Return of the mitochondrial DNA

Case study of mitochondrial genome evolution in the genus Fusarium

Brankovics, Balázs

Publication date
2018

Document Version
Other version

License
Other

Link to publication

Chapter 2

GRAbB: Selective genome assembly

This chapter was published as:

B. Brankovics, H. Zhang, A.D. van Diepeningen, T.A.J. van der Lee, C. Waalwijk, and G.S. de Hoog. (2016) GRAbB: Selective assembly of genomic regions, a new niche for genomic research. *PLoS Computational Biology*, 12(6):e1004753. DOI: 10.1371/journal.pcbi.1004753

Authors' contributions: BB designed & developed the software package. BB, ADD, TAJL & CW designed the test cases. BB carried out the analysis. BB wrote the first draft. All authors (BB, HZ, ADD, TAJL, CW and GSH) helped shaping the manuscript. All authors approved the final version of the manuscript.

Abstract

GRAbB (Genomic Region Assembly by Baiting) is a new program that is dedicated to assemble specific genomic regions from NGS data. This approach is especially useful when dealing with multi copy regions, such as mitochondrial genomes and the rDNA region (repeats), parts of the genome that are often neglected or poorly assembled, although they contain interesting information from phylogenetic or epidemiologic perspectives, but also single copy regions can be assembled. The program is capable of targeting multiple regions within a single run. Furthermore, GRAbB can be used to extract specific loci from NGS data, based on homology, like sequences that are used for barcoding. To make the assembly specific, a known part of the region, such as the sequence of a PCR amplicon or a homologous sequence from a related species must be specified. By assembling only the region of interest, the assembly process is computationally much less demanding and may lead to assemblies of better quality. In this study the different applications and functionalities of the program are demonstrated such as: exhaustive assembly (rDNA region and mitochondrial genome), extracting homologous regions or genes (IGS, RPB1, RPB2 and TEF1a), as well as, extracting multiple regions within a single run. The program is also compared with MITObim, which is meant for the exhaustive assembly of a single target based on a similar query sequence. GRAbB is shown to be more efficient than MITObim in terms of speed, memory and disk usage. The other functionalities (handling multiple targets simultaneously and extracting homologous regions) of the new program are not matched by other programs. The program is available with explanatory documentation at `https://github.com/b-brankovics/grabb`. GRAbB has been tested on Ubuntu 12.04 and 14.04. Furthermore, GRAbB is available as a docker repository: brankovics/grabb.

Introduction

High throughput sequencing of whole genomes is becoming routine due to the advances in sequencing technologies and the reduction of the costs involved. Nowadays, more and more research groups are producing whole genome and/or transcriptome sequencing datasets that are made accessible for the research community.

However, in many studies not all the produced data are used effectively and most effort is dedicated to the nuclear genome. In cases where an assembly is available, the quality of some regions of interest may be poor due to the fact that most of the whole genome assembly programs use global optimization, which may result in suboptimal local assemblies. This is often the case with the genomes of organelles such as the mitochondrion. Mitogenome (mitochondrial genome) and ribosomal DNA region (18S rRNA - ITS1 - 5.8S rRNA - ITS2 - 28S rRNA – IGS) are generally not completely assembled, even when there is sufficient information in the NGS data [1]. These regions contain loci that are predominantly used for phylogenetic comparisons and species identification. Furthermore, there is no program that promises to extract barcoding loci form NGS reads.

The time and computational power required for assembling a specific region of the genome is less than that required for assembling the entire genome, while the quality of the assembled sequences may improve. Few resources exist to examine regions other than the nuclear genome. So far there is only one program, MITObim, which selectively assembles organellar genomes, and was developed for the mitogenome [1]. While other programs are aimed at re-assembly to improve an existing assembly [2] or at closing gaps [3], MITObim is designed to use the sequence of a related species as a reference for the assembly until either one of two completion criteria is met: i) a maximum number of iterations specified at invocation is reached, or ii) no new reads are found.

