Résumés des exposés de

2020

23 et 24 novembre

SeqBIM

Abstract

UMI-Gen: a UMI-based read simulator for variant calling evaluation

Vincent Sater[1,2,3]*, Pierre-Julien Viailly[2,3], Thierry Lecroq[1], Philippe Ruminy[2,3], Élise Prieur-Gaston[1], Caroline Bérard[2,3] and Fabrice Jardin[2,3]

[1]*Normandie Univ, UNIROUEN, LITIS EA 4108, 76000 Rouen, France*

[2]*Centre Henri Becquerel, 76000 Rouen, France*

[2]*Normandie Univ, UNIROUEN, INSERM U1245, Team "Genomics and Biomarkers of Lymphoma and Solid Tumors", 76000 Rouen, France*

***Corresponding author**: vincent.sater@gmail.com

Abstract

With Next Generation Sequencing becoming more affordable every year, NGS technologies asserted themselves as the fastest and most reliable way to detect Single Nucleotide Variants (SNV) and Copy Number Variations (CNV) in cancer patients. These technologies can be used to sequence DNA at very high depths thus allowing to detect abnormalities in tumor cells with very low frequencies. Multiple variant callers are publicly available and are usually efficient at calling out variants. However, when frequencies begin to drop under 1%, the specificity of these tools suffers greatly as true variants at very low frequencies can be easily confused with sequencing or PCR artifacts. The recent use of Unique Molecular Identifiers (UMI) [1] in NGS experiments has offered a way to accurately separate true variants from artifacts. UMI-based variant callers are slowly replacing raw-read based variant callers as the standard method for an accurate detection of variants at very low frequencies. However, benchmarking done in the tools publication are usually realized on real biological data in which real variants are not known, making it difficult to assess their accuracy. We present UMI-Gen, a UMI-based read simulator for targeted sequencing paired-end data. UMI-Gen generates reference reads covering the targeted regions at a user customizable depth. After that, using a number of control files, it estimates the background error rate at each position and then modifies the generated reads to mimic real biological data. Finally, it will insert real variants in the reads from a list provided by the user.

References

[1] Y. Kukita, R. Matoba, J. Uchida, T. Hamakawa, Y. Doki, F. Imamura, K. Kato, High-fidelity target sequencing of individual molecules identified using barcode sequences: de novo detection and absolute quantitation of mutations in plasma cell-free DNA from cancer patients, DNA Res 22 (2015) 269–277. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4535617/. doi:10.1093/dnares/dsv010.

Extended abstract

SeqBIM

Enabling multiscale variation analysis with genome graphs

Brice Letcher[1]*, Martin Hunt[1] and Zamin Iqbal[1]

[1]*EMBL-EBI, Hinxton, United Kingdom*

***Corresponding author**: bletcher@ebi.ac.uk

Abstract

Standard approaches to characterising genetic variation revolve around mapping reads to a reference genome, but high genetic diversity leads to biases in mapping and variation detection. Genome graphs have been proposed as a means to address this and alleviate mapping bias. However when genotyping genome graphs, we need to define which variant sites are in the graph and what reference to express them against. Notably, with enough samples or in highly diverse genomic regions "nested variation" naturally occurs- a long deletion which is an alternate allele to multiple SNPs, or diverged haplotypes with small variants on top of each. There is currently no tool that models these relationships and meaningfully outputs variation at multiple scales.

We demonstrate our software `gramtools` can accurately genotype dense variation at multiple scales, outperforming reference-based variant callers and state of the art genome graph tools on two datasets of microbial pathogens. Many species and genes of great interest harbour high levels of genetic diversity where multiscale variation naturally occurs and requires consideration. We provide a new output format for accessing all variation in directed acyclic genome graphs allowing straightforward genotyping of sample cohorts, finer resolution of genetic variation and the definition of alternate references.

Keywords

Genome graphs— Variant calling — P. falciparum — M. tuberculosis

1. Introduction

Genome graphs are graph structures extending single, linear reference genomes with known population genetic variation or candidate variants. They are used as objects that remove reference bias [1] and as objects that enable genotyping across samples at the same variant sites [2].

In genome graphs built from enough samples or in highly diverse genomic regions, defining which variant sites are present and what reference to express them against becomes non-trivial. In particular in such graphs variation starts to appear at multiple scales, with two naturally occurring cases. First, when analysing structural variants and small variants together, SNPs can occur under long deletions. Second, in genes with divergent forms or in long insertions, SNPs can occur on top of alternate haplotypes.

