Emilian Jungwirth, BSc

Comparison of ddRAD Analysis Pipelines

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Biomedical Engineering

submitted to

Graz University of Technology

Supervisor

Dr Gerhard Thallinger

Institute of Neural Engineering

Institute of Computational Biotechnology

Graz, December 2017

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit[1]

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

An efficient determination of the genotype is important to many studies. Rather than sequencing the whole genome of an individual, genetic markers such as SNPs (single nucleotide polymorphism) are used for analysis. RAD (restriction site associated DNA) tags are DNA sequences, which are next to a restriction endonuclease recognition site. Restriction site associated DNA sequencing (RAD-seq) in combination with RAD analysis pipelines offer an efficient and robust way for the determination of genetic markers. In this thesis, an overview of existing RAD analysis pipelines is given and a comparison between the pipelines Stacks, PyRAD, ipyrad and dDocent is performed. The pipelines were compared by taking a look at the loci the pipelines recovered, phylogenetic and population structure results. Runtimes and memory footprint of the pipelines are also in the focus of this comparison.

Analyses for simulated and empirical data were performed. dDocent recovered more usable loci than the other pipelines and is also overall the fastest pipeline. But dDocent does not come with build-in functions for generating phylogenetic or population structure files. Stacks was also very fast except for one analysis. It also recovered the highest number of overall loci. PyRAD was the slowest pipeline for most of the data sets. But it recovered a relatively high number of loci. ipyrad performed much faster than PyRAD but it did not recover as many loci as PyRAD. The results of the pipelines overlapped strongly for the simulated data sets. But only the results of one of the empirical data sets did show a relatively high overlap. In general, Stacks had the largest memory footprint and was the only analysis pipeline that required more than the available 8 GB of memory on the test system.

Key words

RAD-seq; ddRAD-seq; Stacks; PyRAD; ipyrad; dDocent; rtd; Rainbow; species tree; population structure

Contents

Abstract iv

1 Introduction 1

1.1 RAD and ddRAD . . . 1

1.2 Aims of the Project . . . 3

2 Methods 4

2.1 Sample Data . . . 4

2.2 Pre-processing . . . 7

2.3 Stacks . . . 8

2.3.1 Running Stacks . . . 8

2.3.2 Parameter settings . . . 10

2.4 PyRAD . . . 11

2.4.1 Running PyRAD . . . 11

2.4.2 Parameter settings . . . 12

2.5 ipyrad . . . 13

2.5.1 Running ipyrad . . . 14

2.5.2 Parameter settings . . . 15

2.6 dDocent . . . 15

2.6.1 Running dDocent . . . 17

2.6.2 Parameter settings . . . 18

2.7 Determination of runtime and memory requirements . . . 19

2.8 Comparison of loci sequences . . . 19

2.9 Tree generation . . . 20

2.10 R-Environment . . . 21

2.11 fastStructure 23

3 Results **25**

3.1 RAD analysis pipelines 25
3.1.1 Stacks 26
3.1.2 PyRAD 29
3.1.3 rtd 30
3.1.4 Rainbow 31
3.1.5 ipyrad 32
3.1.6 dDocent 33
3.1.7 Features 35
3.1.8 Support 35
3.2 Results of Stacks analysis 36
3.3 Results of PyRAD analysis 37
3.4 Results of ipyrad analysis 37
3.5 Results of dDocent analysis 37
3.6 Estimated size limits for data sets on an ordinary laptop 43
3.7 Overlap of the results 44
3.7.1 Comparison of the recovered loci 46
3.7.2 Comparison of species trees 50
3.7.3 fastStructure Results 55

4 Discussion **58**

4.1 Installation 59
4.2 Support 60
4.3 Pipeline execution 60
4.4 Pipeline Results Overlap 62
4.4.1 InDel handling 62
4.4.2 Handling of PE-reads 63
4.4.3 Structure analysis 63
4.4.4 Simulated data 63
4.4.5 Empirical data 65

5 Conclusion **69**

Bibliography **71**

Appendix A **78**

A.1 Pipeline Parameters 78

A.1.1 Example dDocent config file 83

A.2 Venn-Diagrams 87

A.3 Phylogenetic Trees 89

A.4 Distruct Graphics 94

A.5 Structure-File 97

A.6 Phylip-File 98

1 Introduction

The genotype is the genetic makeup of an individual or a specific gene. An efficient determination of the genotype is crucial to many studies such as clinical diagnostics and genotype-phenotype association. Sequencing the whole genome of a large number of individuals is often not possible due to limited resources. But a large number of individuals is needed to achieve high statistical power. Hence genetic markers are used to analyse the genome of an individual [1]. Often single nucleotide polymorphisms (SNPs) are used as genetic markers. RAD-seq offers an efficient way to find and determine such genetic markers.

1.1 RAD and ddRAD

Restriction site associated DNA (RAD) tags are short DNA sequences, which neighbour a commonly used restriction endonuclease recognition site. A restriction endonuclease recognises a specific short DNA-sequence (just a few bases long) and cuts the DNA-molecule at that position. The recognition site depends on the used restriction endonuclease. If a RAD tag is associated with sequence polymorphisms, it can be used as a genetic marker and is called RAD marker. With next generation sequencing, the sequencing of RAD tags can be massively parallelized, this is called RAD sequencing (RAD-seq) [2]. In general is RAD-seq designed for interrogating 0.1 to 10% of a genome [3]. Thereby RAD-seq allows to massively lower the sequencing efforts and cost. A good example is the study of Antarctic krill populations [4]. In that study more than 95 Antarctic Krill samples with a genome of the size of 50 Gb were sequenced. 6.8M 100bp Illumina Reads per sample were generated by using the restriction endonuclease *SbfI*. The Illumina Reads were analysed for variant

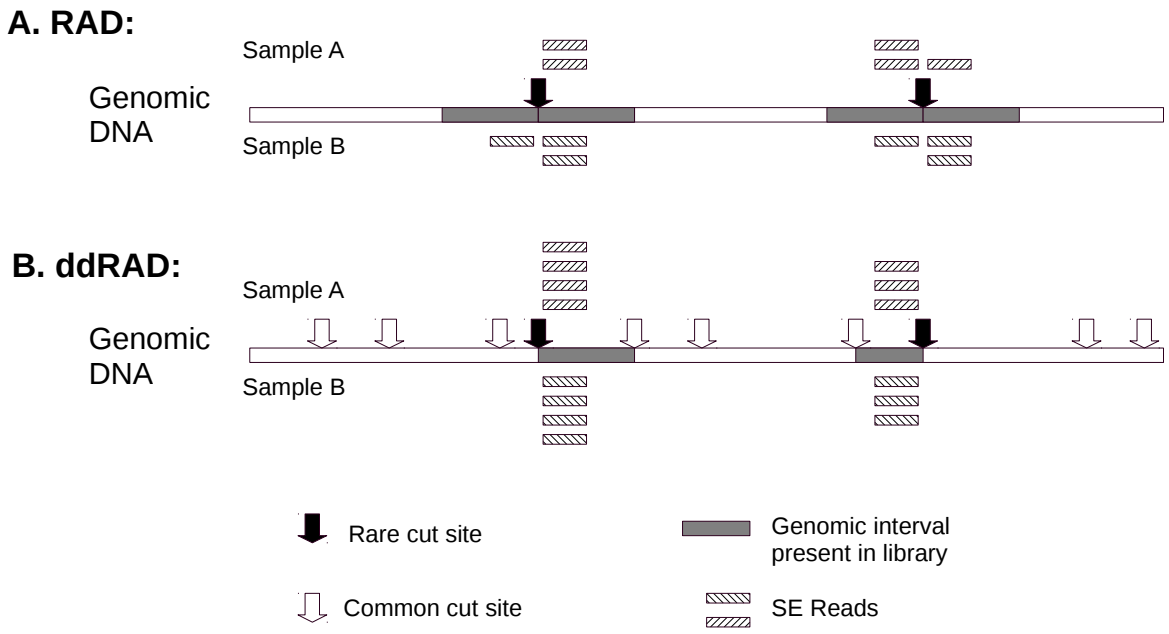

Figure 1.1: Fragment generation with restrictions enzymes. For RAD-seq(A) only one restriction enzyme is used, while ddRAD-seq(B) uses two. Precise and repeatable size selection offers a better fragment select based on the chosen size-selection range in case of ddRAD-seq (Image adapted form [1]).

sites and filtered so that a core set of 2.1k SNPs remained for further genomic analyses. Sequencing the whole genome of an Antarctic Krill sample would cost about $28,000 if we assume a coverage of 40 and sequencing costs of $14 per 1 Gb, which were the sequencing costs in October 2015 [5]. RAD-seq in comparison amounts to only $9.52 per sample (based on the 680 Mb generated for a sample).

Double Digest Restriction Associated DNA (ddRAD) Sequencing

In the case of ddRAD two different restriction enzymes are used. The resulting fragments are cut with the restriction enzymes at both sites (Figure 1.1). This eliminates the random shearing and end repair of the DNA. The ddRAD-seq also allows to only select fragments of a certain length.

Combining precise and repeatable size selection with sequence-specific fragmentation offers two further advantages. The first one is that the probability of sampling both directions from the same restriction site is low, because only a small fraction of the fragments will fall in the chosen size-selection range. This reduces sampling of immediately neighbouring regions and thereby only half as many reads are required to reach high-confidence sampling of a SNP associated with a restriction enzyme cut site [1]. The second advantage is that

fragments with the same size from different samples can be expected to come from the same genomic region. These advantages make ddRAD-seq more efficient and robust against under-sampling than RAD-seq and offers at least a five fold reduction of library preparation cost [1].

RAD and ddRAD pipelines are used for the determination of the genotype of individuals or samples. There are several RAD and ddRAD analysis pipelines available such as Stacks [6], PyRAD [7], rtd [1], ipyrad [8], dDocent [9] and Rainbow [10]. In section 3.1 a short overview of the workflow of these pipelines is given.

1.2 Aims of the Project

The overall goal of this master thesis is to get an overview over existing ddRAD analysis pipelines and perform a comparison. Specifically, the following points should be achieved:

1. Getting acquainted with the terminology and workflow of ddRAD experiments and analysis.
2. Perform an literature research on existing ddRAD analysis pipelines.
3. Setting up a test system with selected pipelines
4. Test the pipelines with simulated and empirical data sets.
5. Compare the pipelines in terms of
    a) Ease of set-up and usage
    b) Number of overall recovered loci
    c) Number of usable loci
    d) Runtime
    e) Memory requirements
6. Comparison of the overlap of usable loci
7. Comparison of the phylogenetic results
8. Comparison of the population structure results
9. Determine if these pipelines allow to analyse a large number of samples on ordinary PC or laptop.

2 Methods

The pipelines were installed on a laptop running Ubuntu 14.04 LTS (64bit) with 8GB RAM and an Intel Core i5-2410M CPU (2.30GHz x 4).

2.1 Sample Data

For this project, both simulated and empirical data was used. The simulated data was taken from the PyRAD tutorials [11]. The tutorials the data was taken from are SE (single-end) RAD Tutorial v.3.0 [12], SE ddRAD Tutorial v.3.0.4 [13], PE (paired-end) ddRAD Tutorial v.3.0.4 [14] and PE ddRAD w/ merged reads Tutorial v.3.0.4 [15].

The simulated data sets comprise 12 individuals each. Two of the data sets are single-ended and two are paired-ended. One of the PE data sets contains overlapping forward and reverse reads. (Table 2.1). The phylogenetic tree of the 12 individuals is presented in SE ddRAD Tutorial v.3.0.4 [13]. The 12 individuals form 3 populations of 4 individuals each (Figure 2.1).

The empirical data sets were retrieved from the NCBI sequence read archive (SRA) and came from the studies SRP035472 (Ber-SE) [16, 18, 19] and SRP068035 (Seb-PE) [17, 20]. Both sets are ddRAD data sets. Because of the limited computational resources, only subsets of the empirical datasets were used.

In focus of the Ber-SE study is the plant species *Berberis alpina*. Additionally to the *B. alpina* the genome of the species *B. moranensis*, *B. trifolia* and *B. pallida* was sequenced using single-end ddRAD-seq. In total, the study comprises 96 individuals. The majority of this

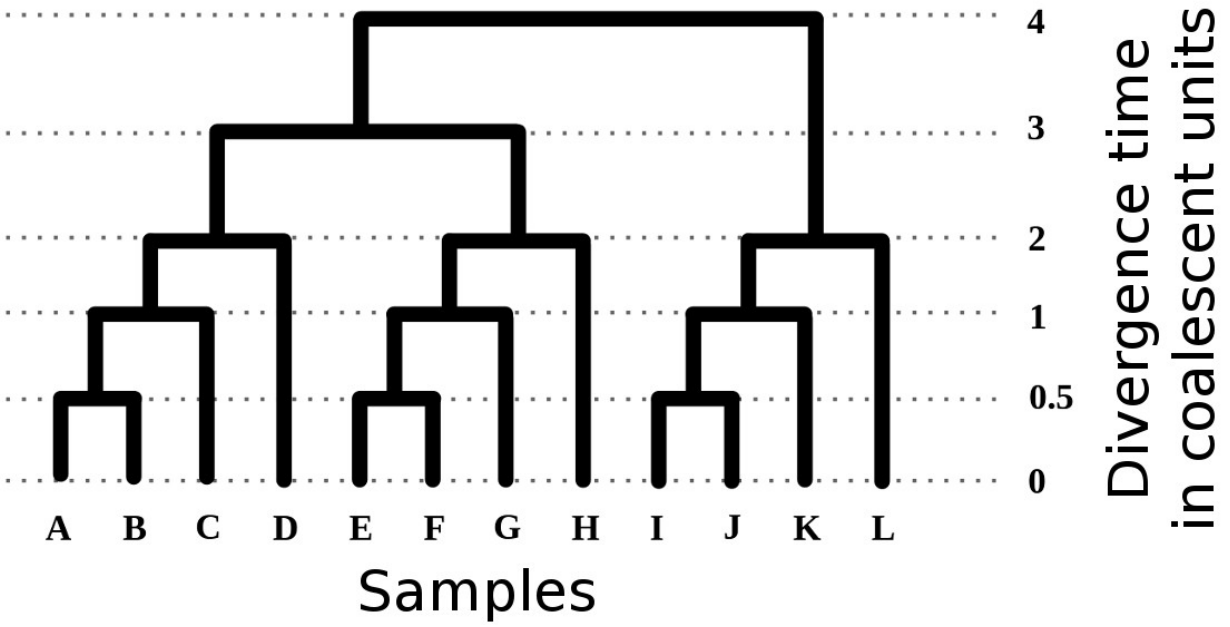

Figure 2.1: Species tree of the simulated data. This figure is adapted from the Species tree presented in SE ddRAD Tutorial v.3.0.4 [13] and shows the phylogenetic relation between the 12 samples.

individuals belonged to the *B. alpina* species [18, 19]. For this study, a subset of 8 individuals was used (Tables 2.2 and 2.3).

The Seb-PE study was conducted to characterise the sex determination system in the sister species *Sebastes chrysomelas* and *S. carnatus*. They performed paired-end ddRAD-seq of the genomic DNA from 40 individuals [20]. The subset used for this study only contained 8 individuals (Tables 2.2 and 2.3).

Additional analyses were performed to test how computational limitations affect the pipelines. To this end, additional samples of the Seb-PE data set were used to determine the maximum number of samples or reads the pipelines can analyse on the used hardware. Alternatively, it was investigated if it is possible to perform an analyses on all 40 samples of the SRP068035 (Seb-PE) data set. For both the extended Seb-PE and the reduced Seb-PE data set, the same pipeline parameters were used. The extended Seb-PE data set was only used to determine the limitations of the pipelines and hardware. No further comparison between the results of these analyses were performed.

Table 2.1: Summary of the simulated data sets.

Name	Abbreviation	Datatype	#Samples	#Reads	Read Length	Restriction Enzymes	Restriction overhang	Expected # of loci	Expected # of pop.	Reference
SE RAD Tutorial v.3.0	SE-RAD	single-end RAD	12	480,000	100	PstI	TGCAG	2,000	3	[12]
SE ddRAD Tutorial v.3.0.4	SE-ddRAD	single-end ddRAD	12	240,000	100	PstI, MluCI	TGCAG, AATT	1,000	3	[13]
PE ddRAD Tutorial v.3.0.4	PE-ddRAD	paired-end ddRAD	12	480,000	2×100	PstI, MluCI	TGCAG, AATT	$2 \times 1,000$	3	[14]
PE ddRAD w/ merged reads Tutorial v.3.0.4	PE-ddRAD_m	paired-end ddRAD	12	81,613	2×100	PstI, MluCI	TGCAG, AATT	340	3	[15]

Table 2.2: Summary of the empirical data sets.

SRA Study	Abbreviation	Genus	Datatype	#Samples total	#Samples used	Read Length	Restriction Enzymes	Restriction overhang	Reference
SRP035472	Ber-SE	*Berberis*	single-end ddRAD	96	8	83	EcoRI-HF, MseI	AATTC, TAA	[16]
SRP068035	Seb-PE	*Sebastes*	paired-end ddRAD	40	8	forward: 96 reverse: 101	SphI, MluCI	CATGC, AATT	[17]

Table 2.3: Summary of the subsets of the empirical data sets Ber-SE and Seb-PE.

Project	RUN	Species	#Reads
Ber-SE	SRR1121343	*Berberis alpina*	457,256
	SRR1121345	*Berberis moranensis*	387,102
	SRR1121355	*Berberis alpina*	432,639
	SRR1121359	*Berberis alpina*	409,883
	SRR1121364	*Berberis alpina*	527,204
	SRR1121369	*Berberis alpina*	158,508
	SRR1121370	*Berberis alpina*	450,469
	SRR1121373	*Berberis alpina*	262,034
Seb-PE	SRR3091916	*Sebastes chrysomelas*	$2 \times 1,241,311$
	SRR3091918	*Sebastes carnatus*	$2 \times 1,261,765$
	SRR3091926	*Sebastes carnatus*	$2 \times 937,847$
	SRR3091928	*Sebastes carnatus*	$2 \times 930,441$
	SRR3091930	*Sebastes carnatus*	$2 \times 1,098,500$
	SRR3091938	*Sebastes chrysomelas*	$2 \times 1,249,476$
	SRR3091940	*Sebastes chrysomelas*	$2 \times 922,310$
	SRR3091941	*Sebastes chrysomelas*	$2 \times 1,352,825$

2.2 Pre-processing

The data sets PE-ddRAD_m and Seb-PE were processed by PEAR [21] (Paired-End reAd mergeR) before they were put through the pipelines. PEAR merges overlapping paired-end reads. It takes as input the forward and reverse reads of a paired-end data set and produces four output files. The four files contain assembled reads (forward and reverse read were merged), discarded reads and the two files for unassembled forward reads and unassembled reverse reads. In case of the PE-ddRAD_m data set only the assembled reads were used for the analyses. In case of the Seb-PE data set the assembled reads were used for analyses like those in the PE-ddRAD_m data set and the unassembled were used for analyses like those in the PE-ddRAD.

With PEAR, the user can set the minimum length for the merged/assembled reads. This

means that two reads only get merged when the resulting read is longer or equally long than the specified minimum length. The minimum length was set to 100 for the PE-ddRAD_m data set and 101 for the Seb-PE data set. The minimum length was chosen based on the length of the reads, because in case of double digested RAD tags it would not make sense if merged/assembled reads end up shorter than the original forward and reverse reads. So the minimum length was set to the length of the reverse reads, because they are longer than the forward reads.

The non-overlapping paired-end forward and reverse reads from the data sets PE-ddRAD and Seb-PE (only the unassembled reads) were concatenated into one file and processed like single-end reads. The Seb-PE assembled reads will further be noted as the data set Seb-PE_m and the concatenated unassembled reads of the Seb-PE data set will simply be noted as Seb-PE.

Reads of the merged data sets had to be trimmed to the same length because Stacks can only perform analysis on data sets with a constant read length within each sample [22]. This was done with the Stacks script `process_shortreads` (see section 2.3.2). The trimmed data sets were processed by Stacks, PyRAD, ipyrad and dDocent and untrimmed data sets were only processed by PyRAD, ipyrad and dDocent because these pipelines can handle reads with varying lengths.

2.3 Stacks

Stacks version 1.41 was used, because it was the most recent version at the time the project was started. It was downloaded from [23].

Stacks comes with a web interface, which presents the results of the Stacks pipeline. To use the web interface Apache, MySQL, PHP and Perl are required.

2.3.1 Running Stacks

Stacks is designed modularly, which modules need to be used depends on the type of analysis that is performed. There are four sets in which the modules can be grouped.

Raw Reads: These are programs, which are used to clean and filter raw sequence data.

Core: The Core programs can be seen as the main Stacks pipeline. They have the task to build loci (ustacks), create the catalog of loci (cstacks) and match samples against the catalog (sstacks). The Core also includes programs for genetic mapping or population genomics analysis.

Execution control: These programs run the whole pipeline by running the Stacks components individually.

Utilities: The utilities are programs for things like: indexing database, exporting Stacks data and building paired-end contigs.

Before the pipeline is run a few preparations need to be done. If Stacks is used with the web interface, you need to create a MySQL database:

```
mysql -e "CREATE DATABASE <database_name>"
mysql <database_name> < /usr/local/share/stacks/sql/stacks.sql
```

<database_name> needs to be replaced by a proper name for the database. However the name of the database should end with "_radtags", this makes it easy to set up database permissions. Note: Without the suffix "_radtags" the database is not visible on the main page of the web interface. (This step can be skipped if Stacks is used without the web interface and the results do not need to be saved in a database)

There also might be some preparations for the sample data necessary. `process_radtags` needs to be executed if it is raw data, that needs to be demultiplexed and cleaned. Otherwise, if your data is already cleaned you can skip this step.

The pipeline is run by executing the `denovo_map` script, or if the sample data is mapped against a reference genome by executing the `ref_map` script. These scripts execute the individual steps (like eg. ustacks, cstacks and sstacks) of the pipeline in the right order for the given sample data.

2.3.2 Parameter settings

The simulated data sets were not demultiplexed and therefore needed to be processed by `process_radtags`. The `process_radtags` parameters (Table A.2) depend on the data set (single/paired end and restriction enzymes).

The most important parameters [24] for the `denovo_map` script are:

- Minimum stack depth of coverage (-m)
  The minimum number of raw reads required to form an exactly matching stack.
- Distance Allowed Between Stacks (-M)
  The maximum number of nucleotides that may differ between two stacks in order to merge them.
- Distance Between Catalog Loci (-n)
  Loci in the catalog get merged into one locus, if they have a smaller distance (distance = number of different nucleotides) than the "Distance Between Catalog Loci" number.

There are also some other `denovo_map` parameters. Those which were used for the analysis are summarised in Table A.1.

For all data sets the `denovo_map` parameters `-m 6 -M 13 -N 15 -n 15` were used.

Since Stacks can only perform analysis on datasets with a constant read length within each sample [22] and the results of the PEAR merges do differ in their read length, these reads must be trimmed to the same length. This trimming is done by `process_shortreads`. The parameters of `process_shortreads` were `-r -c -q and -t <min_length>`. For `<min_length>`, the same value as the minimum length in PEAR was set. Analyses of the trimmed data sets were also performed with the other pipelines.

2.4 PyRAD

PyRAD version 3.0.66 was used for this project, because it was the latest version at the time the project was started. It was downloaded from [11].

PyRAD needs two additional programs MUSCLE [25] and VSEARCH [26] (or USEARCH [27]) and also two common Python packages Numpy [28] and Scipy [29].

2.4.1 Running PyRAD

Before the pipeline can be run a parameter file must be created. This can be done by the following command:

```
pyrad -n
```

This creates a parameter file with the name params.txt. This file can be edited with any text editor. For a further description of the parameter file see section 2.4.2.

The PyRAD pipeline consists of 7 individual steps. These steps are all executed in succession when using the command:

```
pyrad -p params.txt
```

The steps can also be performed individually by using the parameter `-s`. For example with the following command only the first step is performed:

```
pyrad -p params.txt -s 1
```

The `-s` parameter can also be used to execute more than one step:

```
pyrad -p params.txt -s 234567
```

This command performs the steps 2-7, which is useful if one is working with already demultiplexed data sets since the first step just demultiplexes the raw sample data.

2.4.2 Parameter settings

For PyRAD all parameters are in the parameter file (created via `pyrad -n`). Overall there are 37 different parameters. In the parameter file there is one parameter per line and the line also contains a short description (including the affected steps) for the parameter. The first 14 Parameters are required and the others are optional. A description of the parameters is summarised in the appendix (Table A.3).

The parameters, which were the same for all data sets are shown in Table 2.4.

Table 2.4: Parameters for the PyRAD analysis. These parameters were used for all analyses.

Parameter	Value	Description
7	4	Number of processors
8	6	Minimum coverage for a cluster
9	4	Maximum number of low quality sites
10	.85	Clustering threshold
12	1	Minimum taxon coverage
13	3	Maximum shared polymorphic sites
30	*	Output formats

Additional parameters for the analyses are shown in Table 2.5.

Table 2.5: PyRAD parameters for the individual analyses.