Here, we present GRAbB (Genomic Region Assembly by Baiting), a new versatile program that is designed to be flexible in terms of input options, assembly and completion criteria. The program uses a reference file to identify reads corresponding to the target region. For the assembly, the program can use two well-known assemblers, Edena [4, 5] and Velvet [6], but the source code is designed to be easily expandable to employ other assemblers, as demonstrated in the documentation. It is the first program that can handle multiple regions separately within a single run. Moreover, it is the first program that is able to extract homologous sequence regions, such as barcoding sites.

In this paper, the exhaustive assembly capabilities of GRAbB were compared with that of MITObim. Besides this comparison, GRAbB was also run on NGS data to demonstrate its functionalities targeting multiple regions with diverse completion criteria.

Design and Implementation

Algorithm overview

GRAbB is written in Perl and it uses only modules that are part of the core distribution of Perl. In addition to basic UNIX commands, the following third-party programs are used by GRAbB: mirabait (from the MIRA package; [7]), Seqtk (`https://github.com/lh3/seqtk`), Edena [4, 5], Velvet [6] and Exonerate [8].

The program is designed to be versatile and flexible with the following functionalities:

- To use pairing information,
- To use additional bait sequences,
- To assemble multiple regions separately in a single run,
- To use any of a range of completion criteria (different ones for each region are allowed in a multi-assembly run).

These functionalities are detailed below at the appropriate steps of the algorithm.

Input

As shown in Fig. 2.1a, there are two mandatory input files (the reference file and at least one read file). In addition, there is an optional input file (the additional bait file).

The reference file should be a FASTA formatted file that contains one or more sequences and the identifier of each sequence has to be unique (as required by mirabait). When the file contains multiple sequences and the program is run in multi-mode, the reference file is split into separate reference files that contain only a single sequence. Furthermore, the description lines may contain specification for the completion criterion to be used for the given sequence. These criteria are used only if the multi-diff mode is selected. Since the read selection is based on exact k-mer (31 bp) matching, the reference sequence does not have to be highly similar to the target sequence.

Multiple read files can be specified as input. If two read files are given, it is assumed that reads are paired, but in single-mode, reads are considered as single reads. The program recognizes read pairs based on the identifiers of the reads, and thus this should be identical for both sequences. The read files may be in FASTA or FASTQ format and may be compressed (using gnuzip). When Edena is selected as the assembler program, the reads must be of the same length.

The additional bait file has to be in FASTA format. This file is used for the first baiting step by merging it with the reference to create a unified bait file. The file may contain more than one sequence, however, unlike the reference file, the identifiers do not have to be unique.

Main loop

The main loop of GRAbB can be summarized as follows: i) creating the bait file, ii) finding reads by baiting, iii) *de novo* assembly of the selected reads, and iv) testing completion (Fig. 2.1b). When multi-mode is selected, the general baiting step is followed by specific baiting, *de novo* assembly and completion testing steps for each of the threads (Fig. 2.1c). The threads are generated by splitting the reference file into single-entry FASTA files. These newly created reference files are used as bait for the initial specific baiting steps for the given thread (Fig. 2.1c). At the end of each cycle of the main loop, the program checks whether there is any thread that is not completed yet: then it continues or stops accordingly (Fig. 2.1d).