There is currently no tool that models these relationships and meaningfully outputs variation at multiple scales. Here we present a framework to identify, call and output all identified variation in directed acyclic genome graphs using the open-

source software `gramtools` (https://github.com/iqbal-lab-org/gramtools). We give applications in two microbial datasets illustrating genotyping performance compared to the state of the art and a new analysis in a previously inaccessible genomic region.

2. Methods

`gramtools` implements a workflow for building, genotyping and augmenting genome graphs. To genotype, we map reads from whole-genome sequencing experiments to a unique data structure developed for `gramtools` [3] and record coverage with awareness of horizontal (genomic repeats) and vertical (allelic repeats) mapping uncertainty.

Genotyping produces three main outputs: a personalised reference genome for the sample, a VCF of called variants expressed against the standard reference genome, and a JSON of calls at each variant site in the graph. The latter includes variant sites which are "nested" in others and sites which occur on different sequence backgrounds or references.

The algorithm for nested genotyping is illustrated in Fig. 1. We refer to each outgoing branch from a parent site as a **haplogroup**, for group of related haplotypes.

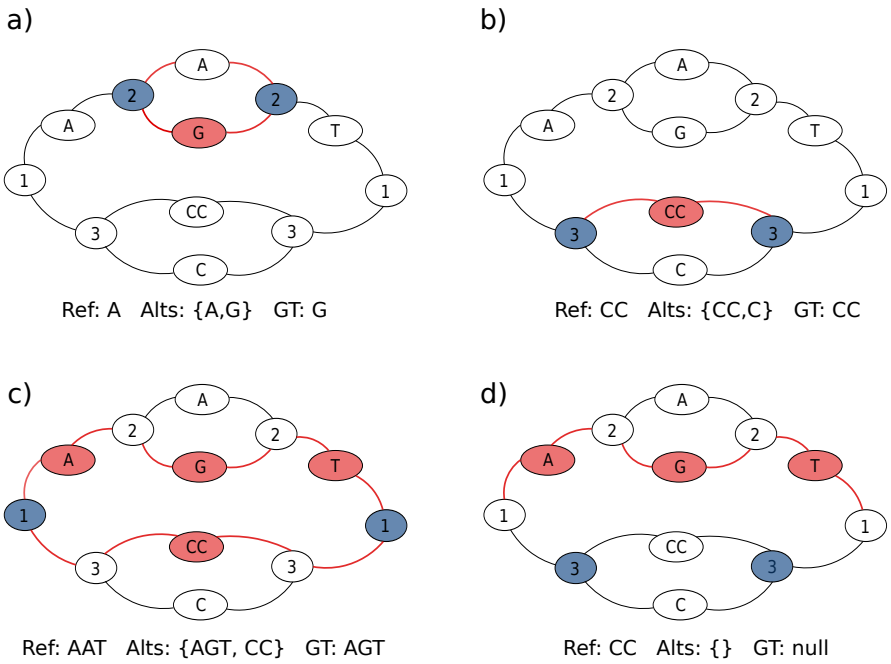


Figure 1. Nested genotyping procedure. Nodes with numbers mark variant sites. In each panel, blue-filled nodes mark which site is being processed, red-filled nodes mark called alleles, and red paths mark alleles considered for genotyping. The example shows haploid genotyping. a. Genotyping of child site 2. b. Genotyping of child site 3. c. Genotyping of parent site 1. d. Invalidation (null calling) of site 3.

3. Results and Discussion

3.1 Multiscale-aware variant call format

We developed a JSON-based output format providing one entry per identified site in a directed acyclic genome graph and storing parent/child relationships between sites. This enables two features. First, it makes incompatibilities between sites explicit allowing genotyping to enforce consistency (Fig.1). Second, it enables defining alternate references based on haplogroups, and which ones variants fall on.

An example is given in Fig. 2. In contrast to VCF, the format also records graph topology allowing queries such as extracting all variant records under a given haplogroup.

