Chapter 4

A Tutorial of EDTA: Extensive De Novo TE Annotator

Weijia Su, Shujun Ou, Matthew B. Hufford, and Thomas Peterson

Abstract

Transposable elements (TEs) are important contributors to genome structure and evolution. With the growth of sequencing technologies, various computational pipelines and software programs have been developed to facilitate TE identification and annotation. These computational tools can be categorized into three types based on their underlying approach: homology-based, structural-based, and de novo methods. Each of these tools has advantages and disadvantages. In this chapter, we introduce EDTA (Extensive de novo TE Annotator), a new comprehensive pipeline composed of high-quality tools to identify and annotate all types of TEs. The development of EDTA is based on the benchmarking results of a collection of TE annotation methods. The selected programs are evaluated by their ability to identify true TEs as well as to exclude false candidates. Here, we present an overview of the EDTA pipeline and a detailed manual for its use. The source code of EDTA is available at https://github.com/oushujun/EDTA.

Key words Transposable elements, TE annotation, De novo TE identification, TE computational pipeline

1 Introduction

Transposable elements (TEs) are mobile DNA sequences that can both change their position in the genome and proliferate. In the 1940s, the iconic geneticist Barbara McClintock discovered the first TE through the observation of patterns of maize kernel pigmentation. She named the first TE system as *Ac/Ds*: *Activator* (*Ac*) is an autonomous TE that can regulate the activity of the non-autonomous TE *Dissociation* (*Ds*) [1, 2]. Since then, many different types of TEs have been identified in both plant and animal genomes. Sequences derived from TEs (both intact elements and TE fragments) make up a large proportion of many eukaryotic genomes; for example, TEs comprise about 85% of the maize and wheat genomes [3] and 40–85% of other annotated grass genomes [4].

One of the most commonly used TE classification systems was proposed by Wicker et al. [5]. This system includes hierarchical levels of class, subclass, order, superfamily, family, and subfamily and uses three-letter codes to denote the class-order-superfamily of a TE. In general, TEs can be categorized into two classes: Class I TEs are also called retrotransposons, which rely on an RNA intermediate to transpose; Class II TEs are called DNA transposons because they do not use RNA for transposition. There are five orders in Class I TEs: LTR-RT, DIRS, PLE, LINE, and SINE; and four orders in Class II TEs: TIR, Helitron, Crypton, and Maverick (Crypton and Maverick are absent in plant genomes). The TE superfamily designation indicates related types based on shared characteristics including protein domains, target site specificity (if any), and length of target site duplications.

As technological advances lead to abundant genome sequence data, researchers need fast and reliable methods for genome annotation. Computational tools for TE annotation generally fall into three main types [6, 7]. The first type is homology-based and uses a library of known TE candidate sequences as queries to search against the target genome to find homologous copies. This approach is fast and efficient but may yield incomplete results if the reference TE library is incomplete. Moreover, homology-based methods are usually unable to identify new TEs in non-model species genomes. The second type is structure-based and uses known TE structures such as terminal motifs and conserved protein domains to annotate TEs. These methods rely on prior knowledge of TE structures and sometimes are limited to full-length TEs. The third type is based on the identification of high copy number, repetitive sequences. These methods have the potential for high sensitivity but can also exhibit high false discovery rates; for example, high copy number genes may be identified as TEs.

As stated above, the existing TE annotation tools have their advantages and disadvantages, and consequently, it is often difficult for a new user to choose the best tools to generate optimal TE annotations. To address this problem, the EDTA (Extensive de novo TE Annotator) pipeline was created based on the benchmarking results of various programs. Best-performing programs for each TE type were assembled into a comprehensive pipeline and downstream filters were developed to reduce false classification, redundant results, and to resolve nested TEs [8]. EDTA is a single-package approach to annotate known types of TEs in both small and large genomes. EDTA had been shown to have robust performances on both plant and animal genomes. Here, we provide an introduction and tutorial for applying the EDTA pipeline to any genome sequence.

2 Materials

2.1 Overview

The EDTA package is composed of eight published programs. LTRharvest [9], LTR_FINDER_parallel [10], and LTR_retriever [11] are incorporated in this package to identify LTR retrotransposons; Generic Repeat Finder [12] and TIR-Learner [13] are included to identify TIR transposons; HelitronScanner [14] identifies *Helitron* transposons; RepeatModeler [15] is used to identify TEs (such as SINEs and LINEs) missed by the other structure-based programs, and finally RepeatMasker (http://repeatmasker.org) is used to annotate fragmented TEs based on homology to structurally annotated TEs (Table 1).

