Received: 4 May 2022 | Revised: 10 January 2024 | Accepted: 23 January 2024

DOI: 10.1111/1755-0998.13933

MOLECULAR ECOLOGY RESOURCES WILEY

RESOURCE ARTICLE

Polly: An R package for genotyping microsatellites and detecting highly polymorphic DNA markers from short-read data

Annabel Perry[1,2] | Dirk Eddelbuettel[3] | Gil Rosenthal[2,4] | Heath Blackmon[2]

[1]Department of Human Evolutionary Biology, Harvard University, Cambridge, Massachusetts, USA

[2]Department of Biology, Texas A&M University, College Station, Texas, USA

[3]Department of Statistics, University of Illinois Urbana-Champagne, Urbana, Illinois, USA

[4]Dipartimento di Biologia, Università degli Studi di Padova, Padova, Italy

Correspondence
Heath Blackmon, Department of Biology, Texas A&M University, College Station, TX, USA.
Email: blackmon@tamu.edu

Funding information
Division of Environmental Biology, Grant/Award Number: 1754669; National Institute of General Medical Sciences, Grant/Award Number: R35GM138098

Handling Editor: Paul A. Hohenlohe

Abstract

Highly polymorphic markers, such as microsatellites, are invaluable for the study of natural populations. However, contemporary methods for genotyping highly polymorphic variants have serious drawbacks that impede their efficiency. We created Polly, an R package with C++ source code that uses Illumina short-read data to genotype microsatellites, detect highly polymorphic variants and identify clusters of highly polymorphic SNPs, indels and microsatellites. We tested Polly on short-read data from *Xiphophorus birchmanni* (Teleostei: Poeciliidae) and *Arabidopsis thaliana*, finding it to be efficient and accurate both for microsatellite genotyping and polymorphic marker detection. This program can be applied to any diploid population for which there exists short-read data and at least one scaffolded reference genome.

KEYWORDS
conservation genetics, ecological genetics, genotype, microsatellite, population genetics–empirical

1 | INTRODUCTION

Polymorphic genetic markers, such as single nucleotide polymorphisms (SNPs), indels and microsatellites are key to a broad range of applications including parentage analysis, pedigree reconstruction and analyses of kinship, diversity, migration and selection (Fischer et al., 2017; Giangregorio et al., 2021; Goudet et al., 2018; Li et al., 2019; Lin et al., 2017; Pemberton, 2008). SNPs, indels and microsatellites have distinct advantages and disadvantages; first, SNPs and most indels are biallelic. This limited variation constrains the questions which can be answered using these markers (Castoe et al., 2012). Though microsatellites are more variable due to their higher mutation rate (Lo et al., 2019), the use of microsatellites for biological research has its own issues. Though recent molecular genetics advances have enhanced affordability and accessibility of microsatellite genotyping (De Barba et al., 2017; Vieira et al., 2016), genotyping microsatellites remains more labour intensive than genotyping of SNPs or indels because each microsatellite locus must be amplified and resolved separately while many 1000s of SNPs or indels can be assessed simultaneously with sequencing (Vieira et al., 2016).

Due to the inconvenience of microsatellite locus detection in some situations, many researchers have created programs to detect microsatellites from computationally stored genomic data (Castoe et al., 2012; Das et al., 2019; Gymrek et al., 2012; Kristmundsdottir et al., 2020; Lo et al., 2019; Metz et al., 2016; Miller et al., 2013;

Pipek et al., 2017; Wang & Wang, 2016). However, none of these programs genotype microsatellites in individuals and thus cannot be used to detect microsatellite loci which are highly polymorphic in the group being studied.

Highly polymorphic DNA markers are very unlikely to have the same genotype in unrelated organisms, so the present inability of the scientific community to efficiently detect highly polymorphic microsatellite loci poses a barrier to studies requiring fine-tuned differentiation between individuals. DNA markers with low variation can only be used to discriminate between very genetically similar individuals if the researcher genotypes many markers but, as mentioned previously, genotyping many low diversity markers is laborious and expensive (Castoe et al., 2012). Instead, many researchers are turning to highly polymorphic DNA markers to identify individuals and kin groups (Castoe et al., 2012; Gymrek et al., 2012; Kristmundsdottir et al., 2020; Lo et al., 2019; Metz et al., 2016; Miller et al., 2013; Pipek et al., 2017).

Currently, no existing microsatellite detection software can detect highly polymorphic microsatellites in all species. All microsatellite detection programs either can only be used for humans (Gymrek et al., 2012), use the outdated SCARF file format (Castoe et al., 2012), require a large sample of scaffolded genome assemblies (Kristmundsdottir et al., 2020; Lo et al., 2019; Pipek et al., 2017) or do not genotype microsatellite loci (Metz et al., 2016; Miller et al., 2013). Software which require scaffolded genomes as input cannot practically be used to detect highly polymorphic microsatellites; despite drastic reductions in sequencing cost and increases in computational efficiency, it is unfeasible to produce the number of scaffolded assemblies that would be needed for most population-level studies. Algorithms to detect microsatellites in short-read data, such as FullSSR and SSR_pipeline (Metz et al., 2016; Miller et al., 2013), only report the locations and not the genotypes of the microsatellite loci and thus cannot be used to identify polymorphic microsatellites.

Furthermore, there does not currently exist a program, to our knowledge, which detects regions enriched in highly polymorphic SNPs, indels and microsatellites. These haplotypes are extremely unlikely to be identical in unrelated individuals, so their detection would be extremely powerful for analyses requiring fine-tuned discrimination between individuals (Goudet et al., 2018; Seroussi & Seroussi, 2007). There is not a compact, efficient method to genotype microsatellites from short-read data, detect polymorphic markers or detect genomic regions of dense polymorphism. Such a program would greatly enhance the efficiency of the plethora of research approaches requiring fine-tuned genetic discrimination between individuals.

We, thus, created Polly, an R package with C++ source code which uses Illumina short-read data to genotype microsatellites, find polymorphic markers and identify genomic regions dense with highly polymorphic microsatellites, SNPs and indels. Polly combines the computational efficiency of C++ with the user-friendliness of the RStudio Graphical User Interface (GUI). The only steps in Polly that require knowledge of coding languages other than R are the command-line Linux pre-processing steps for preparing the input files, and we provide detailed pipelines to perform these steps in a vignette accompanying the package (https://github.com/AnnabelPerry/Polly/tree/main/vignettes), as well as in the Data S1.

To ensure that Polly accurately genotypes microsatellites in species with highly curated and less well-curated genomes, we tested Polly on *Arabidopsis thaliana* and *Xiphophorus birchmanni* sequences with a priori-known genotypes. *Arabidopsis thaliana* is a plant species widely used as an evolutionary model organism while *Xiphophorus* is a fish genus used by a smaller community of researchers to study processes such as hybridization, oncogenesis and sexual selection (Lu et al., 2020). Since a larger community studies *A. thaliana*, the reference genome for this plant species is more closely monitored and more frequently updated than the reference genome for *X. birchmanni* (Li et al., 2017; Lu et al., 2020; Schumer et al., 2018; Sloan et al., 2018). We, thus, tested Polly on both species to ensure this program can accurately genotype microsatellites in genomes of differing curation levels. Polly can output these microsatellite genotypes both in a novel format which provides detailed characteristics of each microsatellite allele in the population, as well as in the commonly used GenePop format (Rousset, 2008).