Data set	Parameter			
	6 (Restriction cutsite overhang)	11 (Datatype)	18 (Path to demultiplexed data)	24 (Maximum of heterozygous sites)
SE-RAD	TGCAG	rad		8
SE-ddRAD	TGCAG,AATT	ddrad		10
PE-ddRAD	TGCAG,AATT	ddrad	location of concatenated forward and reverse reads	10
PE-ddRAD_m trimmed	TGCAG,AATT	ddrad	location of merged reads	10
PE-ddRAD_m untrimmed	TGCAG,AATT	ddrad	location of merged reads	10
Ber-SE	AATTC,TAA	ddrad	location of already demultiplexed sample data	10
Seb-PE_m trimmed	CATGC,AATT	ddrad	location of assembled data	10
Seb-PE_m untrimmed	CATGC,AATT	ddrad	location of assembled data	10
Seb-PE	CATGC,AATT	ddrad	location of concatenated data	10

2.5 ipyrad

The ipyrad version 0.5.15 was used for this project, because it was the latest version at the time the project was started. It was downloaded from [8].

ipyrad requires additional Python packages and executables (Table 2.6).

Table 2.6: Summary of the ipyrad dependencies. ipyrad depends on third party software and Python packages.

Python Packages		Executables	
package-name	reference	software-name	reference
Numpy	[28]	VSEARCH	[26]
Scipy	[29]	MUSCLE	[25]
Pandas	[30]	bwa	[31]
Sphinx	[32]	SMALT	[33]
ipyparallel	[34]	samtools	[35]
jupyter	[36]	bedtools	[37]
Cython	[38]	hdf5	[39]
H5py	[40]	mpich	[41]
Toyplot	[42]	-	-

2.5.1 Running ipyrad

The ipyrad pipeline is very similar to run as the PyRAD pipeline. First a parameter file has to be created:

```
ipyrad -n <assembly_name>
```

`<assembly_name>` must be replaced with the chosen name for this assembly. This command creates a parameter file called `params-<assembly_name>.txt`, which holds all parameters for the given assembly. For further details on the parameter file see section 2.5.2.

To run all 7 individual steps of ipyrad the following command can be used:

```
ipyrad -p params-<assembly_name>.txt -s 1234567
```

The `-s` parameter allows to choose which of the steps should be preformed. For example the following command only runs the frist two steps:

```
ipyrad -p params-<assembly_name>.txt -s 12
```

The `-r` parameter can be used to show a summary of the results of the analysis.

```
ipyrad -p params-<assembly_name>.txt -r
```

With the -b parameter a new branch can be created. Branching can be used to efficiently assemble multiple data sets under a range of parameter settings. If an analysis is repeated with different parameters, a new branch should be created. Only the steps whose results are affected by the changed parameters must be executed on the new branch. Because the new branch can use the data of the previous steps form the original branch. The following commands show an example where after the first two steps a new branch is created.

```
ipyrad -n <assembly_name>
ipyrad -p params-<assembly_name>.txt -s 12
ipyrad -p params-<assembly_name>.txt -b <newbranch_name>
##Edit the newly created parameter file params-<newbranch_name>.txt using a
text editor
ipyrad -p params-<assembly_name>.txt -s 34567
ipyrad -p params-<newbranch_name>.txt -s 34567
```

2.5.2 Parameter settings

All parameters for ipyrad are saved in a parameter file. There are a total of 28 parameters. Details about the individual parameters can be found in the appendix (Table A.4).

In Table 2.7 all non default parameters for the different analyses are shown.

2.6 dDocent

The dDocent version 2.2.10 was used for this project, because it was the latest version at the time the project was started. It was downloaded from [43].

dDocent relies heavily on third party software and therefore provides a script which installs all the required software. The programs dDocent depend on are summarized in Table 2.8.

Table 2.7: All non default parameters for the different analysis in ipyrad.

Data set	Parameter	Value	Data set	Parameter	Value
SE-RAD	2	Location of raw FASTQ files	Ber-SE	4	Location of already demultiplexed sample data
	3	Location of the barcode file		7	ddrad
	9	4		8	AATTC, TAA
	21	1		9	4
	24	3		20	10,10
	27	l, p, s, v, k		21	1
SE-ddRAD	2	Location of raw FASTQ files		24	3
	3	Location of the barcode file		27	l, p, s, v, k
	7	ddrad	Seb-PE_m (trimmed and untrimmed)	4	Location of assembled data
	8	TGCAG, AATT		7	ddrad
	9	4		8	CATGC, AATT
	21	1		9	4
	24	3		20	10,10
	27	l, p, s, v, k		21	1
PE-ddRAD	4	Location of concatenated forward and reverse reads		24	3
	7	ddrad		27	l, p, s, v, k
	8	TGCAG, AATT	Seb-PE	4	Location of concatenated data
	9	4		7	ddrad
	21	1		8	CATGC, AATT
	24	3		9	4
	27	l, p, s, v, k		20	10,10
PE-ddRAD_m (timmed and untrimmed)	4	Locationof merged reads		21	1
	7	ddrad		24	3
	8	TGCAG,AATT		27	l, p, s, v, k
	9	4			
	21	1			
	24	3			
	27	l, p, s, v, k			

Table 2.8: Summary of the required software needed to execute dDocent.

Required Software			
Software	Reference	Software	Reference
FreeBayes	[44]	Rainbow	[10]
vcflib	[45]	gnuplot	[46]
Trimmomatic	[47]	seqtk	[48]
mawk	[49]	cd-hit	[50]
bwa	[31]	bedtools	[37]
samtools	[35]	bamtools	[51]
VCFtools	[52]	GNU-parallel	[53]
PEAR	[21]		

2.6.1 Running dDocent

The raw data has to be pre-processed before dDocent can perform the analysis. The raw reads must be demultiplexed and the input files must follow a strict naming policy so that dDocent can use them. The dDocent homepage [43] suggests to use `process_radtags` from Stacks pipeline for the demultiplexing but other demultiplexing software can be used as well. The demultiplexed data must have one (SE) or two (PE) gzipped FASTQ file(s) for each individual.

The files must be named following the naming convention. The name consists of a population identifier and an individual identifier. The two identifiers must be separated by an '_'. In short the files have to look like `<population identifier>_<individual identifier>.F.fq.gz` for the forward reads and `<population identifier>_<individual identifier>.R.fq.gz` for the reverse reads where `<population identifier>` and `<individual identifier>` are replaced with their population and individual identifiers.

dDocent can be run simply by using the following command, once the data is prepared:

```
dDocent
```

When that command is executed a dialogue starts which asks the user for the values of the different parameters one by one.

Alternatively the pipeline can also be executed with a configuration file:

```
dDocent <config.file>
```

The configuration file must follow a specific format (Appendix A.1.1). If dDocent is executed with a configuration file only two dialogue inputs are necessary. For the first the software presents the number of unique sequences in relation to their coverage (counted within individuals) and the user can define a cutoff value for the minimum coverage. For the second dDocent presents the number of remaining unique sequences in relation to the coverage of the individuals and the user can again choose a cutoff value.

2.6.2 Parameter settings

dDocent was run with a configuration file for each analysis. The configuration file must follow a specific format (example in Appendix A.1.1). Most of the parameters were the same for all analyses and are presented in Table 2.9. Only the Trimming option was set to 'yes' for the data sets Ber-SE, Seb-PE_m trimmed, Seb-PE_m untrimmed and Seb-PE.

Table 2.9: Parameters used for all dDocent parameters. The 'Trimming' parameter was set to 'yes' for the empirical data sets.

Parameter	Value
Number of Processors	4
Maximum Memory	0
Trimming	no; yes
Assembly	yes
Type_of_Assembly	SE
Clustering_Similarity	0.85
Mapping_Reads	yes
Mapping_Match_Value	1
Mapping_MisMatch_Value	3
Mapping_GapOpen_Penalty	5
Calling_SNPs	yes
Email	emilian.jungwirth@student.tugraz.at

Two more parameters are needed to run dDocent which are entered via the command line dialogue. The first parameter is the minimum coverage of unique sequences (counted within individuals) which was set to 6. The second parameter is the minimum number of individuals where a sequence must be present which was set to 1.

The data sets were pre-processed in the same way as for the Stacks analyses, because the data had to be pre-processed and the dDocent documatation suggests `process_radtags`.

2.7 Determination of runtime and memory requirements

The runtime and memory requirements of the analyses are both determined with the Linux command `time` (`/usr/bin/time`). The `time` command keeps track of the time and memory a program requires. The output of the `time` command consist of various time values such as elapsed real time between invocation and termination, user CPU time and system CPU time. In addition, some memory and I/O statistics are presented. In this study, only the elapsed real time (runtime) and the maximum resident set size (maximum memory requirements) were documented for each analyses.

2.8 Comparison of loci sequences

Stacks and dDocent create a consensus sequence for every locus, while PyRAD and ipyrad represents loci differently, they shows a consensus sequence for every sample, where the locus is present. To compare pipelines, a FASTA file for each pipeline was created, one containing the consensus sequences of Stacks, one containing the consensus sequences of dDocent, one containing only the first sequence of each locus of the PyRAD loci and another one, which contained only the first sequence of each locus of the ipyrad loci. This means that the consensus sequences and the first sample sequences may differ in their SNPs. Therefore BLASTN [54], which tolerates minor differences between two sequences, was used to compare the FASTA files. The BLAST searches were performed in both direction for all pipelines for example Stacks against a PyRAD database and vice versa. For BLAST the default parameters were used except for the parameter `-word_size` which was reduced to 14. This was necessary because otherwise matching sequences would not align because they differ in their SNPs. Beside form SNP differences there might also be differences at the other sites because PyRAD and ipyrad replace low quality sites with the wild card character 'N'. Stacks and dDocent do not replace low quality sites. Hence PyRAD and ipyrad sequences may differ at more sites than just the SNP sites from the sequences of Stacks and dDocent.

The BLAST search results were filtered so only matches with a minimum alignment length remain, to ensure that the matching loci align over more than just the minimum exactly matching 14 nucleotides. The minimum alignment length was set to approximately 80% of

the loci length. This results in a minimum alignment length of 63 for the Ber-SE data set and 72 for the other data sets.

From the results Venn diagrams were created, which show how much the results of the pipelines overlap. The Venn diagrams were created with the R package VennDiagram [55].

2.9 Tree generation

Phylogenetic trees were generated from phylip [56] files of the pipeline results. Examples on phylip files can be found in the appendix section A.6. In PyRAD and ipyrad you can specify phylip as the output format or just use the wildcard character * to generate all output formats (Tables tables A.3 and A.4).

A phylip file can be generates with Stacks using the following command:

```
populations -P <stacks_dir> -M <popmap> -b 1 -k -t 36 --phylip_var_all
```

<stacks_dir> was replaced with the path to the directory containing the Stacks files and <popmap> with the path to the population map file.

dDocent does not come with a function to generate a phylip file. The phylip file for dDocent was generated with a modified version of the python script `vcf_to_phylip.py` [57] from the output vcf file (`Final.recode.vcf`) of dDocent. But this only works for the empirical data sets, because dDocent does perform a basic SNP filtering which results in an empty or nearly empty `Final.recode.vcf` file for the simulated data sets. Apparently, the simulated data sets are too uniform so most of the loci get filtered [58]. This means that an unfiltered vcf file had to be created, which was done by the following commands:

```
samtools view -@1 -b -1 -L mapped.bed -o split.temp.bam cat-RRG.bam
samtools index split.temp.bam
freebayes -b split.temp.bam -t mapped.bed -v unfiltered.vcf -f reference.fasta
-m 5 -q 5 -E 3 --min-repeat-entropy 1 -V --populations popmap -n 10
```

The `vcf_to_phylip.py` script was modified to include variant sites which are longer than just one nucleotide and also variant sites which contain a single indel in the generated phylip

file. All necessary modification are in the `parse_vcf` function of the `vcf_to_phylip.py` script (Appendix Listing A.1).

The phylip files were further processed by the program RAxML [59], which generates a tree and saves it in a file using the newick tree format. The substitution model `GTRCAT` was specified. As algorithm, the default 'rapid hill-climbing' was used. The trees were finally plotted using the R package phytools [60].

Stacks was not able to create the phylip file for all loci of the Seb-PE data set, because it ran out of memory. Therefore the phylip file was created for only those loci which are present in all 8 samples. The other pipelines were able to create the phylip file for all loci, but for better comparability, the minimum sample coverage of the loci was also set to 8 samples.

The trees were also compared using the dist.topo function of the R package ape [61]. The dist.topo function returns the topological distance. By default, it uses the PH85 method [62] to determine the topological distance. The PH85 method defines the distance as twice the number of different bipartitions between the two trees and does not take the branch length into account.

The trees of the simulated data sets were compared to the reference tree presented in the SE-ddRAD PyRAD tutorial [13] (Figure 2.1) and the trees of the empirical data sets were compared to the trees of the other pipelines for the same data set.

2.10 R-Environment

The species trees and Venn diagrams were created using R, a framework for statistical analysis [63]. For this, R version 3.3.2 was used. The packages and functions used are summarised in Table 2.10.

Table 2.10: Species trees and Venn diagrams were generated with the following R packages and functions.

Package	Version	Used functions	Describtion	Non default parameters		Reference
				Paramter name	Value	
VennDiagram	1.6.17	draw.pairwise.venn	Used for creating a Venn diagram with two sets	area1, area2, cross.area, category	according to the data	[55]
				euler.d	FALSE	
				scaled	FALSE	
				cex	c(2,2,2)	
				cat.dist	c(0.09,0.09)	
				cat.cex	c(2,2)	
				print.mode	percent	
				sigdigs	5	
		draw.triple.venn	Used for creating a Venn diagram with two sets	area1, area2, area3, n12, n23, n13, n123, category	according to the data	
				euler.d	FALSE	
				scaled	FALSE	
				cex	c(2,2,2,2,2,2,2)	
				cat.dist	c(0.17,0.16,0.05)	
				cat.cex	c(2,2,2)	
				print.mode	percent	
				sigdigs	3	
gridExtra	l 2.2.1	grid.arrange	Used to create a proper margin around the Venn diagram plots	grobs	gTree(children=venn.plot)	[64]
				layout_matrix	rbind(c(NA,NA,NA), c(NA,1,NA),c(NA,NA,NA))	
				widths	c(3,15,3)	
				heights	c(3,15, 3)	
phytools	0.5.64	read.tree	To read in the output tree file of RAxML	file	RAxML output tree	[60]
		reroot	Used to reroot the tree and make it bifurcated	tree	MyTree (output of read.tree)	
				node.number	root_node (=MyTree$Nnode/2+ length(MyTree$tip.label))	
				position	MyTree$edge.length[which(MyTree$edge[,2] == root_node)]	
ape	4.0	plot	To plot the Tree	tree	rooted_tree (output of reroot)	[61]
				direction	downwards	
				type	phylogram	
				use.edge.length	TRUE	
				cex	1.5	
				show.node.label	TRUE	
				tip.color	tip.color=ifelse(rooted_tree$tip.label %in% species, "red","blue") (sample labels in species is an array of those sample labels which should be red)	
		dist.topo	To determine the topological distance between two trees	x	Tree one for the comparison (output of RAxML)	
				y	Tree two for the comparison (output of RAxML)	

2.11 fastStructure

The results of the pipelines were further used for inferring the population structure of the data sets. This is done with the variational framework fastStructure [65]. The version 1.0-4 of fastStructure was used, which was the most recent version at the time the project was started. fastStructure offers three Python scripts `structure.py`, `chooseK.py` and `distruct.py` for analysing the data sets. The `structure.py` script is the main script. It performs the actual structure analysis. To choose the appropriate number of populations the `chooseK.py` script can be used. It provides two different suggestions for the population number K. One is based on the marginal likelihood of the data and the other is the minimum number of populations that have a cumulative ancestry contribution of at least 99.99% [65]. For visualisation of the results the `distruct.py` script is used. The `distruct.py` script generates distruct plots [66] for the results of the `structure.py` script.

The input for the `structure.py` script must be either in the plink [67] bed format or the original Structure format. An Example for a Structure file can be found in the Appendix A.5. In PyRAD and ipyrad you can specify Structure as the output format or just use the wildcard character * to generate all output formats (Tables tables A.3 and A.4).

To generate a Structure file with Stacks the following command is needed:

```
populations -P <stacks_dir> -M <popmap> -b 1 -k -t 36 --structure
--write_single_snp
```

`<stacks_dir>` needs to be replaced with the path to the directory containing the Stacks files and `<popmap>` with the path to the population map file. Unfortunately it was again not possible to create a structure file of all loci for the Seb-PE data set with Stacks. So the structure file for Stacks was generated from only those loci which cover all 8 samples.

For dDocent, the plink bed file format was used. To create the plink files plink 1.9 [68] was used. Plink needs as input a vcf file and a reference FASTA file. dDocent generates a reference file which contains the consensus sequences of the loci and a vcf file (`Final.recode.vcf`). The vcf file `Final.recode.vcf` does not contain all variant sites, because dDocent performs a basic filtering step. This filtering step, unfortunately, causes problems for the simulated data sets, because it filters almost all SNPs. Therefore an unfiltered vcf file had to be created

for the simulated data sets (see section 2.9). The plink files were created with the following command:

```
plink -vcf <vcf-file> --reference <reference.fasta> --allow-extra-chr
```

Non default values and description of the parameters of the scripts are provided in Table 2.11.

Table 2.11: Values and description of the Parameters for the fastStructure scirpts `structure.py`, `chooseK.py` and `distruct.py`.

Script	Parameter	Value	description
`structure.py`	-K	Numbers from 1 to 8	Number of populations
	–input	Name of the structure output of the pipelines	Path to the input file(s)
	–output	Name of the output file	Path for the output files
	–full		To output all variational parameters
	–format	str or bed	The format of the input file. Either bed (default) or str
`chooseK.py`	–input	Name of the output file of `structure.py`	Path to the input file(s)
`distruct.py`	-K	Number of Population suggested by `chooseK.py`	number of populations
	–input	Name of the output file of `structure.py`	Path to the input file
	–output	Name of the output svg file	Path for the output file
	–popfile	File, which contains the names of the samples	File with known categorical labels

3 Results

3.1 RAD analysis pipelines

There are several analysis pipelines for RAD and ddRAD reads. Following is a short overview for the Stacks, PyRAD, rtd, Rainbow, ipyrad and dDocent pipelines.

A RAD and ddRAD analysis pipeline needs to perform certain general steps (Figure 3.1). The demultiplexing step does not necessarily be part of the pipeline because most of the data will already be demultiplexed at the beginning of the analysis.

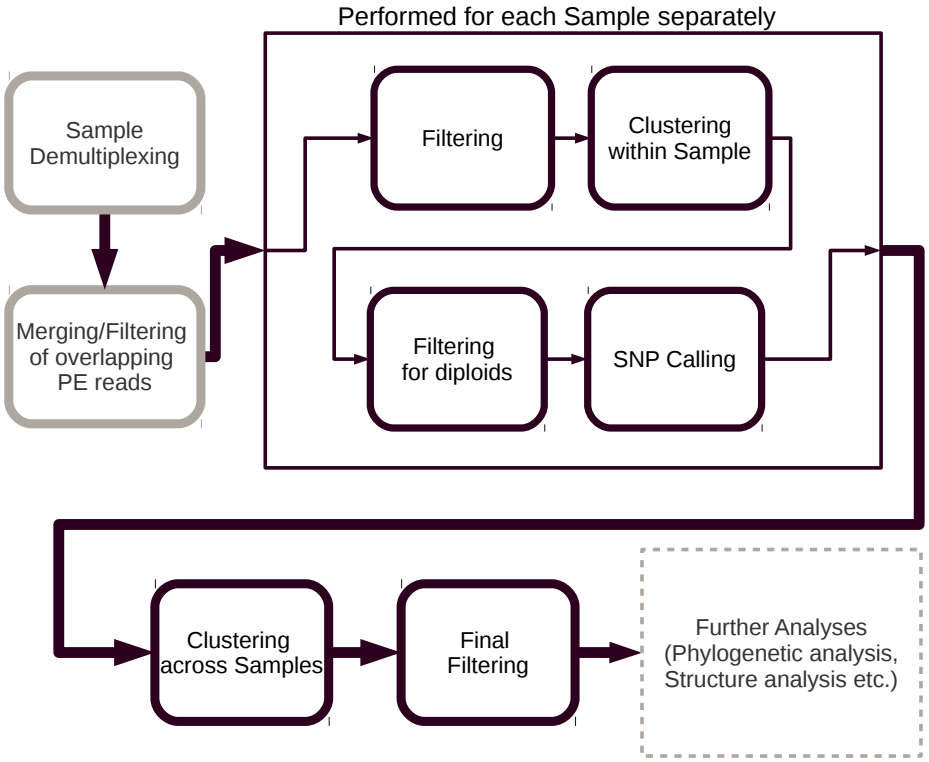

Figure 3.1: The flow diagram with the basic steps of RAD and ddRAD data analysis. Data is often already demultiplexed so the demultiplexing step show in grey can be skipped. The results of a RAD/ddRAD pipeline are used for further analyses which is shown with a grey block at the end of the pipeline.

3.1.1 Stacks

This section is a summary of the workflow of Stacks based on the paper "Stacks: Building and Genotyping Loci *De novo* From Short-Read Sequences" [6].

The way Stacks works is most easily explained with the schematic shown in Figure 3.2.

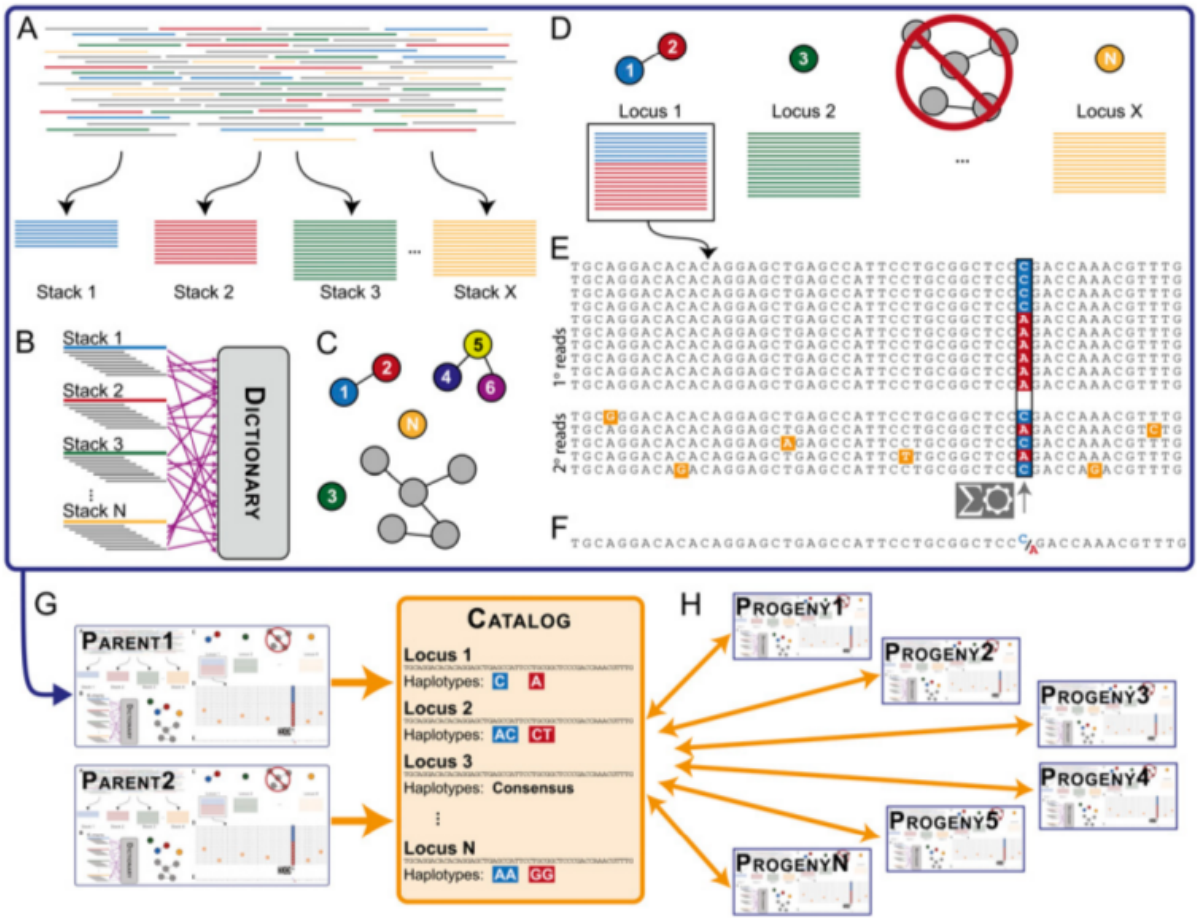

Figure 3.2: Schematic of the Stacks work flow. (A) Exactly matching stacks are created from reads of an individual. (B) These stacks get broken down into k-mers to create a dictionary. A list of potentially matching stacks can be created by again breaking down stacks into k-mers and querying them against the dictionary. (C) Matching stacks get merged to form putative loci. (D) Filtering of the putative loci based on their diplodidy and coverage. (E) Secondary reads (secondary reads are reads which could not form a proper stack in step A) are matched with the putative loci and the SNPs of the loci are identified. (F) A consensus sequence for each locus is created. The steps A to F needs to be performed on each individual separately. (G) A set/catalog of all possible loci is created from the outputs of the steps A to F of every individual. (H) Every individual gets matched against the catalog (Image taken from [6]).

Identifying stacks, inferring loci

- Creating the ustacks:

  The ustacks (unique stacks) program reads cleaned sequences from a FASTA or FASTQ file and creates exactly matching stacks (exactly matching means that each sequence in a stack is exactly the same). These exactly matching stacks are called unique stacks. Those unique stacks, that have fewer reads in them as a configurable threshold (stack-depth parameter), are disassembled and the reads are set aside. The reads in the unique stacks are called primary stacks and the reads, that are set aside, are called secondary reads.