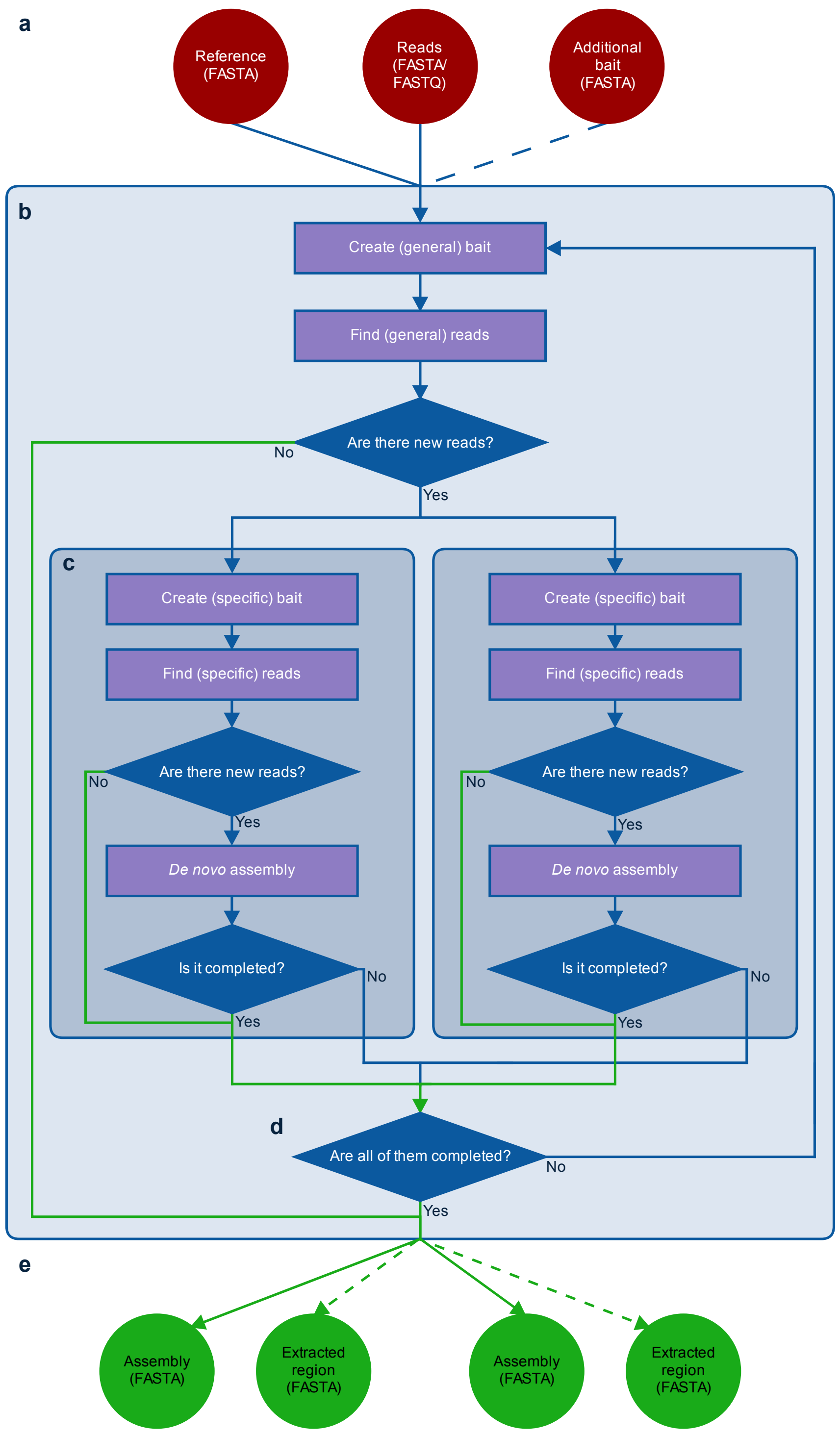

Figure 2.1: **Schematic workflow of GRAbB.**
a: The input files for GRAbB. **b:** The main loop of GRAbB. **c:** Internal loops for the individual threads, the number of threads is based on the number of sequences in the reference file in multi-mode, otherwise there is only a single thread. **d:** Check whether there are active threads left or not. **e:** Output files for each thread. Dashed arrows indicate optional files (only in exonerate-mode). Green arrows indicate termination signal of a given thread or run. If GRAbB is not run in multi-mode, then the first step within the internal loop is the *de novo* assembly, the preceding ones are skipped.

i) Creating the bait file Before the first baiting step, the specified reference and the additional bait file are combined into a single internal bait file. In multi-mode, the reference sequences corresponding to the given thread will be used for the first specific baiting step.

