HairSplitter: separating strains in metagenome assemblies with long reads

Roland Faure, Jean-François Flot, Dominique Lavenier

HairSplitter: separating strains in metagenome assemblies with long reads

Roland Faure[1,2], Jean-François Flot[2] and Dominique Lavenier[1]
[1] Univ. Rennes, INRIA RBA, CNRS UMR 6074, Rennes, France
[2] Service Evolution Biologique et Ecologie, Université libre de Bruxelles (ULB), 1050 Brussels, Belgium

Corresponding author: roland.faure@irisa.fr

Abstract *Long read assemblers struggle to distinguish closely related strains of the same species and collapse them into a single sequence. This is very limiting when analysing a metagenome, as different strains can have important functional differences. We present the first version of a new software called HairSplitter, which recovers the strains from a strain-oblivious assembly and long reads. The originality of the method lies in a custom variant calling step that allows HairSplitter to work with erroneous reads and to separate an unknown number of haplotypes. On simulated datasets, we show that HairSplitter significantly outperforms the state of the art when dealing with metagenomes containing many strains of the same species.*

Keywords Metagenomics, Haplotyping, Genome assembly, Strain separation

1 Introduction

A powerful tool for understanding complex microbial communities is de novo metagenome assembly. Current methods can reconstruct the genomes of sufficiently abundant species, but struggle to differentiate strains within a species, even if they are abundant. While strains of the same species are very similar at the genomic level, the small differences can lead to very significant phenotypic and functional changes. The most famous example of such intra-specific diversity is probably *Escherichia coli* [1], some strains of which can be highly pathogenic while sharing an average nucleotide identity of more than 98.5% with commensal strains [2].

Assemblers are designed to correct for sequencing errors by ignoring bases that occur at low frequencies. As a side effect, they generally discard haplotype differences and collapse the different haplotypes into a single sequence. An additional difficulty in the metagenomic context is that the number of haplotypes is a priori unknown and that haplotypes generally have different frequencies in a sample.

Specific software has been developed to overcome these difficulties. For example, two such software based on short reads are STRONG [3] and strainXpress [4]. However, more and more samples are sequenced using only long reads, as their cost has dropped recently and they allow for more contiguous assemblies.

Using error-prone long reads, assemblers such as metaFlye [5] or Canu [6] attempt to assemble strains separately. However, the authors of [7] showed that these assemblers still struggle to recover multiple strains and proposed a new pipeline, called *Strainberry*, which takes an assembly and the long reads as input and recovers the collapsed strains. Strainberry improves significantly the assembly of samples containing 2 or 3 strains of the same species, but is limited when the number of species increases.

We present *HairSplitter*, a new pipeline for recovering the strains lost when assembling exclusively from (error-prone) long reads. HairSplitter does not make any assumption on the number of strains that should be found in the metagenome. It contains an original procedure that combines a custom variant calling method with a new phasing algorithm. We have extensively tested HairSplitter on simulated Nanopore data, replicating the protocol proposed in [7], and show that HairSplitter significantly improves the completeness of assemblies of metagenomes composed of many strains compared to the state of the art. HairSplitter still needs to be tested on real datasets.

2 Description of the pipeline

The HairSplitter pipeline is composed of four main steps: 1) alignment of the reads on the contigs, 2) separation of the reads in their haplotype of origin, 3) generation of the new contigs and 4) strain-aware contig scaffolding. Steps 2 is done by an original software, while step 1 is performed by minimap2 [8], step 3 by Racon [9] and step 4 by GraphUnzip [10]. The pipeline is illustrated Figure 2

Step 1: Aligning the reads on the contigs

HairSplitter starts by generating base-to-base alignments of the sequencing reads on the assembly with minimap2. For each contig, a multiple sequence alignment is generated using the alignment of the reads to the reference. This is the simplest way to build a multiple sequence alignment, but it creates alignment artifacts, especially at positions where the contig contains errors.