  Furthermore, ustacks excludes those stacks, whose coverage is two standard deviations above the mean depth of coverage (and also those stacks that differ in just one nucleotide from those extremely deep stacks), because they usually represent repetitive sequences (Figure 3.2A).
- Creating a dictionary:

  Ustacks breaks the sequences of each stack into overlapping k-mers (fragments of the length k) and loads them into a dictionary. Each k-mer of a stack gets queried against the k-mer dictionary to find other stacks with matching k-mers (Figure 3.2B).
- Aligning the stacks:

  Stacks, which pair with a sufficient number of matching k-mers, get aligned. Some of the stacks may align with multiple other stacks. Aligned stacks get merged and represent putative loci. The putative loci are displayed as a graph where the nodes represent unique stacks and the edges are weighted by the nucleotide distance between the stacks (Figure 3.2C).
- Filtering for diploids:

  In a diploid genetic cross, only loci with one (homozygous) and two (heterozygous) stacks should exist. Merged stacks with more than three (three to allow some error) unique stacks get analysed by the deleveraging algorithm, which decides which subset of these stacks represent a locus. The deleveraging algorithm is also used for merged stacks, which have a coverage higher than two standard deviations above the mean (figure 3.2D).

- Matching with the secondary reads:
  Finally, the secondary reads get matched against the putative loci. Secondary reads that do not match any of the putative loci get discarded (Figure 3.2E).

Inferring alleles and haplotypes

- Identifying SNPs:
  To identify polymorphisms within a locus ustacks examines for each putative locus one nucleotide position at a time. Polymorphisms are detected via a maximum likelihood framework [69] (Figure 3.2E).
- Creating a consensus sequence:
  A consensus sequence for each locus is created by ustacks and the SNPs are recorded (Figure 3.2F).

Aggregating loci into a Catalog

- Stacks performs the steps from Figure 3.2A to F for each individual (e.g. two parents of a genetic cross, because the progeny of that cross will only have genes of the two parents). The cstacks (catalog stacks) program merges the output of ustacks into a catalog to create a set of loci that possibly appear in members of the population (Figure 3.2G).

Matching the population against the Catalog

- The sstacks (search stacks) program matches every individual in the cross against the catalog to determine the haplotypes at each locus in every individual (Figure 3.2H).

Calling mappable markers

- The program `markers.pl` characterizes the loci into 10 classes of mappable markers (Table 3.1).

Table 3.1: The 10 different mappable markers

Marker type	Individual 1	Individual 2	Number of alleles
ab/aa	Heterozygous	Homozygous	2
aa/ab	Homozygous	Heterozygous	2
ab/ab	Heterozygous	Heterozygous	2
aa/bb	Homozygous	Homozygous	2
ab/–	Heterozygous	Absent	2
–/ab	Absent	Heterozygous	2
ab/cc	Heterozygous	Homozygous	3
cc/ab	Homozygous	Heterozygous	3
ab/ac	Heterozygous	Heterozygous	3
ab/cd	Heterozygous	Heterozygous	4

3.1.2 PyRAD

The PyRAD Framework consists of seven sequential steps, which are summarized here based on "PyRAD: assembly of *de novo* RAD-seq loci for phylogenetic analyses" [7]:

1. De-multiplexing
    - In this step the sequences, which are in a (gzipped) FASTQ file, are divided into separated files by using the sample barcodes.
2. Quality filtering and removal of barcodes, cut sites and adapters
    - This step removes barcodes and Illumina adapters and discards low quality reads by using the quality scores of the bases. Bases with a score below a certain threshold are changed into "N"s. Reads that contain more "N"s than a user defined number are discarded.
3. Clustering within samples and alignment
    - Replicate sequences are merged into individual clusters, but the total number of occurrence is saved. The sequences are clustered by using either VSEARCH [26] or USEARCH [27] (Note that newer versions than usearch_v.7.0.1090 of USEARCH are not supported in PyRAD and in general, VSEARCH is recommended). VSEARCH creates clusters (stacks) by matching sequences to a seed

sequence or creating a new seed. The results of the clustering process are aligned with MUSCLE [25].

4. Joint estimation of error rate and heterozygosity
   - A maximum likelihood equation [70] is used to determine the mean heterozygosity and sequencing error rate.
5. Consensus base calling and paralog detection
   - This step creates the consensus sequences for each stack by using the error rate and heterozygosity.
     Stacks are discarded if they have:
     a) less than the minimum coverage,
     b) more than the allowed number of undetermined sites,
     c) more than the allowed number of heterozygosity sites,
     d) or more than the allowed number of alleles (two for diploid genomes).
6. Clustering across samples
   - The stacks resulting from step 5 are clustered across samples. For the clustering USEARCH/VSEARCH is used like in step 3.
7. Alignment across samples, filtering and formatting
   - The stacks get aligned like in Step 3.

3.1.3 rtd

This summary of the rtd algorithm is based on the paper "Double digest RAD-seq: an inexpensive method for *de novo* SNP discovery and genotyping in model and non-model species" [1].

First all identical sequences are collapsed into a single record, while the number of occurrences in each individual is retained. The resulting unique sequences are clustered by a graph-based distance approach and filtered by a novel "ploidy-aware" quality filter.

The pairwise distances are computed via BLAT [71], which allows the detection of InDel (insertion/deletion) regions. A MCL (Markov Cluster Learning) [72] graph clustering algorithm

is used to discover groups. The fraction of reads in each cluster reporting haplotypes beyond the ploidy of the organism is calculated to estimate if the clusters should be discarded. For a diploid organism, clusters where 90% of the reads are one of the two most frequent unique sequences in that cluster, are retained.

Finally MUSCLE [25] is used to align all sequences in a group.

Unfortunately could rtd not be included in the further comparison because it was not possible to execute the available rtd versions. rtd uses a Google Documents Spreadsheet to store data and uses the ClientLogin which is no longer supported by Google since April 20, 2012 [73] and leads to an error at the start of a analysis. To run rtd, a migration to OAuth 2.0 would be necessary.

3.1.4 Rainbow

The following summary is based on the paper "Rainbow: an integrated tool for efficient clustering and assembling RAD-seq reads" [10].

With the Rainbow algorithm, paired-end reads are clustered into groups. Paired-end sequencing means that a DNA-sequence is sequenced from both ends. In case of RAD the restriction site end is considered as the forward paired-end and the other end is considered as the reverse paired-end.

Rainbow mainly consists of the following steps:

1. Indexing
    - For the indexing of the tagged reads a spaced seeds hash table is used.
2. Primary clustering
    - All indexed reads are clustered into groups:
      A read is added to a group if it has less or equal mismatches than a certain number. Thereby all potential true clusters a clustered correctly by the primary clustering.

3. Top–down cluster division
    - A dividing module is used to find sequencing errors. Stacks are built from the primary clusters and every position gets scanned. At every position the occurrences of the minor base K and its frequency F is determined. The cluster is recursively divided at the most significant position (largest K and F below a minimum requirement). A guided tree is generated, which contains relationship information of the divided clusters.
4. Bottom–up merging process
    - The guided tree from step 3 is used to merge similar clusters in a bottom up manner. To better determine the similarity of two clusters the reverse paired-ends are used.
5. Local assembly of final clusters
    - For the assembling a greedy algorithm is used. All the final divided reads and their reverse read partners are pooled. The two reads with the largest overlap (overlap length and overlap similarity) are merged into a consensus sequence. This process is repeated until the thresholds for overlap length and overlap similarity is reached.

Rainbow was not used for further comparisons because it does not perform an actual SNP calling and would need further software to generate an output for a RAD and ddRAD analysis. But dDocent, which uses Rainbow, was used for the further analysis. So in a certain sense, Rainbow is part of the further comparisons.

3.1.5 ipyrad

This summary is based on the information presented at the ipyrad homepage [8].

ipyrad is a completely re-written version of PyRAD that has been optimized in terms of speed and flexibility. The workflow of the pipeline basically consists of the same seven sequential steps as PyRAD:

1. De-multiplexing / Loading FASTQ files

2. Filtering / Editing reads
3. Clustering / Mapping reads within Samples and alignment
4. Joint estimation of heterozygosity and error rate
5. Consensus base calling and filtering
6. Clustering / Mapping reads among Samples and alignment
7. Filtering and formatting output files

3.1.6 dDocent

The following summary is based on the paper "dDocent: a RADseq, variant-calling pipeline designed for population genomics of non-model organisms" [9] and the dDocent homepage [43].

The dDocent pipeline is written in BASH and depends largely on other bioinformatics software packages. It takes advantage of programs designed specifically for each task of the analysis. This also makes it possible to update each component separately.

1. **Data input requirements**
    - The data needs to be demultiplexed.
    - The data must be in gzipped FASTQ files.
    - The name of the files must follow a specific naming convention:
      `<population identifier>_<individual identifier>.F.fq.gz`
      for the forward reads and
      `<population identifier>_<individual identifier>.R.fq.gz`
      for the reverse reads.
    - To demultiplex different programs can be used such as the `process_radtags` script from the Stacks pipeline [6].
2. **Quality trimming**
    - Trimmomatic [47] is used for a quality filtering, which removes low quality bases and Illumina adapters, if they are present. Initially, dDocent [9] used TrimGalore [74], but this has been replaced by Trimmomatic in newer versions [43].

3. ***De novo* assembly**

   - In this step the reference sequences are assembled. Each reference sequence represents a consensus sequence of a locus. The *de novo* assembly can be skipped if a FASTA file with reference sequences is provided.
   - dDocent offers four different assembly methods (Figure 3.3):
     - **PE (Paired-End without a random shearing step; ddRAD and ezRAD) and RPE (Paired-End with a random shearing step; original RAD) methods:** The clustering program CD-HIT [50] and the assembly program Rainbow [10] are used to create the reference sequences. Each of these sequences represent one of the loci. To check for substantial overlap between the forward and reverse reads the program PEAR [21] is used.
     - **OL (Overlapping forward and reverse reads) method:** The data set is merged with PEAR before they it is clustered using CD-HIT.
     - **SE (Single end) method:** The data set is clustered using CD-HIT.
     - **Data reduction:** For the PE methods the reads are concatenated into one forward and one reverse FASTA file and for the SE method only the forward reads are used. First the program presents the user the number of unique sequences in relation to their coverage (counted within the individuals). The user defines a cutoff for the minimum coverage a sequence must have. The number of remaining unique sequences in relation to the coverage of the individuals is shown and a cutoff for the minimum coverage of the individuals must be defined.

4. **Read mapping**

   - Raw reads are aligned to the reference sequences using the MEM algorithm of BWA [31].

5. **SNP Calling**

   - Variant sites are called with FreeBayes [44] and saved into a single VCF (variant call format) file using VCFtools [52].

6. **SNP filtering**

   - dDocent performs a basic filtering so that only those SNPs which are called in at least 90% of all individuals remain. This is done by using VCFtools.

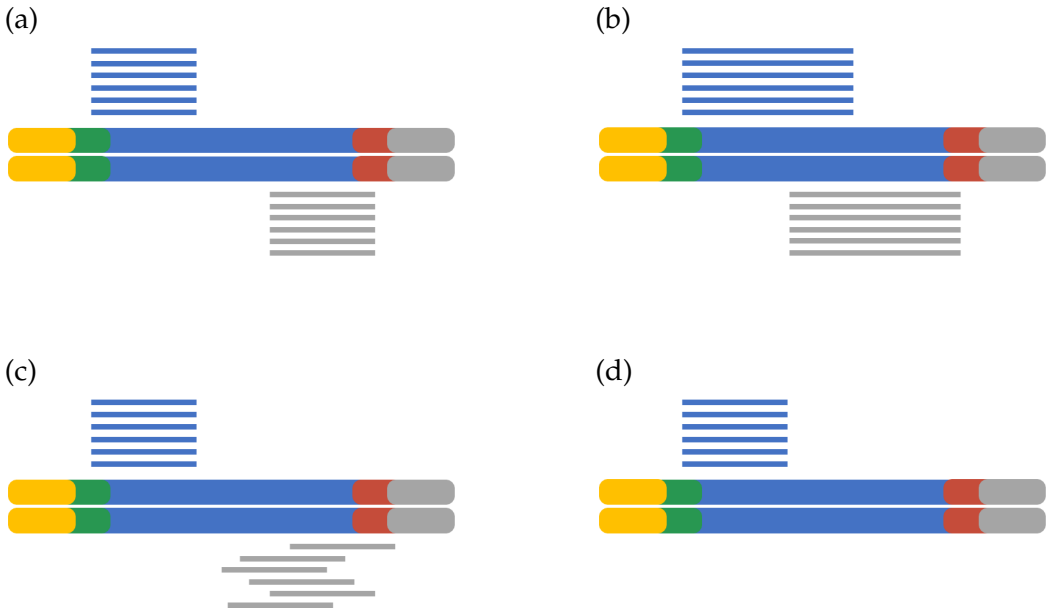

Figure 3.3: dDocent assembly methods: (a) PE method for PE data without a random shearing step (eg. ddRAD and ezRAD). (b) OL method for overlapping forward and reverse reads. (c) RPE method for PE data with a random shearing step (original RAD). (d) SE method for SE data. The thick lines represent the RAD sequences with Illumina adapters (yellow and grey), overhangs of the restriction enzymes (green and red) and RAD sequence (blue). The reads are represented by the thin lines. The forward reads are blue and the reverse reads are grey (Images taken from[43]).

3.1.7 Features

The pipelines provide different features. Some features like e.g. sample demutiplexing can be compensated with third party scripts and programs. dDocent only provides features for the core RAD analysis and needs additional programs to generate output formats like phylip or structure files. The pipelines Stacks, PyRAD and ipyrad provide more features and can generate different output formats. Stacks is the only pipeline which provides a web interface and a database, but it does not directly support PE reads [75] (Table 3.2).

3.1.8 Support

Stacks, PyRAD and dDocent have Google Groups as their support forum. ipyrad has a forum on its homepage but ipyrad support can also be found in the PyRAD Users Google Group (Table 3.3).

Table 3.2: Feature-list of the pipelines.

Features	Stacks	PyRAD	rtd	Rainbow	ipyrad	dDocent
Demultiplexing	✓	✓	✓	✖	✓	✖
PE/SE Support	✖	✓	✓	✓	✓	✓
Loci building	✓	✓	✓	✓	✓	✓
Filtering for diploids	✓	✓	✓	✖	✓	✓
SNP Calling	✓	✓	✓	✖	✓	✓
Parallelization	✓	✓	✓	✓	✓	✓
Creation of phylip files	✓	✓	✖	✖	✓	✖
Creation of structure files	✓	✓	✖	✖	✓	✖
Handling of reads with varying lengths	✖	✓	✓	✓	✓	✓
Web interface and Database	✓	✖	✖	✖	✖	✖

Table 3.3: Support forums of the analysis pipelines.

Pipeline	Support
Stacks	`https://groups.google.com/forum/#!forum/stacks-users`
PyRAD	`https://groups.google.com/forum/#!forum/pyrad-users`
ipyrad	`http://ipyrad.readthedocs.io/support.html`
dDocent	`https://groups.google.com/forum/#!forum/ddocent`
rtd	NA

3.2 Results of Stacks analysis

Important aspects of an RAD analysis are: the number of identified and usable loci, the run time and the memory requirements. Usable loci are those loci which contain a variant site (e.g. SNPs), because loci without any variant sites do not have any information for differentiation between the individuals.

Stacks recovered more loci than the other pipelines, but had relatively high memory footprints. The memory requirements for the simulated data sets were more or less constant with about 1,200 bytes per read or about 290 kB per locus. The memory footprint for the empirical data sets did vary a lot and reached the hardware limitation for the Seb-PE data set. Therefore the phylip and structure file of the Seb-PE data set were created using only

those loci which cover all 8 samples. Stacks is relatively fast compared to the other pipelines, although it was the slowest for the Seb-PE data set. It needed over 6 days to complete the Seb-PE analysis (Table 3.4).

Because the computational limits have already been reached with the reduced Seb-PE data set, no analysis on an extended Seb-PE data set with additional samples was performed.

3.3 Results of PyRAD analysis

PyRAD was for the most part the slowest pipeline. It recovered less loci than Stacks or dDocent, but more than ipyrad. For the simulated data set, PyRAD required the least memory off al pipelines. The memory requirements for the empirical data were also relatively low except for the extended Seb-PE data set were it needed 5.5 GB (Table 3.5).

It was possible to perform the ddRAD analysis with PyRAD for all 40 samples of the extended Seb-PE data set. The analysis needed about 2.4 days and 5.5 GB of RAM.

3.4 Results of ipyrad analysis

ipyrad was faster than PyRAD, but was slower than Stacks or dDocent. The memory requirements of ipyrad were higher than those of PyRAD for the simulated data sets. ipyrad recovered less loci than the other pipelines (Table 3.6).

It was possible to perform the ddRAD analysis with ipyrad for all 40 samples of the extended Seb-PE data set. The analysis needed about 1.5 days and 1.2 GB of RAM.

3.5 Results of dDocent analysis

dDocent was for the most part the fastest pipeline and also had a relatively low memory footprint. It also recovered more usable loci for the empirical data sets than the other pipelines. The memory footprint of dDocent was relatively constant for the simulated data

sets. dDocent needed 90 - 95 MB for each of the simulated data sets, but the memory footprint per read and per locus vary a lot between the data sets (Table 3.7).

It was possible to perform the ddRAD analysis with dDocent for all 40 samples of the extended Seb-PE data set. The analysis needed about 7.7 hours and 1.1 GB of RAM.

Table 3.4: Summary of the Results of Stacks. The total number of recovered loci, the number of usable loci, the needed time and maximum resident memory for each analysis is presented.

Data set	Number of			Time			Max Memory resident		
	Reads	found loci	usable loci	overall [s]	per read [μs]	per locus [ms]	overall [Mbytes]	per read [bytes]	per locus [kbytes]
SE-RAD	480,000	2,058	1,975	143	298	69	580	1,208	282
SE-ddRAD	240,000	1,019	985	60	250	59	287	1,196	282
PE-ddRAD	480,000	2,142	1,986	130	271	61	619	1,290	289
PE-ddRAD_m trimmed	81,613	348	332	22	270	63	105	1,287	302
Ber-SE	3,086,094	13,923	4,751	3,796	1,230	272	2,226	722	160
Seb-PE_m trimmed	531,665	893	377	37	70	41	176	331	197
Seb-PE	16,925,410	158,891	48,629	567,242	33,514	3570	7,290	431	46
Seb-PE 40 samples[1]	109,070,980	NA	NA	NA	NA	NA	NA	NA	NA

[1]The computation limitation with Stacks were already been reached with the Seb-PE data set.

Table 3.5: Summary of the Results of PyRAD. The total number of found loci, the number of usable loci, the needed time and maximum resident memory for each analysis is presented.

Data set	Number of			Time			Max Memory resident		
	Reads	found loci	usable loci	overall [s]	per read [μs]	per locus [ms]	overall [Mbytes]	per read [bytes]	per locus [kbytes]
SE-RAD	480,000	2,000	1,959	660	1,375	330	61	127	31
SE-ddRAD	240,000	1,000	982	295	1,229	295	45	188	45
PE-ddRAD	480,000	2,002	1,961	557	1,160	278	61	190	45
PE-ddRAD_m untrimmed	81,613	340	336	170	2,083	500	37	453	109
PE-ddRAD_m trimmed	81,613	340	332	165	2,022	485	35	429	103
Ber-SE	3,086,094	6,809	2,977	23,513	7,621	3,453	270	88	40
Seb-PE_m untrimmed	531,770	384	300	1,380	2,595	3,594	118	222	307
Seb-PE_m trimmed	531,665	456	330	1,400	2,633	3070	97	182	213
Seb-PE	16,925,410	107,623	47,761	33,545	1,982	312	1,498	89	14
Seb-PE 40 samples	109,070,980	144,522	80,172	209,000	1,916	1,446	5,556	51	38

Table 3.6: Summary of the Results of ipyrad. The total number of found loci, the number of usable loci, the needed time and maximum resident memory for each analysis is presented.

Data set	Number of			Time			Max Memory resident		
	Reads	found loci	usable loci	overall [s]	per read [μs]	per locus [ms]	overall [Mbytes]	per read [bytes]	per locus [kbytes]
SE-RAD	480,000	2,000	1,959	221	460	111	111	231	56
SE-ddRAD	240,000	1,000	982	143	596	143	109	454	109
PE-ddRAD	480,000	2,000	1,959	193	402	97	125	260	63
PE-ddRAD_m untrimmed	81,613	340	336	78	956	229	119	1,458	350
PE-ddRAD_m trimmed	81,613	340	332	68	870	209	118	1,446	347
Ber-SE	3,086,094	5,766	2,646	11,371	3,686	1,972	130	42	23
Seb-PE_m untrimmed	531,770	240	206	638	1,200	2,658	117	220	488
Seb-PE_m trimmed	531,665	284	201	632	1,189	2,225	117	220	412
Seb-PE	16,925,410	83,542	33,809	23,997	1,418	287	420	25	5
Seb-PE 40 samples	109,070,980	119,043	61,361	133,122	1,221	1.118	1,194	11	10

Table 3.7: Summary of the Results of dDocent. The total number of found loci, the number of usable loci, the needed time and maximum resident memory for each analysis is presented.

Data set	Number of			Time			Max Memory resident		
	Reads	found loci	usable loci	overall [s]	per read [μs]	per locus [ms]	overall [Mbytes]	per read [bytes]	per locus [kbytes]
SE-RAD	480,000	2,000	1,960	29	60	15	95	198	48
SE-ddRAD	240,000	1,000	983	17	71	17	91	379	91
PE-ddRAD	480,000	2,000	1,965	39	81	20	95	198	48
PE-ddRAD_m untrimmed	81,613	341	337	15	184	44	90	1,103	264
PE-ddRAD_m trimmed	81,613	341	333	12	147	35	91	1,115	267
Ber-SE	3,086,094	11,835	6,138	715	232	60	783	254	66
Seb-PE_m untrimmed	531,770	707	507	76	143	107	98	184	139
Seb-PE_m untrimmed	531,665	857	545	69	130	81	368	692	429
Seb-PE	16,925,410	141,105	75,118	4,203	248	30	787	46	6
Seb-PE 40 samples	109,070,980	206,660	147,935	27,636	253	134	1,128	10	5

3.6 Estimated size limits for data sets on an ordinary laptop

The pipelines PyRAD, ipyrad and dDocent were able to perform the analysis of the entire 40 samples extended Seb-PE data set. An estimate of the maximum number of samples that could be analysed by the pipelines was made to better understand the analysis limits of the pipelines. Stacks was excluded from this estimation, because the computational limits have already been reached with the reduced Seb-PE data set.

The maximum number of samples that can be analysed by PyRAD, ipyrad and dDocent were determined by assuming a linear approximation between the required memory and the number of reads. The linear approximation is only based on the results of the empirical data sets for each pipeline, because those are represent real data sets better than the simulated data sets. It was assumed that each sample contains 2,726,773 reads which is the average number of reads per samples of the entire 40 samples extended Seb-PE data set. For the operating system 1 GB of RAM was reserved. The maximum number of samples were 50 for PyRAD, 259 for ipyrad and 473 for dDocent. However this is just a rough estimation and more runs would be necessary to create a more accurate function and approximation for the memory prediction (Table 3.8).

Table 3.8: The approximately maximum numbers of samples and reads the pipeline can process. The approximation is based on the linear function which is derived from the results of the empirical data sets. The coefficient of determination (R^2) is a measure of how well the linear functions the data represent.

Pipeline	Max. Samples	Max. reads	linear function: Memory(reads) [Mbytes]	R^2
PyRAD	50	135,745,968	$4.9 * 10^{-5} * \#reads + 267$	0,988
ipyrad	259	706,082,474	$9.7 * 10^{-6} * \#reads + 151$	0,98
dDocent	473	1,289,112,903	$4.96 * 10^{-6} * \#reads + 606$	0,677

3.7 Overlap of the results

The Number of recovered loci in respect to the taxon coverage is an important property of the results the pipeline produce. Each sample is considered as taxon. Ideally, most of the loci cover many taxa. All loci recovered for the simulated data sets by PyRAD, ipyrad and dDocent are shared by all samples. Except for the data sets PE-ddRAD_m untrimmed and PE-ddRAD_m trimmed where dDocent recovered in both case one locus, which is only found in one sample. Most of the loci recovered for the simulated data sets by Stacks are shared by all samples, but there are also loci which are only shared by a view samples. PyRAD, ipyrad and dDocent provide a better taxon coverage than Stacks, because they can handle InDels (Figures 3.4a to d).

The loci recovered by dDocent for the empirical data sets have in general a much higher taxon coverage than the loci recovered by the other pipelines. The results of Stacks and PyRAD show a very similar taxon coverage. The results of ipyrad have the lowest taxon coverage. In case of the Seb-PE data set is the taxon coverage of the ipyrad loci especially low, compared to the other pipelines (Figures 3.4e and f).

The taxon coverage is presented as the number of recovered loci, which are shared by least 1,2,3,... samples (Figures 3.4).