2 | METHODS

2.1 | Inputs

Polly requires three types of input: a lookup table of microsatellite motifs identified from a scaffolded reference genome, one indexed Binary Alignment Map (BAM) file for each individual to be genotyped and a frequency file describing the frequencies of all SNPs and indels identified in this set of BAM files (Figure 1). To generate the microsatellite lookup table, one must apply micRocounter (Lo et al., 2019) to the reference genome (Figure 1). To prepare the BAM files (Figure 1), one must first map the short reads to the same reference genome using the Burrows–Wheeler Alignment (BWA) mem algorithm, sort reads by coordinates and remove duplicate reads using Picard, realign the reads around indels using the Genome Analysis Toolkit (GATK), then convert the resultant Sequence Alignment Map (SAM) files to BAM format and index the BAMs using samtools (Li et al., 2009; Li & Durbin, 2009; McKenna et al., 2010). Finally, one must apply BCFtools mpileup and call commands to the BAMs and reference to generate a Variant Call Format (VCF) file (Danecek et al., 2011), then apply the VCFtools --freq command to obtain the frequency file (Figure 1). These steps are described in detail in the vignette provided in the Polly R package, as well as in the Data S1.

2.2 | Functions

Polly consists of four functions for identifying or filtering genomic variants: MicroGenotyper(), PollyMicros(), PollySI() and Polly() (Figure 1), as well as a function called PollyPop() which converts

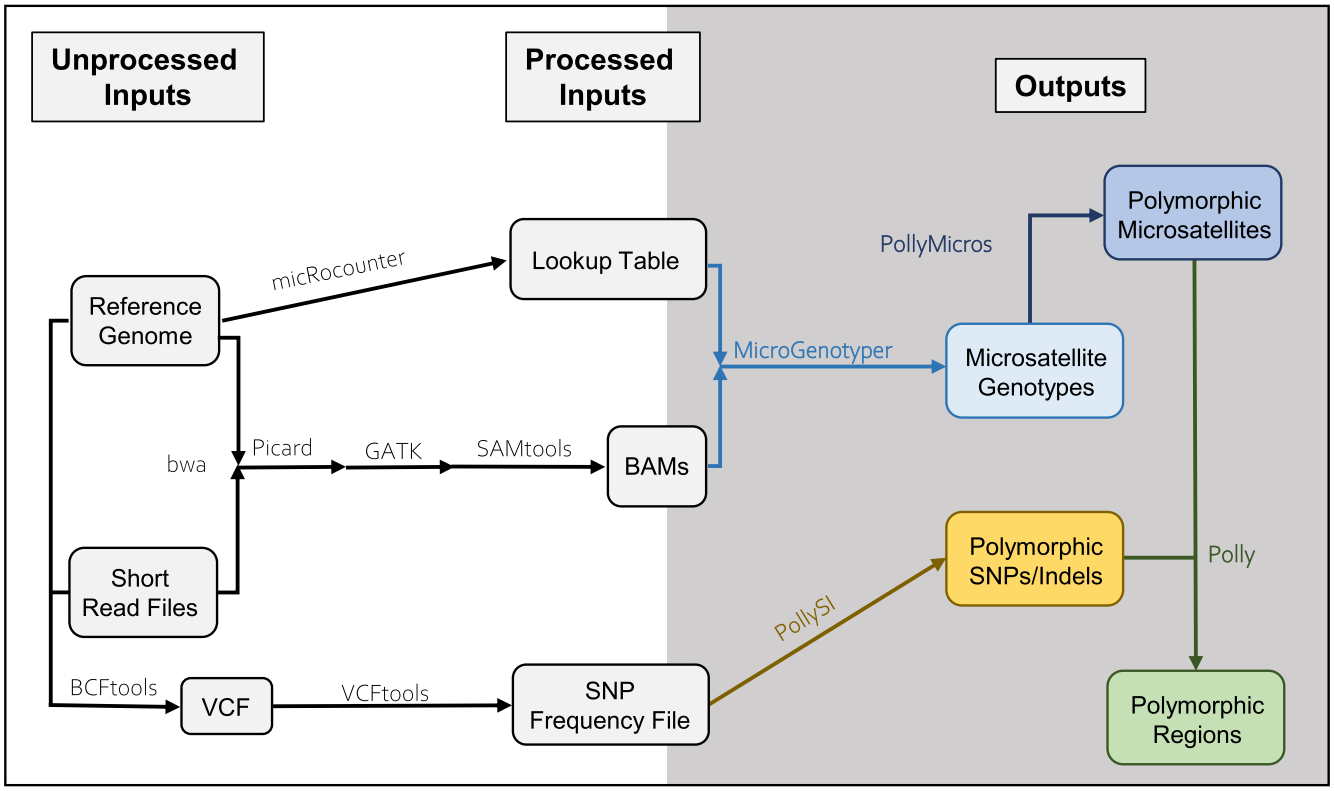

FIGURE 1 Inputs and outputs for both the preprocessing steps (white background) and the in-program functions (grey background) of Polly. Scaffolded reference genomes are converted to lookup tables using micRocounter R package. BAM files are created from the reference genome and short-read files using the command-line bwa mem algorithm, Picard SortSam command, GATK LeftAlignIndels command and SAMtools view and index commands. The frequency file is created by first applying the BCFtools mpileup and call commands to short-read files and a reference genome to generate a VCF file, from which SNP/Indel frequencies are extracted using the VCFtools --freq command. Once the preprocessing steps are complete, Polly's MicroGenotyper function can be applied to the BAMs and lookup table to identify microsatellite genotypes in each BAM, and the resulting files housing the microsatellite genotypes can be run through Polly's PollyMicros command to identify polymorphic microsatellites. Polly's PollySI function can be applied to the frequency file to yield the polymorphic SNPs/indels across the population, and finally, Polly's Polly function can be applied to the outputs of PollySI and PollyMicros to identify the polymorphic regions.

microsatellite genotypes to GenePop format. MicroGenotyper() can genotype microsatellites in whole genomes from one or more individuals, while PollyMicros() can identify polymorphic microsatellites from a sample of individuals genotyped using MicroGenotyper(). PollySI() quantifies variation at SNPs and indels, while Polly() identifies all genomic regions containing multiple polymorphic loci (Figure 1). PollyPop() creates GenePop-formatted files of polymorphic microsatellites from the outputs of PollyMicros() and MicroGenotyper().

The MicroGenotyper() function genotypes microsatellite loci in individuals (Figure 1). MicroGenotyper() uses a lookup table of microsatellite loci generated using micRocounter as a catalogue of loci to be searched for potential microsatellites (See supplemental materials LookupTableCreation.R). Creation of the lookup table requires a single scaffolded genome, but MicroGenotyper() itself genotypes non-scaffolded BAM files (See Data S1 BAMPipeline.txt). MicroGenotyper() also requires the user to specify the names of each output file and the scaffolds to be searched for microsatellites.

MicroGenotyper() uses the HTSlib C++ library to collect reads which map to a microsatellite locus recorded in the lookup table. We used HTSlib version 1.11, compiled using Gnu Compiler Collection (GCC) version 9.3.0. Prior to scanning for microsatellites in the reads, MicroGenotyper() checks the read quality, excluding any reads with MAPQ < 10.

The expected start locus of a microsatellite is the locus listed under "Locus" in the lookup table row corresponding to the read of interest (Figure 2). Since read alterations such as alignment errors or upstream indels may alter the starting coordinate of a microsatellite in a read (Schbath et al., 2012), MicroGenotyper() checks the 5 bp window surrounding each expected start locus rather than just the one starting locus (Figure 2a). MicroGenotyper() starts 2 bp upstream of the expected start locus and scans until it reaches either the first base pair of the expected motif (the motif listed in the corresponding row of the lookup table) or the nucleotide 2 bp downstream of the expected start point, whichever occurs first (Figure 2b).