Besides combining and filtering the raw output candidates from the base programs, EDTA also features downstream screening that helps to minimize the false discovery rate. Furthermore, EDTA uses TE exemplars identified by the base programs for whole-genome TE annotation as well as evaluation of the annotation output (Fig. 1).

2.2 Main Concept

Intact TEs: Intact TEs refer to elements that retain complete TE structures including terminal repeats and protein domains. Intact TEs are often capable of transposition, but maybe silenced and/or rendered immobile due to epigenetic modification(s) [16].

Full-length TEs: Full-length TEs contain two terminal repeat sequences and subterminal motifs, but their internal sequences may be mutated, deleted, or rearranged. Some full-length TEs are non-autonomous due to lack of the transposase-coding function; however, they may be mobilized by transposase provided by autonomous TEs in the same family [17].

Fractured TEs: Fractured TEs contain only one terminal fragment of an otherwise complete element. Fractured TEs may be able to transpose by cooperating with another nearby transposon terminus in alternative transposition reactions, if the genome also contains an autonomous TE of the same family to provide transposase [18].

Table 1
EDTA base programs

TE types	Programs		
LTR	LTR_FINDER	LTRharvest	LTR_retriever
TIR/MITE	Generic Repeat Finder (GRF)		TIR-Learner
Helitron	HelitronScanner		
Others	RepeatModeler, RepeatMasker		

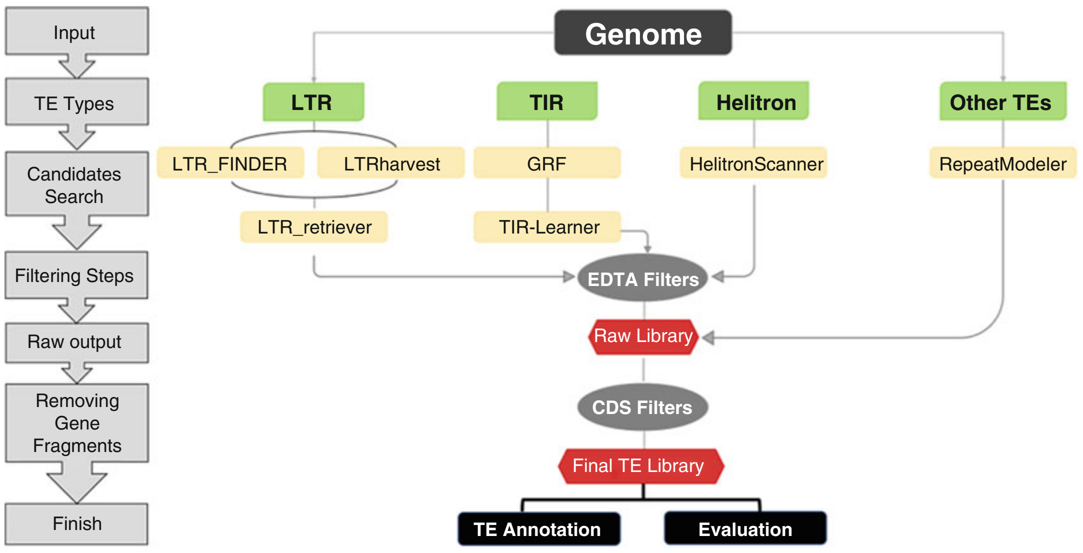


Fig. 1 Schematic flowchart of the EDTA pipeline

TE fragments: TE fragments are DNA sequences that have significant similarity to intact or full-length TEs, but do not have the TE terminal sequences required to transpose. In some cases, these TE fragments have functions in genomic dynamics and epigenetic regulation [19].

Nested TEs: Nested TEs occur when one TE inserts within another TE. Multiple nested insertion events could create highly fragmented TE clusters [20].

Figure 2 shows the characteristics of different TE structures using *Ac/Ds* transposable elements as an example.

Virtual environment: EDTA uses conda (https://docs.conda.io/en/latest/) for package and environment management. This helps to isolate the dependencies required by different base programs. To implement EDTA, users should install and initiate conda first. The detailed conda instructions can be found on the website https://docs.conda.io/projects/conda-build/en/latest/install-conda-build.html. EDTA is also available as a docker container that contains all necessary dependencies. For this installation approach, users should install docker first. The detailed docker installation and instructions can be found on the website https://docs.docker.com/.

