genomic regions as sources for robust qPCR diagnostic primer design, as coding sequences are expected to be relatively conserved.

```
pdp filter --prodigal myconfig.json cds_filtered.json
```

predicted intergenic regions Using the `pdp filter --prodigaligr` option will conduct CDS prediction with Prodigal, but generate a `.gff` file describing regions *between* predicted CDS, plus a short flanking regions that extends into neighbouring predicted CDS. This is included in the output configuration file. The size of the flanking region can be controlled with the `--flanklen` option. This is suggested as an approach to maximise amplicon sequence variation when designing metabarcoding markers, as intergenic regions are expected to be less well-conserved.

```
pdp filter --prodigaligr myconfig.json igr_filtered.json
```

consserved variable regions With the `pdp filter --alnvar <GROUP>` option, `pdp` will pairwise align all genomes annotated in the configuration file with group `<GROUP>`, to identify regions conserved across all members of that group. If the `--min_sim_error_rate <VALUE>` option is used, then only conserved regions that have an error rate (SNP or indel) of at least `<VALUE>``% (constrained in the range [0, 1) in *every* genome of ``<GROUP>` will be retained for primer design. This is suggested as an approach to maximising amplicon sequence variatino for metabarcoding marker design, as a minimum level of source sequence variation is assured.

```
pdp filter --alnvar group01 --min_sim_error_rate 0.1 myconfig.json igr_filtered.json
```

Tip: The `pdp filter` subcommand can be used with options for multiprocessing/SGE-like parallelisation (see below)

5.4.4 3. `pdp eprimer3`

Warning: The EMBOSS `ePrimer3` package uses the PRIMER3 primer design software, but will only work with an old version of that software: `v1.1.4`

The `pdp eprimer3` command takes a `.json` format configuration file, and uses the EMBOSS `ePrimer3` package to design primers to each input genome sequence, writing the output files to a location specified with the `--outdir <OUTDIR>` option. A new output `.json` configuration file is produced which associates primer information with the appropriate input sequence.

```
pdp eprimer3 --outdir primers/ myconfig.json genomes_with_primers.json
```

In order to use the "filtered" genomes specified in an input configuration file, the `pdp eprimer3` subcommand must be run with the `--filter` option.

```
pdp eprimer3 --filter --outdir primers/ myconfig.json genomes_with_primers.json
```

Tip: The `pdp eprimer3` subcommand can be used with options for multiprocessing/SGE-like parallelisation (see below)

5.4.5 4. `pdp dedupe`

The `pdp eprimer3` primer design step treats each genome independently which, especially when several related genomes are used as input, can result in design of many identical primer sets. The `pdp dedupe` subcommand identifies these redundant primer sets and discards them from the design process, generating a new `.json` configuration file that points only to non-redundant primer sets. The location of deduplicated/non-redundant primer sets generated in this process is given with the `--dedupedir <DIRNAME>` option, and `BLASTN+` output written to a location specified by the `--outdir <OUTPUTDIR>` option.

Tip: This step is optional, but it is typical that most primer sets in the primer design process are redundant. As some of the subsequent pipeline stages do not scale well with an increasing number of primer sets, deduplication is *highly* *recommended.*

```
pdp dedupe --dedupedir deduped/ myconfig.json deduped.json
```

5.4.6 5. `pdp blastscreen`

As a fast, preliminary screen of designed primers against a set of off-target sequences, the `pdp blastscreen` subcommand performs `BLASTN+` searches of designed primers against a pre-prepared nucleotide sequence database specified by the `--db <DBPATH>` option, where `<DBPATH>` is the path to the `BLAST+` database.

```
pdp blastscreen --db blastdb/offtargets --outdir blastn_outputmyconfig.json screened.
↪json
```

Tip: The composition of the screening database should be appropriate to your analysis/design goals. For example, if you are interested in designing primers diagnostic to all species in a particular bacterial genus, then a database comprising available genomes from sister genera may be appropriate. Alternatively, a subsampling of complete genomes from the bacterial group containing your genus of interest may be useful. However the screening database is constructed, it should represent a good range of off-target sequences that could reasonably be detected as false positives, to eliminate non-specific primer sets.

Tip: This step is optional, but it is typical that many primer sets in the primer design process have off-target matches and are not useful as diagnostic tools. As some of the subsequent pipeline stages do not scale well with an increasing number of primer sets, screening against a relevant database is *highly recommended.*

Tip: The `pdp blastscreen` subcommand can be used with options for multiprocessing/SGE-like parallelisation (see below)

5.4.7 6. `pdp primersearch`

`pdp primersearch` carries out the most critical step in the design process: predicting which genomes produce amplicons for each of the designed primer sets. Each candidate primer set is tested in turn against all the input sequences to determine whether it has the potential to amplify that sequence. This is the most computationally-demanding step of the analysis.