Once MicroGenotyper() encounters a base pair matching the first base pair of the expected motif, it checks if the downstream base pair in the read matches the second base pair of the known motif (Figure 2c). MicroGenotyper() continues this parallel matching procedure until a full motif is detected, at which point the repeat

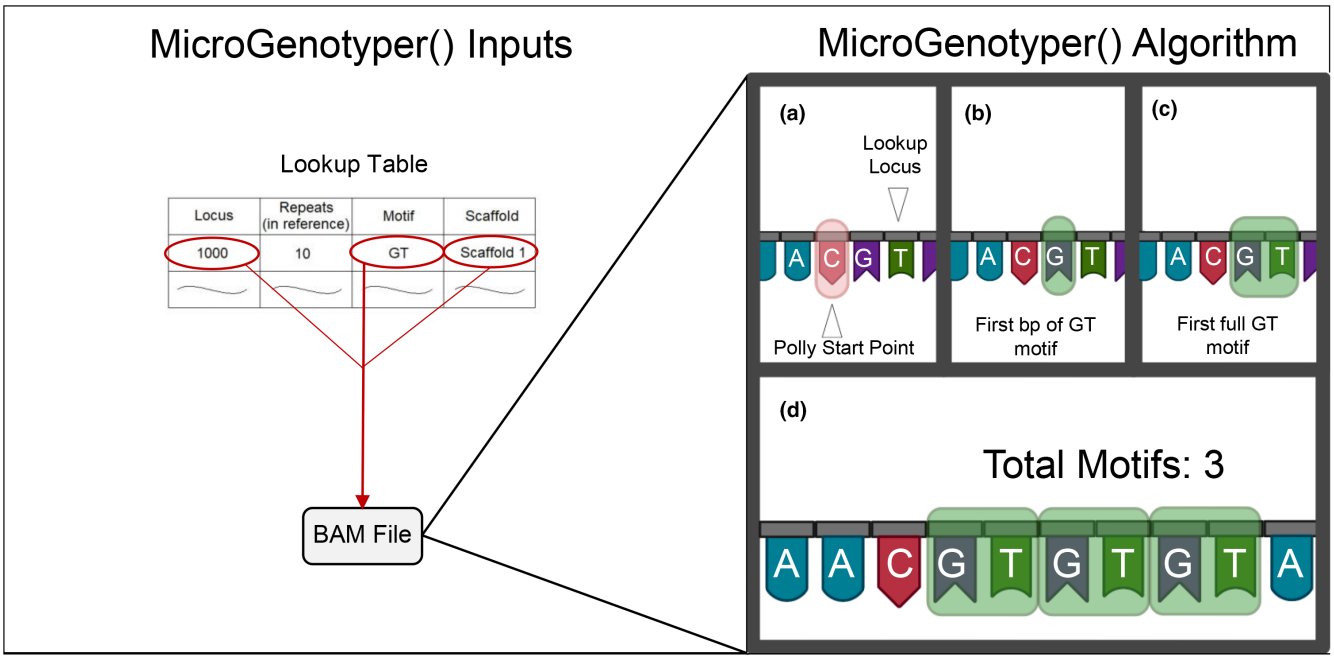

FIGURE 2 Genotyping process of MicroGenotyper(). (a) Genotyping starts 2 bp upstream of the microsatellite start locus indicated in the lookup table. (b) If the first base pair of the lookup motif (in this case, GT) is detected within the 5 bp search window, MicroGenotyper() checks if the downstream base pairs match the remaining base pairs in the lookup motif. (c) If a full motif is detected, the motif count is incremented by 1. (d) Motifs are counted until the repeats are disrupted.

count is incremented by 1 and the matching procedure restarts with the next base pair (Figure 2c). If a discrepancy between motif and read is detected at any point in this process, the program outputs the running motif count as the total repeat count for that read and moves to the next read (Figure 2d). For example, if a "TAG" motif is expected in the sequence "TAGTAGCTAGTAG", Polly will read the sequence as 2 repeats rather than as 4 repeats due to the "C" insertion. This termination of MicroGenotyper()'s genotyping algorithm in response substitutions, insertions or deletions enables it to distinguish between true microsatellites and microsatellites interrupted by SNPs or indels. If a read lacks a motif starting in the 5 bp window, it is skipped.

After every read for a locus has been genotyped, the final genotype of the locus for that individual is determined. If an individual has only two reads for the locus, they are deemed homozygous if both reads have the same repeat count but heterozygous if the repeat counts on the two reads differ. Thus, the minimum required coverage to determine an individual's final genotype is 2×. To determine whether an individual is homozygous or heterozygous when coverage is greater than 2×, we use a simple heuristic. According to this heuristic, an individual is deemed homozygous at a locus if one of the genotypes occurs on twice as many reads as any other genotype. For example, if an individual has 20 reads, 13 of which have allele A, six with allele B, and one with allele C, the individual will be genotyped as homozygous AA. If none of the genotypes are twice as common as all other genotypes, then the individual is deemed heterozygous and the two most common alleles are output as the final genotype for the individual at the locus. For example, if an individual has 20 reads, 12 of which have allele A, seven with allele B, and one with allele C, the individual will be genotyped as heterozygous AB. Once every individual has been genotyped at a locus, the allele frequencies are calculated.

The PollyMicros() function searches for polymorphic microsatellite loci in MicroGenotyper() output files (Figure 1). The user can adjust the polymorphism threshold by inputting an integer specifying the number of alleles which a locus must possess to be considered polymorphic. The user can also specify which scaffolds PollyMicros() should search by inputting a vector of desired scaffolds. PollyMicros() outputs a five-column matrix reporting the scaffold, locus, motif, alleles and allele frequencies of each polymorphic locus. Distinct repeat numbers in the "Alleles" column are separated by commas and correspond to the allele frequencies with the same position in the "Allele Frequencies" column.

With its adjustable polymorphism threshold, PollyMicros() enables the user to control for poor genome assembly in detecting polymorphic microsatellites. If a genome is poorly assembled, some microsatellite loci may not be assembled at all such that the reads corresponding to that locus may erroneously map to a similar locus, inflating the allele counts at this locus. Thus, if the user expects many mis-mapped reads in their input genomes, they can use a high polymorphism threshold in PollyMicros() to control for false positives.

PollySI() is useful for filtering SNP and indel loci, detected using VCFtools (Danecek et al., 2011), to only those which pass a user-specified polymorphism threshold (Figure 1). This function outputs a matrix of SNP and indel loci where the minor allele frequency is greater than or equal to an adjustable threshold between 0 and 0.5.

Its three input arguments are a string denoting the VCFtools frequency file, a vector of desired scaffolds and a numeric allele frequency threshold. PollySI() will only identify polymorphic SNPs and indels occurring on the scaffolds indicated in the vector.

We chose to output microsatellite genotypes in MicroGenotyper() and polymorphic microsatellites in PollyMicros() using a novel format to provide detailed information about the motif, allele frequencies and counts of repeats per alleles. However, for many large-scale population genetics analyses, the GenePop file format is more widely used (Rousset, 2008). There already exist multiple programs to convert VCF-formatted SNP and indel files to GenePop format (Gosselin, 2020; Knaus & Grünwald, 2017), similarly, we created a function to convert the outputs of MicroGenotyper() and PollyMicros() to GenePop format. This function will convert polymorphic microsatellite genotypes from multiple populations into GenePop format allowing results to be imported to other software for downstream analyses.