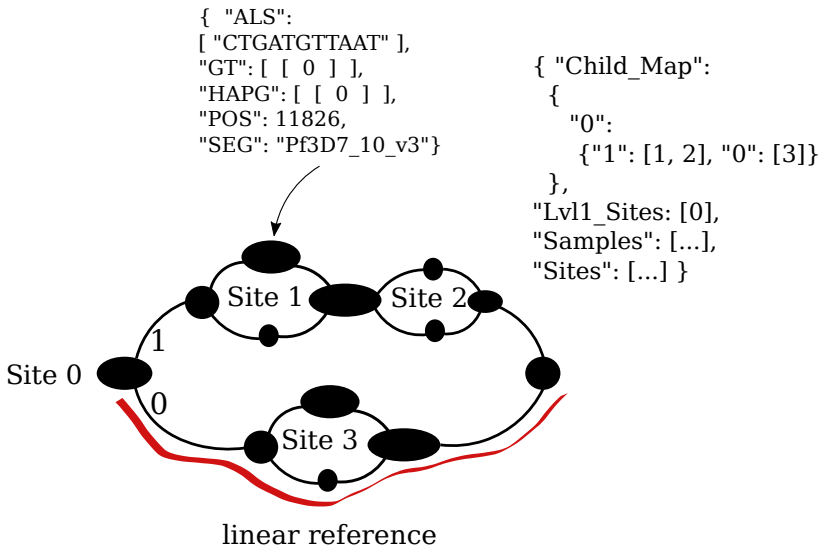


Figure 2. JSON variant call format introduced in `gramtools`. A graph with nested variation is shown; `gramtools` gives each identified site a number ID. Black nodes contain sequence. Haplogroups, groups of related haplotypes in the graph, are labeled on the edges leaving the first node of Site 0. The red path shows the embedded linear reference genome, and Site 1 and 2 occur on a non-reference sequence background. Top-right text shows part of the top-level of the call format. "Child_Map" associates a site ID to sites occurring under it: here we record that site 0 contains Site 1 and Site 2 under haplogroup 1, and Site 3 under haplogroup 0. "Lvl1_Sites" gives site IDs which are not children of any other sites, allowing recursive exploration of the child map. "Sites" is an array indexed by each site ID: each entry is a JSON containing the same information as a VCF line, shown here above Site 1.

3.2 Genotyping performance

3.2.1 Comparison with reference-based variant callers

We performed an experiment on a genome graph of variation from 2,500 samples in four clinically relevant surface antigens of the malaria parasite *P. falciparum*. Using 14 validation samples with long-read assemblies we show `gramtools` genotype calls outperform variant callers `samtools` and `cortex` run against the reference genome alone. We further show the `gramtools` inferred personalised reference genome allows

those tools to discover previously inaccessible variation, and that `gramtools` finds recombinants between input haplotypes in the graph.

3.2.2 Comparison with state of the art genome graph tools

We built graphs containing 45 distinct deletions between 100 and 13,000 bp found in 17 samples and all variation overlapping the deletions in a further 1,000 samples of *M. tuberculosis*. Using long-read assemblies for the 17 samples we show `gramtools` is better able to resolve these regions compared to state of the art tools `vg` [1] and `graphtyper2` [4]. Our nesting-aware genotyping process guarantees mutually exclusively calling deletions and the small variants overlapping them.

3.3 Analysis of variation on top of locally defined references

We genotyped 700 *P. falciparum* samples at the surface antigen DBLMSP2 in which two diverged forms are known to segregate, likely due to balancing selection [5]. We show how `gramtools` recovers the two forms and is able to output variation on top of each diverged form allowing the study of variation on different references.

3.4 Discussion

We have presented a method for identifying, calling and outputting multiscale variation in `gramtools`. Analogous to the recently proposed rGFA format for describing sequences in genome graphs [6], we provide a format for describing variant calls in genome graphs. We believe such formats are required to better study and express variation in genome graphs.

To be useful, genome graphs should support three concepts: compatibility, consistency and interpretability. Compatibility is maintaining support for linear references. Consistency is outputting a fixed set of variants for a given genome graph. Interpretability is providing a simple way of analysing variation at multiple scales or on different references. In `gramtools` we propose a framework and format implementing each of these concepts.

Acknowledgments

The authors thank Rachel Colquhoun for the tool used to construct graphs, Sorina Maciuca for algorithms in `gramtools` and Robyn Ffrancon for software engineering in `gramtools`.

Brice Letcher is a predoctorate fellow at EMBL-EBI, funded by EMBL and registered at the University of Cambridge.

References

[1] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, Benedict Paten, and Richard Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, August 2018.

[2] Jonas Andreas Sibbesen, Lasse Maretty, and Anders Krogh. Accurate genotyping across variant classes and lengths using variant graphs. *Nature Genetics*, 50(7):1054, July 2018.

[3] Sorina Maciuca, Carlos del Ojo Elias, Gil McVean, and Zamin Iqbal. A natural encoding of genetic variation in a Burrows-Wheeler Transform to enable mapping and genome inference. In Springer, editor, *Proceedings of the 16th International Workshop on Algorithms in Bioinformatics, Volume 9838 of Lecture Notes in Computer Science*, pages 222–233, 2016.