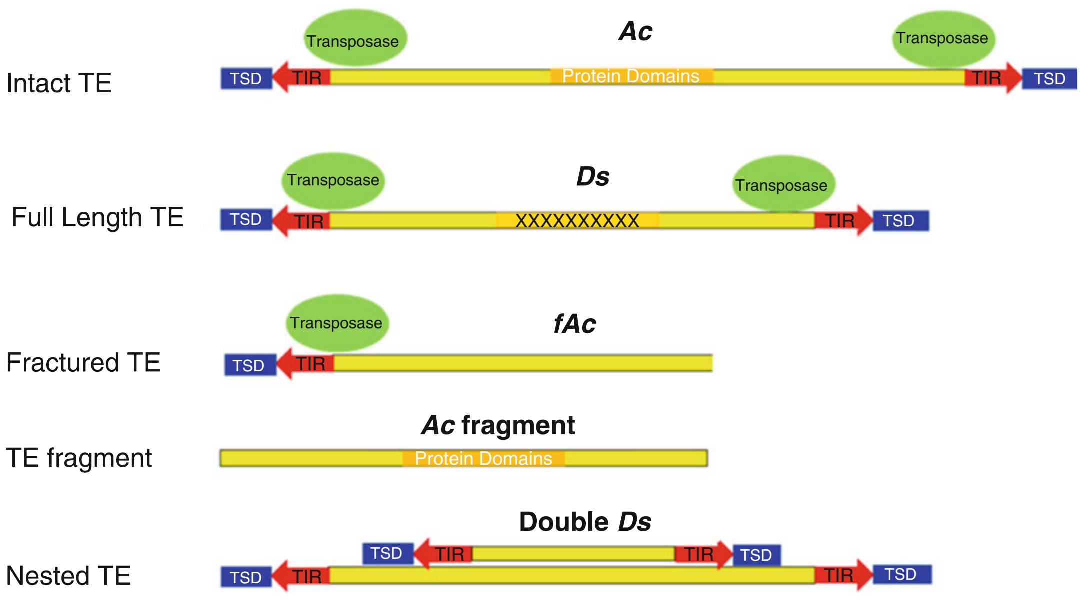


Fig. 2 The characteristics of different TE structures

3 Methods

3.1 Getting Started (Installation)

3.1.1 Install Via Conda

EDTA uses a variety of libraries such as biopython, pandas, glob, scikit-learn, tensorflow, and keras for python version 3.6. Various dependencies were also implemented in EDTA including CD-HIT, MUSCLE, mdust, and RepeatMasker. These packages and dependencies can be managed by conda. After conda is installed, users can install EDTA dependencies by running (*see* **Note 1**):

```
conda create -n EDTA
conda activate EDTA
conda install -n EDTA -y -c bioconda -c conda-forge edta
```

EDTA source code is available at GitHub; clone the EDTA package and initiate it as follows (*see* **Note 2**):

```
git clone https://github.com/oushujun/EDTA
./EDTA/EDTA.pl
```

3.1.2 Install Via Docker

A docker version of EDTA is available through https://hub.docker.com/r/kapeel/edta. The docker version is automatically updated. Users can avoid system and dependency conflicts by using the docker version. Users can install the docker version of EDTA and initiate it as follows (*see* **Note 3**):

```
docker pull kapeel/edta
docker run -v $PWD:/in -w /in kapeel/edta --genome genome.fa
[other parameters]
```

3.2 Running EDTA

EDTA is a flexible pipeline that allows users to customize the start and end steps according to their purpose and the original input data. Furthermore, instead of running the entire pipeline to annotate all TE types, the users can decompose the pipeline in order to focus on a particular type(s) of TEs such as LTR, TIR, or Helitrons. By running the decomposed pipeline in parallel on different TE categories, users can reduce processing time for large genomes (*see* **Note 4**).

To get help:

```
perl EDTA.pl -h
```

To run EDTA in general:

```
perl EDTA.pl [options]
Example:
perl EDTA/EDTA.pl --genome Rice.fa --species Rice --threads 32
Required:
--genome [file] The genome file in fasta format. If the fasta
                file is not in the current directory, please
                specify the absolute or relative path of the
                fasta file: /Path/to/fastafile.fasta (see Note 5)
```

```
Optional:
--species [string] Specify the species for identification of TIR candi-
dates.
Rice: Any rice genomes, using the rice-specific model in TIR step
Maize: Any maize genomes, using the maize-specific model in TIR step
others: Any other genomes, using the general deep learning model in TIR step
Default: others
--step [string] Specify which steps of EDTA you want to run.
all: run the entire pipeline
filter: start from raw TEs to the end.
final: start from filtered TEs to finalizing the run.
anno: perform whole-genome annotation and analysis.
Default: all
--overwrite [0|1] If previous results are found (see Note 6).
 0: do not overwrite, use previous results.
 1: overwrite previous results, re-run all steps.
 Default: 0
--curatedlib [file] Provided a curated library as known TEs (see Note 7).
 Default: No TE file is provided.
```

```
--sensitive [0|1] Use RepeatModeler to identify remaining TEs (1) or not
                  (0) (see Note 8).
 Default: 0
--anno [0|1] Perform (1) or do not perform (0) whole-genome TE annotation.
             Default:0
--evaluate [0|1] Evaluate classification consistency of the TE annotation
                 (see Notes 9).
 Default: 0
--exclude [File] Exclude bed format regions from TE annotation (--anno
                 1 required) (see Note 10).
 Default: undef
--threads|-t [int] Number of threads to run this script
 Default: 4
```