Step 2: Splitting a group of reads in one or more haplotypes

The originality of HairSplitter lies in the second and most crucial step, where reads that align on a contig are separated by haplotype of origin.

This operation is performed locally on window of the contig of size w. The phasing is performed locally to avoid clustering reads that do not overlap. Indeed, clustering together reads that do not overlap could lead to the HairSplitter algorithm clustering very different reads together, as shown in Figure 1. w should thus be chosen to be significantly shorter than the reads.

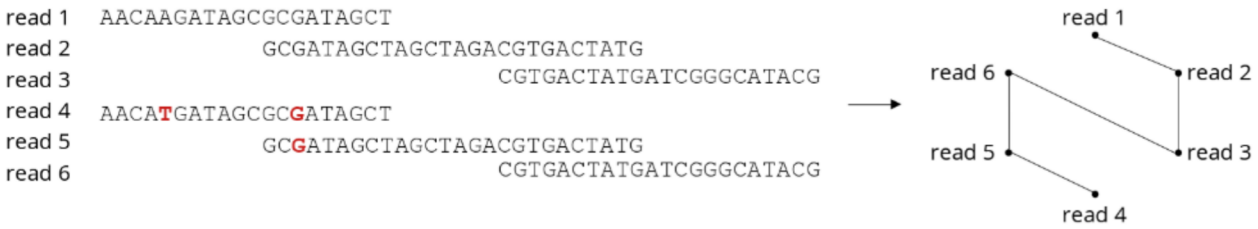

Fig. 1. A read graph is built as follows: each read is a vertex ; reads are connected to the reads they overlaps with 100% identity. Even though read1 and read4 are very different, they are transitively linked through read3 and read6.

A difficulty of the process comes from the error rate of the reads, which can be much higher than the divergence between the haplotypes. Another difficulty is the possibly high number of haplotypes which can be unevenly covered by the sequencing. As this step represents the core of HairSplitter, it is described in detail in section 3.

Step 3: Generating new contigs

The reads on a given window are separated into n groups, the contig sequence in the window is polished n times by Racon using the different groups, yielding n different versions of the window, which we call *subcontigs*. The subcontigs are laid out as an assembly graph, based on the original assembly graph.

Step 4: Strain-aware scaffolding

Due to local homozygosity, some subcontigs will contain multiple haplotypes. These subcontigs limit the contiguity of the graph and must be duplicated to be present once for each haplotype. To do this, the paths of the sequencing reads on the subcontig graph are inventoried. Once all the paths are inventoried, GraphUnzip [10] untangles the graph. From the paths of the read in the graph, it deduces which contig to duplicate and which contig to link to improve the contiguity and completeness of the assembly.

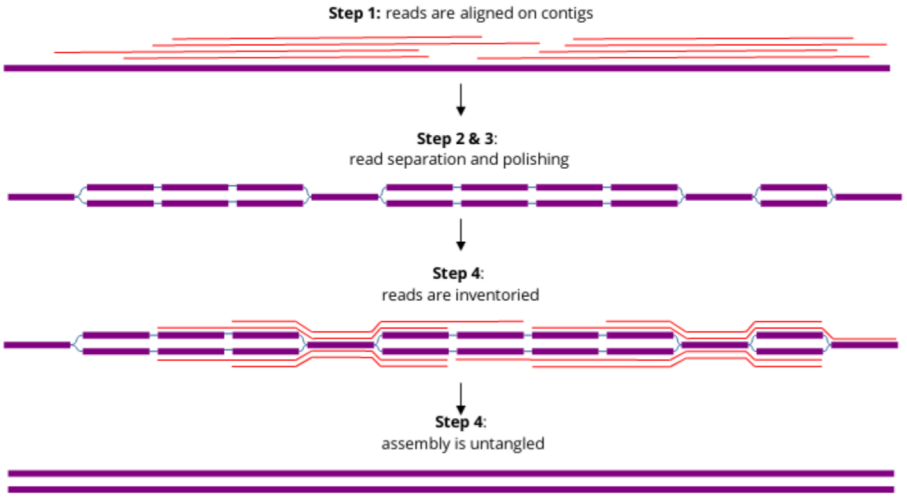

Fig. 2. The HairSplitter pipeline. The purple rectangles represent contigs or subcontigs. The red lines are reads used to build the subcontigs. Only a very small subset of reads is shown here to keep the visualisation readable.

