Supplementary material for:

stringMLST: a fast k-mer based tool for multi locus sequence typing

Anuj Gupta, I. King Jordan, Lavanya Rishishwar

Supplementary Methods

Conceptual overview of the stringMLST algorithm

stringMLST is a tool for detecting the sequence type (ST) of a bacterial isolate directly from the genome sequence reads. stringMLST predicts the ST of an isolate in a completely assembly- and alignment-free manner. The tool is designed in a light-weight, platform-independent fashion with minimal dependencies. It has a rapid runtime and small memory footprint.

stringMLST works on the basis of exact string matching, which computers are capable of performing in a much simpler and faster manner compared to inexact matches that are used for sequence alignment. The tool's simple underlying algorithm works by k-merizing the input FASTQ reads and matching them to the k-merized allele sequences using a standard hash table. For each match, stringMLST records the alleles in which the k-mer was found and at the end of the run, it reports the allele number with the maximum k-mer hits for each locus. The resulting allele numbers for each locus form the allelic profile of the input isolate, which is in turn used to determine its ST.

It should be noted that stringMLST can be run on other larger-scale typing schemes such as rMLST and cgMLST. As the number of loci and allele sequences increases, the runtime and memory consumption of the tool increase.

stringMLST's workflow (Supplementary Fig. S1 and S2) is divided into two routines:

1) Database building
2) ST discovery

1. Database building

The database building routine begins by k-merizing all allele sequences from each of the loci in the typing scheme. k-merizing entails going through a sequence, with a window size of k and a step size of 1, and recording the resulting sequence substring (k-mer) in a hash table data structure. The reverse complement of all the k-mers are also recorded to account for sequences on both strands. stringMLST then creates a k-mer to locus (or loci if more than one contains a given k-mer) relationship matrix by tracking the k-mers and the alleles/loci in which the k-mers were found. By default, stringMLST will create a database with a k value of 35, but this can be overridden by the user. An additional weight file is also created for cases where the alleles differ in over 5% of their sequence length. This database is then used in the next routine to discover the ST of the isolate.

2. ST discovery

The process of ST discovery can conceptually be broken down into three stages – 1) filtering, 2) counting, and 3) reporting.

1) Filtering: In the filtering stage, stringMLST discards sequences that are uninformative to speed up the process of ST discovery. Since MLST loci account for a small fraction of the bacterial genome, the vast majority of the sequencing reads do not come from them and are thereby uninformative for the process of ST discovery. In addition to providing a substantial speed up in the ST discovery process, the filtering process ameliorates the accuracy by reducing the background noise from the redundant sequence reads. For filtering, stringMLST discards a sequence read if the k-mer situated at the middle of the sequence read does not have a match in the stringMLST database created in the previous routine. The choice of the middle k-mer helps in assessing the overall informativeness of the sequence read, as the middle k-mers are most likely to capture any overlap with sequences from the allele database. Evaluation of

middle k-mers also helps to circumvent the issue of sequence contamination from untrimmed primer/adapter present at the ends of the sequence reads.

2) Counting: Sequence reads whose middle k-mers have a match in the database are k-merized in the counting stage. Each k-mer is searched in the database and for each database match, the corresponding locus (or loci) and alleles are recorded. A counter is incremented for each allele whose constituent k-mer was matched. In the event a locus has weights associated with it, the counter is incremented by the associated weight factor. k-mers with no match in the database are discarded. With this approach, users are not required to do any quality control or trimming of the data beforehand, and sequence contaminations will not affect the algorithm's performance. Once all of the sequence reads have been processed, stringMLST identifies alleles at each locus with the maximum counter value, *i.e.*, the allelic profile.

3) Reporting: The generated allelic profile of the input sample is used to find the corresponding ST based on the profile definition file, which is then reported in the final stage.

Description and pseudocode for the stringMLST algorithm

1. Database building

Input: Allele sequence files (one or multiple) in a multi-FASTA format with the sequence description being the locus name and corresponding allele number for each locus in the MLST schema

Output: A database file and a weight file

k-mer Distribution File – A tab-separated file with the following format:

`[k-mer`$_N$`]  [Locus name]  [Comma separated list of alleles for this locus]`

List of alleles in which the k-mer is present is specified in a comma-separated format (with no spaces)

Weight file – A tab-separated file with the following format:

`[Locus name and allele number]  [weight factor]`

k-mer Distribution Algorithm:

1. Open input files, one at a time
2. For each sequence in the file, find:
    a. Locus name
    b. Allele number
    c. All the unique k-mers
3. Store the k-mers in a hash data structure associating the k-mer to the loci and corresponding allele number
4. Print the hash to output file

k-mer Distribution Pseudocode *(Python notations)*:

```
db = {}
for file in (inputFiles) :
       open file;
       for sequence in (file) :
              id = sequence.id.split('_')
              locus = id[0]
              allele_number = id[1]
              find all k-mers and their reverse compliments = k-mers[]
              for k-mer in (k-mers):
                     db[k-mer] = {}
                     db[k-mer][locus] = alleleList.append(allele_number)

for k-mer in (db.keys()):
       print tab separated k-mer, locus and corresponding allele list;
```

Weight File Algorithm:

1. For each locus, calculate the average sequence length of all the alleles
2. For each allele in a locus, calculate the ratio of sequence length to mean sequence length (*i.e.*, the weight factor)
3. Store the allele and the weight factor in a weight file if the ratio > 1.05 or < 0.95

Weight File Pseudocode *(Python notations)*:

```
weightDB ={}
for file in (inputFiles) :
       open file
       average_length = mean(all sequence lengths)
       for sequence in (file){
              factor = sequence_length/average_length
              if factor > 1.05 or factor < 0.95 :
                     weightDB[locus][allele] = factor
```

2. ST discovery

Input: Raw FASTQ sequencing reads (single end or paired end)

Database: DB files created in previous routine

Output: Allelic profile and sequence type

ST Discovery Algorithm:

1. Load the DB file as a hash data structure associating k-mer and locus to the list of alleles the k-mer is found in db[k-mer][locus] = [list of alleles]
2. Filtering stage: For each sequence in the reads file(s) assess its informativeness by checking if the middle k-mer of the read is found in the database. If the middle k-mer is found in the database, advance the read to the next stage,otherwise discard the read
3. Counting stage: k-merize the entire read. k-mers are formed by extracting the substrings from bases 1 to k, 2 to k+1, 3 to k+2, etc. For each k-mer, search for a match in the database. If a match is found, increment a counter corresponding to the allele; f a weight exists for the alleles, increment the counter by the weight factor. If no match is found, the k-mer is discarded
4. Reporting stage: For each locus, find the allele with maximum counter value. Their corresponding allele numbers create the allelic profile of the sample
5. Find the ST corresponding to the allelic profile from the profile definition file

ST Discovery Pseudocode:

```
for k-mer in db_file:
       save db[k-mer][locus] = [list of allele numbers]


for sequence in input_reads_file:
       find middle_k-mer for the read
       if middle_k-mer in db.keys():
              is_useful;
       else:
              discard;


for read is_useful:
       make all_k-mer_list of all k-mers in read
       for k-mer in all_k-mer_list:
              if k-mer in db.keys():
                     for all combinations of loci, allelenumbers for k-mer
                            k-mer_count_profile[loci][allele_number] += 1

for locus in k-mer_count_profile:
       for allele_number in k-mer_count_profile[locus]:
```

```
            weighted_profile = allele_weight * k-mer_count_profile[locus][allele]

for locus in weighted_profile :
      for allele_number in weighted_profile [locus]:
            find allele_number corresponding to max(weighted_profile)
```

Input file format

The database building routine requires the ST profile definition file and allele sequence files. The profile definition file is a tab separated file that contains the ST and the allele profile corresponding to the ST. An example of the profile definition file is shown below:

```
ST    abcZ  adk   aroE  fumC  gdh   pdhC  pgm
1     1     3     1     1     1     1     3
2     1     3     4     7     1     1     3
3     1     3     1     1     1     23    13
4     1     3     3     1     4     2     3
```

Each allele sequence file (one for each locus in the MLST schema) is a standard multi-FASTA file with the description for each allele sequence being the locus name with the allele number. An example *abcZ* allele sequence is shown below:

```
>abcZ_1
TTTGATACTGTTGCCGA…
>abcZ_2
TTTGATACCGTTGCCGA…
>abcZ_3
TTTGATACCGTTGCGAA…
>abcZ_4
TTTGATACCGTTGCCAA…
```

These files can be obtained from PubMLST/BIGSdb or can be created by the user. An accompanying configuration file is also required to describe the location of profile definition and allele sequence files. An example configuration file is shown below:

```
[loci]
abcZ  /data/home/stringMLST/pubmlst/Neisseria_sp/abcZ.fa
adk   /data/home/stringMLST/pubmlst/Neisseria_sp/adk.fa
aroE  /data/home/stringMLST/pubmlst/Neisseria_sp/aroE.fa
fumC  /data/home/stringMLST/pubmlst/Neisseria_sp/fumC.fa
gdh   /data/home/stringMLST/pubmlst/Neisseria_sp/gdh.fa
pdhC  /data/home/stringMLST/pubmlst/Neisseria_sp/pdhC.fa
pgm   /data/home/stringMLST/pubmlst/Neisseria_sp/pgm.fa

[profile]
profile     /data/home/stringMLST/pubmlst/Neisseria_sp/neisseria.txt
```

We provide a set of these files, as downloaded from PubMLST database, at stringMLST's website:
http://jordan.biology.gatech.edu/page/software/stringMLST

The ST discovery routine requires standard FASTQ sequence reads files. The files can be single-end or paired-end files. It should be noted that stringMLST does not account for the pairing information present in the sequence reads files; sequence reads are treated independently of one another. Having the paired end file effectively doubles the number of reads for the sample and thus provides more k-mer support for the correct allele.