In each of the later iterations of the program, the latest assembly is used as (internal) bait file. For the general baiting step in multi-mode the individual assembly files are combined into a single (general) bait file.

ii) Finding reads Reads corresponding to the target sequence are identified by using exact k-mer (31 bp) matching that is implemented by mirabait. The names of the reads thus identified are collected and added to the list of read names from previous iterations (there is a separate list for general baiting and for each of the threads.) If there are no new reads identified, the program stops the iteration: either for the given thread or for all the threads, depending on which baiting step generated this result. The identified reads are collected from the read files using Seqtk into internal read files.

By using a general baiting step before the specific baiting, it is possible to reduce the required runtime, since the large input read file(s) is/are only screened once per iteration and the specific baiting is confined to screening the reads that are already identified to be specific during the general baiting.

iii) *De novo* assembly The program can use two assemblers, Edena and Velvet, by default, however there is a skeleton code to add other assemblers to be used by the program. Even though Edena is the default assembler, by using command line options other assemblers may be selected as well.

The fact that single- or paired-mode is selected is passed on to the assembler program that assembles the specific reads *de novo*. Furthermore, additional arguments may be passed, such as overlap cutoff size (Edena), to the assembler program at the invocation of the main program.

At the invocation of GRAbB it is possible to specify a length filter that excludes the contigs from the assembly that are shorter than the specified length from the completion testing, and these are not used for generating the bait file for the next iteration either. This enables to minimize the effect of repeat regions on the assembly process.

iv) Testing completion There are multiple completion criteria that can be specified for the program. These can be specified for each thread separately or for all of the threads at once. In addition, it is possible to specify multiple criteria for the same thread or run. The program stops when any one of these criteria is met.

The first completion criterion is implicit: the program stops if no new information is found. This means that either there are no new reads found or that the new assembly is identical to the bait used for the current iteration. If no other criterion is specified, then the run will be exhaustive, since it iterates until it cannot find any new information.

The simplest explicit completion criterion is the size criterion. There are three options that can be used for this setting: i) total assembly size, ii) length of the longest contig, and iii) N50 value of the assembly. This criterion is tested independently for each of the threads in multi-mode, or alternatively for the single thread. Multiple criteria can be used in a single run; this also applies to the different size criteria. These settings are useful when exploring the vicinity of a specified sequence region.

The final criterion is matching a homologous sequence. In this case, the specific reference sequence is used to identify the homologous region within the assembly. In order to identify the matching region, GRAbB uses Exonerate with settings that ensure that the entire reference sequence is aligned to the contigs. This makes it possible to match sequences that are somewhat dissimilar to the reference and may contain indels, causing gaps in the alignment. If the reference could be aligned to the contigs over its entire length, the completion criterion is met, and the matched region is extracted from the assembly in the same orientation as the reference sequence and saved to a separate output file. Otherwise, the length of the matched region is compared to that of the previous iteration. In case the length of the matched region has increased, the thread or run will continue, otherwise the thread or run will stop.

Up- and downstream processes

Besides the main program, there are some helper programs included within the package that can help with preprocessing the input data or to further process the output data. These are detailed in the documentation (`https://github.com/b-brankovics/grabb`).

Results

Simulated data

To compare the performance of GRAbB with that of MITObim 1.7 (using MIRA 4.0.2; `https://github.com/chrishah/MITObim.git`; [1]), simulated sequencing data generated based on the *F. graminearum* mitogenome sequence available from NCBI (NC_009493) were used together with either the original *F. graminearum* (NC_009493) or *F. oxysporum* (NC_017930) mitogenome as reference. MITObim has two assembly settings: *de novo* and mapping. However, since the mapping setting did not prove to be applicable to reconstruct the *F. graminearum* sequence by using *F. oxysporum* as reference, only the *de novo* setting was used for the comparison. GRAbB produced a single contig in both cases. MITObim produced multiple contigs in both runs, however a single contig could be obtained from the LargeContigs MAF format file produced by using miraconvert. All the single contigs produced could be circularized using the merge_contigs.pl and were identical to the original sequence. Although both MITObim and GRAbB could assemble the genome correctly, GRAbB did it more efficiently and demanded less disk space for the output folders Table 2.1.