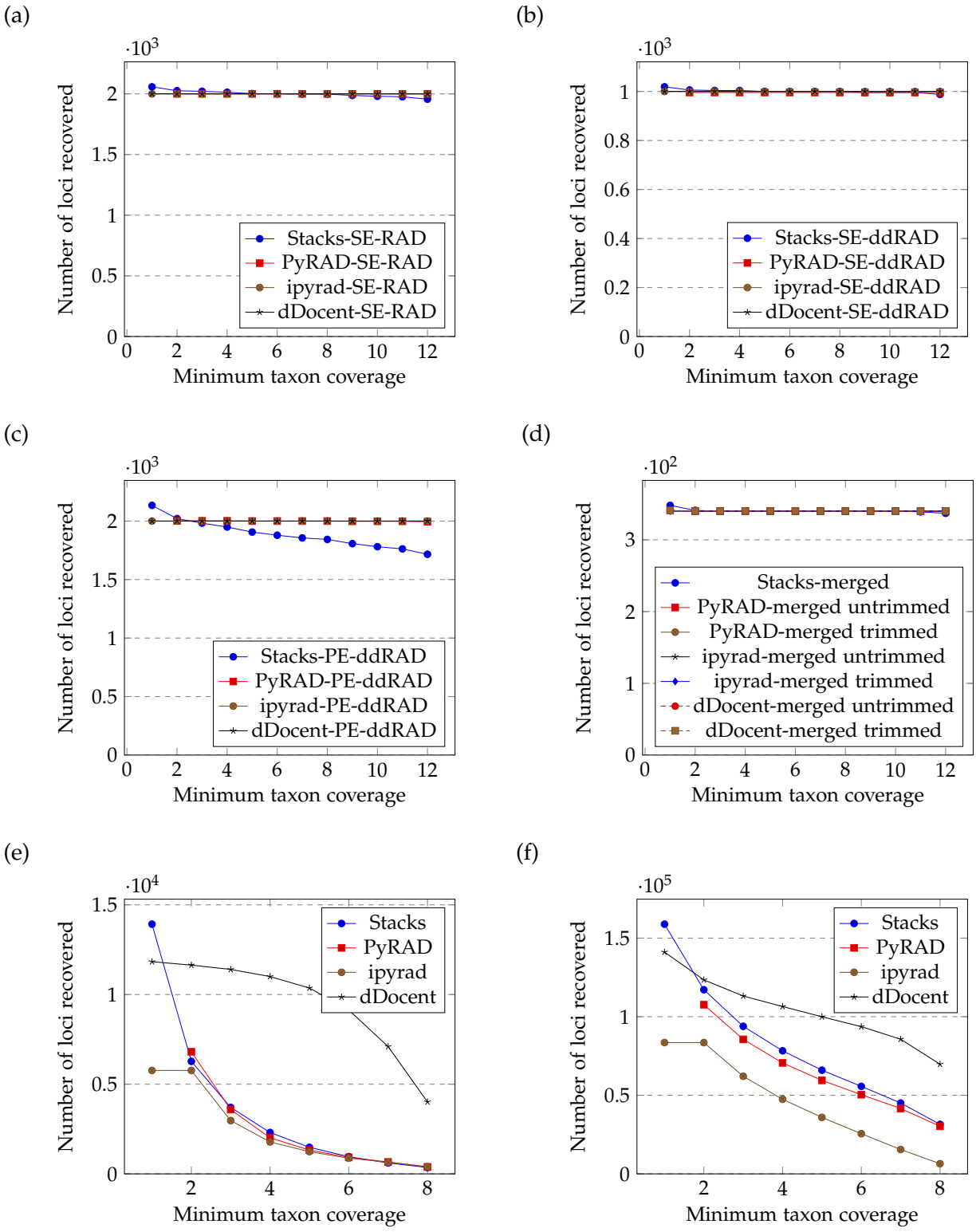

Figure 3.4: Taxon coverage of all four pipelines Stacks, PyRAD, ipyrad and dDocent for the simulated and empirical data sets. The number of found loci in respect to the taxon coverage for the data sets SE-RAD (a), SE-ddRAD (b), PE-ddRAD (c), PE-ddRAD_m (d), Ber-SE (e) and Seb-PE (f) is presented.

3.7.1 Comparison of the recovered loci

Beside of the taxon coverage comparison, the sequences of found usable loci of the pipelines were compared using BLASTN [54].

Stacks and dDocent create a consensus sequence for every locus, while PyRAD and ipyrad represent loci differently, they shows a consensus sequence from every sample (at least from every sample in the locus) per locus. To compare all pipelines, FASTA files were created, one containing the consensus sequences of Stacks, one containing the consensus sequences of dDocent, one containing only the first sequence of each locus of the PyRAD loci and one containing only the first sequence of each locus of the PyRAD loci. If necessary, additional FASTA files for the trimmed or untrimmed data sets were created. All these sequences, the consensus sequences and the first sample sequence of the loci, may differ in their SNPs. Therefore BLASTN, which tolerates minor differences between two sequences, was used to determine the loci overlap. The BLAST search was performed reciprocally: pipeline A against a pipeline B database and vice versa and this was done for each pipeline pair. The BLAST search results were filtered so only matches with an alignment length of at least 80% of the locus-sequence length were counted. This was done to ensure that the matching loci align over more than just the minimum exactly matching 14 nucleotides.

Although a pipeline may have more loci than the other pipelines it might not have any loci that cannot be found in the other pipelines, because sequences can match more than once for the BLASTN comparison. The reason for this multiple matches are InDels (insertions/deletions) in the sequences. Stacks does not handle InDels, it creates a new locus for the sequence(s) with the InDel(s). The other pipelines try to assign sequences where InDels are present to similar sequences where the InDels are not present. So a loci sequence with InDels might get more than one hit when blasted against another pipeline if the InDel sequences are not grouped to one loci in the other pipeline.

dDocent and Stacks identified more usable loci than PyRAD and ipyrad. The number of Stacks loci not found in the results of the other pipelines is for the most part, approximately equal to the numbers of not found loci of PyRAD and ipyrad, although Stacks identified more usable loci. dDocent, on the other hand, has higher numbers of loci that were not found in the results of the other pipelines than PyRAD, ipyrad and Stacks. The recovered

loci of the pipelines overlap almost entirely for each of the simulated data sets. This is not the case for the empirical data sets (Table 3.9).

The data from Table 3.9 was used to create Venn diagrams. The Venn diagrams illustrate the overlap between the pipelines. The results of each of the simulated data sets overlap almost entirely (Figures 3.5 and A.1). The results of the empirical data sets do not overlap as much as those of the simulated data sets. But the overlap between the results of Stacks and dDocent is relatively large (Figures 3.6 and A.2).

Here are only the Venn diagrams for the data sets SE-RAD (Figure 3.5) and Seb-PE (Figures 3.6) presented. The other Venn diagrams can be found in the Appendix (Figures A.1 and A.2).

Table 3.9: Overlap of the usable loci between the pipelines for the different analysis. The comparison was performed using BLASTN which tolerates miner differences in the sequences such as SNPs. The BLASTN search results were filtered for matches with an minimum alignment length of about 80% of the locus-sequence length to ensure only sophisticated matches are counted.

Data set	Pipeline	Number of usable loci	Number of loci not found by						
			Stacks	PyRAD		ipyrad		dDocent	
				untr.	tr.	untr.	tr.	untr.	tr.
SE-RAD	Stacks	1,975	–		1		1		0
	PyRAD	1,959	0		–		0		0
	ipyrad	1,959	0		0		–		0
	dDocent	1,960	0		1		1		–
SE-ddRAD	Stacks	985	–		0		0		0
	PyRAD	982	0		–		0		0
	ipyrad	982	0		0		–		0
	dDocent	983	0		0		0		–
PE-ddRAD	Stacks	1,986	–		0		0		0
	PyRAD	1,961	0		–		0		0
	ipyrad	1,959	0		0		–		0
	dDocent	1,965	5		6		6		–
PE-ddRAD_m	Stacks trimmed	332	–	0	0	0	0	0	0
	PyRAD untrimmed	336	4	–	4	0	4	0	4
	PyRAD trimmed	332	0	0	–	0	0	0	0
	ipyrad untrimmed	333	4	0	4	–	4	0	4
	ipyrad trimmed	332	0	0	0	0	–	0	0
	dDocent untrimmed	337	5	1	5	1	5	–	4
	dDocent trimmed	333	1	1	1	1	1	0	–
Ber-SE	Stacks	4,763	–		1,935		2,812		211
	PyRAD	2,977	1,559		–		1,503		1,057
	ipyrad	2,646	1,618		1,196		–		1,098
	dDocent	6,135	2,405		3,810		4,353		–
Seb-PE_m	Stacks trimmed	377	–	249	231	312	297	38	8
	PyRAD untrimmed	300	177	–	84	162	186	122	123
	PyRAD trimmed	330	196	119	–	210	203	155	140
	ipyrad untrimmed	206	142	71	86	–	88	96	96
	ipyrad trimmed	201	123	89	73	82	–	100	88
	dDocent untrimmed	507	169	321	327	393	405	–	40
	dDocent trimmed	546	183	370	353	435	433	83	–
Seb-PE	Stacks	48,629	–		8,529		19,304		477
	PyRAD	47,761	12,941		–		16,598		2,324
	ipyrad	33,809	10,418		4,028		–		1,744
	dDocent	75,118	29,345		26,892		39,104		–

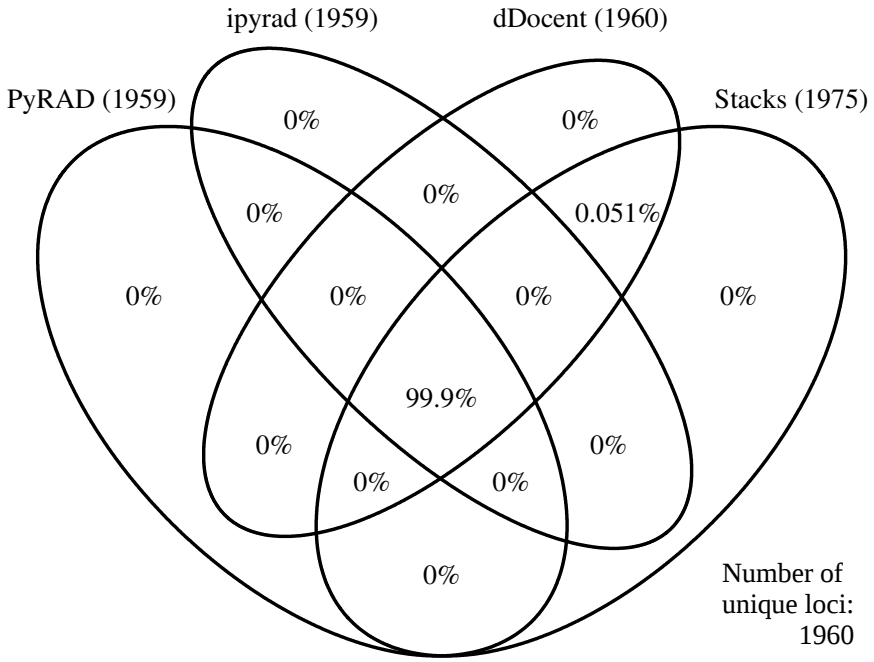

Figure 3.5: Overlap of the usable consensus sequences of the pipelines Stacks, PyRAD, ipyrad and dDocent in percentages for the SE-RAD data set.

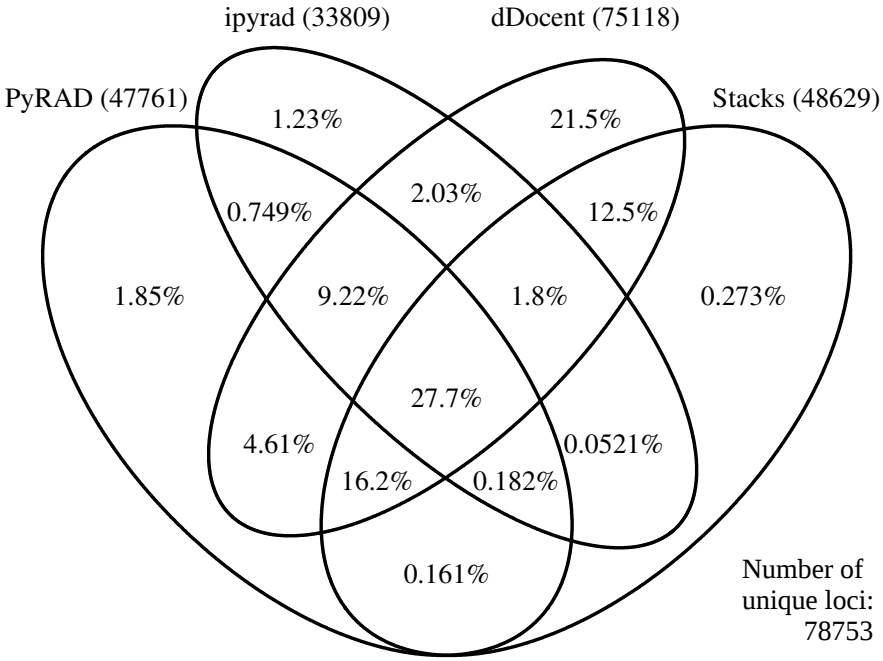

Figure 3.6: Overlap of the usable consensus sequences of the pipelines Stacks, PyRAD, ipyrad and dDocent in percentages for the Seb-PE data set.

3.7.2 Comparison of species trees

Of further interest are the phylogenetic relations of the species. The phylogenetic relations are illustrated with species trees. For the length of the vertical edges of these trees the branch lengths were used. The branch length represents the genetic distance between the samples. The unit of the branch lengths is nucleotide substitutions per site, which is the number of substitutions divided by the length of the sequence [76, 77].

The true species tree for the simulated data sets comprises three subtrees of 4 species each (Figure 3.7). The tree structure of the trees created from the results of the pipelines for the simulated data sets is identical to the true tree, not considering branch lengths. They may differ in the horizontal order of the samples, but the order in which the samples are connected is the same. For example, if we take a look at the subtree of A, B, C and D we can see that from bottom to top A and B are the first samples, which are connected, then C and finally D. Although the samples in the trees of the analyses are ordered B, A, C and D, the order, in which they are connected, is the same (Figure 3.8 and A.3 to A.5). The horizontal distance between two samples does not relate to a genetic distance between these two samples. The horizontal lines are simply for visualisation of the connection between the vertical lines [76]. The trees of the simulated data sets all had a PH85 distance of 0 to each other and to the reference tree presented in figure 3.7 (data not shown).

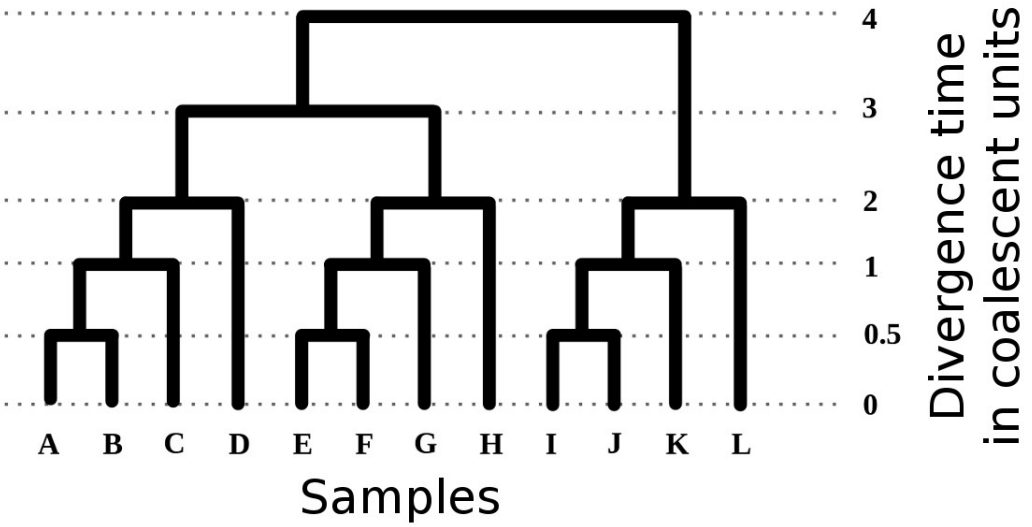

Figure 3.7: Species tree of the simulated data. This figure is adapted from the Species tree presented in SE-ddRAD [13] and shows the phylogenetic relation between the 12 samples.

Unfortunately, no reference tree is available for the empirical data sets. But it can be assumed that individuals from the same species should form a subtree. The tree for the Seb-PE analysis was created by only using those loci which cover all 8 samples, because otherwise Stacks ran out of memory. Apart from the Seb-PE analysis tree are the trees not particularly similar (Figures 3.9, 3.10, A.6 and A.7). The PH85 topological distances between the empirical Stacks, PyRAD, ipyrad and dDocent trees for the different analysis are relatively high. But the PH85 distances between the trees of the Seb-PE analysis are smaller than distances of the other empirical data set analysis (Table 3.10).

Table 3.10: The PH85 topological distances between trees form the different pipelines for the empirical data sets.

Data Set	Stacks			PyRAD		ipyrad
	compared to					
	PyRAD	ipyrad	dDocent	ipyrad	dDocent	dDocent
Ber-SE	10	10	6	10	8	8
Seb-PE_m untrimmed	-	-	-	10	10	6
Seb-PE_m trimmed	10	10	10	6	10	8
Seb-PE	4	8	6	8	6	4

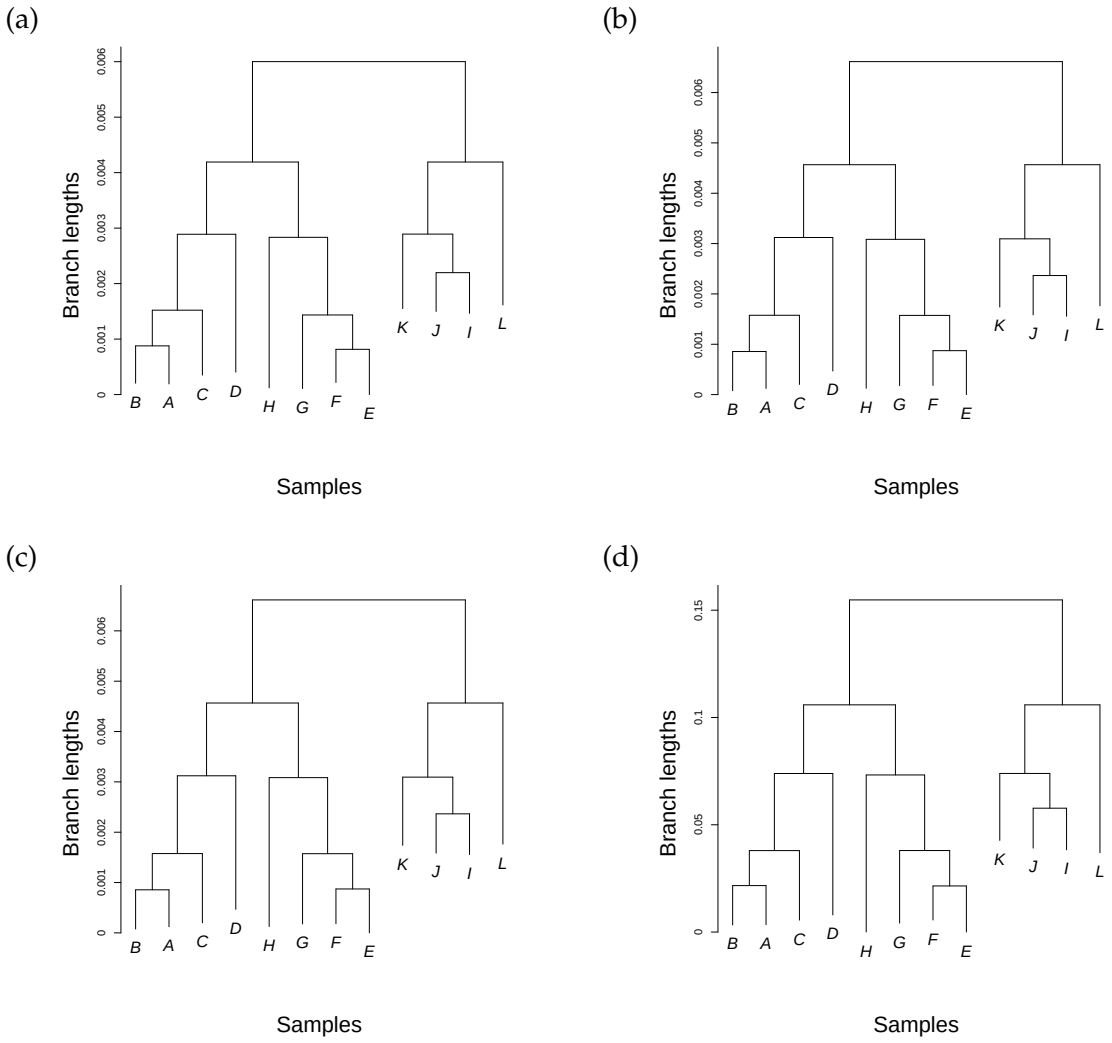

Figure 3.8: Species trees of the SE-RAD analysis created with the phylip files of the Stacks (a), PyRAD (b), ipyrad (c) and dDocent (d). The Stacks phylip file was created from 2,058 loci and each phylip file of PyRAD, ipyrad and dDocent was from 2,000 loci.

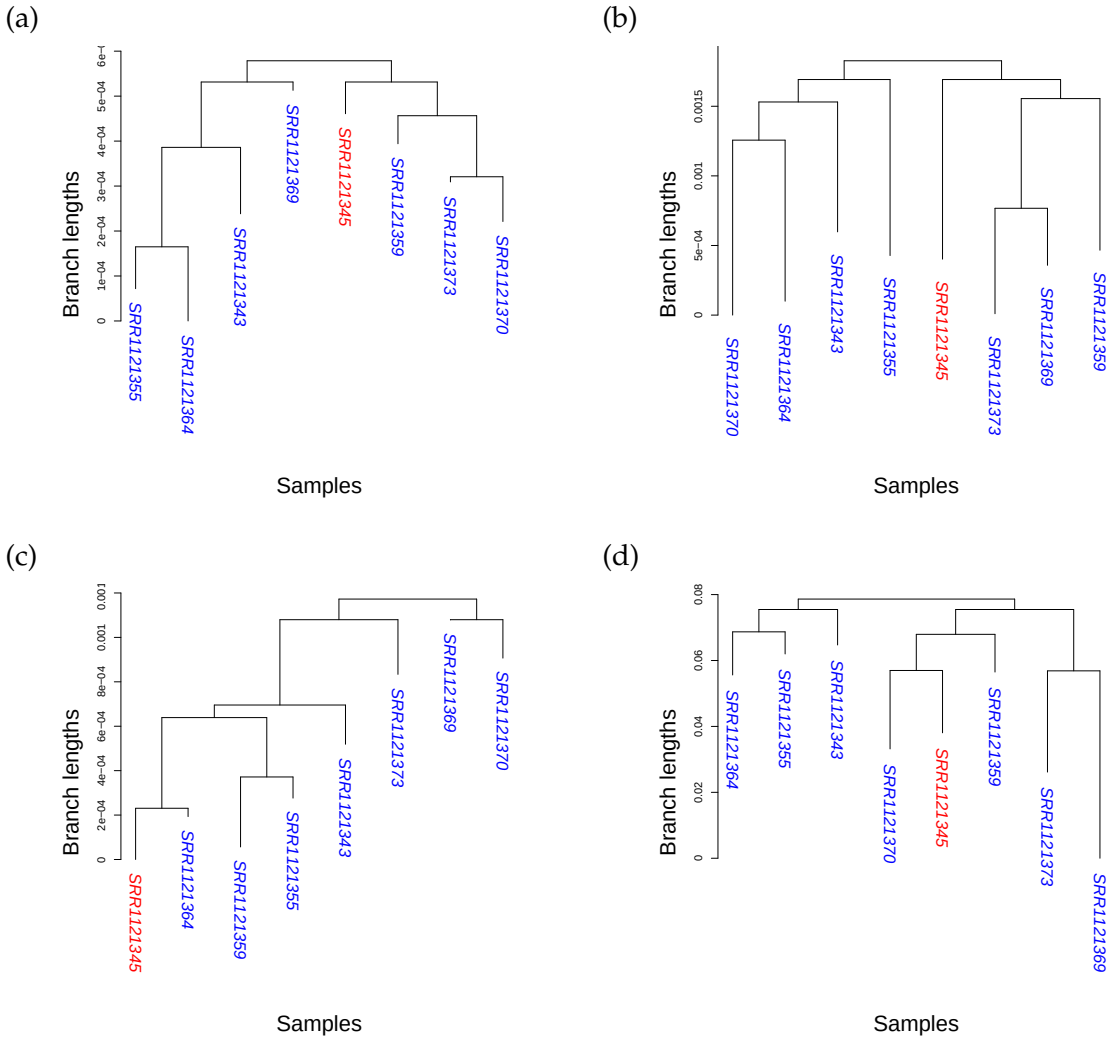

Figure 3.9: Species trees of the Ber-SE analysis created with the phylip files of the Stacks (a), PyRAD (b), ipyrad (c) and dDocent (d). The Stacks phylip file was created from 13,923 loci, the PyRAD phylip file from 6,809 loci, the ipyrad phylip file from 5,766 loci and the dDocent phylip file from 4,082 loci (from the `Final.recode.vcf` file). Samples of the *Berberis alpina* spieces are coloured blue and the *Berberis moranensis* species sample is coloured red.