To run decomposed EDTA for a particular TE class:

```
perl EDTA_raw.pl [options]
Required:
--genome [file] The genome file in fasta format. If the fasta file is not in the
                current directory, please specify the absolute path of the fasta
                file.
```

```
Optional:
--species [string] Specify the species for identification of TIR candidates.
 Rice: Any rice genome, using the rice-specific model in TIR step
 Maize: Any maize genome, using the maize-specific model in TIR step
 others: Any other genomes, using the general deep learning model in TIR step
 Default: others
--type [string] Specify one of the TE types.
 The program will return TE candidates of the specified TE type.
 ltr: Candidates for long terminal repeat transposon
 tir: Candidates for terminal inverted repeat transposon.
 helitron: Candidates for helitron.
 all: Candidates for all TE types.
 Default: all
--overwrite [0|1] If previous results are found.
 0: do not overwrite, use previous results.
 (this will save some time if the previous run was interrupted)
 1: overwrite previous results, re-run all steps.
--threads|-t [int] Number of threads to run this script
 Default: 4
```

To benchmark EDTA results with a curated TE library:

```
1. Mask the genome with the curated library:
RepeatMasker -pa 36 -q -no_is -norna -nolow -div 40 -lib curated.TE.lib.
fasta -cutoff 225 genome.fasta
```

```
2. Mask the genome with the EDTA library:
RepeatMasker -pa 36 -q -no_is -norna -nolow -div 40 -lib EDTA.TE.lib.fasta
-cutoff 225 genome.fasta
3. Test the annotation performance of a particular TE category.
perl lib-test.pl -genome genome.fasta -std curated.TE.lib.RM.out -tst EDTA.
TE.lib.RM.out
-genome [file] FASTA format genome sequence
-std [file] RepeatMasker .out file of the standard (curated) library
-tst [file] RepeatMasker .out file of the EDTA library
-cat  [string]  Testing  TE  category.  Use  one  of  LTR|nonLTR|LINE|SINE|TIR|
               MITE|Helitron|Total|Classified
-N [0|1] Include (1) Ns in total length of the genome or not (0).
 Default: 0.
-unknown  [0|1]  Include unknown annotations to the testing category. This
                should be used when the test library has no classification
                and  you  assume  they  all  belong  to  the  target  category
                specified by -cat.
 Default: 0 (do not include unknowns)
```

3.3 Output

EDTA generates several folders for different tasks (Figs. 3, 4, and 5). The outputs of raw TEs are located in the working directory in a folder named *FastaName*.EDTA.raw (*FastaName* indicates the name of the user-defined genome fasta file). The sub-folders are named LTR, TIR, and Helitron and store the corresponding intermediate results. The raw TE libraries contain both intact and full-length TEs that could be nested in each other. Filtered intact TEs identified from the genome are reported in both the fasta format and the GFF3 format, which can be used to study the biology of TEs. The TE output from the final step is called *FastaName*.fa. EDTA.TElib.fa and the whole-genome TE annotation is included in the file *FastaName*.fa.EDTA.TEanno.gff3 (*see* **Note 11**).

3.4 Test Run

To illustrate different options of EDTA, we use the rice genome (*Oryza sativa* MSU v7) as an example. All tests were done on a high-performance computation cluster by using 32 threads.

The annotation results (Rice.fa.EDTA.TElib.fa, Rice.fa.EDTA. TEanno.gff3, and Rice.fa.MAKER.masked) could be found in the root-working folder. Specifically, the file "Rice.fa.EDTA.TElib.fa" is the non-redundant TE library, in which each sequence could be regarded as a TE family; the file "Rice.fa.EDTA.TEanno.gff3" is the whole-genome TE annotation file annotated by the non-redundant library; the file "Rice.fa.MAKER.masked" is the genome file with TEs masked in a lower threshold, so that genic regions are largely unmasked to facilitate downstream gene predictions.