Supplementary Results

Testing data used for performance evaluation

As described in the Performance Evaluation section, a total of 1,042 sequence read samples from four species – *Campylobacter jejuni*, *Chlamydia trachomatis*, *Neisseria meningitidis* and *Streptococcus pneumonia* – were obtained from PubMLST/EBI ENA database (Jolley and Maiden, 2010) (Supplementary Table 1). The four species were chosen because each had at least 10 sets of whole genome sequence reads, and known ST information, available for analysis. The majority of the samples were of *Neisseria meningitidis* (1,012 samples) followed by *Streptococcus pneumoniae*, *Campylobacter jejuni* and *Chlamydia trachomatis* (10 samples each). The complete list of samples tested here can be found in Supplementary File S1. These samples were all sequenced on Illumina GAIIx or Illumina HiSeq 2000 or 2500 sequencing platforms. The samples varied in the read length and the depth of sequencing (Supplementary Fig S3). The relationship of sequencing depth to the number of k-mers supporting the best matching allele in each locus, in the four species (40 samples), is shown in Supplementary Fig S4. The read length varied from as low as 37 bp to a maximum of 151bp. The depth of sequencing showed extremes with the lowest sequencing depth of ~5X to an excess of 1,500X. This allowed us to assess the performance of stringMLST on sequence read data sets from across wide range of parameters. It should be also be noted that at least 43 out of the 1,042 samples tested here had some level of sequence contamination (residual primer/adapter sequence) present in them. No quality control or preprocessing was done on any of the FASTQ samples tested here.

Testing environment

Most of the code testing was done on a Linux based computation server (24-core; 64 GB memory). For a randomly chosen small set of reads, stringMLST was also run on a Windows based small desktop environment (2-core; 4 GB memory).

Notes on choosing the optimal k value

The size of the database and the memory footprint of stringMLST are dependent on the size of the k-mer (Supplementary Fig S5). Increasing the k-mer value increases the specificity of the tool and increases the runtime. However, higher values of k also increases the size of the database in an exponential manner. Surprisingly, the values of k tested here do not seem to affect the performance of the tool (Supplementary Table S2). We recommend a minimum k-mer length of 31 bp.

Running stringMLST on larger-scale gene-based typing schemes

stringMLST can be used for any gene-based typing scheme including user-designed schemes and larger-scale schemes that employ scores or hundreds of loci. To demonstrate its ability to work on larger-scale typing schemes, stringMLST was additionally run on two other popular typing schemes – ribosomal MLST (rMLST) on 53 loci and core genome MLST (cgMLST) on 1,605 loci. For both of these schemes, profile definitions and allele sequences were obtained from PubMLST. These files were utilized to construct the stringMLST database and then were used to assign scheme-specific STs for 20 *N. meningitidis* samples corresponding to the 10 most common STs defined by the traditional 7 loci MLST scheme (subset from accuracy test data; Supplementary Table 1 and Supplementary File 1). Pairs of genomes from the same MLST groups were chosen owing to the expectation that pairs of genomes from the same ST should group together when typed using different (larger-scale) typing methods, as has been shown previously for *N. meningitidis* (Bratcher, et al., 2014). The resulting rMLST and cgMLST ST and allele assignments were compared to the assignments obtained by performing a manual BLAST (Camacho, et al., 2009) based searching approach. For the BLAST approach, read sequences were assembled using the SPAdes assembler (Bankevich, et al., 2012) followed by BLAST searches against the allele sequences. A small fraction of loci (41 out of 32,979) were not predicted by BLAST (potentially due to the draft nature of the assembly). The rest of loci that were detected by the BLAST-based method, stringMLST was correctly able to predict 95.2% (1,009 alleles out of 1,060) of the rMLST alleles and 90.8% (28,976 alleles out of 31,919) of the cgMLST alleles. Given the variable file sizes (sequencing depth) of the files tested here, the runtime of the tool was measured as rate defined as the number of kilobytes (Kb) of data processed per second. The average rate for predicting the cgMLST and rMLST for all the samples was 43.0 Kb/s and 516.7 Kb/s (see Supplementary File 1, sheet "N. meningitidis_20SamplesSubset"). It should be noted that the stringMLST approach

is fully automated, requiring only a single command, whereas the BLAST based approach requires numerous independent analysis steps.