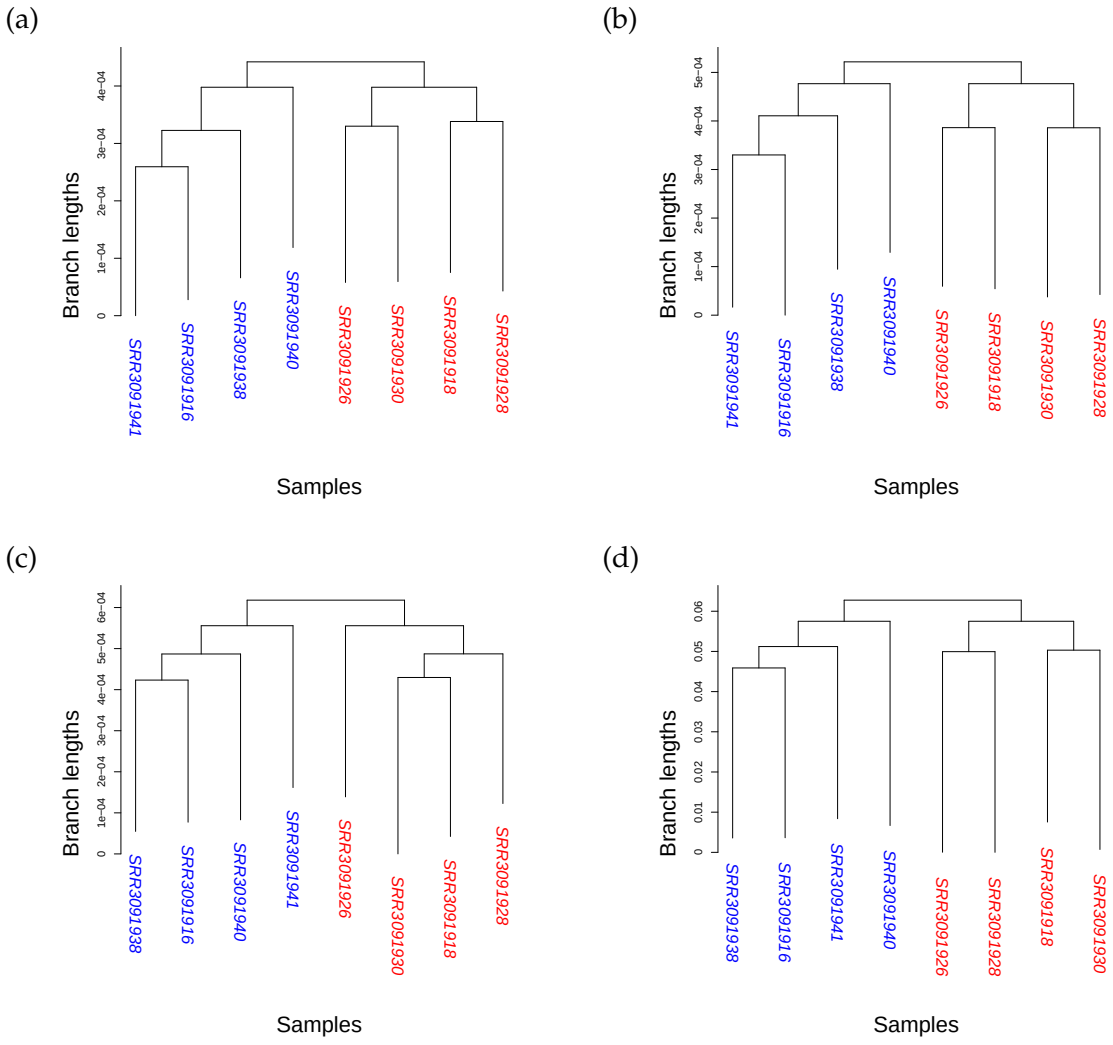

Figure 3.10: Species trees of the Seb-PE analysis created with the phylip files of the Stacks (a), PyRAD (b), ipyrad (c) and dDocent (d). The phylip files were created only by using loci, which cover all 8 samples. The Stacks phylip file was created from 33,575 loci, the PyRAD phylip file from 30,285 loci, the ipyrad phylip file from 6,476 loci and the dDocent phylip file from 61,095 loci (from the `Final.recode.vcf` file which was filtered for a minimum coverage of 8). Samples of the *Sebastes chrysomelas* spieces are coloured blue and samples of the *Sebastes carnatus* species are coloured red.

3.7.3 fastStructure Results

The result of the fastStructure script `chooseK.py` are two estimations for the number of populations: The model complexities K^*_ε and $K_{\varnothing^c}$. The model complexity K^*_ε is based on the marginal likelihood of the data and $K_{\varnothing^c}$ is the number of relevant model components, which is the minimum number of populations that have a cumulative ancestry contribution of at least 99.99% [65].

In case of the simulated data sets, `chooseK.py` suggest for the most part 3 populations. The simulated data sets comprise 3 populations, but two of those populations closer related to each other than to the third population. The suggested number of populations beside 3 is 2, in this case are the two closer related populations counted as only one population (Table 3.11).

For the most part did `chooseK.py` suggest 1 population for the empirical data sets. The empirical data sets are comprised of 2 different species, which should lead to at least 2 different populations (Table 3.11).

As with the phylip file for the Seb-PE Stacks analysis, it was also only possible to create a Structure file from only those loci which cover all 8 samples, because otherwise Stacks ran out of memory.

Table 3.11: Results of the `chooseK.py` script. K^*_ε and $K_{\varnothing^c}$ are suggestions for the number of populations.

Data set	Stacks		PyRAD		ipyrad		dDocent	
	K^*_ε	$K_{\varnothing^c}$	K^*_ε	$K_{\varnothing^c}$	K^*_ε	$K_{\varnothing^c}$	K^*_ε	$K_{\varnothing^c}$
SE-RAD	3	3	2	3	3	3	3	3
SE-ddRAD	2	3	2	3	3	3	3	2
PE-ddRAD	2	2	3	3	3	3	2	3
PE-ddRAD_m untrimmed	-	-	3	3	3	3	3	3
PE-ddRAD_m trimmed	2	3	3	3	3	3	3	3
Ber-SE	1	1	2	3	1	3	1	1
Seb-PE_m untrimmed	-	-	1	1	1	4	1	1
Seb-PE_m trimmed	1	1	1	3	1	4	1	1
Seb-PE	1	2	1	1	1	1	1	1

Distruct graphics were generated using the the fastStructure script `distruct.py`. For the simulated analysis, the distruct graphics for 2 and 3 populations generated, because the suggestions of the `chooseK.py` script were either 2 or 3 for the these data sets. All samples were assign the correct population (Figures 3.11 and A.8 to A.10).

For the most part suggested fastStructure a single population for the empirical data sets, although each data set consist of 2 different species. Because a distruct plot for only 1 population does not carry much information, the distruct plots for the 2 populations and the other population numbers (besides 1) suggested by `chooseK.py` are presented instead. The populations shown in the distruct plots for the empirical data sets do not show a separation based on the sample species or location, where they were found (Figures 3.13, A.11 and A.12).

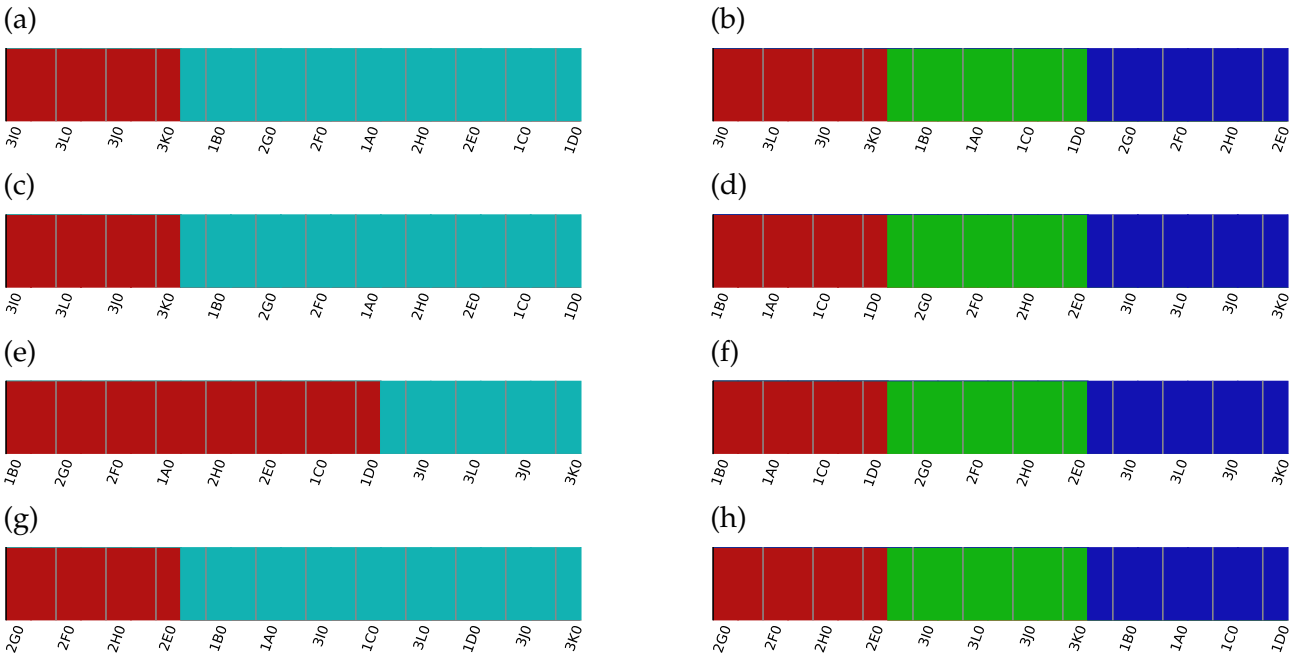

Figure 3.11: Distruct graphics for the SE-RAD data set. a (Stacks), c (PyRAD), e (ipyrad) and g (dDocent) show the distruct graphics for 2 populations. b (Stacks), d (PyRAD), f (ipyrad) and h (dDocent) show the distruct graphics for 3 populations.

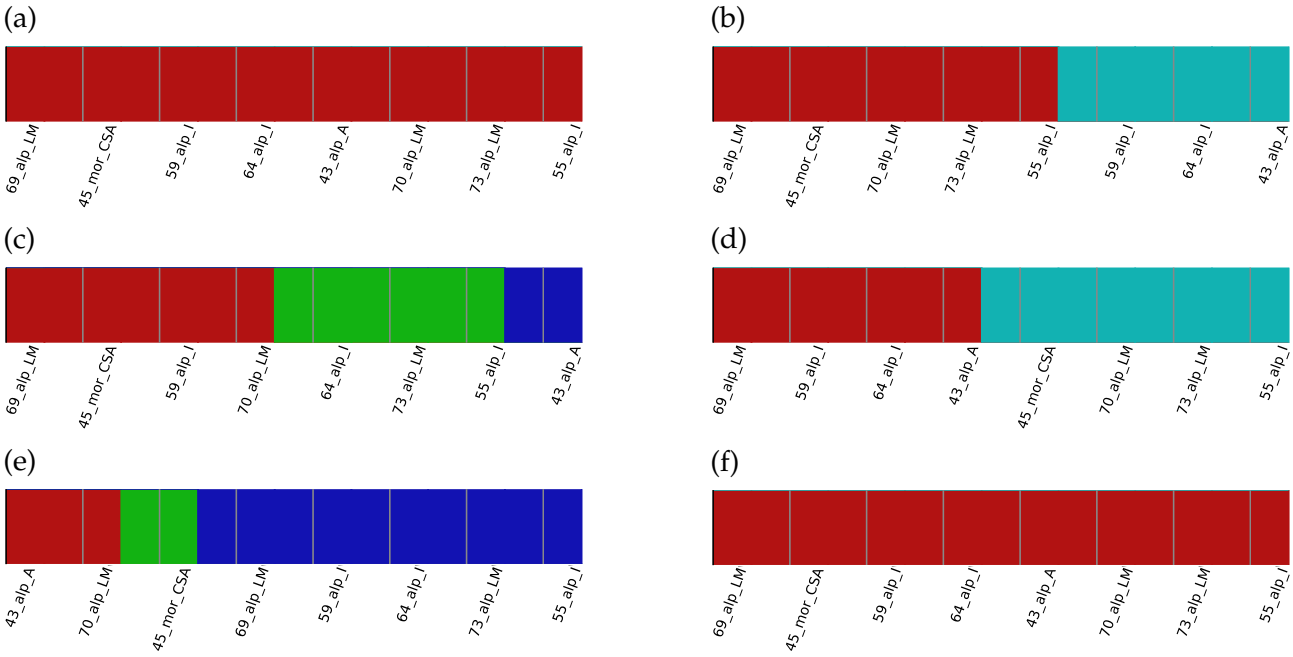

Figure 3.12: Distruct graphics for the Ber-SE data set. The samples are labelled with the last two digits of their Run number, an abbreviation for their species and an abbreviation for the location where they were found. The species are abbreviated with alp for *Berberis alpina* and mor for *Berberis moranensis*. The locations are abbreviated with A for Ajusco, CSA for Cerro San Andres, I for Iztaccihuatl and LM for La Malinche. The shown distruct plots are from: Stacks with 2 population (a), PyRAD with 2 populations (b), PyRAD with 3 populations (c), ipyrad with 2 population (d), ipyrad with 3 populations (e) and dDocent with 2 populations (f).

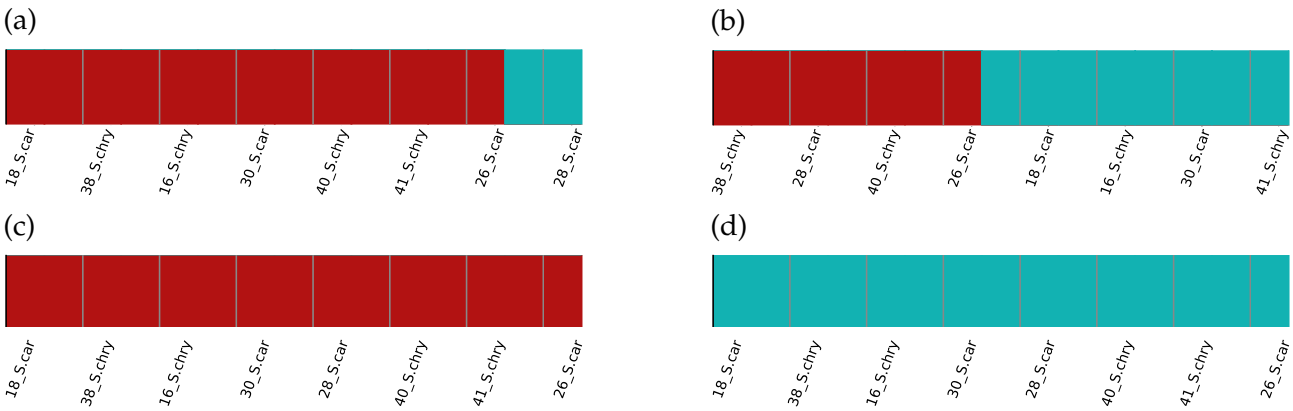

Figure 3.13: Distruct graphics for the Seb-PE data set. The samples are labelled with the last two digits of their Run number and an abbreviation for their species. The species are abbreviated with S.car for *Sebastes carnatus* and S.chry for *Sebastes chrysomelas*. Figures a (Stacks), b (PyRAD), c (iyprad) and d (dDocent) show the distruct graphics for two populations.

4 Discussion

In this thesis, pipelines to analyse RAD and ddRAD data were investigated and their results compared. 4 simulated and 2 empirical data sets were analysed with Stacks [6], PyRAD [7], ipyrad [8], and dDocent [9]. A further pipeline rtd [1] had to be excluded from the comparison because it was not possible to run the available rtd version. A migration of rtd to OAuth 2.0 would have been necessary to run it. The Rainbow [10] pipline was not used directly to analyse the data sets. However, since Rainbow is used by dDocent, it was used indirectly for the analyses. The identified RAD loci as well as the run time and the memory footprint of the pipelines were compared.

Stacks identified the highest number of overall loci and was generally relatively fast except for one analysis. But it had the highest memory footprint and was the only pipeline which exceeds the available 8 GB of memory on the test system. It is also the only pipeline which creates a new locus for a sequence with InDels. The other pipelines try to assign sequences where InDels are present to similar sequences where the InDels are not present.

PyRAD was for most of the analyses the slowest pipeline. But it recovered more loci than ipyrad.

ipyrad is the successor of PyRAD and is much faster than PyRAD. But it did not identified as many loci as PyRAD. ipyrad had a higher memory footprint for small data sets but a much smaller for large data sets.

dDocent was overall the fastest pipeline and recovered more usable loci than the other pipelines. But it is the only pipeline that does not come with build-in functions for generating phylogenetic or population structure files. The memory footprint of dDocent was relatively small.

4.1 Installation

The Stacks pipeline can run independently and does not need any additional software. Unlike the other pipelines, Stacks provides an optional web interface, which is used to visualize data. The web interface depends on additional software and configurations to work properly: Apache, MySQL, Perl, and PHP. After their installation, the web interface has to be enabled in the Apache web server and access to the MySQL database has to be ensured.

PyRAD needs two common Python packages (Numpy and Scipy) and two additional programs (muscle and vsearch).

ipyrad also needs additional software, but ipyrad and all dependencies can be easily installed with conda [78]. Conda is a package management and environment management system for installing software packages and their dependencies. With conda ipyrad can be installed via a single command.

For dDocent several dependencies are needed, which is not surprising since dDocent relies heavily on third party software. dDocent comes with a script to install all required third party software, which is simply executed in a Linux terminal.

rtd is build in python but also requires additional software. It was not possible to set up or connect a database, because it uses ClientLogin API which is no longer supported by Google. No workaround was found to overcome this problem. A migration of the pipeline to OAuth 2.0 would be necessary.

The installation of all four pipelines is straightforward and well documented. However, if Stacks is installed with the web interface, it needs more effort than PyRAD and especially more effort than ipyrad and dDoncent because these two can be installed by using conda or the provided installation script of dDocent. Although the activation of the Stacks web interface is well described and therefore not that difficult, it requires some effort and is probably not that easy for someone who is not familiar with Linux.

Updated/newer versions of the pipelines Stacks, ipyrad and dDocent are seem to be regularly published. No newer versions of PyRAD will be available, because it was discontinued and

replaced by ipyrad. All four pipelines can be run on a high performance cluster (HPC), although the installation depends on the cluster and the administrator should be consulted. Because all four pipelines are invoked via the command line interface, the pipelines can be executed on a HPC via ssh.

4.2 Support

The pipelines are supported by user forums. Stacks, PyRAD and dDocent are supported via Google Groups. ipyrad has its own forum on its homepage but ipyrad support can also be found in the PyRAD Users Google Group. There is no support forum available for rtd (Table 3.3). The forums offer enough support for most or all needs.

4.3 Pipeline execution

All four pipelines are invoked via the command line interface. The commands for PyRAD and ipyrad do not vary much for the different analyses because all parameters are stored in the parameter files for the respective analysis. The same applies for dDocent where most of the parameters are stored a configuration file as well. Stacks is build modularly and allows a separate execution of each module. The whole pipeline is invoked with the execution control programs `denovo_map` or `ref_map`. All parameters are handed over to the execution control program at the start. This makes the execution command of Stacks more complex than the command of the other three pipelines. To make the command shorter and also save the parameters of Stacks, a short script was implemented, which executed the `denovo_map` program with the appropriate parameters. This has the advantage that the command can be very simple and also that the parameters are saved.

PyRAD and ipyrad can take all sample data files of a format e.g. fastq.gz of a directory as input, while in Stacks you either need to specify each sample file or provide a population map and a sample directory. In the population map, each sample is categorised in a group/population. If a population map and sample directory are provided than Stacks will use only those samples of the sample directory, which are also in the population map. So

either way each sample must be specified in Stacks, which might be annoying for data sets with a huge number of individuals, but also ensures that the analysis is only performed for the desired data. For dDocent the sample data files must be in the working directory and must be in gzipped FASTQ file format. The gzipped FASTQ files must also follow a strict naming convention which could be annoying for larger data sets, although dDocent comes with a script which helps renaming the demultiplexed sample files.

The runtime of the pipelines varies a lot. For most of the performed analyses, dDocent turned out to be much faster than any other pipeline. Stacks is faster than PyRAD and ipyrad except for the Seb-PE analysis where Stacks needed over six days while the second slowest pipeline PyRAD finished in under 10 hours. During the Stacks analysis of the Seb-PE data set, the CPU was fully utilised, while for the most part enough RAM was free. Although ipyrad is much faster than PyRAD it is still very slow compared to dDocent.

Stacks had the highest memory footprints and exceeds the available 8 GB of memory for the Seb-PE analysis. The other pipelines had much smaller memory footprints. Especially ipyrad and dDocent only required a relatively small amount of RAM for large data sets.

Performing the analyses on an ordinary laptop

PyRAD, ipyrad and dDocent were able to perform the analysis of the entire 40 samples extended Seb-PE data set. This shows that RAD/ddRAD analyses can be performed for large data sets on an ordinary desktop PC or laptop.

PyRAD has a much larger memory footprint for the extended Seb-PE data set than ipyrad and dDocent. This indicates that ipyrad and dDocent are able to analyse more samples than PyRAD (Tables tables 3.5 to 3.7). The maximum number of samples that can be analysed by PyRAD, ipyrad and dDocent was estimated based on the results of the empirical data sets. A linear approximation between the needed memory and the number of reads was used for the estimation. More runs would be necessary to create a more accurate non linear function and approximation for the memory prediction. According to this estimate, dDocent can analyse larger data sets than PyRAD and ipyrad (Table 3.8).

4.4 Pipeline Results Overlap

4.4.1 InDel handling

Variants sites are used to differentiate between individuals. The simplest form of a variant site are SNPs. But insertions/deletions (InDels) are also variant site and can also be used for differentiation between individuals. Because of that a proper identification and handling of InDels is important to ensure that all variant sites can be used for further analyses such as phylogenetic and structure analysis.

Stacks does not really handle InDels, it just creates a new locus for the sequence with the InDel(s). PyRAD, ipyrad and dDocent instead can handle InDels and try to assign sequences with an InDel to a locus of a similar sequence where the InDel is not present. But only dDocent marks an InDel as an variant site of a sequence. PyRAD and ipyrad only mark polymorphisms as variant sites. This means that loci with only InDels and no SNP will show up in the usable loci of dDocent but not in the usable loci of PyRAD and ipyrad. Although PyRAD and ipyrad do not mark InDels as variant sites they do use them for the generation of their output files like phylip files.

The different InDel handling of the pipelines also cause multiple BLASTN search hits. Because Stacks creates a new for sequences with InDels and the other pipelines try to assign sequences where InDels are present to similar sequences where the InDels are not present. This has two important effects on the usable loci. If an InDel is near the end of the sequence, Stacks may align the sequence with those sequences without the InDel, but all the following mismatches than get incorrectly interpreted as SNPs. Another effect is, that some usable loci may not be considered usable, because a new locus is created for only one individual, which has an InDel but no SNPs in that locus. In this case, the locus is not marked as a usable locus by Stacks, although the locus of the individuals without that InDel has SNPs either within an individual or across individuals.

4.4.2 Handling of PE-reads

PE reads may overlap at their ends if the RAD/ddRAD tags are shorter than twice the sequencing length. Analysing overlapping forward and reverse reads individually may introduce noise. Because of that it is beneficial, to merge or filter the overlapping reads. PEAR filters PE data and merges overlapping forward and reverse reads. With the results of PEAR 3 options are available to proceed the analysis. The first option is to only analyse the non-merged reads, the second option is to only analyse the merged reads and the third option is to combine the merged and non-merged data and analyse them. If the majority of the data does not overlap it is usually not recommended to combine them with the merged reads, because it may just introduce noise instead of signal. The same applies for a data set where most of the reads overlap [15].

In case of the Seb-PE data set most of the reads do not overlap. We can see that the results of the Seb-PE_m trimmed and Seb-PE_m untrimmed data sets are more noisy than the results of the Seb-PE (unassembled) data set.

4.4.3 Structure analysis

The fastStructure script `chooseK.py` provides different values for the estimated number of populations. The model complexity K_ε^* is based on the marginal likelihood of the data and $K_{\varnothing^c}$ is the number of relevant model components, which is the minimum number of populations that have a cumulative ancestry contribution of at least 99.99% [65]. In general is K_ε^* robust in identifying strong structures, while $K_{\varnothing^c}$ identifies model components of weak structures.

4.4.4 Simulated data

For the most part, the results of all four pipelines do overlap for the simulated data sets. Stacks seems to find more loci than PyRAD, ipyrad and dDocent, because Stacks does not really handle InDels. It just creates a new locus for the sequence(s) with the InDel(s). The

other pipelines handle InDels better, they try to assign sequences where a InDels are present to similar sequences where the InDels are not present (Figures 3.5, 3.6, A.1 and A.2.).

If we take closer look at the comparison of the usable loci of the simulated data sets we observe that Stacks and dDocent tend to have more loci than PyRAD and ipyrad. Stacks has more loci because it creates a new locus for a sequence with InDels. dDocent has more usable loci than PyRAD and ipyrad because it marks InDels as variant sites, but PyRAD and ipyrad do not mark them. For the simulated data, the regular dDocent output could not be used because dDocent performs a filtering step before the SNP calling. dDocent filters loci which have a high coverage across the samples, because these tend to be multi copy loci in real data sets [58]. An unfiltered vcf file therefore had to be generated for the simulated data sets.

dDocent is the fastest pipeline and on average it is 15 times faster than PyRAD, which is generally the slowest pipeline. PyRAD has the smallest memory footprint and is on average 6 times smaller than the memory footprint of Stacks. Stacks has except for the PE-ddRAD_m trimmed data set the largest memory footprint.

Phylogeny:

The species trees of all four pipelines are very similar to the reference tree (Figure 3.7) after which the simulated data was modelled. If branch lengths are not considered, the topology of the trees is identical apart from the order of the samples, which is also shown by their PH85 distances of 0 to the reference tree (Figures 3.8 and A.3 to A.5). This is not surprising given the fact that the loci overlap as much as they do.

Structure:

The `chooseK.py` script suggests 2 and 3 populations for the simulated data sets. The simulated data sets comprise 3 populations, but the suggested 2 population make also sense, because the third population is the first one which separates from the other two populations (Figure 3.7). The distruct graphics show that the samples get correctly grouped into populations (Figures 3.11 and A.8 to A.10).