[4] Hannes P. Eggertsson, Snaedis Kristmundsdottir, Doruk Beyter, Hakon Jonsson, Astros Skuladottir, Marteinn T. Hardarson, Daniel F. Gudbjartsson, Kari Stefansson, Bjarni V. Halldorsson, and Pall Melsted. GraphTyper2 enables population-scale genotyping of structural variation using pangenome graphs. *Nature Communications*, 10(1):5402, November 2019. Number: 1 Publisher: Nature Publishing Group.

[5] Alfred Amambua-Ngwa, Kevin K. A. Tetteh, Magnus Manske, Natalia Gomez-Escobar, Lindsay B. Stewart, M. Elizabeth Deerhake, Ian H. Cheeseman, Christopher I. Newbold, Anthony A. Holder, Ellen Knuepfer, Omar Janha, Muminatou Jallow, Susana Campino, Bronwyn MacInnis, Dominic P. Kwiatkowski, and David J. Conway. Population Genomic Scan for Candidate Signatures of Balancing Selection to Guide Antigen Characterization in Malaria Parasites. *PLOS Genetics*, 8(11):e1002992, November 2012.

[6] Heng Li, Xiaowen Feng, and Chong Chu. The design and construction of reference pangenome graphs. *arXiv:2003.06079 [q-bio]*, March 2020. arXiv: 2003.06079.

Extended abstract

SeqBIM

Evaluation of open search methods based on theoretical mass spectra comparison

Albane Lysiak[1,3], Guillaume Fertin[1]*, Géraldine Jean[1], Dominique Tessier[2,3]

[1] *Université de Nantes, CNRS, LS2N, F-44000, Nantes, France*
[2] *INRAE, BIBS facility, F-44316, Nantes, France*
[3] *INRAE, UR BIA, F-44316*
***Corresponding author**: guillaume.fertin@ls2n.fr

Abstract

Mass spectrometry remains the privileged method to characterize proteins. Nevertheless, most of the spectra generated by an experiment remain unidentified after their analysis, mostly because of the modifications they carry. Open Modification Search (OMS) methods offer a promising answer to this problem. However, assessing the quality of OMS identifications remains a difficult task.
Aiming at better understanding the relationship between (i) similarity of pairs of spectra provided by OMS methods and (ii) relevance of their corresponding peptide sequences, we used a dataset composed of theoretical spectra only, on which we applied two OMS strategies. We also introduced two appropriately defined measures for evaluating the above mentioned spectra/sequence relevance in this context: one is a color classification representing the level of difficulty to retrieve the peptide sequence that generated the identified spectrum ; the other, called LIPR, is the proportion of common masses, in a given Peptide Spectrum Match (PSM), that correspond to dissimilar sequences. These two measures were also considered in conjunction with the classical False Discovery Rate (FDR). The three above mentioned measures allowed us to clearly determine which of the two studied OMS strategies outperformed the other, both in terms of number and of accuracy of identifications. Even though quality evaluation of PSMs in OMS methods remains challenging, the study of theoretical spectra is a favorable framework for going further in this direction.

1. Introduction

Mass spectrometry in tandem MS mode (MS/MS) is the most powerful method to identify proteins and characterize their modifications on a large scale. However, most of the spectra are left unidentified after their analysis by a dedicated software. This is likely due to the large proportion of spectra generated from proteins carrying modifications [1]. Software usually infer the identification of an experimental spectrum from its similarity to reference spectra. When a peptide carries a modification, its mass is by nature modified, which prevents its identification by conventional methods that compare each experimental spectrum with only a *restricted* set of reference spectra, approximately sharing the same mass in order to avoid excessive runtime. On the other hand, Open Modification Search (OMS) methods compare each experimental spectrum to *all* the reference spectra representing a proteome. Thus, this comparison produces a list of Peptides to Spectrum Matches (PSMs) per experimental

spectrum, and a mass difference $\Delta m \neq 0$ between the experimental spectrum and its associated peptide is assumed to be due to one or several modifications that differentiate them. Many scores exist to evaluate the similarity between two spectra, which all take into account, at a certain level, the number of peaks (i.e., of masses) that are shared by the two spectra, a number called shared peaks count (SPC). Despite the scientific relevance of better spectra identifications, OMS methods are still underused, notably because their reliability remains debated. It is therefore important to better describe the advantages and limitations of these methods. We focused our study on a thorough understanding of two widely spread strategies to determine the best PSM for each experimental spectrum. In the first strategy *Strategy1*, the best PSM is chosen according to a score that does not take Δm into account. The second strategy *Strategy2* tries to improve the alignment – and thus the score – of all the PSMs according to Δm before the choice of the best PSM (see Figure 1). In order to determine the most efficient strategy, a prerequisite was to be able to implement both strategies using the same software, which implies the availability of very efficient spectra comparison and alignment algorithms. The SpecOMS software [2], which we have previously developed, fulfills these conditions.