The typing accuracy of stringMLST for larger-scale rMLST and cgMLST approaches was also compared to results from Average Nucleotide Identity (ANI) (Richter and Rossello-Mora, 2009). ANI is a whole genome based comparative measure that quantifies similarity between a pair of genomes. For each pair of genomes, ANI values were computed using the dnadiff utility of the MUMmer package (Kurtz, et al., 2004). ANI values were converted to distances by subtracting the ANI value from 100. The resulting distance matrix was utilized to construct a Neighbor-Joining (Saitou and Nei, 1987) tree as implemented in MEGA 7 (Kumar, et al., 2016). For the construction of rMLST and cgMLST trees, allele calls from stringMLST for rMLST and cgMLST were converted to allelic distances (defined as the fraction of alleles each pair of genome differs by) and the resulting distance matrix was utilized to construct a distance tree using the Neighbor-Joining (Saitou and Nei, 1987) tree as implemented in MEGA 7 (Kumar, et al., 2016). For both the rMLST and cgMLST trees generated with stringMLST, pairs of genomes from 9 out of 10 STs were grouped together as terminal, monophyletic clades as is expected (90% accuracy; Supplementary Figure 6). The ANI whole genome phylogeny groups pairs of genomes from the same STs together 8 out of 10 times (80% accuracy). Overall, stringMLST and ANI generate phylogenies were found to be largely concordant with each other (Supplementary Figure 6), further elucidating the ability of stringMLST in dealing with larger typing schemes.

The simple paradigm of exact string matching that powers stringMLST is a departure from the current ideological framework in the field of sequence typing, and stringMLST clearly demonstrates that this paradigm has the potential of being valuable with larger typing schemes. Despite k-mer matching being widely adopted in many fields of bioinformatics (Andrews, 2010; Bankevich, et al., 2012; Ondov, et al., 2016; Wood and Salzberg, 2014), it remains largely unexplored in the area of locus-based sequence typing. We are currently in the process of coupling this paradigm with efficient data structures to make the successor of stringMLST more efficient and accurate in dealing with larger typing schemes.

References

Andrews, S. FastQC: A quality control tool for high throughput sequence data. In.; 2010.
Bankevich, A*., et al.* SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J Comput Biol* 2012;19(5):455-477.
Bratcher, H.B*., et al.* A gene-by-gene population genomics platform: de novo assembly, annotation and genealogical analysis of 108 representative Neisseria meningitidis genomes. *BMC Genomics* 2014;15:1138.
Camacho, C*., et al.* BLAST+: architecture and applications. *BMC Bioinformatics* 2009;10:421.
Jolley, K.A. and Maiden, M.C. BIGSdb: Scalable analysis of bacterial genome variation at the population level. *BMC Bioinformatics* 2010;11:595.
Kumar, S., Stecher, G. and Tamura, K. MEGA7: Molecular Evolutionary Genetics Analysis Version 7.0 for Bigger Datasets. *Mol Biol Evol* 2016;33(7):1870-1874.
Kurtz, S*., et al.* Versatile and open software for comparing large genomes. *Genome Biol* 2004;5(2):R12.
Ondov, B.D*., et al.* Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biol* 2016;17(1):132.
Richter, M. and Rossello-Mora, R. Shifting the genomic gold standard for the prokaryotic species definition. *Proc Natl Acad Sci U S A* 2009;106(45):19126-19131.
Saitou, N. and Nei, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evol* 1987;4(4):406-425.
Wood, D.E. and Salzberg, S.L. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biol* 2014;15(3):R46.

Supplementary Tables

Supplementary Table S1. **Samples used for testing the accuracy and runtime performance of MLST detection tools.**

Dataset	Organism	# Samples[1]	# STs[2]	Avg Read Len[3]	Avg Seq Depth[4]	Avg k-mer[5]	Platform[6]
Comparative	*Neisseria meningitidis*	10	9	78 [54-151]	221 [70 - 1059]	42,912	GAII/ HiSeq
	Streptococcus pneumoniae	10	8	73 [55-76]	380 [142 - 1317]	57,036	GAII/ HiSeq
	Campylobacter jejuni	10	8	75 [75]	125 [115 - 157]	28,532	HiSeq
	Chlamydia trachomatis	10	6	45 [38-55]	623 [304 - 1594]	27,025	GAII
Accuracy	*Neisseria meningitidis*	1002	393	95 [54-151]	200 [5 -1,278]	38,019	GAII/HiSeq