PollyPop uses CSV files of microsatellites found to be polymorphic within a single population to determine which microsatellite genotypes to extract from the corresponding population's microsatellite genotype files, converts the alleles of these polymorphic microsatellites to GenePop 2-digit codes, then organizes the genotypes of each input population into GenePop format. This function takes four inputs: an output file name, a GenePop header to be reported as the first line in this output file, a vector where each element is the name of a PollyMicros output CSV describing a single population's polymorphic microsatellites and a list of vectors where each vector represents a population and each element of a population's vector is an individual in that population's MicroGenotyper() output CSV name. To decide which PollyMicros() output names and vectors of MicroGenotyper() output names belong to the same population, PollyPop() uses the order in which the names were input. For example, say Population A consists of four individuals whose MicroGenotyper() files are named "A1.csv", "A2.csv", "A3.csv" and "A4.csv" and whose PollyMicros() output file is "PopulationA.csv", while Population B consists of three individuals whose MicroGenotyper() files are named "B1.csv", "B2.csv" and "B3.csv" and whose PollyMicros() output file is "PopulationB.csv". For each population's microsatellite genotype files and polymorphic microsatellite files to be read as belonging to the appropriate population, one would input these files into PollyPop() in the following order:

```
PollyMicrosInputNames <- c("PopulationA.csv",
"PopulationB.csv")
MicroGenotyperInputNames <- list(c("A1.csv", "A2.
csv", "A3.csv", "A4.csv"),
c("B1.csv", "B2.csv", "B3.csv"))
```

PollyPop() starts by using the PollyMicros() outputs to find all polymorphic loci found across all populations. PollyPop() names microsatellite identifiers with a numeric scaffold identifier and the starting locus of the microsatellite, separated by a dash. For each of these unique microsatellite identifiers, PollyPop() then creates a key describing each allele of this identifier alongside a two-digit GenePop code. PollyPop() then uses this key to convert each MicroGenotyper() genotype of this locus to a four-digit diploid GenePop genotype, then organizes those genotypes into a tab-delimited file where the first line is the GenePop header, the second line indicates each of the unique microsatellite identifiers, and the subsequent lines describe each individual's genotype at the locus indicated in the second line. If this microsatellite locus is not present in or not polymorphic in the individual indicated on the current row, 0000 is output as the individuals' genotype. Populations are separated by a line housing only the word "pop". More details on GenePop format can be found here (Rousset, 2008). To reduce memory usage, PollyPop converts each scaffold name to a numeric identifier described in the "Scaffold Key.csv" which is output alongside the GenePop-formatted microsatellite genotype file.

The Polly() function attempts to identify polymorphic markers in positive linkage disequilibrium by grouping polymorphic regions identified by PollySI() and PollyMicros() which occur within a user-specified range of base pairs from a microsatellite locus (Figure 1). Polly() takes as input the desired window of base pairs into which polymorphic loci ought to be grouped, a vector of desired scaffolds, two CSVs housing the outputs of PollySI() and of PollyMicros(), as well as the desired name of the output file. Polly() then uses the loci in these files, which were originally detected using VCFtools and micRocounter, to group polymorphic sites occurring within a user-specified base pair window of a microsatellite locus in the centre of such a cluster.

Polymorphic markers clustered by the Polly() function are output in a novel format which gives detailed information about the genomic architecture. Each row of the final output, csv file represents a cluster of polymorphic loci. The row describes the scaffold on which the loci occur, the total number of loci, total number of alleles, SNP locus count, indel locus count, dinucleotide microsatellite allele count, trinucleotide microsatellite allele count, tetranucleotide microsatellite allele count, pentanucleotide (and above) microsatellite allele count, loci positions, the alleles and frequencies of each locus, the length of the longest allele of the central microsatellite locus and the minimum difference in repeat number between alleles of the central locus. In the allele and frequency column, each locus' alleles are separated by brackets, alleles of the same locus are separated by forward slashes and alleles are separated from their respective frequencies by colons. For example, a diallelic AG SNP locus with 50/50 allele frequencies would be represented as follows: [A:0.5/G:0.5]. Since Polly()'s grouping algorithm searches microsatellites first, the first loci in a region will be microsatellite loci and some regions may consist of single polymorphic microsatellite loci. Polly() also removes SNPs/Indels which overlap with the longest microsatellite allele in the region.

To group loci occurring within a custom range, Polly() starts with a microsatellite (termed the "central" microsatellite because the algorithm will check for markers surrounding this microsatellite). After filtering for polymorphism and ensuring the microsatellite has yet to be recorded, Polly() checks if any other polymorphic

microsatellites occur within the specified range. As microsatellites are clustered together, the beginning and ending loci of their longest possible alleles are recorded to later check for overlap with SNPs and indels. Once all microsatellites occurring within range of the central microsatellite have been recorded, Polly() searches the polymorphic SNPs and indels for any which occur within range of the central microsatellite but do not overlap with the longest allele of any in-range microsatellites. Polly() thus groups polymorphic loci while avoiding redundancy.

2.3 | Accuracy testing

We tested the accuracy of each of the Polly R package's functions by running these algorithms on files with known outputs. To check the accuracy of our microsatellite genotyping algorithm and confirm that it works on species with highly curated genomes and on species with less curated genomes, we then ran MicroGenotyper() on input files derived from *A. thaliana* (highly curated) and *X. birchmanni* (less curated). To check the accuracy of Polly and collect runtime data, we then ran each of the four functions on *X. birchmanni*.

To ensure that the MicroGenotyper() function accurately detects microsatellites in highly curated genomes, we first ran MicroGenotyper() on an *A. thaliana* input file with a priori-known microsatellite genotypes. Since there currently does not exist an open-source panel of a priori-known *A. thaliana* genotypes, we constructed a BAM file with researcher-inserted microsatellites to test the genotyping capabilities of MicroGenotyper(). To create the BAM file, we first downloaded an existing *A. thaliana* short-read file (accession no: SRR18089322) from the National Center for Biotechnology Information's Sequence Read Archive (SRA) and began applying the steps required to obtain a sorted SAM file, described in the vignette and in the Data S1. For the BWA mem step, we mapped the short reads to The Arabidopsis Information Resource's *A. thaliana* reference genome, version 10.1 (N50 = 23.46 MB; contig count = 7; Sloan et al., 2018). We selected reads from the resulting SAM file to insert microsatellites with a priori-known genotypes. Since MicroGenotyper() detects eight main types of microsatellite genotypes (homo- or heterozygous loci with 2, 3, 4, or >= 5 bp per motif), we selected eight reads from our *A. thaliana* short-read file into which we inserted microsatellites of a priori-known genotype. To synthesize heterozygous loci and vary the start locus within the 5 bp detection window of the microsatellite search algorithm, we duplicated each of the eight reads 20 times and varied the start locus of all reads while varying the repeat count of reads intended to be genotyped as heterozygous. We then applied SAMtools view and index to this SAM file (EditedThaliana.sam in Data S1), and then created a lookup table specifying the eight microsatellite loci described by the read sets.

To ensure that the MicroGenotyper() accurately detects naturally occurring microsatellites in less curated genomes, such as the *X. birchmanni* assembly (N50 = 30.8 MB; contig count = 5987), we then counted the number of repeats in 10 naturally occurring microsatellite loci from a single 13.03GB *X. birchmanni* BAM file (accession no: SRR6511930), then compared the manual genotypes to the output of MicroGenotyper().