3 Splitting a group of reads in one or more haplotypes

3.1 Detecting variants

To separate the set of reads that align on a contig in several haplotypes, rudimentary variant calling is performed. In this context, we define variants as positions where the different haplotypes are not identical. Once the variants are clearly identified, the reads can be split into the different haplotypes based on these positions, as can be seen in Figure 3. However, due to errors in the reads and in the reference, differences between read and reference do not systematically highlight a variant, as can be seen in Figure 4.

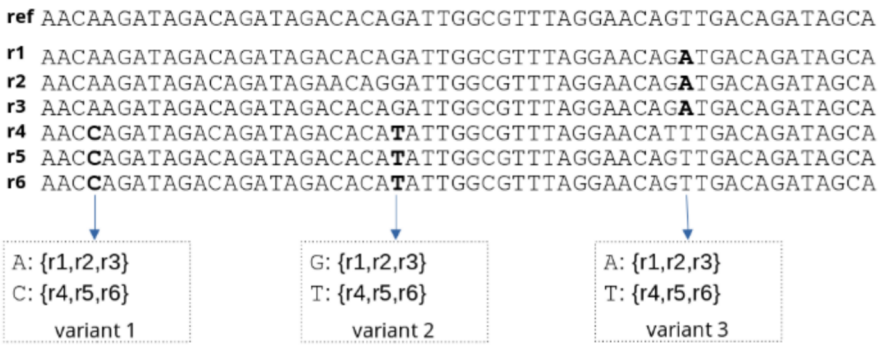

Fig. 3. In this error-free alignment, the variants clearly separate the reads in two haplotypes, one containing r1, r2 and r3 and the other containing r4, r5 and r6.

3.2 Dealing with errors

To distinguish variants from errors, all positions of the alignment are iteratively inspected. For each position, the program inventories the triplet of bases centered there in the reads. A base triplet is defined as the base at the given position flanked by the nucleotide on the left and the nucleotide on the right in the given read. At conserved regions, which make up the majority of the contig, the reference triplet will be the most common, while some residual triplets may be due to sequencing errors or alignment artifacts. At positions of true genomic variants, two triplets (corresponding to the two variants) will be common, with some residual triplets due to sequencing errors or alignment artifacts. *Suspect positions* are defined as positions where the second most abundant triplet is significantly (by a factor s) more abundant than the third most abundant triplet. All variant should generate at least one suspicious positions, therefore only suspect positions are considered for phasing. This will discard most positions where there are only a few random errors. Some positions where there are no true

variants will also be flagged as suspicious, typically positions with alignment artifacts. The selection of suspicious position is illustrated Figure 4.