EDTA.pl: Get a high-quality TE library from a genome				
Test 1. Command Line: perl EDTA.pl --genome Rice.fa --species Rice --threads 32 --anno 1				
Task	Run the entire pipeline and perform whole-genome annotation			
Time	19:06:41			
CPU	920%			
Output	Rice.fa.EDTA.TElib.fa	Rice.fa.EDTA.TEanno.gff3	Rice.fa.MAKER.masked	
Test 2. Command Line: perl EDTA.pl --genome Rice.fa --species Rice --threads 32 --anno 1 --evaluate 1 --overwrite 1				
Task	Run the entire pipeline, perform whole-genome annotation and evaluate classification consistency without using existing results			
Time	45:58:21			
CPU	1403%			
Output	Rice.fa.EDTA.TElib.fa	Rice.fa.EDTA.TEanno.gff3	Rice.fa.MAKER.masked	Rice.fa.EDTA.anno/: Rice.fa.EDTA.TE.fa.stat.all.sum Rice.fa.EDTA.TE.fa.stat.nested.sum Rice.fa.EDTA.TE.fa.stat.redun.sum

Fig. 3 Examples of comprehensive EDTA pipeline output using rice genome (Genome name: Rice.fa)

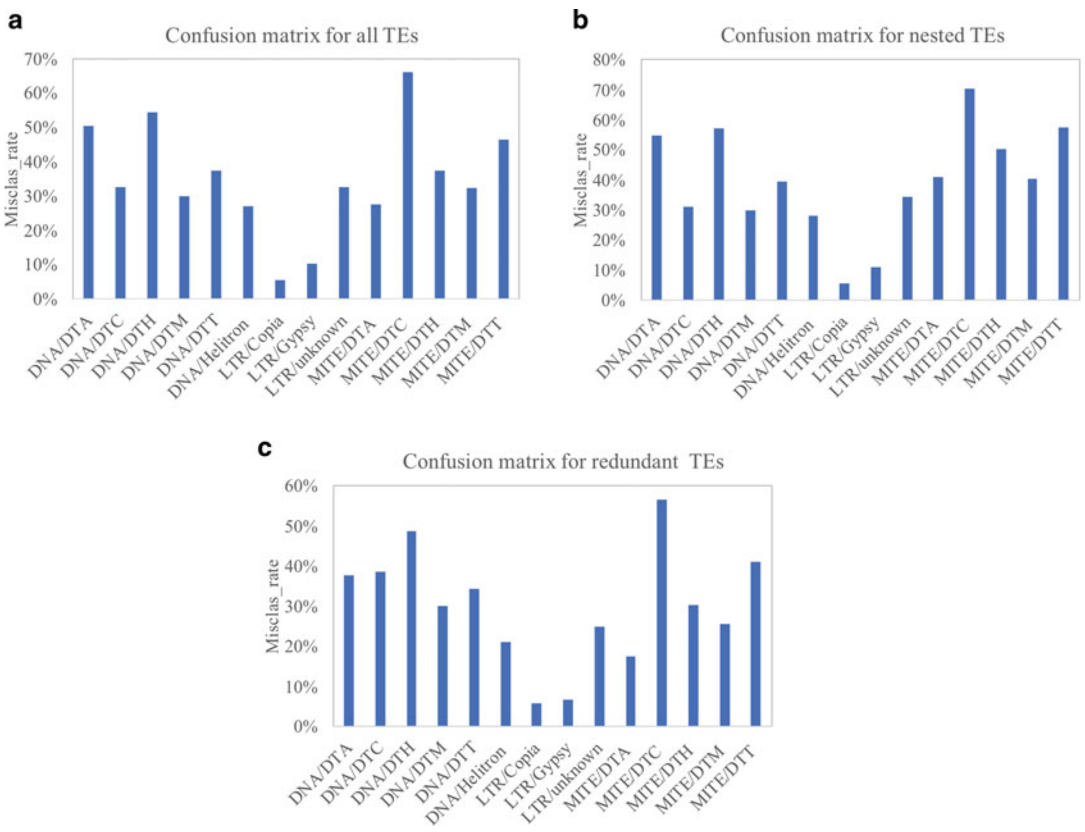
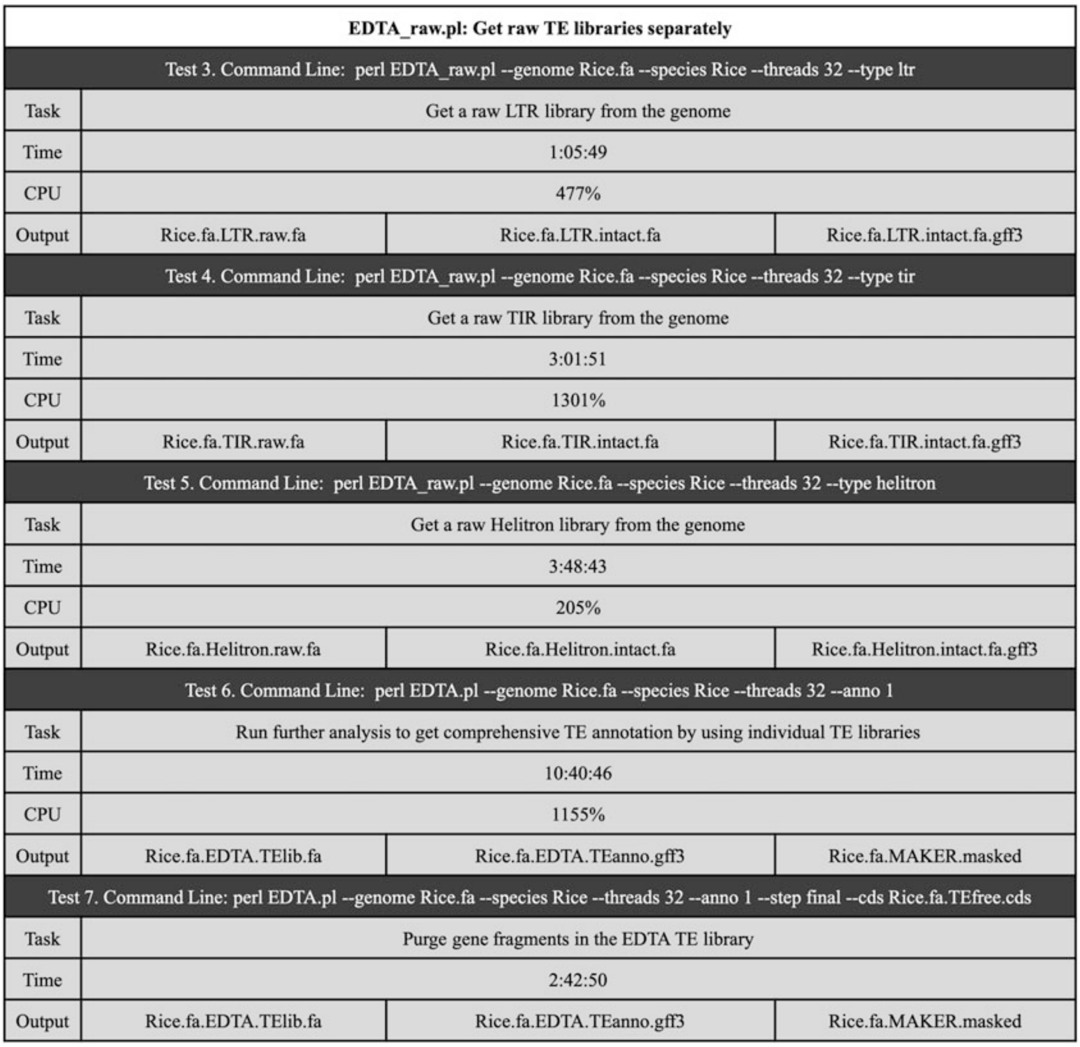