NGS Data

To demonstrate the capabilities of GRAbB, it was run on paired-end reads derived from *Fusarium oxysporum* f. sp. *cubense* race 4 strain B2 (SRR550152; [9]) and the results were compared to the assembly published for this strain (AMGQ01). In the single run multiple

Table 2.1: **Comparison between GRAbB and MITObim using our *in silico* generated paired-end read library.**

Program	**Reference**	**N**	**CPU usage (%)***	**Time elapsed (m:s)**	**Maximum memory usage (Mb)**	**Output folder size (Mb)**
GRAbB	*F. graminearum*	2	157	00:15.52	285.09	70
GRAbB	*F. oxysporum*	10	161	01:49.24	286.05	527
MITObim	*F. graminearum*	3	150	09:42.15	1,354.86	4,126
MITObim	*F. oxysporum*	9	150	25:37.71	1,358.81	10,667

Data were generated by /usr/bin/time on Ubuntu 14.04.1. Both programs were run on the same simulated read library derived from *F. graminearum* mitogenome. Reference column shows which mitogenome was the input for the run. N is the number of iteration it took to complete the assembly. Output folder size refers to the disk space occupied by the output files of the run. This was assessed using /usr/bin/du on Ubuntu 14.04.1.
* The program, /usr/bin/time, assigns 100 % for each of the cores of the processor, and the computer used has four cores, thus the maximal CPU usage is 400 %.

regions were targeted simultaneously: mitochondrial genome, rDNA region, barcoding loci (IGS, TEF1a, RPB1 and RPB2). The goal of the threads were as follows: assemble the complete mitogenome, assemble the complete rDNA region, extract barcoding loci, compare the extraction results using sequences from different species and test the effect on the extraction when the query contains a gap (Table 2.2).

Mitochondrial genome The mitochondrial genome of the strain was assembled into a single contig, which had 9-9 oligoC sequences at both ends. To solve this problem, Edena was rerun on the sequences specific for the mitogenome, but the reads were trimmed to be 70 bp long. By removing part of the ‘3 end of the reads, the assembly improved. The final assembly contained a single contig, and the two ends of the contig overlapped by 60 bp, indicating a circular nature. This is in agreement with the fact that the mitochondrial genome is circular in *Fusarium* spp. [10].

The circularized sequence was 49697 bp long. The reference sequence (NC_017930) lacked a region (approximately 12 kbp long) that is present in the mitogenomes of all other *Fusarium* spp. sequenced thus far [10]. The new sequence contained this region, in addition to three new introns in protein coding genes.

In the AMGQ01 assembly there are 30 contigs containing mitochondrial sequences. These sequences cover only 39.07 % of the mitogenome.

Ribosomal DNA region (18S rRNA - ITS1 - 5.8S rRNA - ITS2 - 28S rRNA - IGS) The assembly for the rDNA region contained two contigs, and the contigs could be joined because they overlapped by 99 bp, indicating a circular or repetitive morphology. The ribosomal DNA region is not circular, but since it is present within the genome in

Table 2.2: **Overview of the multi-query run.**

Thread number	**Target**	**Reference**				**Result size (bp)**	**Assembly type**	**Number of iterations**	
		Region	**Species**	**Accession number**	**Size (bp)**			**Specific**	**General**
1	mt	mt	*F. oxysporum*	NC_017930	34,477	49,697	exhaustive	18	3
2	rDNA	IGS	*F. oxysporum*	FD_00403_IGS	1,449	7,872	exhaustive	2	10
3	IGS	IGS	*F. oxysporum*	FD_00403_IGS	1,449	1,446	exonerate	1	1
4	TEF1a	TEF1a	*F. oxysporum*	FD_00403_EF-1a	689	684	exonerate	1	1
5	TEF1a	TEF1a	*F. oxysporum*	FD_00403_EF-1a*	652	647	exonerate	1	1
6	TEF1a	TEF1a	*F. graminearum*	FD_00001_EF-1a	643	647	exonerate	3	1
7	TEF1a	TEF1a	*F. solani*	FD_01036_EF-1a	677	647	exonerate	4	1
8	RPB1	RPB1	*F. oxysporum*	FD_02003_RPB1	1,606	1,606	exonerate	1	1
9	RPB2	RPB2	*F. oxysporum*	FD_02003_RPB2	1,762	1,899	exonerate	1	1
10	RPB2	RPB2	*F. oxysporum*	FD_00120_RPB2-57 FD_00120_RPB2-711	879 860	1,876	exonerate	1	1