4.4.5 Empirical data

The loci for the empirical data do not overlap as much as the loci for the simulated data. Stacks recorded overall more loci than PyRAD, ipyrad and dDocent. dDocent found nearly as much loci as Stacks. Although Stacks had the highest number of overall loci the number of usable loci is only the second largest and is much closer to the number of usable loci recorded by PyRAD. ipyrad recorded the least amount of loci.

PyRAD only identifies/keeps loci which cover at least 2 individuals. Stacks on the other hand identifies also loci which only occur in a single individual. In case of the empirical data sets did Stacks identify many loci, which only cover a single individual. The taxon coverage of Stacks and PyRAD for 2 - 8 taxa is very similar (Figures 3.4e and f). Most of the Stacks loci, which cover only a single individual, do not contain a SNP and thereby do not represent a usable locus. For example did Stacks recover 158,891 loci (Table 3.4) for the Seb-PE data set, but 41,617 of these loci only cover a single individual. Only 1,457 of these 41,617 loci contain SNPs. This means that the majority of the loci, which cover only a single individual, are not part of the usable loci set of Stacks. Therefore, the number of usable loci recovered by Stacks is so close to the number of usable loci recovered by PyRAD, although Stacks recovered a lot more loci overall than PyRAD.

dDocent does have the most usable loci sequences which can not be found in the results of the other pipelines. InDels are marked as variant sites by dDocent and they are thereby counting to the usable loci, but PyRAD and ipyrad do not mark InDels as variant sites. This means that a sequence with InDels is not part of the usable loci of PyRAD and ipyrad unless the sequence does also have a polymorphism site. Stacks does not detect InDels at all. The amount of Stacks loci which can not be found in the dDocent loci is relatively low (Table 3.9). This suggests that the Stacks results are very close to the dDocent results. In the same way we observe that the results of PyRAD and ipyrad are closer to each other than those of Stacks. The number of loci that could not be found in ipyrad is relatively high for each pipeline. This is probably due to the relatively small number of overall and usable loci that ipyrad recorded for each analysis.

dDocent is significantly faster than the other pipelines except for the data set Seb-PE_m trimmed where Stacks is the fastest. Stacks is the second fastest of these pipelines except for

the Seb-PE, where it needed over six days for the analysis while the second longest time of PyRAD was under 10 hours. PyRAD and ipyrad are both significantly slower than dDocent although ipyrad is much faster than PyRAD it is still far behind dDocent.

The memory footprint of each pipeline varies greatly for the empirical data sets. For the most part Stacks has the highest memory footprint. It also exceeded the available memory during the generation of the phylip and structure files of the Seb-PE data set. It was only possible to generate these files from loci which cover all 8 samples. The other pipelines were able to complete the entire Seb-PE analysis. The analyses of the extended Seb-PE 40 samples data set could also be completed with PyRAD, ipyrad and dDocent.

Phylogeny:

The branch lengths of the trees represent the genetic distance between the samples. The unit of the branch lengths is nucleotide substitutions per site, which is the number of substitutions divided by the length of the sequence [76, 77]. The dDocent trees have the longest branches. The reason for this is the used `vcf_to_phylip.py` script which only uses SNPs and not the whole sequences of the loci to generate the sequences of the phylip file. This leads to larger genetic distances because the ratio between number of substitutions and length of the sequence is larger. Stacks, PyRAD and ipyrad use the whole sequences of the loci to generate the phylip file. The Stacks trees have the shortest branches.

The species trees of the Seb-PE analysis are very similar to each other. The samples form two sub trees one for the *S. carnatus* species and one for the *S. chrysomelas* species. But the topology of the four trees has differences in their subtrees. The branch lengths of the Seb-PE Stacks, PyRAD and ipyrad indicate that the samples only have a small genetic distance. Some of the samples in the subtrees are not on the same position in the different Seb-PE trees because of the small genetic distances and the variations of the results of the pipelines (Figure 3.10). The PH85 topological distance is smaller than those of the other empirical data sets (Table 3.10).

The Ber-SE analysis trees do not look very similar which is also reflected by the PH85 distances (Figure 3.9). Most of the trees have a PH85 distance of 10 which is the maximum PH85 distance for two trees with 8 tips. In that case the two trees differ in every bipartition. The distance between the PyRAD and dDocent trees as well as the distance between the

ipyrad and dDocent trees is with 8 not quit as high as the distances between the others (Table 3.10).

The trees for the Seb-PE_m analysis have very different branch lengths. Except for the PyRAD and the ipyrad tree which have branch lengths of the same order of magnitude. Unfortunately none of these trees separates the two species *S. carnatus* and *S. chrysomelas* in two subtrees. Although the ipyrad tree shows some grouping of the samples based on their species. The other trees do not show a separation of the two species (Figure A.6 and A.7). Most of the trees have a high PH85 distance to each other, but the Stacks and PyRAD tree are relatively close to each other (Table 3.10).

Structure:

None of the distruct plots for the Ber-SE data set show a separation of the two species *B. alpina* and *B. moranensis*. The results of the structure analysis for the Stacks Structure file only show one population no matter with which number of population the structure.py script is executed. The distruct plots for the analysis of the other pipelines show different populations, but those seem not to be based on the species or location, where they were found (Figure 3.12).

The phylogenetic information available for the Ber-SE data set does not include all samples. It does also not include all of the 8 samples used in this thesis (Dendrograms: Appendix S3 of [19]).

The `chooseK.py` script suggests only one population for the Seb-PE_m data set except for ipyrad were it suggested four populations and PyRAD were it suggested three populations for the trimmed data set. The distruct plots for ipyrad with four populations and PyRAD with three populations do not show a separations based on the species or loaction of the samples. If fastStructure is performed with 2 as number of populations the distruct graphic of the Stacks and dDocent results still only show one population. The distruct plot for the PyRAD and ipyrad Structure files show two population. But the samples are not separated based on their species into two populations (Figure A.11 and A.12).

The distruct plots for the Seb-PE data set do not show a separation of the two species *S. carnatus* and the *S. chrysomelas*. The distruct plot for Stacks only has one sample in the second population and the distruct plot for ipyrad and dDocent have only one population. In the

distruct plot of RyRAD are two populations with each four samples. But those populations contain samples of the *S. carnatus* and the *S. chrysomelas* species (Figure 3.13).

In the original study of the Seb-PE data set was no phylogenetic analysis among the different individuals performed [20]. The phylogenetic information of this study could have been used as a reference, if a phylogenetic analysis among the different individuals had been performed. The two species *S. chrysomelas* and *S. carnatus* are very close related. The F_{ST}-value between these two species is 0.046, which is about an order of magnitude lower than comparisons between other closely related *Sebastes* species [79]. The F_{ST} [80] describes the differences between populations and takes also the differences within the population into account. fastStructure could not separate the species into two different populations. The two species are very closely related, but they each form a subtree in the species trees of the Seb PE dataset (Figure 3.10). The separation of the species in the species trees indicates that a separation into two population should be possible. Further investigation would be needed to clarify the reason why fastStructure could not determine the two populations.

5 Conclusion

Each pipeline has different advantages and disadvantages:

Stacks had several problems with the Seb-PE data set. Besides the very long run time it was only possible to run the `populations` script to generate the phylip file and the structure file for loci which cover all 8 samples. Stacks performed well for the other data sets, but handles InDels as loci on its own.

All pipelines except Stacks could complete the analysis of the entire 40 Sample Seb-PE data set on a state of the art laptop hardware, whereas Stacks does require a high-end computer server in that case.

PyRAD was the slowest pipeline for most of the data sets. But it recovered a relatively high number of loci. PyRAD is easy to execute because of all parameters are saved in one file. But PyRAD was discontinued and replaced by ipyrad.

ipyrad performed much faster than PyRAD but it did not recover as many loci as PyRAD. ipyrad (like PyRAD) can be executed via a single command which also generates all the desired output files without any further commands or software.

dDocent recovered more usable loci than any other pipelines and was also the fastest pipeline (Table 3.7). There is also a very significant overlap between the results of dDocent and the other pipelines (Table 3.9). One drawback of dDocent is that additional software and scripts were needed to create input files for the phylogenetic and structure analysis while the other pipelines had built-in functions for generating these files (Table 3.2).

Indels are handled differently by each pipeline. In that respect, dDocent seems to come closest in providing the expected results.

If an ordinary desktop PC or laptop is used dDocent would be the best choice. The memory footprint of dDocent is relatively small and it also was the fastest pipeline. But additional software is needed to generate desired output files. Because of this more effort is needed for the set-up of the whole system.

Bibliography

[1] Peterson BK, Weber JN *et al.*: **Double Digest RADseq: An Inexpensive Method for *De Novo* SNP Discovery and Genotyping in Model and Non-Model Species**. *PloS ONE* 2012. **7**(5): e37135.

[2] Baird NA, Etter PD *et al.*: **Rapid SNP Discovery and Genetic Mapping Using Sequenced RAD Markers**. *PloS ONE* 2008. **3**(10): e3376.

[3] Floragenex: RAD Sequencing. `http://www.floragenex.com/rad-seq/`, Date accessed: 16.03.2017.

[4] Deagle BE, Faux C *et al.*: **Antarctic krill population genomics: apparent panmixia, but genome complexity and large population size muddy the water**. *Molecular Ecology* 2015. **24**(19): 4943–4959.

[5] DNA Sequencing Costs: Data. `https://www.genome.gov/sequencingcostsdata/`, Date accessed: 16.05.2017.

[6] Catchen JM, Amores A *et al.*: **Stacks: Building and Genotyping Loci *De novo* From Short-Read Sequences**. *G3: Genes, Genomes, Genetics* 2011. **1**(3): 171–182.

[7] Eaton DA: **PyRAD: assembly of *de novo* RADseq loci for phylogenetic analyses**. *Bioinformatics* 2014. **30**(13): btu121.

[8] Eaton DA. ipyrad documentation. `http://ipyrad.readthedocs.io/`, Date accessed: 03.01.2017.

[9] Puritz JB, Hollenbeck CM *et al.*: **dDocent: a RADseq, variant-calling pipeline designed for population genomics of non-model organisms**. *PeerJ* 2014. **2**: e431.

[10] Chong Z, Ruan J *et al.*: **Rainbow: an integrated tool for efficient clustering and assembling RAD-seq reads**. *Bioinformatics* 2012. **28**(21): 2732–2737.

[11] Eaton DA. PyRAD Homepage. `http://dereneaton.com/software/pyrad/`, Date accessed: 05.07.2016.

[12] Eaton DA. Example *de novo* RADseq assembly using pyRAD. `http://nbviewer.ipython.org/gist/dereneaton/1f661bfb205b644086cc/tutorial_RAD_3.0.ipynb`, Date accessed: 18.10.2016.

[13] Eaton DA. Tutorial: single-end ddRAD analysis (or other two-cutter based datatypes) pyRAD v.3.0.4. `http://nbviewer.ipython.org/dc6241083c912519064e/tutorial_ddRAD_3.0.4.ipynb`, Date accessed: 18.10.2016.

[14] Eaton DA. Tutorial: paired-end ddRAD analysis w/ merged reads (or other two-cutter based datatypes) pyRAD v.3.0.4. http://nbviewer.ipython.org/dc6241083c912519064e/tutorial_pairddRAD_3.0.4.ipynb, Date accessed: 18.10.2016.

[15] Eaton DA. Tutorial: paired-end ddRAD w/ merged reads (or other two-cutter based datatypes) pyRAD v.3.0.4. http://nbviewer.ipython.org/gist/dc6241083c912519064e/tutorial_pairddRAD_3.0.4-merged.ipynb, Date accessed: 18.10.2016.

[16] University of East Anglia. NCBI SRA: SRP035472. http://www.ncbi.nlm.nih.gov/Traces/study/?acc=SRP035472, Date accessed: 25.10.2016.

[17] Juniata College. NCBI SRA: SRP068035. http://www.ncbi.nlm.nih.gov/Traces/study/?acc=SRP068035, Date accessed: 25.10.2016.

[18] Mastretta-Yanes A, Zamudio S *et al.*: **Gene Duplication, Population Genomics, and Species-Level Differentiation within a Tropical Mountain Shrub**. *Genome Biology and Evolution* 2014. **6**(10): 2611–2624.

[19] Mastretta-Yanes A, Arrigo N *et al.*: **Restriction site-associated DNA sequencing, genotyping error estimation and *de novo* assembly optimization for population genetic inference**. *Molecular Ecology Resources* 2015. **15**(1): 28–41.

[20] Fowler BL and Buonaccorsi VP: **Genomic characterization of sex-identification markers in *Sebastes carnatus* and *S. chrysomelas* rockfishes**. *Molecular Ecology* 2016. **25**(10): 2165–2175.

[21] Zhang J, Kobert K *et al.*: **PEAR: a fast and accurate Illumina Paired-End reAd mergeR**. *Bioinformatics* 2014. **30**(5): 614–620.

[22] Catchen JM. Stacks Google Group: Does Stacks requires [sic] equal length of Rad-Seq reads? https://goo.gl/LHhGU5, Date accessed: 10.10.2016.

[23] Catchen JM. Stacks Homepage. http://catchenlab.life.illinois.edu/stacks, Date accessed: 14.07.2016.

[24] Catchen JM. Tutorial: How do the major Stacks parameters control the *de novo* formation of stacks and loci? http://catchenlab.life.illinois.edu/stacks/param_tut.php, Date accessed: 18.10.2016.

[25] Edgar RC: **MUSCLE: multiple sequence alignment with high accuracy and high throughput**. *Nucleic Acids Research* 2004. **32**(5): 1792–1797.

[26] Rognes T, Flouri T *et al.*: **VSEARCH: a versatile open source tool for metagenomics**. *PeerJ* 2016. **4**: e2584.

[27] Edgar RC: **Search and clustering orders of magnitude faster than BLAST**. *Bioinformatics* 2010. **26**(19): 2460–2461.

[28] van der Walt S, Colbert SC *et al.*: **The NumPy Array: A Structure for Efficient Numerical Computation**. *Computing in Science & Engineering* 2011. **13**(2): 22–30.

[29] Jones E, Oliphant T *et al.* SciPy: Open source scientific tools for Python. `http://www.scipy.org/`, Date accessed: 25.10.2016.

[30] Pandas homepage. `http://pandas.pydata.org/talks.html`, Date accessed: 10.04.2017.

[31] Li H and Durbin R: **Fast and accurate short read alignment with Burrows–Wheeler transform**. *Bioinformatics* 2009. **25**(14): 1754–1760.

[32] Sphinx homepage. `http://www.sphinx-doc.org/en/stable/develop.html`, Date accessed: 10.04.2017.

[33] Ponstingl H. SMALT Sanger Institute. `http://www.sanger.ac.uk/science/tools/smalt-0`, Date accessed: 10.04.2017.

[34] The IPython Development Team - ipython/ipyparallel: Interactive Parallel Computing with IPython. `https://github.com/ipython/ipyparallel`, Date accessed: 10.04.2017.

[35] Li H, Handsaker B *et al.*: **The Sequence Alignment/Map format and SAMtools**. *Bioinformatics* 2009. **25**(16): 2078–2079.

[36] Project Jupyter homepage. `http://jupyter.org/index.html`, Date accessed: 10.04.2017.

[37] Quinlan AR and Hall IM: **BEDTools: a flexible suite of utilities for comparing genomic features**. *Bioinformatics* 2010. **26**(6): 841–842.

[38] Cython homepage. `http://cython.org/`, Date accessed: 10.04.2017.

[39] Hierarchical Data Format, version 5. `http://www.hdfgroup.org/HDF5/`, Date accessed: 10.04.2017.

[40] H5py homepage. `http://www.h5py.org/`, Date accessed: 10.04.2017.

[41] MPICH — High-Performance Portable MPI. `http://www.mpich.org/`, Date accessed: 10.04.2017.

[42] Shead TM - toyplot: Interactive plotting for Python. `https://github.com/sandialabs/toyplot`, Date accessed: 10.04.2017.

[43] Puritz JB. dDocent homepage. `http://ddocent.com/`, Date accessed: 03.03.2017.

[44] Garrison E and Marth G: **Haplotype-based variant detection from short-read sequencing**. *arXiv preprint arXiv:1207.3907* 2012.

[45] Garrison E - vcflib: A C++ library for parsing and manipulating VCF files. `https://github.com/ekg/vcflib`, Date accessed: 12.04.2017.

[46] Gnuplot homepage. `http://www.gnuplot.info`, Date accessed: 12.04.2017.

[47] Bolger AM, Lohse M *et al.*: **Trimmomatic: a flexible trimmer for Illumina sequence data**. *Bioinformatics* 2014. **30**(15): 2114.

[48] Li H - seqtk: Toolkit for processing sequences in FASTA/Q formats. https://github.com/lh3/seqtk, Date accessed: 12.04.2017.

[49] Dickey TE. mawk – pattern scanning and text processing language. http://invisible-island.net/mawk/, Date accessed: 12.04.2017.

[50] Fu L, Niu B *et al.*: **CD-HIT: accelerated for clustering the next-generation sequencing data**. *Bioinformatics* 2012. **28**(23): 3150–3152.

[51] Barnett DW, Garrison EK *et al.*: **BamTools: a C++ API and toolkit for analyzing and managing BAM files**. *Bioinformatics* 2011. **27**(12): 1691–1692.

[52] Danecek P, Auton A *et al.*: **The variant call format and VCFtools**. *Bioinformatics* 2011. **27**(15): 2156–2158.

[53] Tange O: **GNU Parallel - The Command-Line Power Tool**. *The USENIX Magazine* 2011. **36**(1): 42–47, Frederiksberg, Denmark.

[54] Camacho C, Madden T *et al.*: **BLAST Command Line Applications User Manual**. *BLAST® Help. Bethesda, MD: National Center for Biotechnology Information (US)* 2008.

[55] Chen H: *VennDiagram: Generate High-Resolution Venn and Euler Plots*, 2016. http://CRAN.R-project.org/package=VennDiagram. R package version 1.6.17. Date accessed: 14.10.2016.

[56] Felsenstein J: {**PHYLIP**}**: phylogenetic inference package, version 3.5 c.** 1993.

[57] Scharmann M - vcf_to_phylip.py. https://github.com/mscharmann/python/blob/master/vcf_to_phylip.py, Date accessed: 20.03.2017.

[58] dDocent User Help Forum: Problem with new SNP-calling implementation. https://goo.gl/p42W8U, Date accessed: 20.05.2017.

[59] Stamatakis A: **RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies**. *Bioinformatics* 2014. **30**(9): 1312–1313.

[60] Revell LJ: **phytools: an R package for phylogenetic comparative biology (and other things)**. *Methods in Ecology and Evolution* 2012. **3**(2): 217–223.

[61] Paradis E, Claude J *et al.*: **APE: Analyses of Phylogenetics and Evolution in R language**. *Bioinformatics* 2004. **20**(2): 289–290.

[62] Penny D and Hendy MD: **The Use of Tree Comparison Metrics**. *Systematic Zoology* 1985. **34**(1): 75–82.

[63] R Core Team: *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. https://www.R-project.org/. Date accessed: 10.10.2016.

[64] Auguie B: *gridExtra: Miscellaneous Functions for "Grid" Graphics*, 2016. https://CRAN.R-project.org/package=gridExtra. R package version 2.2.1. Date accessed: 10.10.2016.

[65] Raj A, Stephens M *et al.*: **fastSTRUCTURE: Variational Inference of Population Structure in Large SNP Data Sets**. *Genetics* 2014. **197**(2): 573–589.

[66] Rosenberg NA: **DISTRUCT: a program for the graphical display of population structure**. *Molecular Ecology Notes* 2004. **4**(1): 137–138.

[67] Purcell S, Neale B *et al.*: **PLINK: A Tool Set for Whole-Genome Association and Population-Based Linkage Analyses**. *The American Journal of Human Genetics* 2007. **81**(3): 559–575.

[68] Chang C. PLINK 1.9 home. `https://www.cog-genomics.org/plink/1.9/`, Date accessed: 20.03.2017.

[69] Hohenlohe PA, Bassham S *et al.*: **Population Genomics of Parallel Adaptation in Threespine Stickleback using Sequenced RAD Tags**. *PLoS Genetics* 2010. **6**(2): e1000862.

[70] Lynch M: **Estimation of Nucleotide Diversity, Disequilibrium Coefficients, and Mutation Rates from High-Coverage Genome-Sequencing Projects**. *Molecular Biology and Evolution* 2008. **25**(11): 2409–2419.

[71] Kent WJ: **BLAT—The BLAST-Like Alignment Tool**. *Genome Research* 2002. **12**(4): 656–664.

[72] Li L, Stoeckert CJ *et al.*: **OrthoMCL: Identification of Ortholog Groups for Eukaryotic Genomes**. *Genome Research* 2003. **13**(9): 2178–2189.

[73] ClientLogin for Installed Applications. `https://developers.google.com/identity/protocols/AuthForInstalledApps`, Date accessed: 26.11.2017.

[74] Krueger F. Trim Galore! `http://www.bioinformatics.babraham.ac.uk/projects/trim_galore/`, Date accessed: 26.10.2017.

[75] Catchen J, Cresko WA *et al.*: *Stacks Manual*, 2016. `http://catchenlab.life.illinois.edu/stacks/manual/`. Date accessed: 14.07.2016.

[76] How to read a phylogenetic tree . `http://epidemic.bio.ed.ac.uk/how_to_read_a_phylogeny`, Date accessed: 04.11.2016.

[77] Three questions regarding the interpretation and meaning of branch lengths. `https://goo.gl/9gsVQ9`, Date accessed: 04.11.2016.

[78] Anaconda Inc. Conda homepage. `https://conda.io/docs/`, Date accessed: 22.04.2017.

[79] Narum SR, Buonaccorsi VP *et al.*: **Genetic Divergence between Gopher Rockfish (*Sebastes carnatus*) and Black and Yellow Rockfish (*Sebastes chrysomelas*)**. *Copeia* 2004. **2004**(4): 926–931.

[80] Weir BS and Cockerham CC: **ESTIMATING F-STATISTICS FOR THE ANALYSIS OF POPULATION STRUCTURE**. *Evolution* 1984. **38**(6): 1358–1370.

[81] Eaton DA. PyRAD Users Google Group: pyrad output contains fabricated sequences. `https://groups.google.com/forum/#!topic/pyrad-users/bSR-4jh_-pA`, Date accessed: 18.10.2016.

[82] Felsenstein J. Molecular Sequence Programs. http://evolution.genetics.washington.edu/phylip/doc/sequence.html, Date accessed: 09.02.2017.

[83] Pritchard JK, Wen X *et al.*: *Documentation for structure software: Version 2.3*, 2010. http://web.stanford.edu/group/pritchardlab/structure_software/release_versions/v2.3.4/structure_doc.pdf. Date accessed: 13.11.2016.

Appendix

Appendix A

A.1 Pipeline Parameters

Table A.1: Summary of the Stacks denovo_map parameter.

Parameter	Description
-N	Maximum distance between aligned secondary reads and primary stacks.
-T	Number of threads to execute.
-B	The database, in which the data is loaded.
-t	Remove highly repetitive RAD-Tags.
-b	Batch id.
-D	Description of the batch.
-o	Path to the output directory.
-O	Path to the population map.
−−samples	Path to the directory of the samples. Only samples of the population map are used.

Table A.2: Summary of the Stacks process_radtags parameter.

Parameter	Description
-P	The files in the input directory (specified with -p) are paired.
-p	Path to the directory of the raw data.
-o	Path to the output directory.
-b	Path to the barcode file.
-E	Encoding of the quality score (eg. phred33).

-r	Rescue barcodes. In case of an error barcodes and the RAD-Tag are rescued if the barcode has a distance of less than 2 (2 is the default value, but it can be changed) to a barcode in the barcode file.
-c	Clean the data.
-q	Discard low quality reads.
-i	The type of the input files.
-renz_1	The used restriction enzyme.
-renz_2	In case of double digestion the second used restriction enzyme.

Table A.3: Summary of the PyRAD parameter.

Number	Name	Description	Default value
1	working directory	Path to the working directory. By default "./" which is the current directory.	./
2	Raw sequence data	Path to the directory with the raw sequence data in it. In case of paired end the pairs should only differ in "R1" and "R2".	./*.fastq.gz
3	Barcodes	Location of the barcodes.	./*.barcodes
4	Vsearch / Usearch	The command to execute vsearch or usearch.	vsearch
5	Muscle	The command to execute muscle.	muscle
6	Restriction cut-site overhang	Overhang(s) of the Restriction enzyme cut-site(s). In case of ddRAD both overhangs are seperated by a comma and the first one is from the common cutter and the second from the rare cutter.	TGCAG
7	Processors	The number of jobs that will run parallel. For the non-clustering steps this equals the number of used processing cores. For the clustering steps (3 and 6) the number of threads per job can be set with parameter 37.	2

8	Minimum depth	The minimum coverage for a cluster.	6
9	Maximum N	The maximum number of low quality sites (which are represented by a N) in a sequence.	4
10	Clustering threshold	The clustering threshold used by vsearch for the sequence clustering. Entered as a decimal, which is calculated by (raw read length - barcode length - allowed distance between reads)/(raw read length - barcode length).	.88
11	Datatype	Six different options which describe the data set type. Options: rad, ddrad, gbs, pairddrad, pairgbs.	rad
12	Minimum taxon coverage	Minimum number of samples represented in a found locus.	4
13	Maximum shared polymorphic sites	The maximaum number of shared polymorphic sites in a locus. This is used to detect potential paralog.	3
14	Output Prefix	Prefixfor the names of the output files.	c88d6m4p3
Only a few of the optional parameters were used			
18	Path to demultiplexed data	If your data is already cleaned and demultiplex, you can enter the path to the data here and skip step 1.	
21	Strictness of the filter	The strictness of the filtering in the quality filtering step. 0...Only filters by using the quality score of the bases. 1...Trims of cut sites and Illumina adapter. 2...Allows some errors in the cut sites and adapter	0
24	Maximum of heterozygous sites	The maximaum number of heterozygous sites in a consensus sequence.	5

30	Output formats	The wildcard character * can be used to generate all output formats.	
32	Minimum length of trimmed reads	The minimum length a read will be trimmed to.	