Fig. 4 Classification consistency for all TEs (**a**), nested TEs (**b**), and redundant TEs (**c**)



EDTA_raw.pl: Get raw TE libraries separately			
Test 3. Command Line: perl EDTA_raw.pl --genome Rice.fa --species Rice --threads 32 --type ltr			
Task	Get a raw LTR library from the genome		
Time	1:05:49		
CPU	477%		
Output	Rice.fa.LTR.raw.fa	Rice.fa.LTR.intact.fa	Rice.fa.LTR.intact.fa.gff3
Test 4. Command Line: perl EDTA_raw.pl --genome Rice.fa --species Rice --threads 32 --type tir			
Task	Get a raw TIR library from the genome		
Time	3:01:51		
CPU	1301%		
Output	Rice.fa.TIR.raw.fa	Rice.fa.TIR.intact.fa	Rice.fa.TIR.intact.fa.gff3
Test 5. Command Line: perl EDTA_raw.pl --genome Rice.fa --species Rice --threads 32 --type helitron			
Task	Get a raw Helitron library from the genome		
Time	3:48:43		
CPU	205%		
Output	Rice.fa.Helitron.raw.fa	Rice.fa.Helitron.intact.fa	Rice.fa.Helitron.intact.fa.gff3
Test 6. Command Line: perl EDTA.pl --genome Rice.fa --species Rice --threads 32 --anno 1			
Task	Run further analysis to get comprehensive TE annotation by using individual TE libraries		
Time	10:40:46		
CPU	1155%		
Output	Rice.fa.EDTA.TElib.fa	Rice.fa.EDTA.TEanno.gff3	Rice.fa.MAKER.masked
Test 7. Command Line: perl EDTA.pl --genome Rice.fa --species Rice --threads 32 --anno 1 --step final --cds Rice.fa.TEfree.cds			
Task	Purge gene fragments in the EDTA TE library		
Time	2:42:50		
Output	Rice.fa.EDTA.TElib.fa	Rice.fa.EDTA.TEanno.gff3	Rice.fa.MAKER.masked