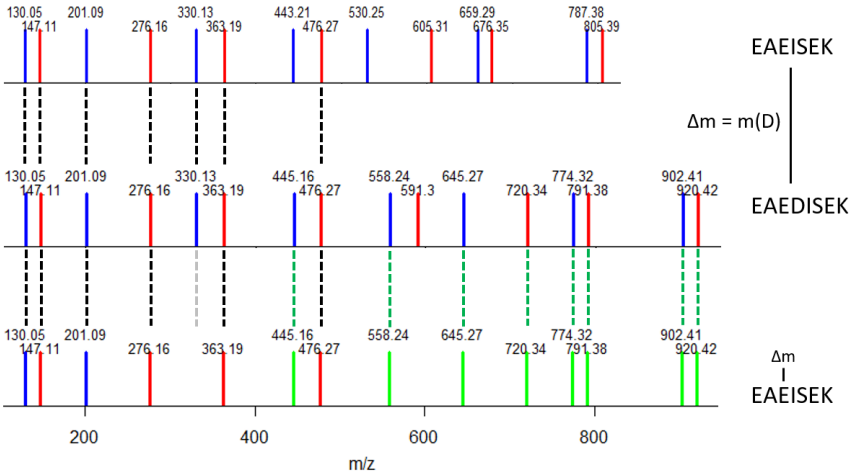


Figure 1. MS/MS spectra matches and their peptide sequences. *b*-ions (containing masses information from the prefix of the peptide) are displayed in blue, *y*-ions (containing masses information from the suffix of the peptide) in red, and matches between spectra in dashed lines. Intensities of all peaks are set to an arbitrary unit value. The middle EAEDISEK MS/MS spectrum shares 7 masses (black dashed lines) with the above native EAEISEK spectrum. After a shift of Δm at position 3 in EAEISEK (below), 8 new masses match with EAEDISEK (shown in green), and one match is removed (grey dashed line). The SPC is then improved from 7 (*raw SPC*) to 14 (*shift SPC*).

To compare in-depth the limits of each strategy, we decided to ground this study using the theoretical spectra derived from the human proteome, considering successively each theoretical spectrum in the role of an experimental spectrum. Doing so, we eliminate the inherent identification difficulties due to the imperfection of experimental spectra (e.g. noise, missing peaks) and concentrate on the benefits of each strategy. Hence, PSMs with $\Delta m \neq 0$ can only be explained by differences in terms of sequence, namely insertions, deletions and/or substitutions of one or several amino acids. Every PSM matching a peptide to itself was considered irrelevant and

consequently forbidden. Many OMS methods estimate the FDR of their results with a target/decoy approach [3]. We also used this approach to compare both strategies, even though it is still unclear whether or not this method underestimates the incorrect identifications [4, 5]. That is why we propose two additional measures of the PSM characteristics to evaluate their quality and compare the strategies.

2. Methods

Peptide identification using SpecOMS We implemented two strategies to find the best PSMs in SpecOMS. Next, we applied these two strategies to compare the set of theoretical spectra generated from the human proteome against themselves. A peptide that plays the role of an experimental spectrum in PSM is called the *bait*, whereas a peptide associated to a bait in a PSM is called a *hit*. Parameters were set in such a way that SpecOMS extracted from its data structure SpecTrees [6] all pairs of spectra of the form (bait,hit) whose SPC is greater than or equal to 7. Depending on the run, the parameter "shift" of SpecOMS was set to `false` (*Strategy1*) or `true` (*Strategy2*) (see Figure 2). More precisely, in *Strategy1*, for each bait, SpecOMS selects the best PSM based on the highest SPC, a score that we call *raw SPC*. In *Strategy2*, the best PSM for a given bait b is selected after the following two-step procedure is applied: first, for every candidate hit h for b such that $\Delta m \neq 0$, SpecOMS realigns h to b by shifting its masses (by Δm) at each possible relevant location in the spectrum, and retains the shift location in h that yields the best newly computed SPC. Second, SpecOMS chooses the best PSM among the candidate PSMs for b, based on the newly computed SPC, that we call *shift SPC*.