The `pdp primersearch` command uses the EMBOSS tool `primersearch` to carry out *in silico* hybridisation of each of the candidate primer sets against each input (*unfiltered*) genome. The output directory into which `primersearch` result files are written can be specified with `--outdir`:

```
pdp primersearch --outdir primersearch_results myconfig.json primersearch.json
```

Tip: It is strongly recommended that primers are deduplicated, and an off-target pre-screen is performed using `pdp blastscreen` or `pdp diamondscreen` before carrying out this step.

Tip: The `pdp primersearch` subcommand can be used with options for multiprocessing/SGE-like parallelisation (see below)

5.4.8 7. `pdp classify`

Each primer set can be assessed for potential specificity using the `pdp classify` subcommand to determine whether it is predicted to amplify specifically genomes only from one of the groups specified in the configuration file.

```
pdp classify myconfig.json classified_primers/
```

No new configuration file is produced, but new `.json` and `.ePrimer3` format files are written to the specified output directory for each of the groups specified in the configuration file. These contain accounts of the primer sets determined to be specific to that group.

```
$ tree tests/walkthrough/classify/
tests/walkthrough/classify/
├── Pectobacterium_primers.ePrimer3
├── Pectobacterium_primers.json
├── atrosepticum_NCBI_primers.ePrimer3
├── atrosepticum_NCBI_primers.json
├── betavasculorum_NCBI_primers.ePrimer3
├── betavasculorum_NCBI_primers.json
├── gv1_primers.ePrimer3
├── gv1_primers.json
├── gv2_primers.ePrimer3
├── gv2_primers.json
├── gv7_primers.ePrimer3
├── gv7_primers.json
├── results.json
├── summary.tab
├── wasabiae_NCBI_primers.ePrimer3
└── wasabiae_NCBI_primers.json
```

Two summary files are also written: `results.json` and `summary.tab`. The `summary.tab` file is a tab-separated plain text file that describes how many primer sets were predicted to be diagnostic for each input class, and describes a path to the `.json` file describing their results:

```
$ cat tests/walkthrough/classify/summary.tab
Group   NumPrimers      Primers
Pectobacterium  4       tests/walkthrough/classify/Pectobacterium_primers.json
atrosepticum_NCBI       1       tests/walkthrough/classify/atrosepticum_NCBI_primers.
↪json
betavasculorum_NCBI     2       tests/walkthrough/classify/betavasculorum_NCBI_
↪primers.json
gv1     1       tests/walkthrough/classify/gv1_primers.json
gv2     2       tests/walkthrough/classify/gv2_primers.json
gv7     2       tests/walkthrough/classify/gv7_primers.json
wasabiae_NCBI   2       tests/walkthrough/classify/wasabiae_NCBI_primers.json
```

5.4.9 8. `pdp extract`

The `pdp extract` subcommand extracts the amplicon sequences for each of the primers in a specified `pdp classify` output primer file (specific for a particular genome group), aligns those sequences using `MAFFT` and then produces summary statistics about the sequence diversity of the amplicons.

`pdp extract` requires the configuration file used for `pdp classify`, the path to the `.json` file describing the primers specific to a particular genome group, and the path to an output directory for result files. Given the output path `<OUTDIR>` and the group `group01`, the extracted sequences and summary files will be written to `<OUTDIR>/group01`.

```
pdp extract myconfig.json classified_primers/group01.json extracted/
```

The output directory will contain, for each primer set:

- a FASTA format file describing all the amplicon sequences (ending in `.fasta`)
- a `MAFFT`-aligned output file (ending in `.aln`) describing the aligned sequences
- a summary tab-separated plain text file called `distances_summary.tab`

The `distances_summary.tab` file is a table with one row per primer set, (and one row for headers), describing:

- primer set identifier
- mean pairwise distance between aligned sequences
- standard deviation of pairwise distance between aligned sequences
- minimum pairwise distance between aligned sequences
- maximum pairwise distance between aligned sequences
- the count of unique amplicons
- the number of non-unique amplicons
- the Shannon Index of sequence diversity (larger is more diverse)
- the Shannon Evenness of sequence diversity ([0, 1]: closer to 1 is more even)

Tip: The `pdp extract` subcommand can be used with options for multiprocessing/SGE-like parallelisation (see below)

5.4.10 Multiprocessing/SGE-like parallelisation

Attention: Several stages in the `pdp` pipeline (principally those that call third-party software tools) can take advantage of multicore systems or clusters using an SGE-like scheduler. The syntax for doing this is the same for each of the stages.