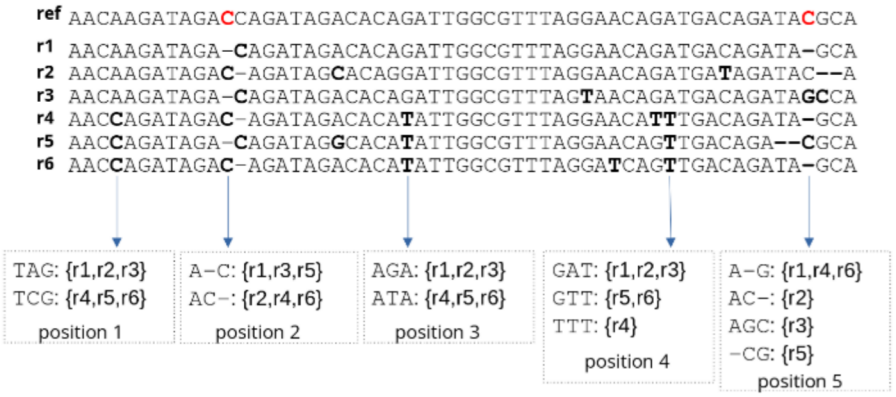

Fig. 4. Selection of suspicious positions. The red bases in the reference represent errors (unknown to HairSplitter). Note that position 2 will be flagged as suspicious because of an alignment artifact. Position 5 will not be considered suspicious because even though there are many different triplets at this position, there is no clear alternative to the triplet 'A-G'.

To distinguish variants from alignment artifacts, HairSplitter implements a method based on the intuition that alignment artifacts are generally randomly distributed on the reads, while genomic variants are not. On the one hand, reads carrying sequencing and alignment errors are not correlated between two positions. On the other hand, reads carrying variants should be strongly correlated between two variants.

Suspicious positions are clustered hierarchically. A cluster is represented by its consensus. Two clusters are merged if more than 90% of each part of each consensus consists of reads from a part of the other consensus. At the end of the process, consensuses consisting of more than p positions are considered solid. We call them the *solid bipartitions* of the reads.

For each suspect position, its correlation with all solid bipartitions is computed by a one-degree-of-freedom chi-square test of independence. If the result is greater than five (strong correlation), the position is marked as *interesting*. The positions that do not correlate well with any solid bipartitions are considered untrustworthy and are discarded. The set of interesting positions corresponds to the set of positions that will be considered as a bi-allelic variants by the algorithm.

This variant calling procedure is quite conservative and may miss existing variants. This is not a problem, as the goal of this step is to confidently identify a set of variants, not to call all variants exhaustively.

3.3 Phasing the reads using the variants

Once a set of variants has been largely extracted from the noise, reads are split into different haplotypes.

A graph is generated for each window of the contig. The reads that cover the window from end to end are the vertices of the graph. Pairwise distances between the reads are calculated as the number of divergent interesting positions divided by the total number of interesting positions overlapped by both reads. Each read is connected to its k nearest neighbors, as shown Figure 5.

The resulting graph forms clusters, grouping reads that share identical variants, corresponding to haplotypes. Empirically, the best algorithm to cluster this graph without knowing a priori the number of clusters seems to be the Chinese Whispers algorithm [11]. However, a limitation of this algorithm is that it is not deterministic, especially when the number of nodes in the graph is small. With low frequency, clusters will be merged or split. To avoid this, HairSplitter exploits the property that if the Chinese Whispers algorithm is initialized with an approximate solution, it will converge to the solution without splitting or merging clusters. The approximate solutions can be found at the interesting positions: each position contains a variant that separates two groups of reads with some

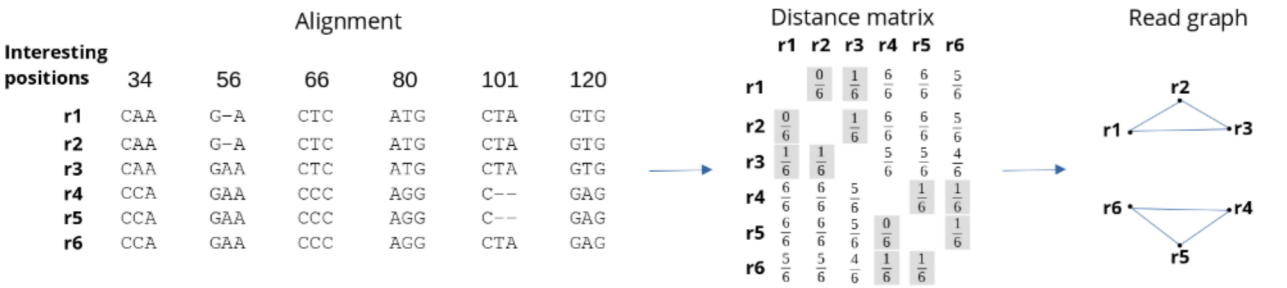

Fig. 5. Generating the read graph from the list of interesting positions with $k = 2$. Light gray squares highlight the k closest neighbors of each read in the distance matrix, which become ones in the adjacency matrix of the read graph.