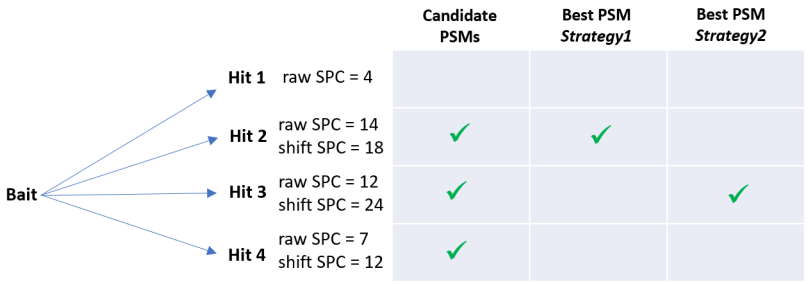


Figure 2. Determining the best PSM in each strategy. Suppose, fictitiously, that a given bait is to be compared to 4 peptides (called hits). Hit 1 is discarded as its *raw SPC* with bait is below the imposed threshold of 7. Hits 2, 3 and 4 are candidate PSMs for bait. Since $\Delta m \neq 0$ for Hits 2, 3 and 4, a shift may be applied, and in that case *shift SPC* is obtained (with *shift SPC* $\geq$ *raw SPC* by definition). In *Strategy1*, the best PSM for bait is Hit 2, as it is based on *raw SPC*. In *Strategy2*, the best PSM for bait is Hit 3, as it is based on *shift SPC*.

Data and theoretical spectra generation The human proteome was downloaded from Ensembl 99, release GrCh38. Proteins predicted with the annotation "protein coding" were added to 116 contaminant proteins downloaded from the cRAP contaminant database. The resulting set of proteins is referred to as the *target* database, *in silico* digested by trypsin. Each peptide is fragmented *in silico* by SpecOMS, so as to transform it into a *theoretical spectrum*. For this, ions from the b and y series are generated. For a given peptide, the set of generated masses represents its theoretical spectrum.

Measuring the quality of PSMs

False Discovery Rate (FDR). The first classical measure that we used is the number of PSMs we can validate at a given False Discovery Rate (FDR). We calculated the FDR as the proportion of best PSMs of the form (bait,hit) for which the hit is a decoy, over the total number of best PSMs. In this work, we were essentially interested by PSMs for which the FDR is less than 1%.

Color classification. Another parameter which we consider as informative for validating MS/MS results, notably in this context, is our ability to explain a PSM of the form (bait,hit) obtained by a given strategy ; by "explain", we mean unambiguously determine the transformation (in terms of amino acid sequence) that is required to retrieve the bait starting from the hit. Thus, the question we ask ourselves is the following: given a PSM (bait,hit) together with Δm, *shift SPC* and its corresponding best shift location, how difficult is it to precisely explain bait from hit ? For this, we introduce here a classification of PSMs into three colors (Green, Orange or Red), depending on this level of difficulty, from the easiest (Green) to the hardest (Red). In a nutshell, Green means that we are able to explain the link between hit and bait unambiguously, Orange contains some level of ambiguity, and Red means that further information and/or computational efforts are necessary to explain the relationship between bait and hit. See Figure 3 for examples of the color classification.

Bait	*Hit*	*Δm* (Da)	Shift location	Interpretation : from the hit to the bait
TSSDSISR	TSDSISR	87.0320	3	TSDSISR + S = TSSDSISR
YQEFQNR	EPPNPEYQEFQNR	-663.2864	6	EPPNPEYQEFQNR – EPPNPE = YQEFQNR
ETVHIPGAR	ETIPGAR	236.1273	2	ETIPGAR + Δm = ET-Δm-IPGAR
VISPEDGK	VIESPDGK	0	/	/
VCASIAQK	VTIQCQK	0	/	/
WFSIYDQR	FWSIQDYFR	-147.0684	/	/

Figure 3. Illustration of the Green/Orange/Red classification of PSMs. The first two rows present PSMs with a bait unambiguously deductible from the information given by the PSM, thus classified as Green. In the first example, Δm corresponds to the mass of S, which can thus be added in the hit at the given location to retrieve the bait. In the second example, the absolute value of Δm corresponds to the mass of EPPNPE, which can be deleted from the hit to retrieve the bait. In the third row, Δm can correspond to two possible amino acid sequences (VH or HV). Such PSM is thus classified as Orange. In the last three rows, transforming hit into bait is too ambiguous, although sequences may be close (e.g., first red row). In all cases, such PSMs are classified as Red.