Fig. 5 Examples of decomposed EDTA pipeline output using rice genome (Genome name: Rice.fa)

The evaluation results (Rice.fa.EDTA.TE.fa.stat.all.sum, Rice.fa.EDTA.TE.fa.stat.nested.sum, and Rice.fa.EDTA.TE.fa.stat.redun.sum) from the "--evaluate 1" option show confusion matrices of all TEs, nested TEs, and non-nested (redundant) TEs, respectively.

The "--evaluate 1" option did not affect the TE identification and annotation process but adds further analysis of annotated TEs by evaluating their classification consistency. This single step took an additional ~27 h. Figure 4 shows the results of the evaluation output.

EDTA_raw.pl allows users to decompose the pipeline and obtain the raw LTR, TIR, and Helitron libraries separately. This will shorten the amount of time by parallelizing the first step of EDTA. Users can generate LTR, TIR, and Helitron libraries in separate jobs by specifying "--type LTR", "--type TIR", and "--type Helitron", respectively.

Tests 3–6 represent the sequential breakdown of Test 1, in which EDTA first produced raw libraries of each TE type (Tests 3–5). Then, Test 6 runs further analyses to process the raw libraries and compile a comprehensive TE annotation from the results of Tests 3–5. Test 6 took an additional ~10 h to finish. On the other hand, if existing raw library results were present in the working directory (e.g., one or all steps of Tests 3–5 were finished), the default "--overwrite 0" option allows the EDTA program to use existing results, which saves ~8 h in Test 6 compared to Test 1.

3.5 Removal of Gene Fragments

Structurally annotated TEs likely contain gene fragments, especially for *Helitrons* and *Mutators*, which are known to capture gene sequences. On the other hand, false TEs that were identified in genic regions likely contain functional genes in full or in part. For both cases, gene-related sequences should be removed; otherwise, these sequences will mask similar sequences in the actual genes during the homology-based TE annotation process. Test 7 shows that the "--cds" function of EDTA could be used to remove gene-related sequences in the TE library (Fig. 6). The input gene sequence file must be TE-free (including only coding regions, no UTRs, no introns), which could be derived from the same genome, the previous version of the genome, or a genome of a closely related individual or species.

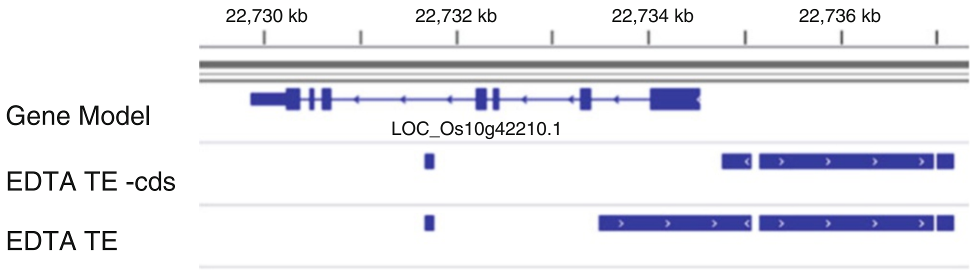


Fig. 6 Removal of gene fragments in the EDTA library using the "--cds" function. The putative, expressed enoyl-CoA-hydratase gene is partially masked by the EDTA library without using the "--cds" function

4 Notes

1. All of the commands in this tutorial are operated on Linux64 system.
2. Source code may be updated, the latest version of EDTA can be found at https://github.com/oushujun/EDTA.
3. Root privilege is required for docker installation and usage, which could be an inconvenience for HPC users.
4. We recommend users run the program on an HPC (High-Performance Computing) Cluster.
5. Names of the sequence in the input genome file should be short ($\leq$15 characters) and simple (i.e., letters, numbers, and underscores).
6. The "--overwrite" option will save some time if the previous run was interrupted.
7. All of the sequences in the file provided by "--curatedlib" option will be considered as the true TEs. Therefore, only provide high-quality TE libraries, otherwise, ignore this option.
8. The "--sensitive" mode is slow and MAY help to recover some TEs.
9. To enable the "--evaluate 1" function, "--anno 1" is required. The "--evaluate 1" step is slow and does not affect the annotation result.
10. Users can provide known gene position of this version of the genome assembly [BED format]. Coordinates specified in this file will be whitelisted from TE annotation to avoid over-masking.
11. TEs are classified into the superfamily level using the three-letter naming system reported in Wicker et al. [5].