errors. Running the Chinese whispers algorithm with one position as initialization will cluster the graph into two parts. Each interesting positions will yield a bipartition. All the bipartitions can then be aggregated into a single partition: two reads will be clustered together if and only if they are not separated in any bipartition. This process is illustrated Figure 6.

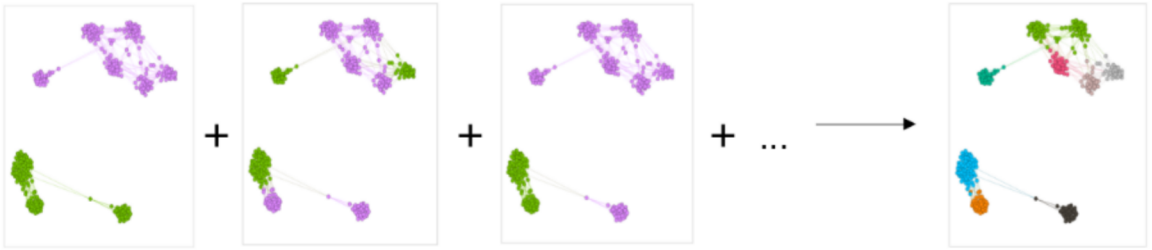

Fig. 6. The read graph is clustered using interesting positions as initialisation, resulting in a series of bipartitions. All these bipartitions can then be aggregated into the final partition.

Each part of the resulting partition corresponds to a haplotype.

4 Results

4.1 Protocol

The protocol is a replica of the protocol proposed in [7].

To systematically test the performance of HairSplitter, we composed a benchmark consisting of different strains of *Escherichia coli*. The reference genomes of the strains were obtained from NCBI. For each strain, we simulated "mediocre" Nanopore sequencing (around 7% error rate), using Badreads [12] with default settings, from the reference genomes.

For each experiment, the sequencing of different strains was concatenated to create a simulation of the sequencing of a metagenomic sample. The reads were then assembled using metaFlye with default settings. Missing strains were then recovered from the assembly using Strainberry and HairSplitter with default settings. For HairSplitter, the parameters are set to $s = 5$, $k = 5$, $p = 5$ and $w = 2000$, but the few tests we ran suggest that the algorithm is not very sensitive to these settings.

4.2 Evaluation metrics

Two metrics were retained to evaluate the quality of the recovered assembly with respect to the known solution genomes.

The first one is the proportion of the 21-mers found in the genomes that are not found in the assembly. This measures how well the different strains are covered. A large number of missing 21-mers indicates that some strains have not been well assembled.

The second metric is the proportion of 21-mers found in the assembly that are found in the genome. This evaluates the accuracy of the assembly. A low number of 21-mers found in the genomes indicates that the assembly contains many errors.

4.3 Influence of strain coverage, divergence and number of strains on strain separation

Divergence A factor that can influence the strain reconstruction is the degree of divergence between the strains. Seven datasets were created, composed of the simulated sequencing of the K12 strain mixed with the simulated sequencing of seven strains having varying degree of divergence with K12. The strains have been chosen to replicate exactly the experiment shown in [7].

Coverage Another crucial factor to reconstruct the strains is the depth at which each strain is covered. To evaluate how this affected the HairSplitter algorithm, tests were carried out on a mixture of the IAI1 and 12009 strains. In a first experiment, the two strains were sequenced at depths ranging from 5x to 50x. In a second experiment, the 12009 strain was sequenced systematically at 50x coverage, while the IAI1 strain was sequenced at depths ranging from 5x to 50x.