To test PollyMicros()' ability to categorize polymorphic loci and calculate allele frequencies, we ran this function on MicroGenotyper() outputs of known allele counts and frequencies. We used two *X. birchmanni* individuals (accession nos: SRR5170591 and SRR5172867), the minimum sample size of diploid organisms required to detect polymorphic loci with an allele frequency threshold of three. We also chose two loci: one with fewer than three alleles in the two individuals (a "non-polymorphic" locus) and a second with three alleles in the two individuals (a "polymorphic" locus). We determined the allele counts of two microsatellite loci in the *X. birchmanni* individuals using the "samtools view" command from samtools v1.10, compiled with GCC 8.3.0 (Li et al., 2009), manually counted microsatellite repeats in the output reads, and calculated allele frequencies. After running MicroGenotyper() on these same two *X. birchmanni* individuals, we ran PollyMicros() on the MicroGenotyper() outputs with a polymorphism threshold of three and compared the output to the polymorphism categorizations and allele frequencies calculated manually.

To test whether PollySI() could accurately sort markers whose VCFtools-calculated minor allele frequencies passed a user-specified threshold, we tested this function on a frequency file where the marker's polymorphism categorizations were known a priori. This frequency file consisted of nine lines. The first four lines consisted of two SNPs and two indels whose major and minor allele frequencies either met or exceeded the user-specified polymorphism threshold so we could test whether PollySI() appropriately considered these loci polymorphic. The fifth and sixth lines described a SNP and an indel whose major or minor allele frequency did not exceed the polymorphism threshold so we could ensure that PollySI() discarded these loci. We also included a SNP and an indel with undesired scaffold names to ensure PollySI() discarded these lines and included an indel where one of the alleles consisted of a single base pair to make sure PollySI() categorized this line as an indel rather than a SNP.

To test Polly()'s ability to group markers based on physical proximity, we ran this function on files with known outputs. On two *X. birchmanni* individuals (accession nos: SRR5170591 and SRR5172867), we ran PollySI() and MicroGenotyper(), then ran PollyMicros() on the outputs of the latter. We then manually recorded two polymorphic microsatellite loci and used the PollySI() outputs to manually record all polymorphic SNPs and indels within 100 bp of these polymorphic microsatellite loci. We then ran Polly() on the PollyMicros() and PollySI() outputs from these two individuals with a base pair region size of 100 bp and checked whether the predetermined haplotype groupings matched the output of Polly().

To ensure PollyPop() could accurately convert alleles of the same locus from different populations to GenePop format, we tested this function on six real *X. birchmanni* samples (accession nos: SRR5170591, SRR5172867, SRR6509133, SRR6511793,

SRR6511812 and SRR6511844). We arbitrarily partitioned the MicroGenotyper() outputs of these six individuals into two "populations" of three individuals each. We ran PollyMicros() on both of these "populations". We manually checked the genotypes of all six individuals at four randomly selected microsatellite loci and predicted the GenePop four-digit codes corresponding to these alleles, then ran GenePop on the PollyMicros outputs and MicroGenotyper() outputs corresponding to these "populations".

3 | RESULTS

3.1 | Accuracy of MicroGenotyper()

In the *A. thaliana* test files, we found that MicroGenotyper() yielded the expected genotypes at all eight loci, indicating that MicroGenotyper() is accurate in genotyping microsatellites from short-read files. In the *X. birchmanni* test files, MicroGenotyper() yielded the expected genotypes under most conditions, with some exceptions: this function did not yield the same genotype as the manual genotype if one or more reads were low quality, many of the reads had common sequencing errors, or the lookup table differed in locus indexing from the reference genome.

MicroGenotyper() will not genotype a locus if all reads are low quality. Since these functions throw out all reads with MAPQ < 10, an individual will not be genotyped at a microsatellite locus if all reads for that locus have MAPQ < 10, causing false negatives; if an individual is not genotyped at a locus, then their microsatellite alleles will not count towards the total count of alleles at the locus. Due to this, a truly polymorphic locus may appear to not meet the polymorphism threshold. However, if MicroGenotyper() did genotype low-quality reads, these genotypes would not be trustworthy due to the high rate of sequencing and mapping errors in low-quality reads. Thus, this issue can only be solved if higher quality reads are obtained.

Another limitation of MicroGenotyper() is that it cannot accurately genotype heterozygotes when a sequencing error repeat difference is present in the same number of reads as a true second allele. In these cases, our approach will use the variant which was recorded most recently as the minor allele. Thus, MicroGenotyper() may mistake sequencing errors for minor alleles and mis-genotype heterozygotes in these cases. However, it is unlikely for reads containing the exact same sequencing error to be as or more abundant than reads containing the true allele (Westen et al., 2012). So, our approach should deal well with most sequencing errors.

MicroGenotyper() also cannot accurately genotype reads where the microsatellite start point occurs two or more base pairs away from the start site predicted by the micRocounter lookup table. This limitation only impacts the final genotype of the individual if inclusion of the shifted microsatellite allele would alter which allele was considered most or second-most common. Such a shift would likely only occur if there were a difference in locus indexing between the micRocounter lookup table and the reference genome to which the reads were mapped.

If the MicroGenotyper() algorithm encounters a SNP or indel while genotyping a microsatellite, it will report the repeat count stored up until the mutation as the genotype for this locus. The consequence of this is that any repeats downstream the SNP or indel will not be genotyped. However, this termination enables more accurate distinguishing between true microsatellites and microsatellites affected by non-microsatellite mutations.

3.2 | Accuracy of PollyMicros()

PollyMicros() categorized both the non-polymorphic and the polymorphic loci into the appropriate categories, indicating that PollyMicros() is as accurate as a human researcher in counting whether the alleles of a locus meet or exceed an integer threshold. PollyMicros() also returned allele frequencies identical to those calculated manually, indicating that PollyMicros() is as accurate at calculating allele frequencies as a human researcher.

3.3 | Accuracy of PollySI()

PollySI() correctly sorted every SNP and indel in the test frequency file. The SNPs and indels whose major and minor allele frequencies met or exceeded the threshold frequency for polymorphism were correctly retained in the table of polymorphic SNPs and indels. Any indels where one of the alleles consisted of a single base pair insertion or deletion were correctly categorized as indels rather than SNPs. All SNPs and indels occurring below the threshold or on undesired scaffolds were appropriately discarded. This indicates that PollySI() is as accurate as a human researcher in determining which markers' VCFtools-calculated frequencies pass a user-specified polymorphism threshold.

3.4 | Accuracy of Polly()

Polly() grouped the same variants as a human researcher into 100 bp regions, demonstrating that Polly() is as accurate as a human researcher in grouping polymorphic microsatellites, SNPs and indels based on physical proximity to one another.

3.5 | Accuracy of PollyPop()

For each of the six *X. birchmanni* individuals in the two populations, the genotypes in the GenePop-formatted file were consistent with the genotypes from MicroGenotyper()'s output format, and the formatting of the output file was appropriate for a two-population GenePop file. This indicates that PollyPop() can be used to convert

from PollyMicros() and MicroGenotyper()'s information-rich output formats to the more commonly used GenePop format.

3.6 | Runtime

To test runtime for each of Polly's functions, we created a lookup table, set of BAM files and frequency file from publicly available *X. birchmanni* data. We used micRocounter and an *X. birchmanni* reference genome to generate the lookup table (Schumer et al., 2018). We then downloaded short-read sequence files for *X. birchmanni* from the SRA and preprocessed them according to the steps described in the Methods Section (Section 2; Leinonen et al., 2011). We used these reference and BAM files to generate a frequency file according to the steps in the Methods (Section 2).

We tested the runtime of all four functions using Texas A&M University's Linux CentOS 7 High-Performance Research Computing clusters with Lenovo DSS-G260 disks.