[1]Number of samples from each species tested in comparative and accuracy tests
[2]Number of unique STs the samples belonged to
[3]Average read length (in basepairs) of the sequencing sample [range]
[4]Average sequencing depth of the samples [range]
[5]Average number of k-mers supporting each locus
[6]Sequencing platforms utilized for the generation of data, GAII = Illumina GA II and HiSeq = Illumina HiSeq 2000/2500

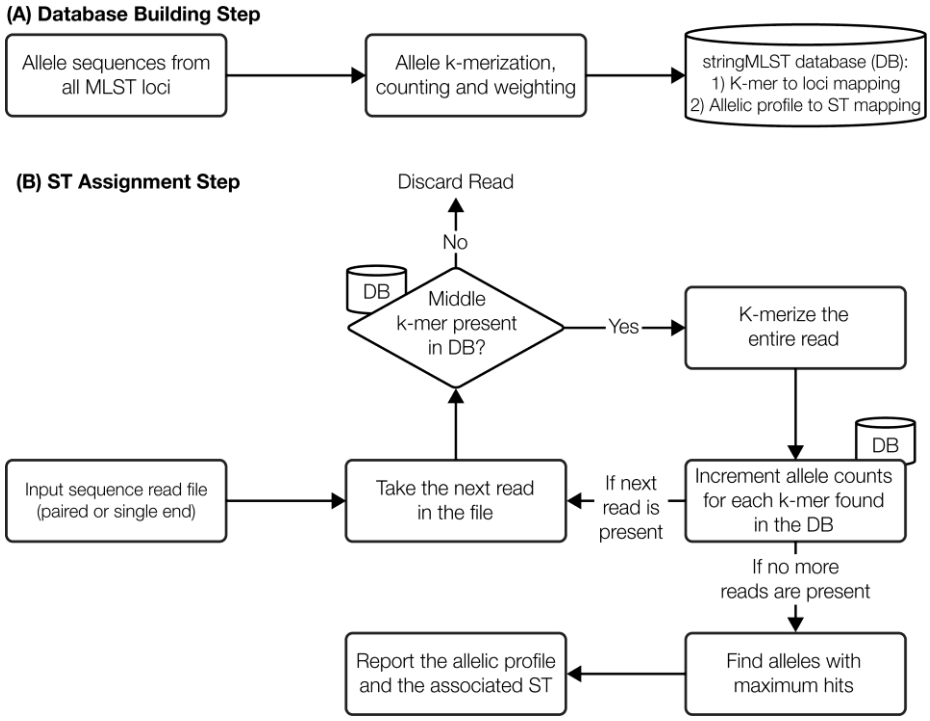
Supplementary Table S2. **Performance evaluation for stringMLST on the large-scale *N. meningitidis* dataset.** The performance is measured in terms of the accuracy and runtime of the tool.

Accuracy parameters								
	Desktop	K=15	K=21	K=31	K=35	K=45	K=55	K=66
Total samples tested	6	1,002	1,002	1,002	1,002	1,002	964	964
Samples (STs) correctly predicted	6	991	991	991	991	991	956	956
Mis-annotated samples (correct prediction)	0	9	9	9	9	9	8	8
Samples predicted incorrectly	0	2	2	2	2	2	0	0
Percent incorrect	0.00	0.20	0.20	0.20	0.20	0.20	0.00	0.00
Allele-wise breakdown								
Total alleles tested	42	7,014	7,014	7,014	7,014	7,014	6,748	6,748
Alleles predicted correctly	42	7,012	7,012	7,012	7,012	7,012	6,748	6,748
Percent incorrect	0.00	0.03	0.03	0.03	0.03	0.03	0.00	0.00
Runtime parameters								
Average runtime (in seconds)	23.8	78.2	70.4	45.2	40.7	32.6	28.3	24.5
Average sample size (in GB)	0.51	0.89	0.89	0.89	0.89	0.89	0.89	0.89
Maximum memory used (in GB)	1.8	0.52	0.57	0.64	0.67	0.73	0.78	0.88