Low Information Peaks Rate (LIPR). In an MS/MS experiment, spectra are considered similar to each other if they share a high number of masses. Ions from the same series (i.e., y-ions or b-ions in our case), which represent the same fragment, necessarily possess the same mass. Consequently, common masses represent relevant information concerning sequence similarity. However, the converse is not always true : identical masses may not represent identical sequences, for example when amino acids are permuted (e.g., AEAE and EEAA have the same mass) or in more complex situations when combinations of different amino acids turn out to have

the same total mass (e.g., KE and GVT have the same mass). Following the above discussion, we introduce here a new measure, that we call Low Information Peaks Rate (or LIPR). For a given PSM (bait,hit), LIPR(bait,hit) is the percentage of common masses between bait and hit that *do not* correspond to identical sequences. A LIPR close to 0 implies that the two amino acid sequences of bait and hit are very similar. In that case, one can argue that the PSM at hand is relevant, and that retrieving bait from hit may be feasible. On the other hand, when LIPR is close to 100, both sequences, although sharing a non negligible number of masses, represent very dissimilar sequences, and the PSM can thus be considered as debatable.

3. Results and Discussion

We successively implemented *Strategy1* and *Strategy2* to compare all the theoretical spectra generated from the human proteome (572 063 spectra) against a database merging the target and decoy human proteins (1 148 608 spectra). About 80% of the 572 063 tryptic peptides from the human proteome share at least 7 peaks with any other peptide, and about 23% of them share at least 10 peaks (target or decoy). Respectively to an FDR less than 1%, *Strategy1* validates 17 160 PSMs with a minimal *raw SPC* of 17 (i.e. considering best PSMs for which *raw SPC* $\geq$ 17), while *Strategy2* validates 57 784 PSMs with a minimal *shift SPC* of 21. *Strategy2* recruits more than three times more PSMs than *Strategy1*, thus we can conclude that *Strategy2* behaves better than *Strategy1* according to the number of validated PSMs. But one may wonder to what extent the information given by these PSMs is enough to restore the correct amino acid sequence of the baits.

We could get the distributions of the sets PSM_1 and PSM_2 obtained respectively by *Strategy1* and *Strategy2* among the 3 color classes, as well as the evaluation of the LIPR feature. Both strategies behave in a similar fashion, but at less than 1% FDR, *Strategy2* validates roughly three times more Green PSMs than *Strategy1* (27 211 vs 9 153). Thus, at first glance, the number of additional identifications obtained by *Strategy2* (compared to *Strategy1*) does not come at the cost of a deterioration of the quality of the results. In terms of LIPR, it can be noted that its average value is higher for PSM_1 (38.5% for PSM_1 vs 22.97% for PSM_2).

The performances obtained by *Strategy2* also lead us to conclude that, among the two, *Strategy2* is the one that should be implemented in OMS software. We saw that although *Strategy2* recruits more Green and Orange PSMs than *Strategy1*, it contains proportionally more Red PSMs. However, the average LIPR from the Red class obtained by *Strategy2* is much lower than for *Strategy1*. This leads us to think that a proportion of the Red PSMs from *Strategy2*, that share enough peaks corresponding to common subsequences, could be considered as "almost valid" PSMs. More precisely, we believe that with additional methodological and computational effort, some of these Red PSMs could be transfered to the Orange or Green category, an effort that methods implementing *Strategy2* should pursue.

By comparing two OMS strategies with theoretical peptides and new indicators, we also developed an environment which allowed us to see and understand elements that are more difficult to comprehend in an experimental context. This protocol could be used to understand principles that are at the heart of other (OMS) MS identification tools in order to configure and calibrate them.

References

[1] J. Griss, Y. Perez-Riverol, S. Lewis, D. L. Tabb, J. A. Dianes, N. Del-Toro, M. Rurik, M. W. Walzer, O. Kohlbacher, H. Hermjakob, R. Wang, and J. A. Vizcaíno. Recognizing millions of consistently unidentified spectra across hundreds of shotgun proteomics datasets. *Nature Methods*, 13(8):651–656, 2016.

[2] M. David, G. Fertin, H. Rogniaux, and D. Tessier. SpecOMS: A Full Open Modification Search Method Performing All-to-All Spectra Comparisons within Minutes. *Journal of Proteome Research*, 16(8):3030–3038, 2017.

[3] J. E. Elias and S. P. Gygi. Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. *Nature Methods*, 4(3):207–214, 2007.

[4] W. S. Noble. Mass spectrometrists should search only for peptides they care about. *Nature Methods*, 12(7):605–608, 2015.

[5] A. Sticker, L. Martens, and L. Clement. Mass spectrometrists should search for all peptides, but assess only the ones they care about. *Nature Methods*, 14(7):643–644, 2017.