Acknowledgments

We thank all the authors of the EDTA paper [8]. We also thank the Iowa State University High-Performance Computing facility for assistance with the storage and processing of large datasets. This research is supported by the USDA National Institute of Food and Agriculture Hatch project number IOW05282 (TP, WS), by State of Iowa funds (TP, WS), and by the NSF Plant Genome Research Program grant IOS-1744001 (MBH, SO). The authors declare there is no competing financial interest.

References

1. McClintock B (1948) Mutable loci in maize. Carnegie Inst Wash Year Book 47:155–169
2. McClintock B (1950) The origin and behavior of mutable loci in maize. Proc Natl Acad Sci 36:344–355
3. Schnable PS, Ware D, Fulton RS, Stein JC, Wei F, Pasternak S, Liang C, Zhang J, Fulton L, Graves TA (2009) The B73 maize genome: complexity, diversity, and dynamics. Science 326:1112–1115
4. Feschotte C, Jiang N, Wessler SR (2002) Plant transposable elements: where genetics meets genomics. Nat Rev Genet 3:329–341
5. Wicker T, Sabot F, Hua-Van A, Bennetzen JL, Capy P, Chalhoub B, Flavell A, Leroy P, Morgante M, Panaud O et al (2007) A unified classification system for eukaryotic transposable elements. Nat Rev Genet 8:973–982
6. Goerner-Potvin P, Bourque G (2018) Computational tools to unmask transposable elements. Nat Rev Genet 19:688–704
7. Lerat E (2010) Identifying repeats and transposable elements in sequenced genomes: how to find your way through the dense forest of programs. Heredity (Edinb) 104:520–533
8. Ou S, Su W, Liao Y, Chougule K, Agda JRA, Hellinga AJ, Lugo CSB, Elliott TA, Ware D, Peterson T, Jiang N, Hirsch CN, Hufford MB (2019) Benchmarking transposable element annotation methods for creation of a streamlined, comprehensive pipeline. Genome Biol 20 (1):275. https://doi.org/10.1186/s13059-019-1905-y
9. Ellinghaus D, Kurtz S, Willhoeft U (2008) LTRharvest, an efficient and flexible software for de novo detection of LTR retrotransposons. BMC Bioinformatics 9:18
10. Ou S, Jiang N (2019) LTR_FINDER_parallel: parallelization of LTR_FINDER enabling rapid identification of long terminal repeat retrotransposons. bioRxiv. https://doi.org/10.1101/722736
11. Ou S, Jiang N (2018) LTR_retriever: a highly accurate and sensitive program for identification of long terminal repeat retrotransposons. Plant Physiol 176:1410–1422
12. Shi J, Liang C (2019) Generic repeat finder: a high-sensitivity tool for genome-wide de novo repeat detection. Plant Physiol 180:1803–1815
13. Su W, Gu X, Peterson T (2019) TIR-learner, a new ensemble method for TIR transposable element annotation, provides evidence for abundant new transposable elements in the maize genome. Mol Plant 12:447–460
14. Xiong W, He L, Lai J, Dooner HK, Du C (2014) HelitronScanner uncovers a large overlooked cache of Helitron transposons in many plant genomes. Proc Natl Acad Sci U S A 111:10263–10268
15. Smit A, Hubley RR (2010) Open-1.0. Repeat masker website, 863
16. Pray LA (2008) Transposons: the jumping genes. Nat Educ 1:204
17. Szuplewska M, Czarnecki J, Bartosik D (2014) Autonomous and non-autonomous Tn. Mob Genet Elem 4:1–4
18. Su W, Sharma SP, Peterson T (2018) Evolutionary impacts of alternative transposition. In: Origin and evolution of biodiversity. Springer, Cham, pp 113–130
19. Anderson SN, Stitzer MC, Brohammer AB, Zhou P, Noshay JM, O'Connor CH, Hirsch CD, Ross-Ibarra J, Hirsch CN, Springer NM (2019) Transposable elements contribute to dynamic genome content in maize. Plant J 100:1052–1065
20. Kronmiller BA, Wise RP (2008) TEnest: automated chronological annotation and visualization of nested plant transposable elements. Plant Physiol 146:45–59