We evaluated the runtime of MicroGenotyper() on 19 *X. birchmanni* BAM-format genomes of varying sizes, using the same lookup table for each run (Figure 3; Table 1). Runtime increased approximately linearly with increasing file size (Figure 3a) and number of microsatellite loci (Figure 3b), while read depth had a less pronounced impact on runtime (Figure 3c). For example, the smallest file tested (SRR6511793, 8.92 GB) took 61.2 min and 1870 GB of memory to genotype its 310,469 microsatellite loci distributed across its 150,870,772 reads, while the largest file tested (SRR5172867, 22.2 GB) took 61.2 min and 1890 GB of memory to genotype its 312,842 microsatellite loci distributed across its 1,948,626,237 reads (Table 1).

To analyse the impact of input file size and number on runtime, we ran PollyMicros() multiple times, including an additional MicroGenotyper() .csv in each run. The runtime of PollyMicros() was notably shorter than that of MicroGenotyper(): runtime stayed under 20 s regardless of the number of files used (Figure 4a,b). The runtime of PollyMicros() appears to scale approximately linearly with each additional BAM input file (Figure 4a,b) and approximately exponentially with the total number of polymorphic loci detected across all inputs (Figure 4c). Since PollyMicros() uses MicroGenotyper()-generated CSVs as opposed to BAM files as input, runtime was not evaluated as a function of read depth.

We recorded PollyPop()'s runtime as we tested its accuracy on the two populations of three individuals each. The first population consisted of three MicroGenotyper() output CSVs with 312,769, 312,842 and 312,769 microsatellite loci as well as a PollyMicros() output CSV describing 35,576 polymorphic loci, while the second consisted of three individuals with 310,469, 309,375 and 311,621 loci as well as their PollyMicros() output CSV describing the 42,705 loci which were deemed polymorphic across these three individuals. The runtime on these two populations was 37 min.

Code necessary for installing the R package Polly and for reproducing the analyses, as well as a vignette providing step-by-step instructions for preparing input files and using Polly, is available in the GitHub repository: https://github.com/AnnabelPerry/Polly.

4 | DISCUSSION

Here, we present Polly, a software program which leverages the friendly user interface of R and the efficiency of C++ to enable any researcher with basic knowledge of command-line Linux and R to genotype microsatellites and detect highly polymorphic genetic markers even in non-traditional species. Polly enables variant detection using Illumina short reads so long as the reads are of sufficient quality to minimize PCR and sequencing errors and at least one fully scaffolded genome is available for a species closely related to the target organism. Polly also condenses data on polymorphic indels, SNPs and microsatellites such that researchers can choose which variant class they would like to focus their studies on. Not only does Polly report polymorphic microsatellites in a detail-rich output format to enable in-depth investigations of genomic architecture, but Polly also enables the user to convert the output into the widely used GenePop file format (Rousset, 2008). This enables the polymorphic microsatellites

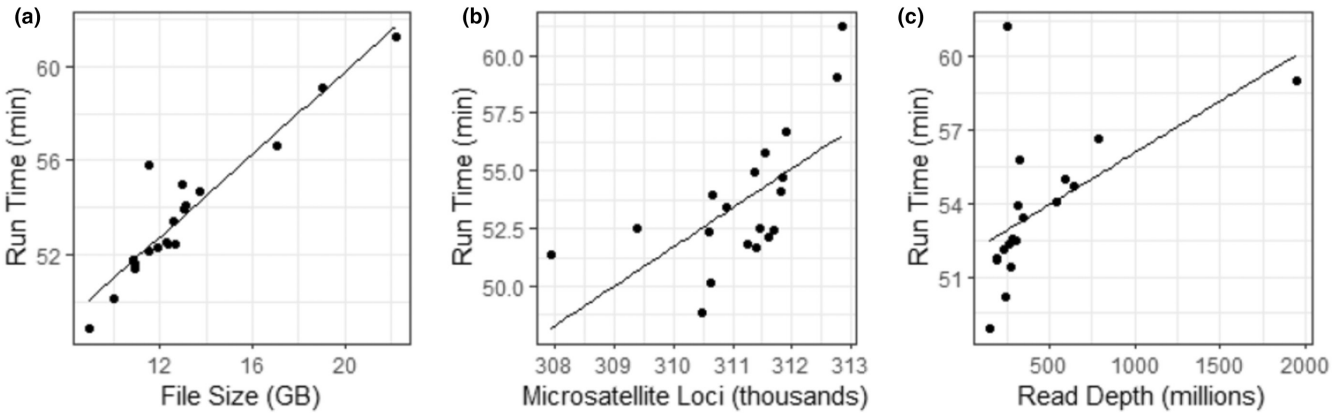

FIGURE 3 Runtime for MicroGenotyper() as a function of…. (a) size of input BAM files, (b) total number of microsatellite loci identified in a BAM file and (c) number of reads (millions). Each run took between 1860 and 1890 GB of RAM (see Table 1).

TABLE 1 MicroGenotyper() run time as a function of number of reads, number of microsatellite loci and file size for each of the 18 *Xiphophorus birchmanni* short-read genomes upon which MicroGenotyper()'s run time was tested.

Accession	File size (GB)	Memory (GB)	Run time (min)	Number of reads	Number of microsatellite loci
SRR5170591	10.92	1860	51.40	279,358,276	307,930
SRR5172867	22.2	1890	61.25	258,238,423	312,842
SRR6509133	19.04	1890	59.05	1,948,626,237	312,769
SRR6511793	8.92	1870	48.89	150,870,772	310,469
SRR6511812	12.29	1880	52.53	289,262,385	309,375
SRR6511844	11.53	1870	52.15	232,490,275	311,621
SRR6511845	10.83	1870	51.79	192,317,148	311,241
SRR6511861	12.62	1880	53.39	344,558,409	310,884
SRR6511862	11.56	1860	55.80	327,412,977	311,557
SRR6511930	13.03	1880	53.92	312,825,139	310,658
SRR6511931	12.96	1870	54.97	592,810,858	311,376
SRR6511932	17.08	1880	56.66	784,455,586	311,900
SRR6511933	13.71	1860	54.71	646,851,104	311,838
SRR6511934	9.99	1880	50.18	246,549,600	310,634
SRR6511970	12.37	1880	52.43	270,866,167	311,686
SRR6511971	13.1	1880	54.10	546,352,226	311,827
SRR6511973	11.89	1860	52.33	262,984,003	310,599
SRR6511974	12.68	1870	52.50	304,652,134	311,456

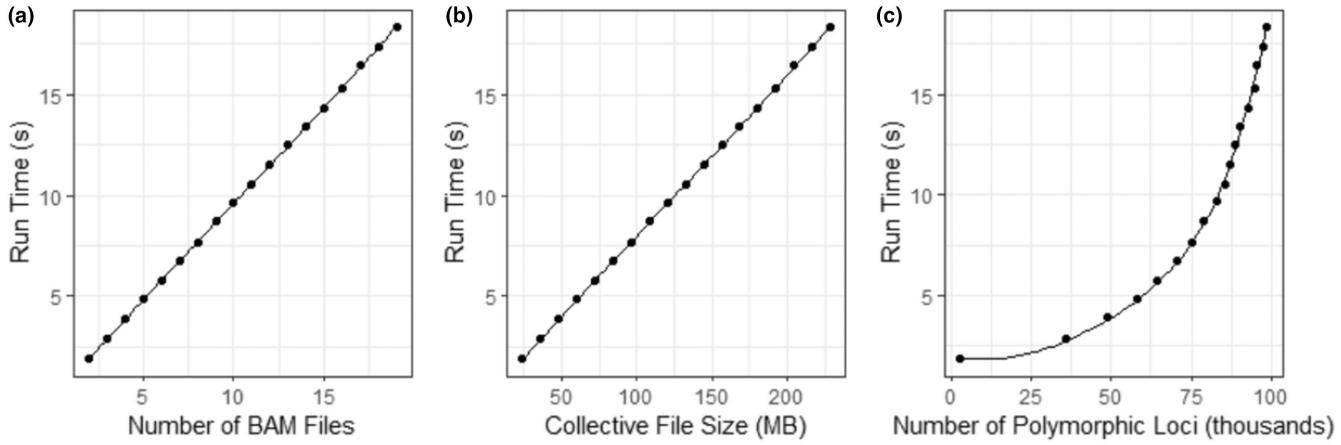

FIGURE 4 Runtime for the PollyMicros() function as a function of…. (a) the number of BAM inputs from MicroGenotyper(). (b) the summed sizes of all BAM files. (c) the total number of polymorphic loci detected across all input BAM files.

identified using Polly to not only be used for population genetics programs such as GenePop but also facilitates easy conversion of this output to the format used by other population genetics programs such as STRUCTURE (Stanley et al., 2017). Polly, thus, ameliorates issues of polymorphic variant detection, facilitating more efficient ecological studies.