multiprocessing By default, subcommands that can use parallelism will attempt to distribute jobs to local cores using Python's built-in `multiprocessing` module. This can be explicitly enabled with the `-s multiprocessing` option, and the number of workers controlled with the `-w <N>` option, to limit the total number of workers to a maximum of `<N>`.

```
pdp eprimer3 --outdir primers -s multiprocessing -w 4 myconfig.json eprimer3.json
```

SGE-like schedulers The `-s SGE` option can be provided to use an SGE-like scheduler (one you can invoke with `qsub`). To cause minimal problems with queues, individual jobs are batched into job arrays, with a default array size of 10000 jobs (this can be controlled with the `--SGEgroupsize <N>` option). If you need to pass further arguments to SGE, this can be done with the `--SGEargs <ARGUMENTS>` option.

```
pdp eprimer3 --outdir primers -s SGE --SGEgroupsize 5000 --SGEargs "-M me@domain.org -
↪m bes" myconfig.json eprimer3.json
```

5.5 Testing

We are currently writing tests formatted for the nosetests package, for testing `pdp`.

Warning: We are aware that `nosetests` is in maintenance mode and, while we have no timetable for the move, our plan is to change the test framework to use pytest at some future date

5.6 Contributing to `pdp`

5.6.1 Reporting bugs and errors

If you find a bug, or an error in the code or documentation, please report this by raising an issue at the GitHub issues page for `pdp`:

- GitHub issues page

5.6.2 Contributing code or documentation

We gratefully accept code contributions, if you would like to fix a bug, improve documentation, or extend `pdp`. To make everyone's lives easier in this process, we ask that you please follow the guidelines below:

Pre-commit checks and style guide

So far as is possible, we aim to follow the coding conventions as described in PEP8 and PEP257, but we have adopted `black` code styling, which does vary from the PEPs in places.

We use the *flake8* tool for style checks, and this can be installed as follows (with two useful plugins):

```
pip install flake8 flake8-docstrings flake8-blind-except
```

flake8 can then be run directly on the codebase with

```
flake8 bin/
flake8 pyani/
```

We use the *black* tool for code style checking, which can be installed with:

```
pip install black
```

The `flake8` and ``black` styles can be enforced as pre-commit hooks using the pre-commit package (included in `requirements.txt`).

The `black` and `flake8` hooks are defined in `.pre-commit-config.yaml`; custom settings for `flake8` are held in `.flake8` (all files are under version control).

To enable pre-commit checks in the codebase on your local machine (once `pre-commit` has been installed), execute the following command in the root directory of this repository:

```
pre-commit install
```

Checking changes to the documentation

Much of the repository documentation is written in Markdown files, but the main documentation (which you are reading) is prepared for ReadTheDocs, which uses reStructuredText and `Sphinx`. The `Sphinx` configuration is described in `docs/conf.py` (under version control).

So long as `Sphinx` is installed on your machine, you can check your documentation changes locally, building inplace by changing to the `docs/` directory and issuing:

```
make html
```

This will place a compiled version of the documentation under `_build/html`, which you can inspect before committing to the repository.

Tip: To build the documentation in `ReadTheDocs` style, you will need to install the corresponding theme with `pip install sphinx_rtd_theme` or `conda install sphinx_rtd_theme`

For now, docstrings in the source code are not required to be in any controlled syntax, such as reStructuredText, but this may change.

Making changes and pull requests

1. Fork the `find_differential_primers` repository under your account at GitHub.
2. Clone your fork to your development machine.
3. Create a new branch in your forked repository with an informative name like `fix_issue_107`, using `git` (e.g. with the command `git checkout -b fix_issue_107`).
4. Make the changes you need and commit them to your local branch.
5. Run the repository tests (see the *Testing* documentation for more details).
6. If the tests all pass, push the changes to your fork, and submit a pull request against the original repository.
7. Indicate one of the `pdp` developers as an assignee to review your pull request when you submit your pull request.

The assigned developer will then review your pull request, and merge it or continue the conversation, as appropriate.

5.6.3 Suggestions for improvement

If you would like to make a suggestion for how we could improve `pdp`, we welcome contributions at the GitHub issues page.

5.7 Citing pdp

Please cite one or more of the following manuscripts in your work, if you have found `pdp` useful:

> Pritchard L, Holden NJ, Bielaszewska M, Karch H, Toth IK (2012) Alignment-Free Design of Highly Discriminatory Diagnostic Primer Sets for Escherichia coli O104:H4 Outbreak Strains. *PLOS ONE* **7** (4): e34498. doi:10.1371/journal.pone.0034498
>
> Pritchard, L. , Humphris, S. , Saddler, G. S., Parkinson, N. M., Bertrand, V. , Elphinstone, J. G. and Toth, I. K. (2013), Detection of phytopathogens of the genus Dickeya using a PCR primer prediction pipeline for draft bacterial genome sequences. *Plant Pathology*, **62** : 587-596. doi:10.1111/j.1365-3059.2012.02678.x

```
@article{10.1371/journal.pone.0034498,
    author = {Pritchard, Leighton AND Holden, Nicola J. AND Bielaszewska,␣
↪Martina AND Karch, Helge AND Toth, Ian K.},
    journal = {PLOS ONE},
    publisher = {Public Library of Science},
    title = {Alignment-Free Design of Highly Discriminatory Diagnostic␣
↪Primer Sets for Escherichia coli O104:H4 Outbreak Strains},
    year = {2012},
    month = {04},
    volume = {7},
    url = {https://doi.org/10.1371/journal.pone.0034498},
    pages = {1-8},
    abstract = {Background An Escherichia coli O104:H4 outbreak in Germany␣
↪in summer 2011 caused 53 deaths, over
    4000 individual infections across Europe, and considerable economic,␣
↪social and political impact. This outbreak
    was the first in a position to exploit rapid, benchtop high-throughput␣
↪sequencing (HTS) technologies and
    crowdsourced data analysis early in its investigation, establishing a␣
↪new paradigm for rapid response to
    disease threats. We describe a novel strategy for design of diagnostic␣
↪PCR primers that exploited this rapid
    draft bacterial genome sequencing to distinguish between E. coli O104:H4␣
↪outbreak isolates and other pathogenic
    E. coli isolates, including the historical hæmolytic uræmic syndrome␣
↪(HUSEC) E. coli HUSEC041 O104:H4 strain,
    which possesses the same serotype as the outbreak isolates.  ␣
↪Methodology/Principal Findings Primers were
    designed using a novel alignment-free strategy against eleven draft␣
↪whole genome assemblies of E. coli O104:H4
    German outbreak isolates from the E. coli O104:H4 Genome Analysis Crowd-
↪Sourcing Consortium website, and a
    negative sequence set containing 69 E. coli chromosome and plasmid␣
↪sequences from public databases. Validation
    in vitro against 21 'positive' E. coli O104:H4 outbreak and 32␣
↪'negative' non-outbreak EHEC isolates indicated
    that individual primer sets exhibited 100% sensitivity for outbreak␣
↪isolates, with false positive rates of
    between 9% and 22%. A minimal combination of two primers discriminated␣
↪between outbreak and non-outbreak E.
    coli isolates with 100% sensitivity and 100% specificity.   Conclusions/
↪Significance Draft genomes of isolates
    of disease outbreak bacteria enable high throughput primer design and␣
↪enhanced diagnostic performance in
```

(continues on next page)

(continued from previous page)

```
    comparison to traditional molecular assays. Future outbreak␣
↪investigations will be able to harness HTS rapidly
    to generate draft genome sequences and diagnostic primer sets, greatly␣
↪facilitating epidemiology and clinical
    diagnostics. We expect that high throughput primer design strategies␣
↪will enable faster, more precise responses
    to future disease outbreaks of bacterial origin, and help to mitigate␣
↪their societal impact.},
    number = {4},
    doi = {10.1371/journal.pone.0034498}
}

@article{doi:10.1111/j.1365-3059.2012.02678.x,
    author = {Pritchard, L. and Humphris, S. and Saddler, G. S. and␣
↪Parkinson, N. M. and Bertrand, V. and Elphinstone, J. G. and Toth, I. K.},
    title = {Detection of phytopathogens of the genus Dickeya using a PCR␣
↪primer prediction pipeline for draft bacterial genome sequences},
    journal = {Plant Pathology},
    volume = {62},
    number = {3},
    pages = {587-596},
    keywords = {diagnostics, Erwinia spp., Pectobacterium spp., potato, real-
↪time PCR},
    doi = {10.1111/j.1365-3059.2012.02678.x},
    url = {https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-3059.2012.
↪02678.x},
    eprint = {https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-3059.
↪2012.02678.x},
    abstract = {This study used a novel computational pipeline to exploit␣
↪draft bacterial genome sequences in order
    to predict, automatically and rapidly, PCR primer sets for Dickeya spp.␣
↪that were unbiased in terms of
    diagnostic gene choice. This pipeline was applied to 16 draft and four␣
↪complete Dickeya genome sequences to
    generate >700 primer sets predicted to discriminate between Dickeya at␣
↪the species level. Predicted diagnostic
    primer sets for both D. dianthicola (DIA-A and DIA-B) and 'D. solani'␣
↪(SOL-C and SOL-D) were validated against
    a panel of 70 Dickeya reference strains, representative of the known␣
↪diversity of this genus, to confirm primer
    specificity. The classification of the four previously sequenced strains␣
↪was re-examined and evidence of
    possible misclassification of three of these strains is presented.}
}
```

5.8 Indices and tables

- genindex
- modindex
- search