The accession numbers are either GenBank accessions or FusariumID (http://isolate.fusariumdb.org) accessions. The assembly type specifies which assembly mode was selected for the given thread during the GRAbB run. Multiple specific iterations are run only during the first general iteration.
*: This sequence was truncated to span the same region as FD_00001_EF-1a and FD_01036_EF-1a.

multiple copies as direct repeats [11] it can be considered as a circular sequence from a bioinformatics perspective.

The circularized sequence was 7,872 bp long. In the AMGQ01 assembly there are 14 contigs containing rDNA region sequences. These sequences cover only 63.89 % of the rDNA region.

Barcoding loci (IGS, TEF1a, RPB1 and RPB2) All the threads targeting barcoding loci finished by matching the query sequence in the assemblies. The result for the threads of the three TEF1a sequences from different species (*F. oxysporum*, *F. graminearum* and *F. solani*) returned identical sequences. The IGS sequence had only partial matches when compared with the AMGQ01 assembly, but matched completely with the newly assembled rDNA region sequence. The TEF1a, RPB1 and RPB2 sequences were identical to matching sequences from the assembly (AMGQ01), however, the RPB2 locus was found at the junction of two contigs in the assembly, which overlapped by 5 bp.

Availability and Future Directions

GRAbB is available with explanatory documentation at https://github.com/b-brankovics/grabb. Furthermore, GRAbB is available as a docker repository: brankovics/grabb. The GitHub package has been tested on Ubunutu 12.04 and 14.04. The docker repository can be used on any platform that has a working installation of docker. The program is made available with MIT license.

Bibliography

[1] Hahn C, Bachmann L, Chevreux B. Reconstructing mitochondrial genomes directly from genomic next-generation sequencing reads - a baiting and iterative mapping approach. Nucleic Acids Research. 2013;41(13):e129.

[2] Green RE, Malaspinas AS, Krause J, Briggs AW, Johnson PLF, Uhler C, et al. A complete Neandertal mitochondrial genome sequence determined by high-throughput sequencing. Cell. 2008 aug;134(3):416–426.

[3] Tsai IJ, Otto TD, Berriman M. Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps. Genome Biol. 2010;11(4):R41.

[4] Hernandez D, François P, Farinelli L, Osterås M, Schrenzel J, Østerås M, et al. *De novo* bacterial genome sequencing: millions of very short reads assembled on a desktop computer. Genome Res. 2008;18(5):802–809.

[5] Hernandez D, Tewhey R, Veyrieras JB, Farinelli L, Østerås M, François P, et al. *De novo* finished 2.8 Mbp *Staphylococcus aureus* genome assembly from 100 bp short and long range paired-end reads. Bioinformatics. 2014;30(1):40–49.

[6] Zerbino DR, Birney E. Velvet: Algorithms for *de novo* short read assembly using de Bruijn graphs. Genome Res. 2008;18(5):821–829.

[7] Chevreux B, Wetter T, Suhai S. Genome sequence assembly using trace signals and additional sequence information. In: Computer Science and Biology: Proceedings of the German Conference on Bioinformatics (GCB).; 1999. p. 45–56.

[8] Slater GSC, Birney E. Automated generation of heuristics for biological sequence comparison. BMC Bioinformatics. 2005;6:31.

[9] Guo L, Han L, Yang L, Zeng H, Fan D, Zhu Y, et al. Genome and transcriptome analysis of the fungal pathogen *Fusarium oxysporum* f. sp. *cubense* causing banana vascular wilt disease. PLoS ONE. 2014;9(4):e95543.

[10] Fourie G, van der Merwe NA, Wingfield BD, Bogale M, Tudzynski B, Wingfield MJ, et al. Evidence for inter-specific recombination among the mitochondrial genomes of *Fusarium* species in the *Gibberella fujikuroi* complex. BMC genomics. 2013;14(1):605.