Table A.4: Summary of the ipyrad parameter.

Number	Name	Description	Default value
0	Assembly name	Name of the project. Used to name the output directories.	–
1	Project directory	Path to the working directory. By default "./" which is the current directory.	./
2	Raw sequence data	Path to the directory with the raw sequence data in it. In case of paired end the pairs should only differ in "R1" and "R2".	
3	Barcodes	Location of the barcodes.	
4	Path to demultiplexed data	If your data is already cleaned and demultiplex, you can enter the path to the data here and skip step 1.	
5	Assembly method	The assembly method that should be used. (denovo, reference, denovo+reference, denovo-reference)	denovo
6	Reference sequence	The reference sequence if a assembly method with reference sequence is selected.	
7	Datatype	The datatype of the reads. (rad, gbs, ddrad, etc.)	rad
8	Restriction cut-site overhang	Overhang(s) of the Restriction enzyme cut-site(s). In case of ddRAD both overhangs are seperated by a comma and the first one is from the common cutter and the second from the rare cutter.	TGCAG,

9	Maximum N	The maximum number of low quality sites (which are represented by a N) in a sequence.	5
10	Phred Q score offset	Offset of the phred q score	33
11	Min depth statistical	Minimum depth for the statistical base calling	6
12	Min depth majrule	Minimum depth for the majority-rule base calling	6
13	Max depth	Maximum cluter depth within samples	10000
14	Clustering threshold	The clustering threshold used by vsearch for the sequence clustering. Entered as a decimal, which is calculated by (raw read length - barcode length - allowed distance between reads)/(raw read length - barcode length).	.85
15	Max Barcode mismatch	Maximum number of allowed barcode mismatches.	0
16	Filter adapters	Filter for adapters or primers (0=no filtering,1=filtering or 2=stricter filtering)	0
17	Filter min trim length	Minimum length a read must have after the adapter trimming.	35
18	Max alleles consensus	Maximum number of of alleles per site in a consensus sequence.	2
19	Max Ns consensus	Maximum number of N's (uncalled bases) in a consensus sequence. (R1, R2)	5, 5
20	Max Hs consensus	Maximum number of H's (heterozygotes) in a consensus sequence. (R1, R2)	8, 8
21	Min samples locus	Minimum number of samples represented in a found locus.	4
22	Max SNPs locus	Maximum number of SNP's per locus. (R1, R2)	20, 20

23	Max indels locus	Maximum number of indels per locus. (R1, R2)	8, 8
24	Max shared Hs locus	The maximaum number of heterozygous sites per locus.	0.5
25	Edit cutsites	Edit cut-sites (R1, R2)	0, 0
26	Trim overhang	Trim overhang (R1>, <R1, R2>, <R2)	0, 0, 0, 0
27	Output formats	The wildcard character * can be used to generate all output formats.	l, p, s, v
28	Pop assign file	Path to a populations assignment file.	

A.1.1 Example dDocent config file

```
Number of Processors
4
Maximum Memory
0
Trimming
no
Assembly?
yes
Type_of_Assembly
SE
Clustering_Similarity%
0.85
Mapping_Reads?
yes
Mapping_Match_Value
1
Mapping_MisMatch_Value
3
Mapping_GapOpen_Penalty
5
```

```
Calling_SNPs?
yes
Email
emilian.jungwirth@student.tugraz.at
```

Modified `vcf_to_phylip` script

Listing A.1: Modified `parse_vcf` function of the `vcf_to_phylip` script. Throw this modifications an inclusion of variant sites which are longer than just one nucleotide and also variant sites which contain a single indel is achieved.

```
1  def parse_vcf(vcf_file):
2
3    with open(vcf_file, "r") as INFILE:
4      for line in INFILE:
5        if line.startswith("##"):
6          continue
7        if line.startswith("#"):
8          header_line = line.lstrip("#").strip("\n").split("\t")
9          print header_line
10         break
11
12   samples = sorted(header_line[9:])
13
14 ### freebayes vcf line:
15 #  E3379_L96  43  .  ATCG  ATTG,ATCA,GTCA,GTCG  4179.3  .  AB=BLABLA  0/0:1:
16
17 ### STACKS vcf line:
18 #  #CHROM  POS  ID  REF  ALT  QUAL  FILTER  INFO  FORMAT  sample_203-ampullaria-Brunei-3
19 #  un  2312  25  G  A  .  PASS  NS=17;AF=0.529,0.471  GT:DP:GL  1/1:32:.,.,.
20
21   samples_vcf_idx = {}
22   for sample in samples:
23     print sample
24     vcf_idx = header_line.index(sample)
25     samples_vcf_idx[sample] = vcf_idx
26 #  print popdict_vcf_idx
27
28 # now start the main business of walking through the vcf:
29   out_columns = []
30   with open(vcf_file, "r") as INFILE:
31
32     linecnt = 0
33     snpcnt = 0
```

```
    for line in INFILE:
      linecnt += 1
      #print linecnt, snpcnt
      if line.startswith("#"):
        continue
      if len(line) < 2: # empty lines or so
        continue
      fields = line.strip("\n").split("\t")
#begin modification

      column = []
      variants = [fields[3]] + fields[4].split(",")
      max_length_of_variants,longest_variant = max([(len(x),x) for x in variants])
      min_length_of_variants = min([(len(x)) for x in variants])
      if max_length_of_variants > min_length_of_variants + 1:
        #skip multi nucleodide deletions
        continue

      for sample in samples:
        idxes = [int(x) for x in fields[samples_vcf_idx[sample]].split(":")[0].split("/") if
            not x == "." ]
        if len(idxes) > 0:
          alleles = [variants[x] for x in idxes ]
          sequence = ""
          shift=0
          if len(alleles[0]) != len(alleles[1]):
            column = [""] * len(samples)
            break

          for i in range(max_length_of_variants):
            if i < len(alleles[0]):
              nucleotide0 = alleles[0][i-shift]
            else:
              nucleotide0 = "-"
            if i < len(alleles[1]):
              nucleotide1 = alleles[1][i-shift]
            else:
              nucleotide1 = "-"
            nucleotide_longest = longest_variant[i]

            if len(alleles[0]) < max_length_of_variants:
              if nucleotide0 != nucleotide_longest and nucleotide1 != nucleotide_longest:
                nucleotide0 = "-"
                nucleotide1 = "-"
                shift += 1

            nucleotides = [nucleotide0,nucleotide1]
            sequence += get_IUPAC_amb_code ( "".join(sorted(nucleotides)) )
          if shift > 1 :
```

```
82                #something went worng skip: possible indel+ ploymorphism mix
83                column = [""] * len(samples)
84                break
85            column.append( sequence )
86          else:
87            string_val = "-" * max_length_of_variants
88            column.append(string_val)
89        snpcnt += len(column[0])
90        out_columns.append(column)
91  #end modification
92    # columns to rows/lines:
93    outlines = []
94
95    ntaxa = len(samples)
96    len_align = snpcnt
97    header = str(ntaxa) + "␣" + str(len_align)
98    outlines.append(header)
99
100   for idx, sample in enumerate(samples):
101     seq = "".join( [ x[idx] for x in out_columns ] )
102     out_line = sample + "␣␣␣␣" + seq
103     outlines.append(out_line)
104
105   with open(vcf_file + ".phylip-with-indels.txt", "w") as OUTFILE:
106     OUTFILE.write("\n".join(outlines))
```

A.2 Venn-Diagrams

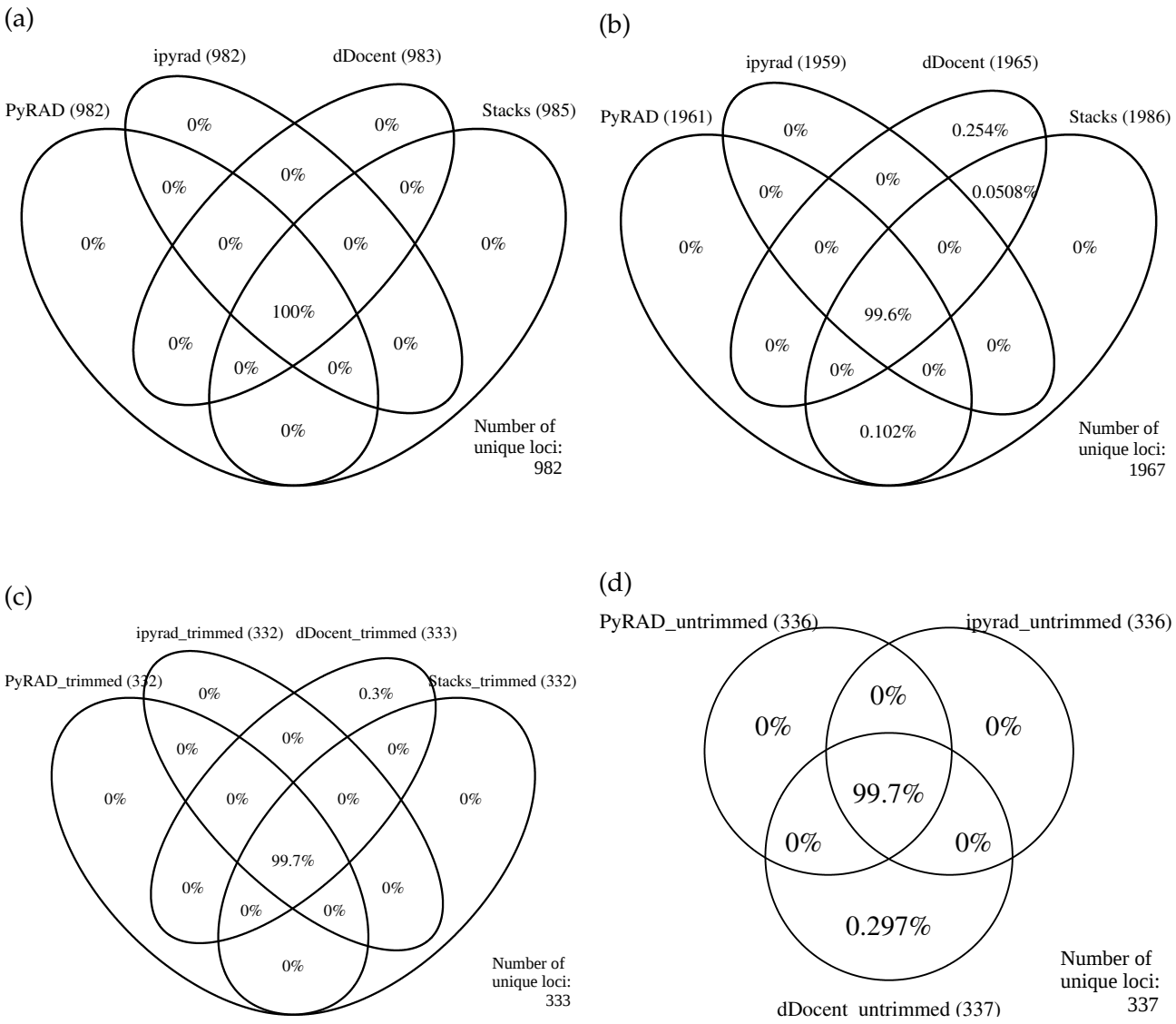

Figure A.1: Venn diagrams: Overlap of the usable consensus sequences in percentages for the rest of the simulated data set. The presented simulated data sets are: SE ddRAD Tutorial v.3.0.4 (a), PE ddRAD Tutorial v.3.0.4 (b) and PE ddRAD w/ merged reads Tutorial v.3.0.4 with trimmed (c) and untrimmed (d) reads.

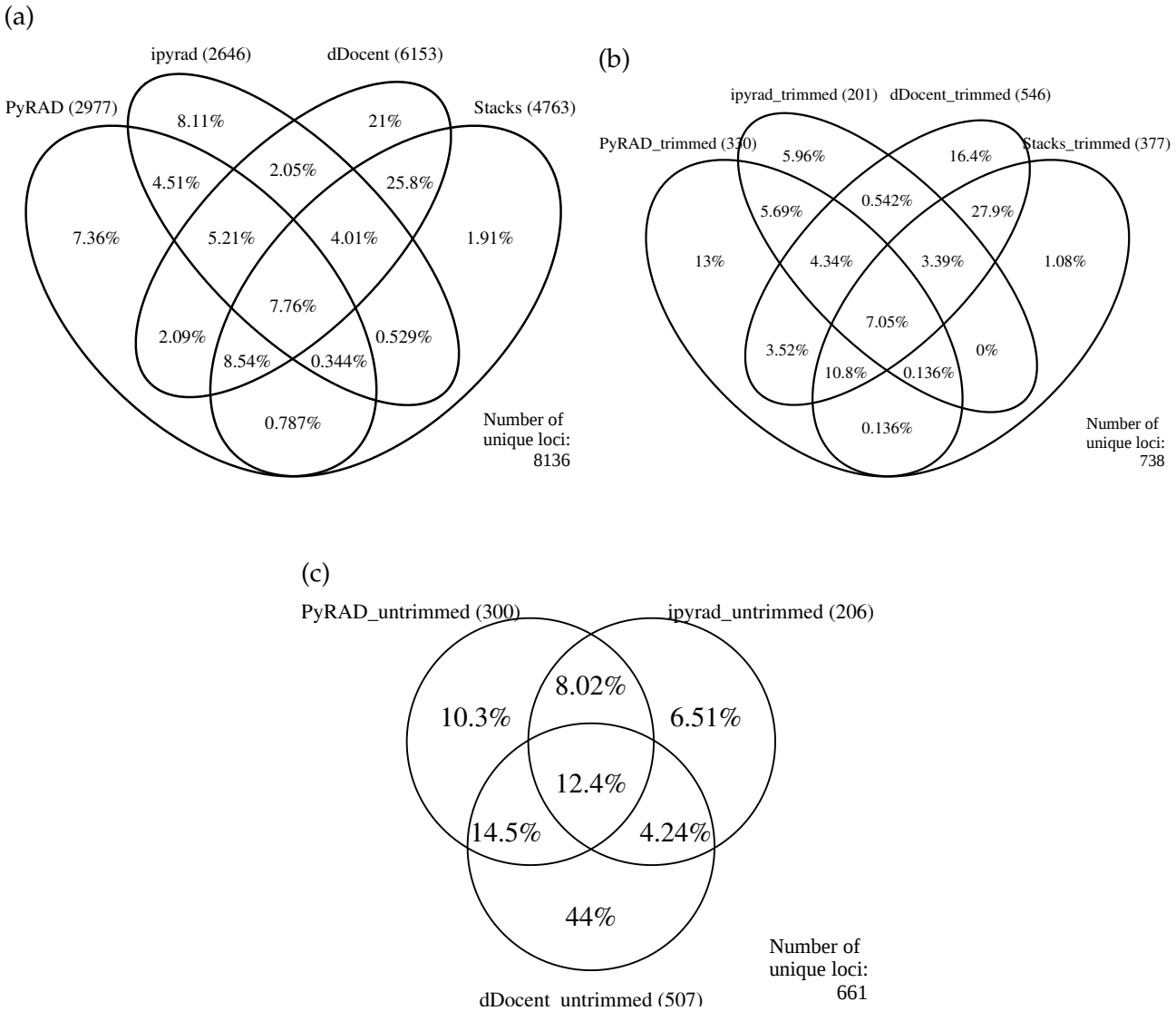

Figure A.2: Venn diagrams for the rest of the empirical data sets. The presented empirical data sets are: SRP035472 (Ber-SE) (a), SRP068035 assembled trimmed (Seb-PE_m trimmed) (b) and SRP068035 assembled untrimmed (Seb-PE_m trimmed) (c).

A.3 Phylogenetic Trees

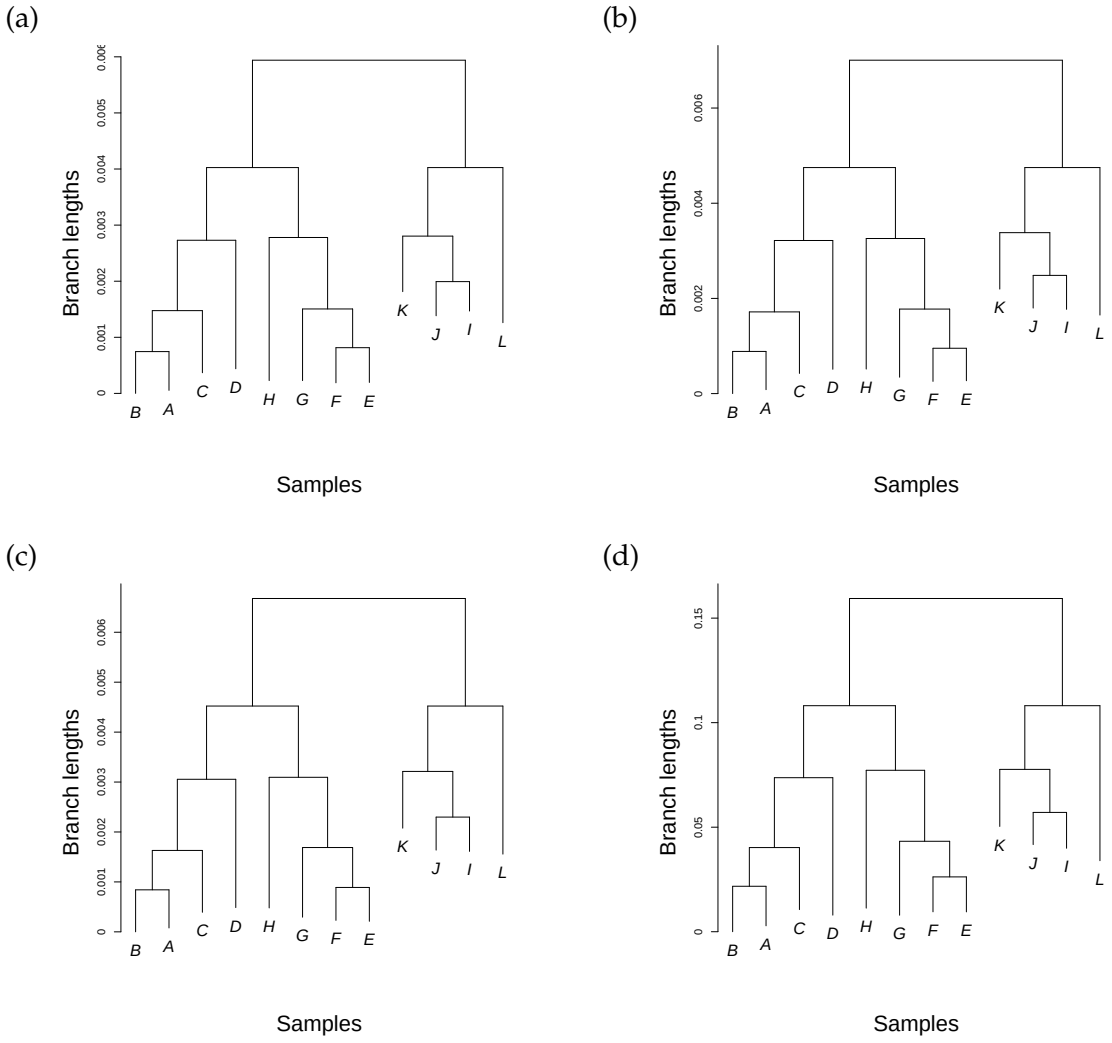

Figure A.3: Species trees of the SE ddRAD Tutorial v.3.0.4 analysis created with the phylip files of the Stacks (a), PyRAD (b), ipyrad (c) and dDocent (d). The Stacks phylip file was created from 1,019 loci and each phylip file of PyRAD, ipyrad and dDocent was from 1,000 loci.

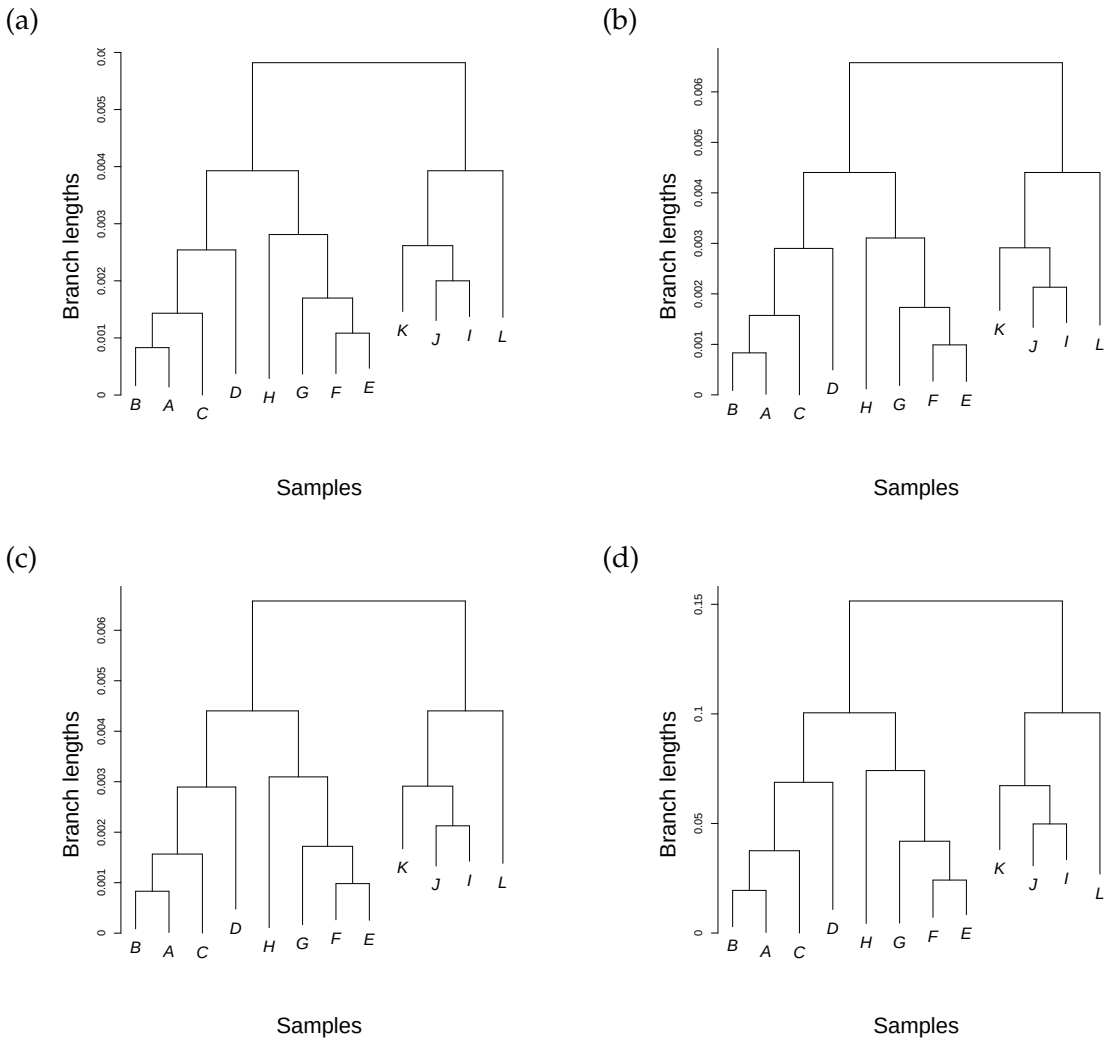

Figure A.4: Species trees of the PE ddRAD Tutorial v.3.0.4 analysis created with the phylip files of the Stacks (a), PyRAD (b), ipyrad (c) and dDocent (d). The Stacks phylip file was created from 2,142 loci, the PyRAD phylip file from 2,002 loci, the ipyrad phylip file from 2000 loci and the dDocent phylip file also from 2000 loci.