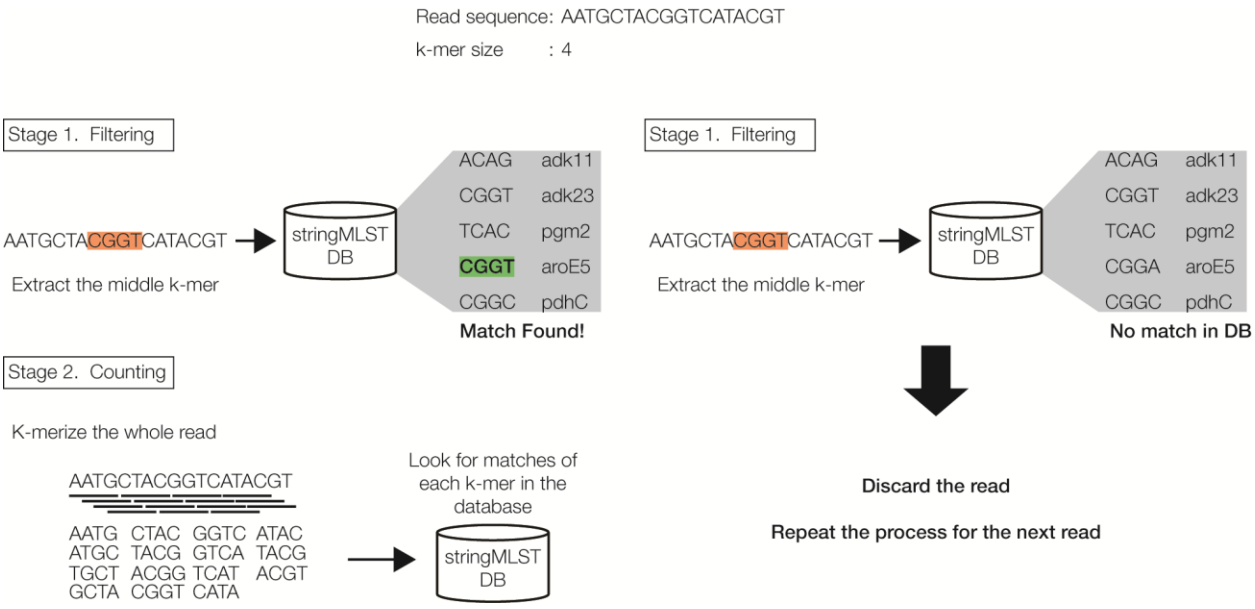
Supplementary Table S3. **List of isolates with predicted ST different from ST reported on PubMLST.** A total of 11 isolate ST predictions differed from the ST reported on PubMLST. Isolates are divided into three categories based on the result of predicted ST and ST reported on PubMLST. stringMLST was able to correctly assign nine of these isolates, one isolate was mis-annotated and was predicted incorrectly and one isolate was predicted incorrectly but was reported correctly.

Isolate	ST predicted	ST on pubMLST	Isolate	ST predicted	ST on pubMLST
Misannotated samples with correct prediction			*Misannotated samples with correct prediction*		
ERR026522	0	10253	ERR310540	11	5757
ERR033097	9330	9332	ERR957622	154	6697
ERR036115	11	672	*Misannotated samples with incorrect prediction*		
ERR133727	106	1087	ERR033104	0	3706
ERR133738	116	106	*Incorrect prediction*		
ERR133744	10307	989	ERR170870	0	9893
ERR137178	144	102			