[11] Hillis DM, Dixon MT. Ribosomal DNA: molecular evolution and phylogenetic inference. Q Rev Biol. 1991 dec;66(4):411–453.

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

[13] Edgar RC. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics. 2004 aug;19(5):113.

[14] Bradley RK, Roberts A, Smoot M, Juvekar S, Do J, Dewey C, et al. Fast statistical alignment. PLoS Comput Biol. 2009;5(5):e1000392.

Supplementary material

Sequence similarity between target and query sequences

Sequence similarity between query and target have been calculate by two approaches: sequence alignment and 31-mer comparison.

Sequence alignment: Target and query have been aligned using MUSCLE [12, 13] or FSA [14], and the identical positions have been reported. The number of identical positions have been divided by the length of the alignment to calculate the percentage.

31-mer approach: The number of 31-mers shared by the query and target have been calculated and reported. The percentage is calculate by dividing the number of shared 31-mers with the number of unique 31-mers in the target sequence.

Table 2.S1: Sequence similarity of published mitochondrial genomes

Species	**Length (bp)**	**Identical (bp)**	**Identical (%)**	**Shared 31-mers**	**31-mer identity (%)**
F. graminearum[a]	95676	95676	100.00	189584	100.00
F. oxysporum[b]	47409	24669	24.04	18304	9.65

[a]GenBank accession: NC_009493
[b]GenBank accession: NC_017930

Table 2.S2: Sequence similarity of TEF1a sequences

Species	**Length (bp)**	**Identical (bp)**	**Identical (%)**	**Shared 31-mers**	**31-mer identity (%)**
F. oxysporum[a]	647	647	100.00	1234	100.00
F. oxysporum[b]	652	636	97.40	702	56.89
F. graminearum[c]	643	524	77.86	200	16.21
F. solani[d]	677	499	71.49	232	18.87

[a]GenBank accession: AMGQ01 (Sequence was truncated to match FD_00001_EF-1a.)
[b]FusariumID (http://isolate.fusariumdb.org) accession: FD_00403_EF-1a (Sequence was truncated to match FD_00001_EF-1a.)
[c]FusariumID (http://isolate.fusariumdb.org) accession: FD_00001_EF-1a
[d]FusariumID (http://isolate.fusariumdb.org) accession: FD_01036_EF-1a

Table 2.S3: Sequence similarity between queries and targets in the multi-query run.

Thread number	Target	Species	Accession number	Query size (bp)	Target size (bp)	Identical (bp)	Identical (%)	Shared 31-mers	31-mer identity (%)
1	mt	*F. oxysporum*	NC_017930	34,477	49,697	24669	65.24	45522	46.17
2	rDNA	*F. oxysporum*	FD_00403_IGS	1,449	7,872	1406	17.85	1336	8.49
3	IGS	*F. oxysporum*	FD_00403_IGS	1,449	1,446	1406	96.90	1336	47.18
4	TEF1a	*F. oxysporum*	FD_00403_EF-1a	689	684	673	97.54	774	59.17
5	TEF1a	*F. oxysporum*	FD_00403_EF-1a*	652	647	636	97.40	702	56.89
6	TEF1a	*F. graminearum*	FD_00001_EF-1a	643	647	524	77.86	200	16.21
7	TEF1a	*F. solani*	FD_01036_EF-1a	677	647	499	71.49	232	18.80
8	RPB1	*F. oxysporum*	FD_02003_RPB1	1,606	1,606	1590	99.00	2090	66.31
9	RPB2	*F. oxysporum*	FD_02003_RPB2	1,762	1,899	1753	92.31	2902	77.64
10	RPB2	*F. oxysporum*	FD_00120_RPB2-57 FD_00120_RPB2-711	879 860	1,876	1724	91.90	2486	67.33

The accession numbers are either GenBank accessions or FusariumID (http://isolate.fusariumdb.org) accessions.
*: This sequence was truncated to span the same region as FD_00001_EF-1a and FD_01036_EF-1a.