Polly's ability to accurately genotype microsatellites is dependent upon the quality of the input Illumina short reads, which must be of sufficient quality to minimize PCR and sequencing errors. Though Polly's capacity to accurately genotype microsatellites is limited by the quality of the input sequences, recent results in skim sequencing indicate that, for most population genetics results, results are more robust if one genotypes many individuals to low accuracy rather than genotyping a few individuals to high accuracy (Kumar et al., 2021). Thus, though Polly may inaccurately genotype microsatellites if the input short reads contain microsatellite-disrupting sequencing errors and PCR artefacts, this limitation ought not to substantially impact most large-scale population genetics analyses.

MicroGenotyper()'s dependence upon short reads, which are inherently limited in length, imposes some limitations upon the length of microsatellite alleles which can be detected by this algorithm. Microsatellite alleles which extend past the end of a read will be counted to have fewer repeats than are truly present at that locus. Our approach helps to ameliorate mis-genotyping due to truncation of reads prior to microsatellite end by throwing out reads with anomalously low repeat counts. However, microsatellite alleles whose total length exceeds that of the read length cannot be genotyped. For example, a 200 bp microsatellite allele cannot be genotyped if

reads are limited to 100 bp. Thus, since MicroGenotyper() depends on short-read data, the length of the input reads limits the maximum microsatellite allele size which can be detected using this genotyping algorithm.

Despite these limitations, we demonstrated that Polly can be applied to diverse species by using the microsatellite genotyping algorithm to detect microsatellites in a plant species (*A. thaliana*) and a fish species (*X. birchmanni*). More broadly we believe that Polly will prove to be an important tool allowing researchers to quickly and inexpensively genotype many individuals at many loci in an efficient and reproducible fashion.

AUTHOR CONTRIBUTIONS

AP designed, developed and tested the C++ source code and the R interface for the Polly R package. AP also analysed the results of testing and wrote the manuscript. DE provided guidance in converting the Polly C++ source code to an R package and helped to troubleshoot this portion of the project. GR aided in conceptualizing the Polly project and also edited the manuscript. HB provided guidance in troubleshooting the Polly R package C++ source code, provided computational resources required for troubleshooting and edited the manuscript.

ACKNOWLEDGEMENTS

This work was supported by an NIH NIGMS R35GM138098 to HB, a TAMU Chancellor EDGES Fellowship and NSF DEB 1754669 to GGR and the Texas A&M University College of Science Undergraduate Research Opportunities Program to ARP. Thanks to Dr. Adam Jones for comments on an earlier version of this manuscript.

CONFLICT OF INTEREST STATEMENT

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

DATA AVAILABILITY STATEMENT

Code necessary for installing the R package Polly is available in the GitHub repository: https://github.com/AnnabelPerry/Polly. All supplementary materials are available from a FigShare repository: https://doi.org/10.6084/m9.figshare.19686408.v1. The *X. birchmanni* reference genome is available as "xiphophorus_birchmanni_10x_12Sep2018_yDAA6.fasta" on Dryad: https://datadryad.org/stash/dataset/doi:10.5061/dryad.z8w9ghx82. For all *X. birchmanni* accuracy and runtime measurements, we used the following short-read accessions which are publicly available on the NCBI SRA: SRR5170591, SRR5172867, SRR6509133, SRR6511793, SRR6511812, SRR6511844, SRR6511845, SRR6511861, SRR6511862, SRR6511863, SRR6511930, SRR6511931, SRR6511932, SRR6511933, SRR6511934, SRR6511970, SRR6511971, SRR6511973, SRR6511974. All *X. birchmanni* samples were wild caught from the Río Calnali, Calnali, Hidalgo, Mexico. We used *A. thaliana* reference genome (TAIR10.1 version of the Columbia (Col-O) ecotype of *Arabidopsis*), publicly available through The Arabidopsis Information Resource. The edited Arabidopsis short-read file used for accuracy validation is publicly available on FigShare: https://doi.org/10.6084/m9.figshare.19686408.v1. Scripts and example outputs included in a vignette are included in the R package.

BENEFIT SHARING STATEMENT

Benefits from this research are publicly available from the data and results described in the above section. Non-monetary benefits of our data to the public include open-access and extensively documented code which can be adapted such that other researchers can build upon our research. The research described herein complies with the Convention on Biological Diversity and Nagoya Protocol. All collaborators are listed as co-authors.

ORCID

Heath Blackmon https://orcid.org/0000-0002-5433-4036

REFERENCES

Castoe, T. A., Poole, A. W., de Koning, A. P. J., Jones, K. L., Tomback, D. F., Oyler-McCance, S. J., Fike, J. A., Lance, S. L., Streicher, J. W., Smith, E. N., & Pollock, D. D. (2012). Rapid microsatellite identification from Illumina paired-end genomic sequencing in two birds and a Snake. *PLoS One*, *7*(2), e30953. https://doi.org/10.1371/journal.pone.0030953

Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., McVean, G., Durbin, R., & 1000 Genomes Project Analysis Group. (2011). The variant call format and VCFtools. *Bioinformatics*, *27*(15), 2156–2158. https://doi.org/10.1093/bioinformatics/btr330

Das, R., Arora, V., Jaiswal, S., Iquebal, M., Angadi, U., Fatma, S., Singh, R., Shil, S., Rai, A., & Kumar, D. (2019). PolyMorphPredict: A universal web-tool for rapid polymorphic microsatellite marker discovery from whole genome and transcriptome data. *Frontiers in Plant Science*, *9*, 1966. https://doi.org/10.3389/fpls.2018.01966

De Barba, M., Miquel, C., Lobréaux, S., Quenette, P. Y., Swenson, J. E., & Taberlet, P. (2017). High-throughput microsatellite genotyping in ecology: Improved accuracy, efficiency, standardization and success with low-quantity and degraded DNA. *Molecular Ecology Resources*, *17*(3), 492–507. https://doi.org/10.1111/1755-0998.12594

Fischer, M. C., Rellstab, C., Leuzinger, M., Roumet, M., Gugerli, F., Shimizu, K. K., Holderegger, R., & Widmer, A. (2017). Estimating genomic diversity and population differentiation—An empirical comparison of microsatellite and SNP variation in *Arabidopsis halleri*. *BMC Genomics*, *18*(1), 69. https://doi.org/10.1186/s12864-016-3459-7

Giangregorio, P., Naldi, L., Mengoni, C., Greco, C., Padula, A., Zaccaroni, M., Fani, R., Argenti, G., & Mucci, N. (2021). Cross-amplification in strigiformes: A new STR panel for forensic purposes. *Genes*, *12*(11), 1721. https://doi.org/10.3390/genes12111721