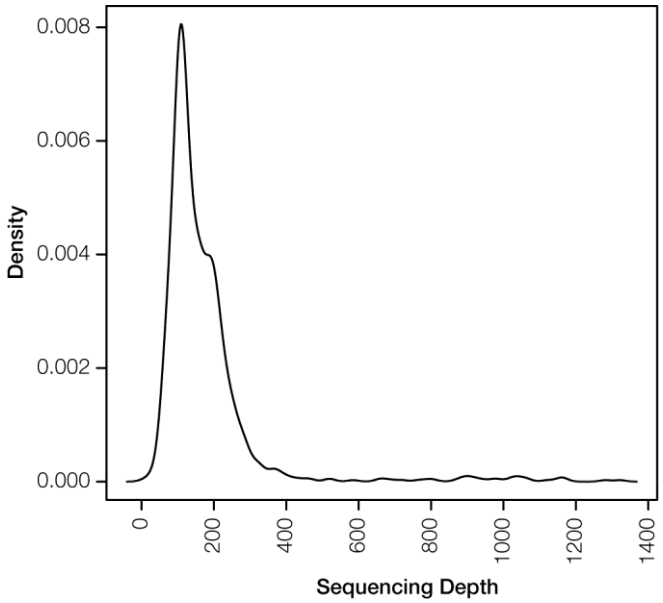
Supplementary Figures



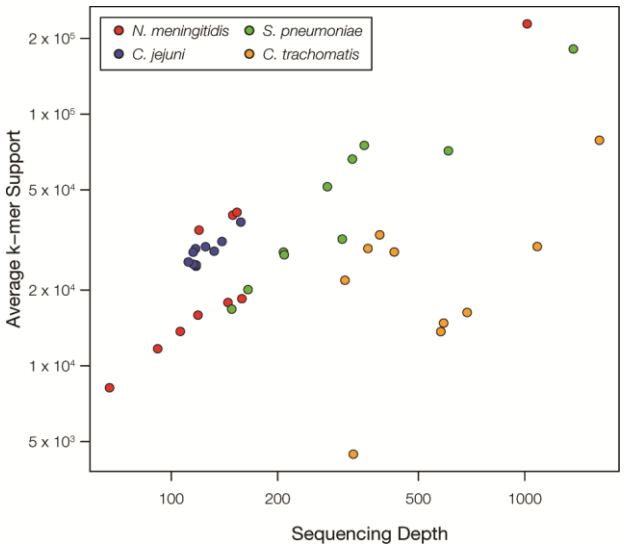
Supplementary Figure S1. **Conceptual workflow of stringMLST's (A) database building routine and (B) ST discovery routine.**



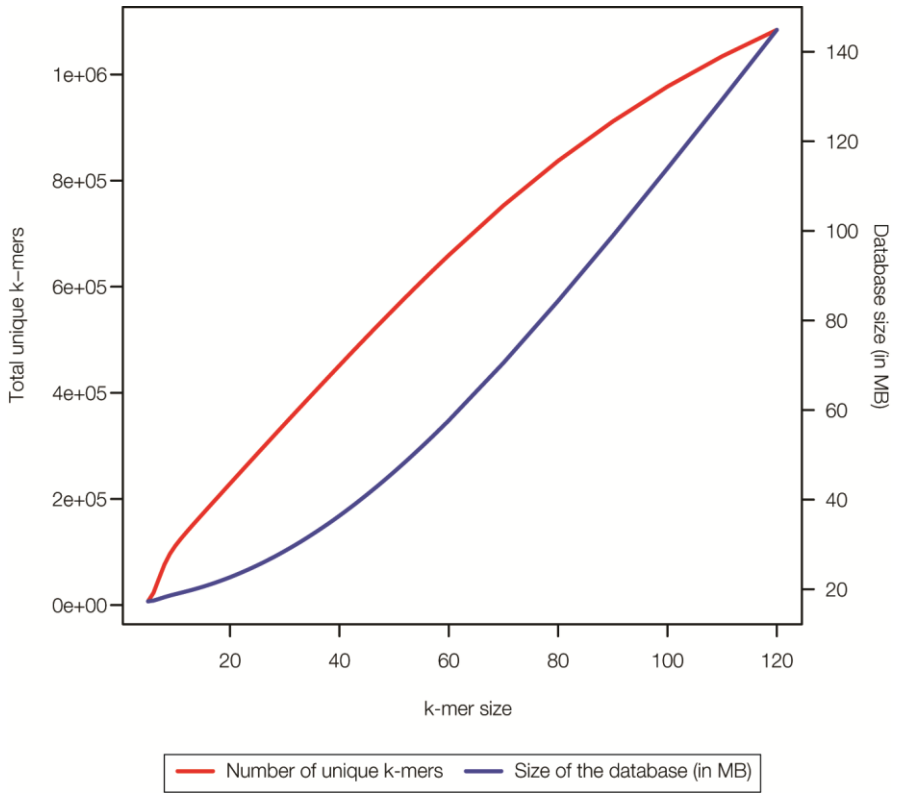
Supplementary Figure S2. **Illustration of stage 1 and 2 of stringMLST's prediction step.** As an example, consider the above shown read sequence and k-mer size of 4. The middle k-mer "CGGT" is searched in the database as part of filtering stage. In the event a match is found in the database (left panel), the read sequence is further k-merized and each resulting k-mer is searched in the database for matches and a counter tracks their prevalence in stage 3. Otherwise (right panel), the read sequence is discarded and the algorithm proceeds to the next available read sequence. This process is repeated till all the available read sequences has been exhausted. Use of the middle k-mer allows for reads to be discarded in a single matching step, thereby providing efficiency and speed to the algorithm.