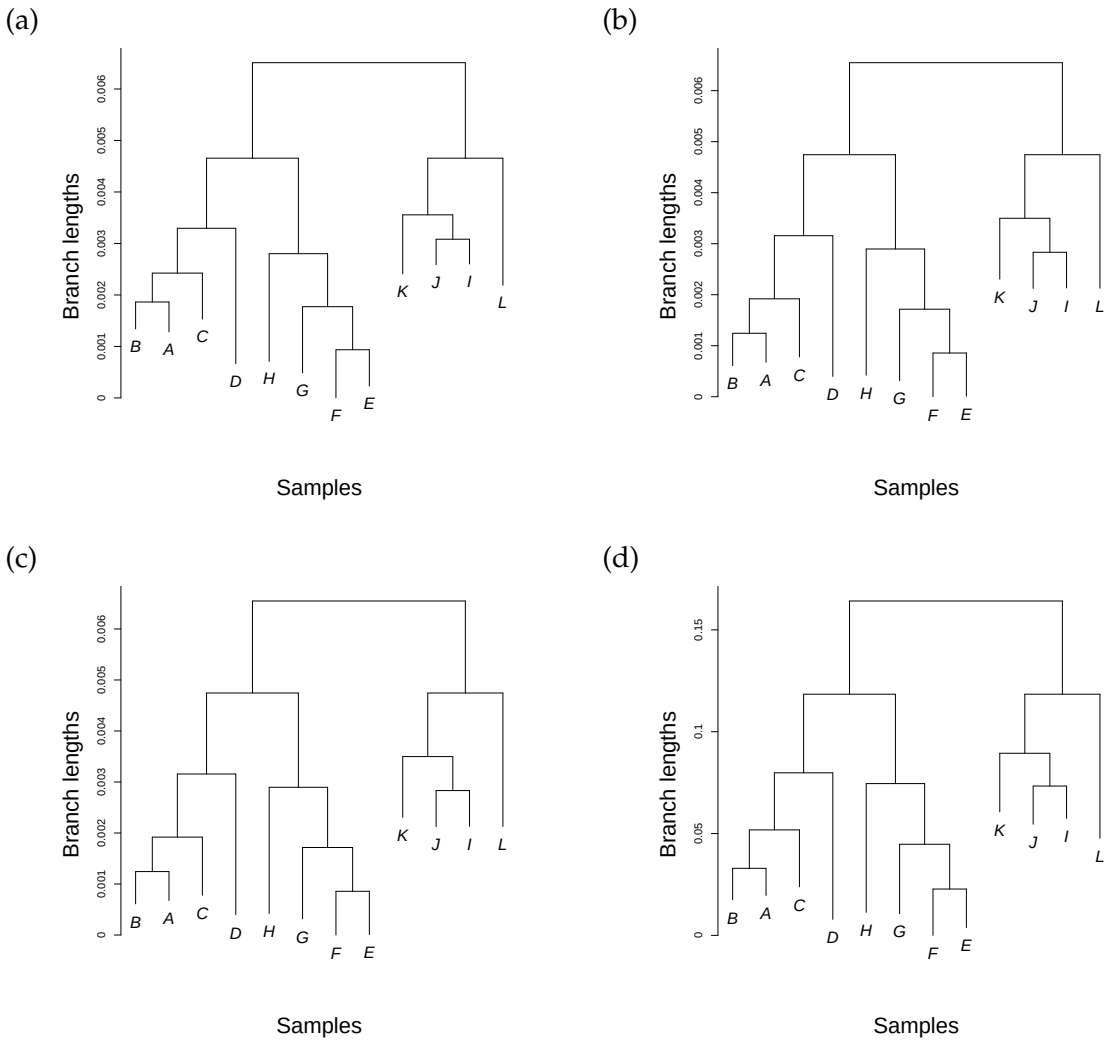

Figure A.5: Species trees of the PE ddRAD w/ merged reads Tutorial v.3.0.4 analysis created with the phylip files of the Stacks (a), PyRAD (b), ipyrad (c) and dDocent (d). The Stacks phylip file was created from 348 loci, the PyRAD phylip file from 340 loci, the ipyrad phylip file from 340 loci and the dDocent phylip file from 341 loci.

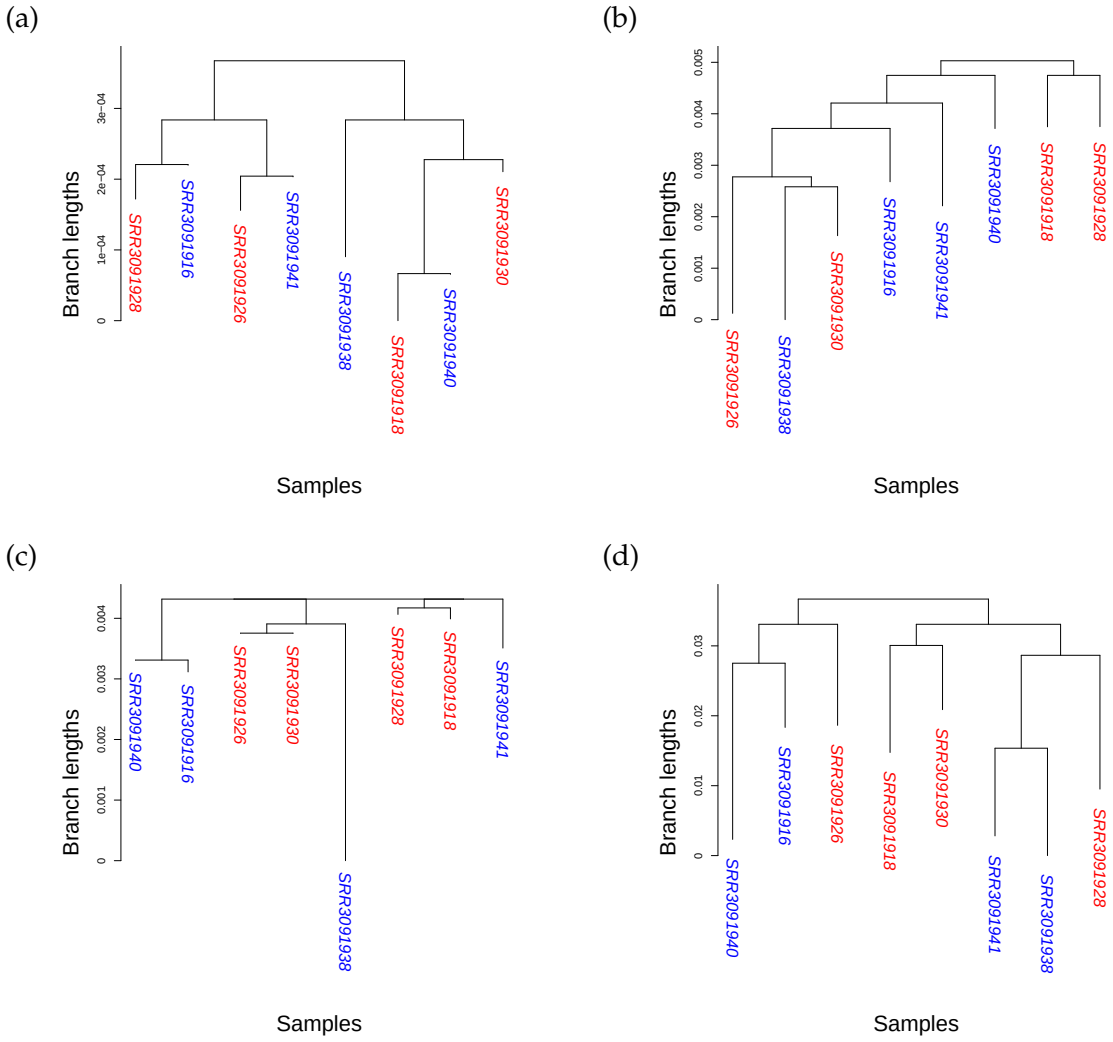

Figure A.6: Species trees of the Seb-PE_m trimmed analysis created with the phylip files of the Stacks (a), PyRAD (b), ipyrad (c) and dDocent (d). The Stacks phylip file was created from 893 loci, the PyRAD phylip file from 456 loci, the ipyrad phylip file from 284 loci and the dDocent phylip file from 363 loci (form the `Final.recode.vcf` file). Samples of the *Sebastes chrysomelas* spieces are coloured blue and samples of the *Sebastes carnatus* species are coloured red.

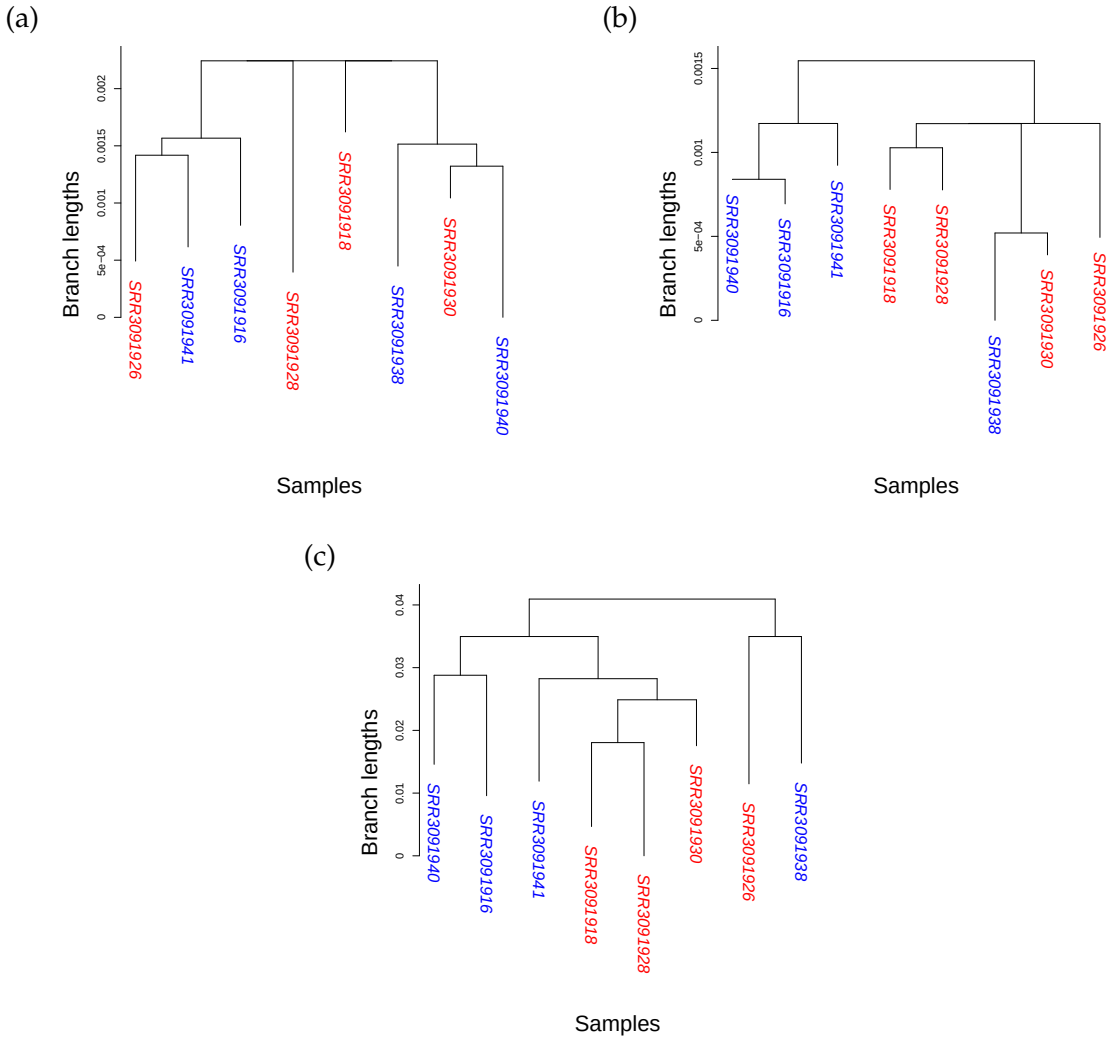

Figure A.7: Species trees of the Seb-PE_m untrimmed analysis created with the phylip files of the PyRAD (a), ipyrad (b) and dDocent (c). The PyRAD phylip file from 384 loci, the ipyrad phylip file from 240 loci and the dDocent phylip file from 317 loci (form the `Final.recode.vcf` file). Samples of the *Sebastes chrysomelas* spieces are coloured blue and samples of the *Sebastes carnatus* species are coloured red.

A.4 Distruct Graphics

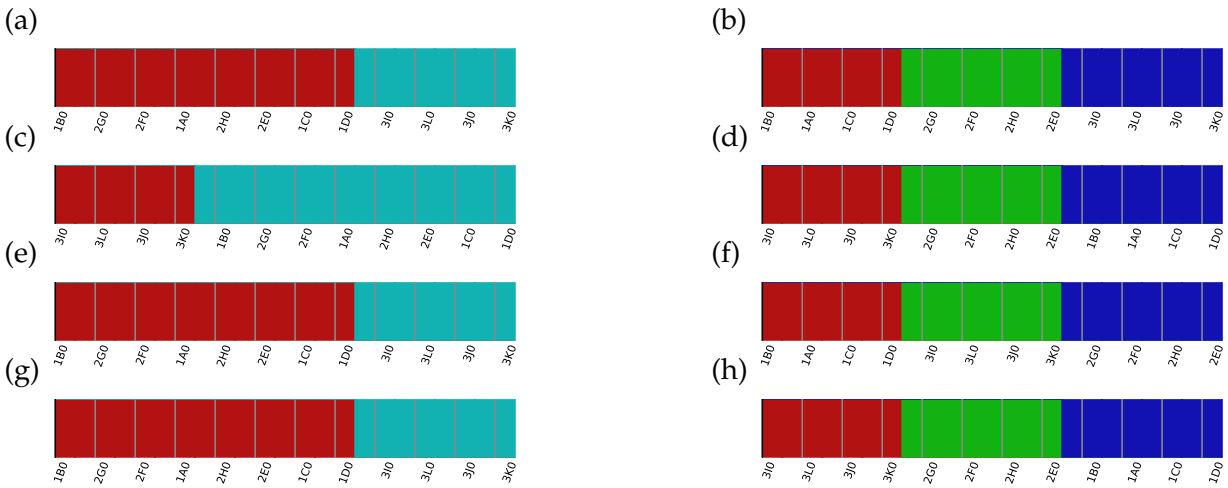

Figure A.8: Distruct graphics for the SE ddRAD Tutorial v.3.0.4 data set. In the figures a (Stacks), c (PyRAD), e (ipyrad) and g (dDocent) the distruct graphics for two populations are shown and in the figures b (Stacks), d (PyRAD), f (ipyrad) and h (dDocent) the distruct graphics for three populations are shown.

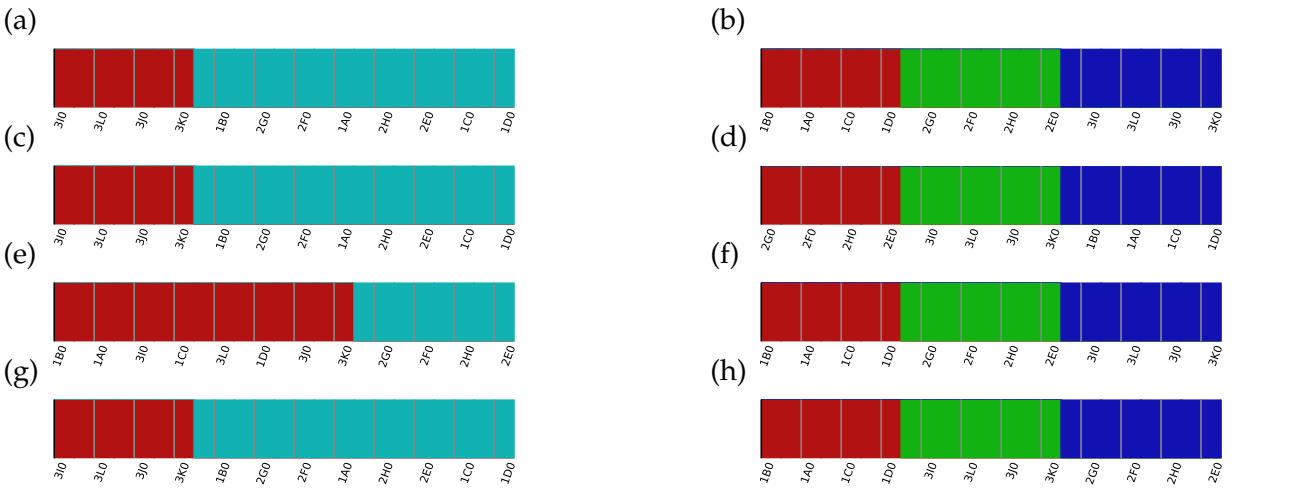

Figure A.9: Distruct graphics for the PE ddRAD Tutorial v.3.0.4 data set. In the figures a (Stacks), c (PyRAD), e (ipyrad) and g (dDocent) the distruct graphics for two populations are shown and in the figures b (Stacks), d (PyRAD), f (ipyrad) and h (dDocent) the distruct graphics for three populations are shown.

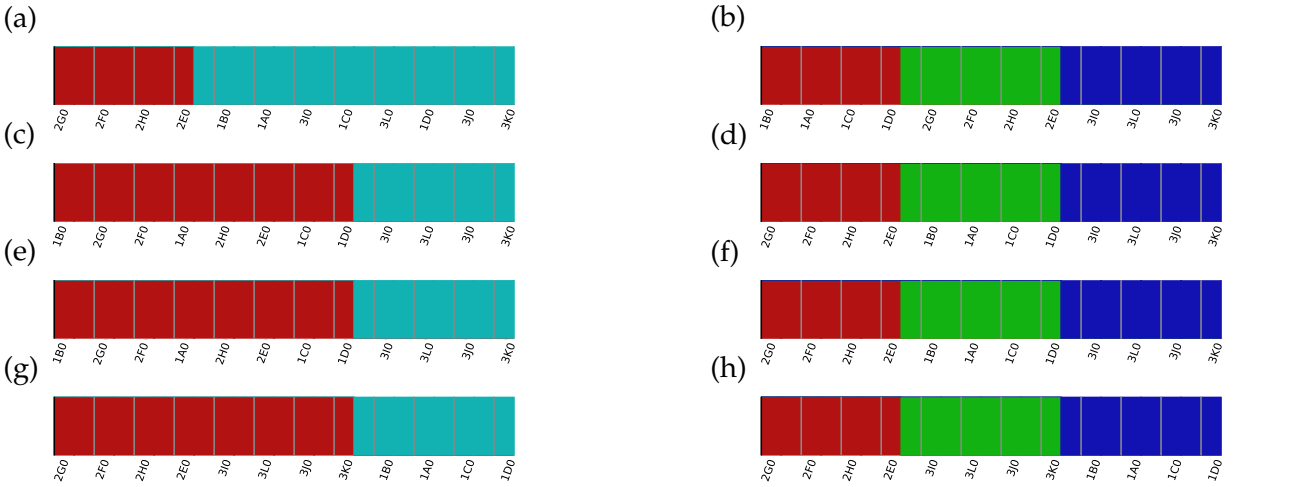

Figure A.10: Distruct graphics for the PE ddRAD w/ merged reads Tutorial v.3.0.4 data set. In the figures a (Stacks), c (PyRAD), e (ipyrad) and g (dDocent) the distruct graphics for two populations are shown and in the figures b (Stacks), d (PyRAD), f (ipyrad) and h (dDocent) the distruct graphics for three populations are shown.

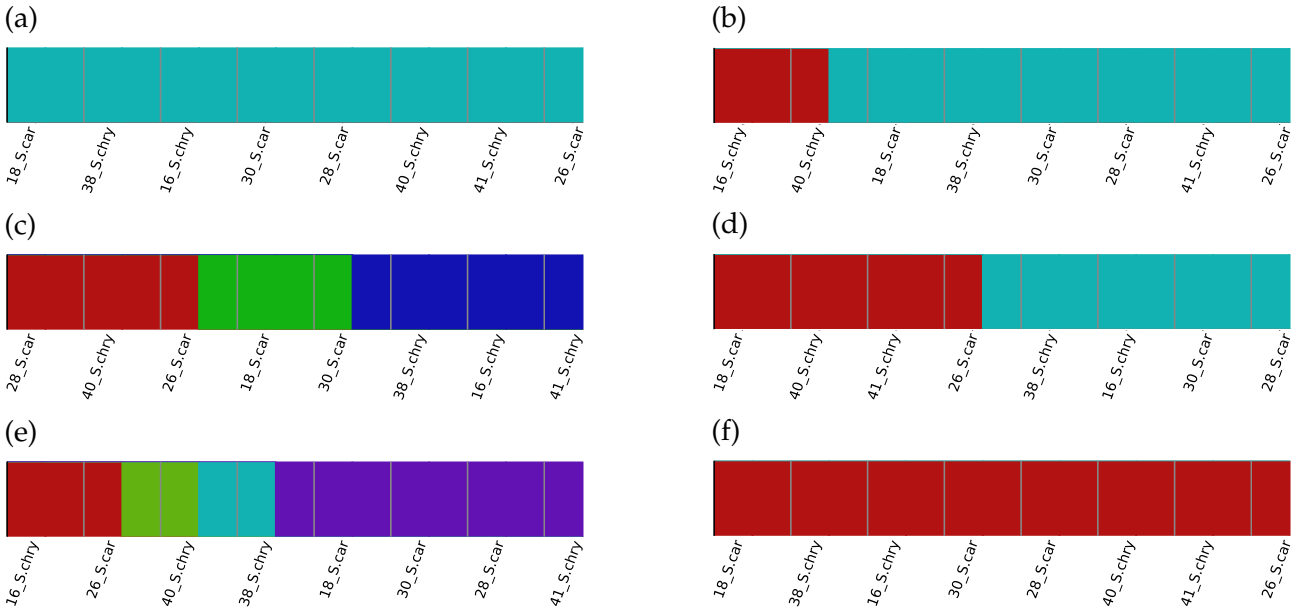

Figure A.11: Distruct graphics for the Seb-PE_m trimmed data set. The samples are labelled with the last two digits of their Run number and an abbreviation for their species. The species are abbreviated with S.car for *Sebastes carnatus* and S.chry for *Sebastes chrysomelas*. The shown distruct plots are from: Stacks with 2 population (a), PyRAD with 2 populations (b), PyRAD with 3 populations (c), ipyrad with 2 populations (d), ipyrad with 4 populations (e) and dDocent with 2 populations (f).

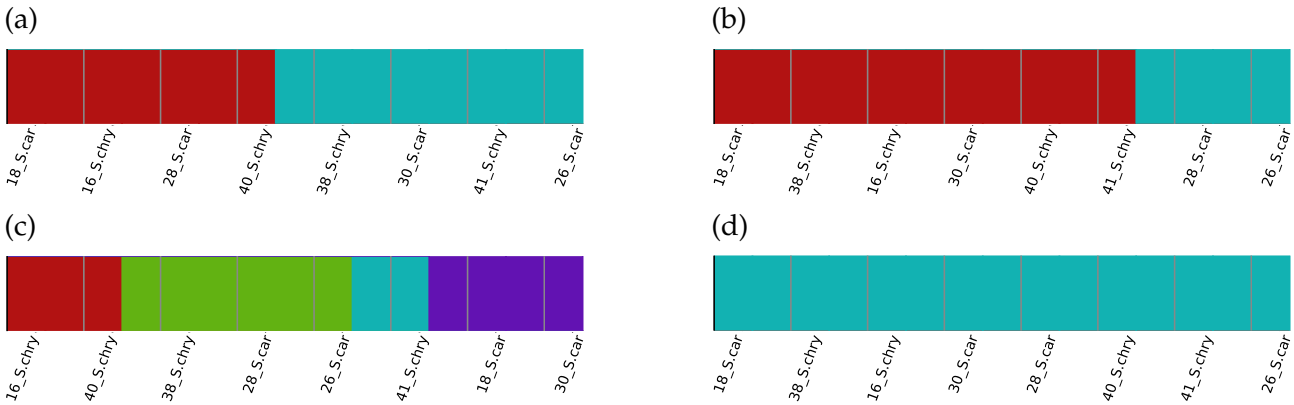

Figure A.12: Distruct graphics for the Seb-PE_m untrimmed data set. The samples are labelled with the last two digits of their Run number and an abbreviation for their species. The species are abbreviated with S.car for *Sebastes carnatus* and S.chry for *Sebastes chrysomelas*. The shown distruct plots are from: PyRAD with 2 populations (a), ipyrad with 2 populations (b), ipyrad with 4 populations (c) and dDocent with 2 populations (d).

A.5 Structure-File

In a Structure file the data set is arranged in a matrix. The individual samples are in rows and the loci are in columns. The alleles of a diploid organism can either be stored in two consecutive rows, where each locus is in one column, or in one row, where each locus consists of two consecutive columns [83].

The following example for a Structure file is taken from [83]. It shows sample data for 7 individuals with 5 loci.

```
1                 loc_a   loc_b   loc_c   loc_d   loc_e
2  George  1      -9      145     66      0       92
3  George  1      -9      -9      64      0       94
4  Paula   1      106     142     68      1       92
5  Paula   1      106     148     64      0       94
6  Matthew 2      110     145     -9      0       92
7  Matthew 2      110     148     66      1       -9
8  Bob     2      108     142     64      1       94
9  Bob     2      -9      142     -9      0       94
10 Anja    1      112     142     -9      1       -9
11 Anja    1      114     142     66      1       94
12 Peter   1      -9      145     66      0       -9
13 Peter   1      110     145     -9      1       -9
14 Carsten 2      108     145     62      0       -9
15 Carsten 2      110     145     64      1       92
```

A.6 Phylip-File

The following two examples are taken from [82]. There are two phylip formats, the interleaved format and the sequential format.

The interleaved format has multiple parts and each part has one line for each sequence.

```
5    42
Turkey     AAGCTNGGGC ATTTCAGGGT
Salmo gairAAGCCTTGGC AGTGCAGGGT
H. SapiensACCGGTTGGC CGTTCAGGGT
Chimp      AAACCCTTGC CGTTACGCTT
Gorilla    AAACCCTTGC CGGTACGCTT

GAGCCCGGGC AATACAGGGT AT
GAGCCGTGGC CGGGCACGGT AT
ACAGGTTGGC CGTTCAGGGT AA
AAACCGAGGC CGGGACACTC AT
AAACCATTGC CGGTACGCTT AA
```

The sequential format has one whole sequence before the next starts.

```
  5    42
Turkey     AAGCTNGGGC ATTTCAGGGT
GAGCCCGGGC AATACAGGGT AT
Salmo gairAAGCCTTGGC AGTGCAGGGT
GAGCCGTGGC CGGGCACGGT AT
H. SapiensACCGGTTGGC CGTTCAGGGT
ACAGGTTGGC CGTTCAGGGT AA
Chimp      AAACCCTTGC CGTTACGCTT
AAACCGAGGC CGGGACACTC AT
Gorilla    AAACCCTTGC CGGTACGCTT
AAACCATTGC CGGTACGCTT AA
```