Number of strains The authors of Strainberry pointed to the number of strains as a limiting factor in strain reconstruction, with the completeness of the reconstruction decreasing significantly when more than 3 strains were sequenced [7]. Mixtures were made with different numbers of strains. The strains used for the mixtures were 12009, IAI1, F11, S88, Sakai, SE15, *Shigella flexneri*, UMN026, HS and K12. The strains were chosen to cover a wide range of the *Escherichia coli* phylogenetic tree, with some very close strains (such as F11 and S88) and others much more distance strains (such as SE15 and K12).

The results Figure 7 show that HairSplitter and Strainberry recover a very similar amount of k-mers on mixtures of 2 strains. More specifically, both software perform well on pairs of strains with more than 0.3% divergence, as defined by the ANI [13] (Figures 7b), and when the coverage is 20x or more (Figures 7d, 7f). This confirms the results presented in [7]. However, while HairSplitter recovers a similar amount of missing 21-mers compared to Strainberry, it tends to generate much fewer erroneous 21-mers (Figure 7a, 7c and 7e).

The most spectacular result is that even when the mixture contained six or more strains, HairSplitter was able to recover most of the missing 21-mers, producing assemblies with significantly fewer missing 21-mers than metaFlye and even Strainberry assemblies (Figure 7h). For example, in the metaFlye assembly of the mixture of 10 strains, 46% of the 21-mers of the solution were missing. This dropped to 39% when using Strainberry and 16% when using HairSplitter.

4.4 Performance

In all these tests, HairSplitter finished in less than 30 minutes using four threads and less than 5G of RAM. This is similar to Strainberry and small compared to the assembly time, which took at least 5 times longer.

5 Discussion

In this work, we introduced HairSplitter, a new pipeline for performing strain separation on assemblies using only long reads. HairSplitter shows a significant improvement over state-of-the-art methods when the number of strains is high. One of the reasons for this is that, unlike Strainberry, HairSplitter can separate a contig into a variable number of strains. To confirm these results, HairSplitter needs to be benchmarked on real sequencing datasets.

The data we simulated was of low quality to test HairSplitter in the hardest possible case. We expect HairSplitter to perform even better as the quality of the sequencing improves, but this remains to be tested. We also want to see if HairSplitter can improve on the de novo assembly performed by hifiasm in the case of HiFi sequencing.

Another potential application of HairSplitter that deserves investigation is the phasing of polyploid species. Powerful software, such as WhatsHap [14], already exists when the number of haplotypes is known a priori and can be used for polyploid assembly. However, knowing the number of haplotypes in each contig is not necessarily an easy task and using an agnostic approach such as HairSplitter could improve the results.