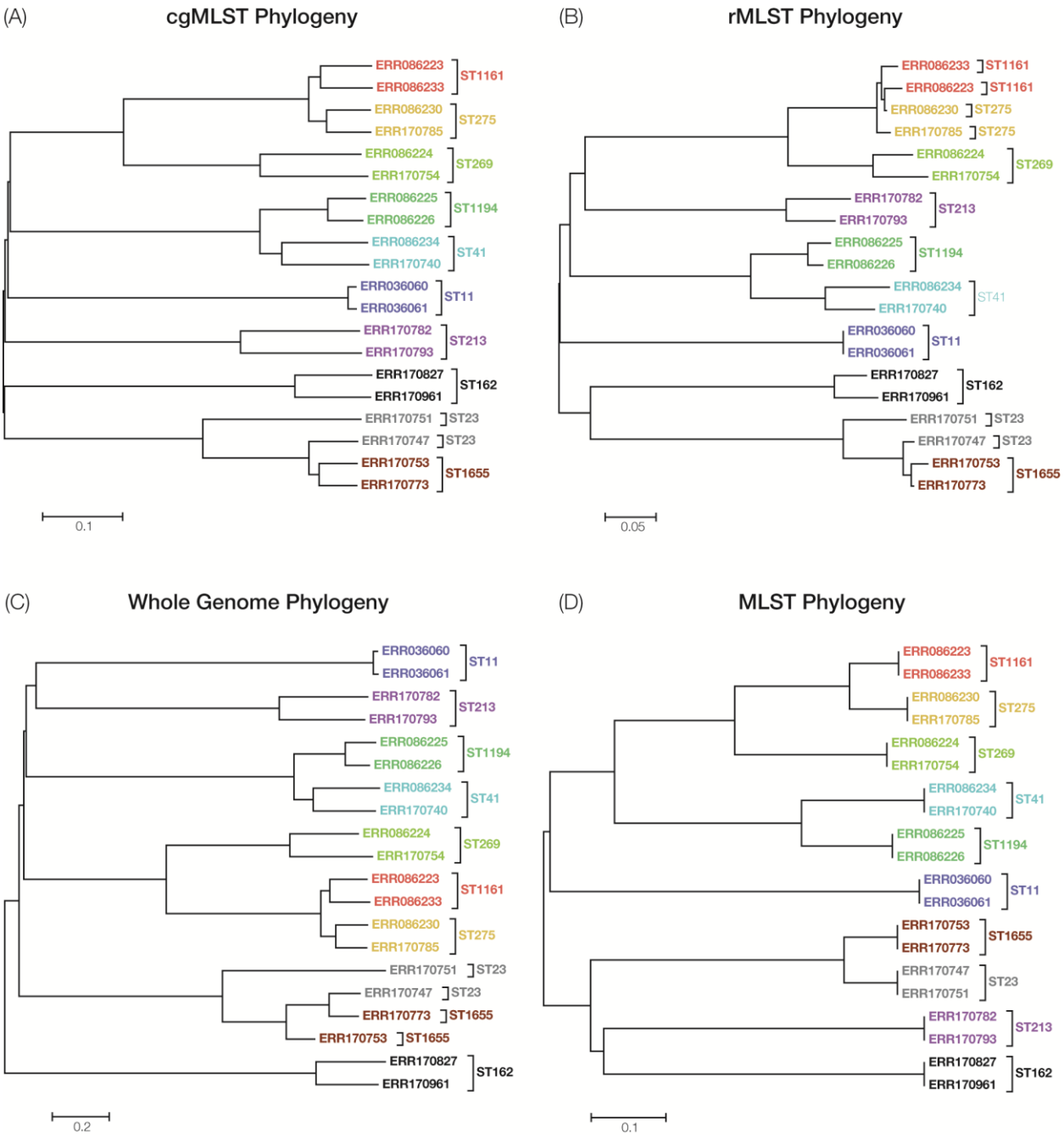
Supplementary Figure S3. **Sequencing depth distribution of the 1,002 *Neisseria meningitidis* samples tested by stringMLST.**



Supplementary Figure S4. **Relationship of sequencing depth and average number of per locus k-mers supporting the best allele for the four species tested here.** As can be expected, for each organism, as the sequence depth increases, the number of average k-mer support also increases. The spread in the correlation can be attributed to the different sequence read length for each isolate, random placement of sequence reads and random sequencing errors.



Supplementary Figure S5. **Increase in the size of the database with increasing k value.** Total number of unique k-mers in the resulting database increases in a near logarithmic fashion (red) whereas the database file size (in MB) increase in an exponential manner (blue) as the value of k is increased from k=5 to k=120.



Supplementary Figure S6. **Phylogenies constructed based on the stringMLST's allelic predictions compared to the whole genome phylogeny.** (A) cgMLST and (B) rMLST prediction results from stringMLST for the 20 *N. meningitidis* isolates were used to compute pairwise allelic distances and a Neighbor-Joining phylogeny was reconstructed for these isolates. The ST for each isolate, based on the traditional seven loci MLST scheme, is also denoted in the tree. Neighbor-Joining based phylogeny was also constructed for (C) whole genome comparisons using Average Nucleotide Identity (ANI) and (D) MLST allele profile. Predictions from stringMLST generates trees that are consistent with the sequence types of each isolate as well as to the ANI-based whole genome phylogeny and MLST based phylogeny.