Gosselin, T. (2020). *Thierrygosselin/radiator: Update (1.1.4) [computer software]*. Zenodo. https://doi.org/10.5281/ZENODO.3687060

Goudet, J., Kay, T., & Weir, B. S. (2018). How to estimate kinship. *Molecular Ecology*, *27*(20), 4121–4135. https://doi.org/10.1111/mec.14833

Gymrek, M., Golan, D., Rosset, S., & Erlich, Y. (2012). lobSTR: A short tandem repeat profiler for personal genomes. *Genome Research*, *22*(6), 1154–1162. https://doi.org/10.1101/gr.135780.111

Knaus, B. J., & Grünwald, N. J. (2017). VCFR: A package to manipulate and visualize variant call format data in R. *Molecular Ecology Resources*, *17*(1), 44–53. https://doi.org/10.1111/1755-0998.12549

Kristmundsdottir, S., Eggertsson, H. P., Arnadottir, G. A., & Halldorsson, B. V. (2020). popSTR2 enables clinical and population-scale

genotyping of microsatellites. *Bioinformatics*, *36*(7), 2269–2271. https://doi.org/10.1093/bioinformatics/btz913

Kumar, P., Choudhary, M., Jat, B. S., Kumar, B., Singh, V., Kumar, V., Singla, D., & Rakshit, S. (2021). Skim sequencing: An advanced NGS technology for crop improvement. *Journal of Genetics*, *100*, 38.

Leinonen, R., Sugawara, H., Shumway, M., & International Nucleotide Sequence Database Collaboration. (2011). The sequence read archive. *Nucleic Acids Research*, *39*, D19–D21. https://doi.org/10.1093/nar/gkq1019

Li, C., Lin, F., An, D., Wang, W., & Huang, R. (2017). Genome sequencing and assembly by long reads in plants. *Genes*, *9*(1), 6. https://doi.org/10.3390/genes9010006

Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, *25*(14), 1754–1760. https://doi.org/10.1093/bioinformatics/btp324

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup. (2009). The sequence alignment/map format and SAMtools. *Bioinformatics*, *25*(16), 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

Li, M.-M., Li, B.-L., Jiang, S.-X., Zhao, Y.-W., Xu, X.-L., & Wu, J.-X. (2019). Microsatellite-based analysis of genetic structure and gene flow of *Mythimna separata* (Walker; Lepidoptera: Noctuidae) in China. *Ecology and Evolution*, *9*(23), 13426–13437. https://doi.org/10.1002/ece3.5799

Lin, Y., Chen, Y., Yi, C., Fong, J. J., Kim, W., Rius, M., & Zhan, A. (2017). Genetic signatures of natural selection in a model invasive ascidian. *Scientific Reports*, *7*, 44080. https://doi.org/10.1038/srep44080

Lo, J., Jonika, M. M., & Blackmon, H. (2019). micRocounter: Microsatellite characterization in genome assemblies. *G3 (Bethesda, Md.)*, *9*, 3101–3104. https://doi.org/10.1534/g3.119.400335

Lu, Y., Sandoval, A., Voss, S., Lai, Z., Kneitz, S., Boswell, W., Boswell, M., Savage, M., Walter, C., Warren, W., Schartl, M., & Walter, R. (2020). Oncogenic allelic interaction in *Xiphophorus* highlights hybrid incompatibility. *Proceedings of the National Academy of Sciences of the United States of America*, *117*(47), 29786–29794. https://doi.org/10.1073/pnas.2010133117

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., & DePristo, M. A. (2010). The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, *20*(9), 1297–1303. https://doi.org/10.1101/gr.107524.110

Metz, S., Cabrera, J. M., Rueda, E., Giri, F., & Amavet, P. (2016). FullSSR: Microsatellite finder and primer designer. *Advances in Bioinformatics*, *2016*, 6040124. https://doi.org/10.1155/2016/6040124

Miller, M. P., Knaus, B. J., Mullins, T. D., & Haig, S. M. (2013). SSR_pipeline: A Bioinformatic infrastructure for identifying microsatellites from paired-end Illumina high-throughput DNA sequencing data. *Journal of Heredity*, *104*(6), 881–885. https://doi.org/10.1093/jhered/est056

Pemberton, J. M. (2008). Wild pedigrees: The way forward. *Proceedings. Biological Sciences*, *275*(1635), 613–621. https://doi.org/10.1098/rspb.2007.1531

Pipek, O., Ribli, D., Molnár, J., Póti, Á., Krzystanek, M., Bodor, A., Tusnády, G. E., Szallasi, Z., Csabai, I., & Szüts, D. (2017). Fast and accurate mutation detection in whole genome sequences of multiple isogenic samples with IsoMut. *BMC Bioinformatics*, *18*(1), 73. https://doi.org/10.1186/s12859-017-1492-4

Rousset, F. (2008). genepop'007: A complete re-implementation of the genepop software for windows and Linux. *Molecular Ecology Resources*, *8*(1), 103–106. https://doi.org/10.1111/j.1471-8286.2007.01931.x

Schbath, S., Martin, V., Zytnicki, M., Fayolle, J., Loux, V., & Gibrat, J.-F. (2012). Mapping reads on a genomic sequence: An algorithmic overview and a practical comparative analysis. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, *19*(6), 796–813. https://doi.org/10.1089/cmb.2012.0022

Schumer, M., Xu, C., Powell, D. L., Durvasula, A., Skov, L., Holland, C., Blazier, J. C., Sankararaman, S., Andolfatto, P., Rosenthal, G. G., & Przeworski, M. (2018). Natural selection interacts with recombination to shape the evolution of hybrid genomes. *Science (New York, N.Y.)*, *360*(6389), 656–660. https://doi.org/10.1126/science.aar3684

Seroussi, Y., & Seroussi, E. (2007). TraceHaplotyper: Using direct sequencing to determine the phase of an indel followed by biallelic SNPs. *BioTechniques*, *43*(4), 452, 454, 456. https://doi.org/10.2144/000112552

Sloan, D. B., Wu, Z., & Sharbrough, J. (2018). Correction of persistent errors in *Arabidopsis* reference mitochondrial genomes. *The Plant Cell*, *30*(3), 525–527. https://doi.org/10.1105/tpc.18.00024

Stanley, R. R. E., Jeffery, N. W., Wringe, B. F., DiBacco, C., & Bradbury, I. R. (2017). GENEPOPEDIT: A simple and flexible tool for manipulating multilocus molecular data in R. *Molecular Ecology Resources*, *17*(1), 12–18. https://doi.org/10.1111/1755-0998.12569

Vieira, M. L. C., Santini, L., Diniz, A. L., & Munhoz, C. F. (2016). Microsatellite markers: What they mean and why they are so useful. *Genetics and Molecular Biology*, *39*(3), 312–328. https://doi.org/10.1590/1678-4685-GMB-2016-0027

Wang, X., & Wang, L. (2016). GMATA: An integrated software package for genome-scale SSR mining, marker development and viewing. *Frontiers in Plant Science*, *7*, 1350. https://doi.org/10.3389/fpls.2016.01350

Westen, A. A., Grol, L. J. W., Harteveld, J., Matai, A. S., de Knijff, P., & Sijen, T. (2012). Assessment of the stochastic threshold, back- and forward stutter filters and low template techniques for NGM. *Forensic Science International: Genetics*, *6*(6), 708–715. https://doi.org/10.1016/j.fsigen.2012.05.001

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Perry, A., Eddelbuettel, D., Rosenthal, G., & Blackmon, H. (2024). Polly: An R package for genotyping microsatellites and detecting highly polymorphic DNA markers from short-read data. *Molecular Ecology Resources*, *24*, e13933. https://doi.org/10.1111/1755-0998.13933