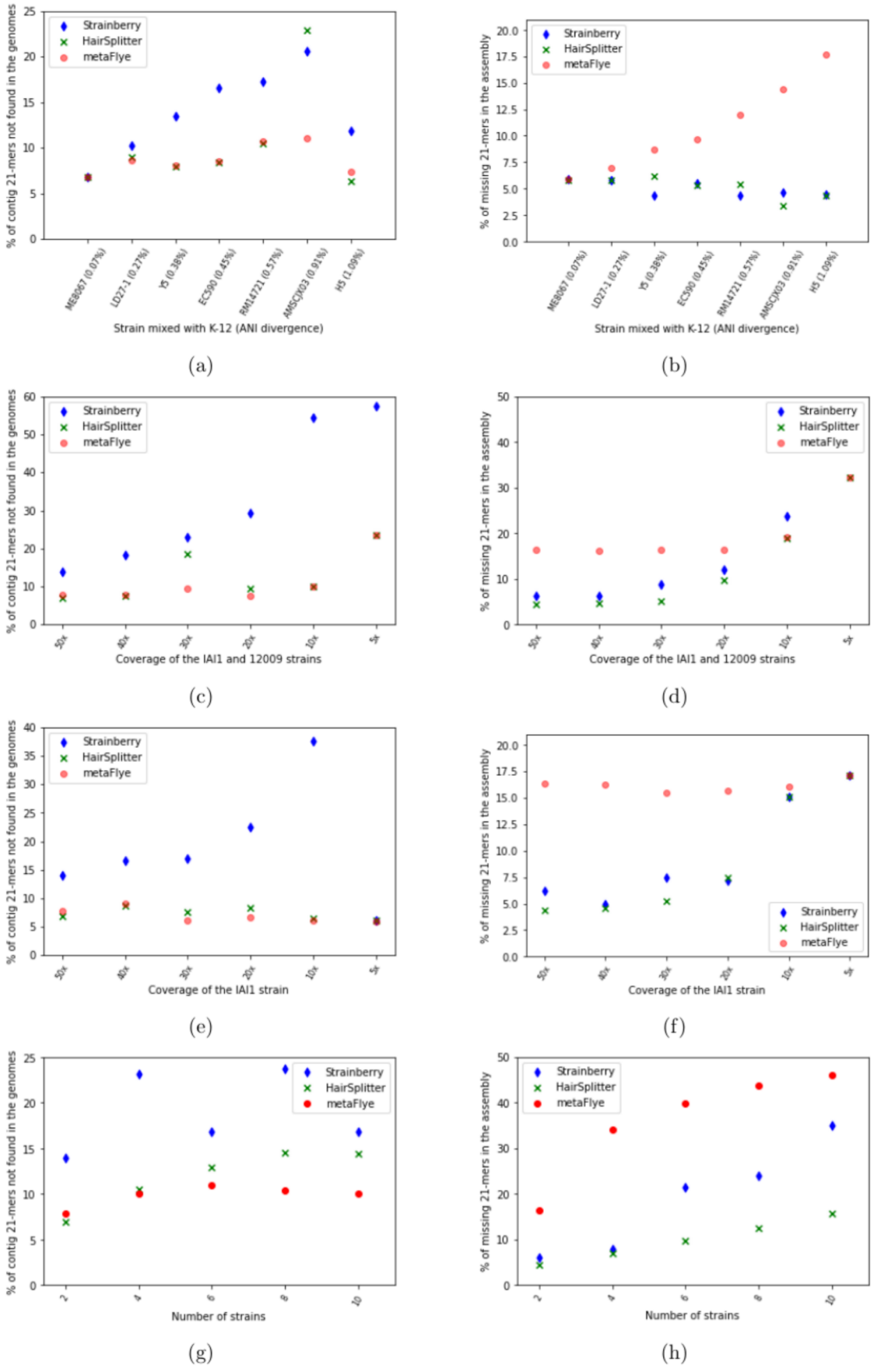

Fig. 7. Evaluation of assemblies obtained using HairSplitter or Strainberry on the metaFlye assembly in different mixes of strains. a and b: mix of K12 strain and another strain at 50x coverage. c and d: mix of the IAI1 and 12009 strains at varying coverage. e and f: mix of the IAI1 and 12009 strains, with 12009 at 50x coverage and IAI1 at varying coverage. g and h: mix of varying number of strains at 50x coverage.

Currently, HairSplitter has two shortcomings that we are trying to fix as a priority. First, it needs at least 20x coverage to distinguish a strain. This is quite high and will undoubtedly be limiting in most real-world applications where rare strains are common. Improving the quality of the data may solve this problem. The second shortcoming is that the contig scaffolding step is still not fully satisfactory and the contiguities of the assemblies obtained is lower than those obtained using Strainberry. This is due to the fact that we are using GraphUnzip slightly outside the use case for which it was designed. We will adapt GraphUnzip to this specific task.

Acknowledgements

We thank the Genouest facility (genouest.org) for providing us the hardware, software and support necessary to run our computations. The software Tablet [15] and Bandage [16] were used to visualize data while developing HairSplitter. For the purpose of Open Access, a CC-BY public copyright licence has been applied by the authors to the present document and will be applied to all subsequent versions up to the Author Accepted Manuscript arising from this submission

References

[1] Olivier Tenaillon, David Skurnik, Bertrand Picard, and Erick Denamur. The population genetics of commensal Escherichia coli. *Nature Reviews Microbiology*, 8(3):207–217, March 2010.

[2] S Hudault. Escherichia coli strains colonising the gastrointestinal tract protect germfree mice against Salmonella typhimurium infection. *Gut*, 49(1):47–55, July 2001.

[3] Christopher Quince, Sergey Nurk, Sebastien Raguideau, Robert James, Orkun S. Soyer, J. Kimberly Summers, Antoine Limasset, A. Murat Eren, Rayan Chikhi, and Aaron E. Darling. Metagenomics Strain Resolution on Assembly Graphs. preprint, Bioinformatics, September 2020.

[4] Xiongbin Kang, Xiao Luo, and Alexander Schönhuth. StrainXpress: strain aware metagenome assembly from short reads. *Nucleic Acids Research*, 50(17):e101–e101, September 2022.

[5] Mikhail Kolmogorov, Derek M. Bickhart, Bahar Behsaz, Alexey Gurevich, Mikhail Rayko, Sung Bong Shin, Kristen Kuhn, Jeffrey Yuan, Evgeny Polevikov, Timothy P. L. Smith, and Pavel A. Pevzner. metaFlye: scalable long-read metagenome assembly using repeat graphs. *Nature Methods*, 17(11):1103–1110, November 2020.

[6] Sergey Koren, Brian P. Walenz, Konstantin Berlin, Jason R. Miller, Nicholas H. Bergman, and Adam M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k -mer weighting and repeat separation. *Genome Research*, 27(5):722–736, May 2017.

[7] Riccardo Vicedomini, Christopher Quince, Aaron E. Darling, and Rayan Chikhi. Strainberry: automated strain separation in low-complexity metagenomes using long reads. *Nature Communications*, 12(1):4485, July 2021.

[8] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, September 2018.

[9] Li Fang and Kai Wang. Polishing high-quality genome assemblies. *Nature Methods*, 19(6):649–650, June 2022.

[10] Roland Faure, Nadège Guiglielmoni, and Jean-François Flot. GraphUnzip: unzipping assembly graphs with long reads and Hi-C. preprint, Bioinformatics, February 2021.

[11] Chris Biemann. Chinese whispers: An efficient graph clustering algorithm and its application to natural language processing problems. *Proceedings of TextGraphs*, pages 73–80, 07 2006.

[12] Ryan Wick. Badread: simulation of error-prone long reads. *Journal of Open Source Software*, 4(36):1316, April 2019.

[13] Johan Goris, Konstantinos T. Konstantinidis, Joel A. Klappenbach, Tom Coenye, Peter Vandamme, and James M. Tiedje. DNA–DNA hybridization values and their relationship to whole-genome sequence similarities. *International Journal of Systematic and Evolutionary Microbiology*, 57(1):81–91, January 2007.

[14] Sven D. Schrinner, Rebecca Serra Mari, Jana Ebler, Mikko Rautiainen, Lancelot Seillier, Julia J. Reimer, Björn Usadel, Tobias Marschall, and Gunnar W. Klau. Haplotype threading: accurate polyploid phasing from long reads. *Genome Biology*, 21(1):252, September 2020.

[15] I. Milne, G. Stephen, M. Bayer, P. J. A. Cock, L. Pritchard, L. Cardle, P. D. Shaw, and D. Marshall. Using Tablet for visual exploration of second-generation sequencing data. *Briefings in Bioinformatics*, 14(2):193–202, March 2013.

[16] Ryan R. Wick, Mark B. Schultz, Justin Zobel, and Kathryn E. Holt. Bandage: interactive visualization of *de novo* genome assemblies. *Bioinformatics*, 31(20):3350–3352, October 2015.