[6] M. David, G. Fertin, and D. Tessier. SpecTrees: An Efficient Without a Priori Data Structure for MS/MS Spectra Identification. In *Algorithms in Bioinformatics (WABI)*, volume 9838 of *Lecture Notes in BioInformatics*, pages 65–76, 2016.

Abstract

SeqBIM

DeepG4 : A deep learning approach to predict active G-quadruplexes

ROCHER V.[1], GENAIS M.[1], NASSEREDDINE E.[1] and MOURAD R.[1]*

[1]*LBCMCP, Centre de Biologie Intégrative (CBI), CNRS, Université de Toulouse, UT3*
***Corresponding author**: raphael.mourad@ibcg.biotoul.fr

Abstract

G-Quadruplex (G4) are alternative DNA secondary structures composed of Guanine-rich DNA sequences which can form a four-stranded structure based on a simple strand, and let the second one free [1]. These structures have been found initially on telomeres, but more recent studies found an enrichment of theses structures on promoters of active genes, and suggest an active role in transcription of these genes [2]. Former in-silico methods to detect and study G4 remained mostly on the detection of a specific motif chain [3], but recent methods have been developed to identify G4 at genome-wide scale using Next Generation sequencing approach, like G4-seq [4](in-vitro G4) and BG4-seq [5](in-vivo). Here, we propose a sequence-based computational Deep learning model to predict in-vivo DNA G4 using the DNA sequences of BG4-seq peaks, in order to detect new motifs involved in the G4 prediction. Deep learning is a recent and popular Machine learning set of approaches where model learn features directly from the data, meaning that we could identify de-novo motifs that are related to G4 prediction. This model can be applied to any DNA sequence to predict the G4 formation, and be used in genetics to study the impact of SNP's on the DNA G4 formation propensities.

References

[1] D. Sen and W. Gilbert. Formation of parallel four-stranded complexes by guanine-rich motifs in DNA and its implications for meiosis. *Nature*, 334(6180):364–366, Jul 1988.

[2] J. Spiegel, S. Adhikari, and S. Balasubramanian. The Structure and Function of DNA G-Quadruplexes. *Trends Chem*, 2(2):123–136, Feb 2020.

[3] J. L. Huppert and S. Balasubramanian. Prevalence of quadruplexes in the human genome. *Nucleic Acids Res.*, 33(9):2908–2916, 2005.

[4] G. Marsico, V. S. Chambers, A. B. Sahakyan, P. McCauley, J. M. Boutell, M. D. Antonio, and S. Balasubramanian. Whole genome experimental maps of DNA G-quadruplexes in multiple species. *Nucleic Acids Res.*, 47(8):3862–3874, 05 2019.

[5] R. Hänsel-Hertsch, D. Beraldi, S. V. Lensing, G. Marsico, K. Zyner, A. Parry, M. Di Antonio, J. Pike, H. Kimura, M. Narita, D. Tannahill, and S. Balasubramanian. G-quadruplex structures mark human regulatory chromatin. *Nat. Genet.*, 48(10):1267–1272, 10 2016.

Abstract

Prediction of auxin response elements based on data fusion approach

Nesrine Sghaier[1]*, Rayda ben Ayed[2], Ahmed Rebai[2]

[1]Department of Computer Science, *University of Sousse, Sousse, Tunisia*
[2]Center of Biotechnology of Sfax, *University of Sfax, Sfax, Tunisia*
***Corresponding author**: nesrine.sghaier@ymail.com

Abstract

The plant hormone "auxin" is a key regulator of plant development and environmental responses. It has critical roles in directing plant cell division, differentiation, and elongation [1]. The identification of Auxin-response element (AuxRE) is one of the most important issues to understand the auxin regulation of gene expression. Over the past few years, a large number of motif identification tools have been developed. Despite this considerable efforts provided by computational biologists, building reliable models to predict regulatory elements has still been a difficult challenge [2, 3].

We propose in this work a data fusion approach for the prediction of AuxRE. Our method is based on the combined use of Dempster–Shafer evidence theory and fuzzy theory. Our method is based on a combination of predictions coming from two techniques commonly used in pattern finding: Overrepresented motifs and Linear Discriminant Analysis. The idea is to extract, for each method, some features and to combine it using the Dempster–Shafer (DS) rule, called orthogonal sum.

To evaluate our model, we have scanning the DORNRÖSCHEN promoter by our model. All proven AuxRE present in the promoter has been detected. At the 0.9 threshold we have not false positive. The comparison of the results of our model and some previous motifs finding tools, show that our model can predict AuxRE more successfully than the other tools and produce less false positive. The comparison of the results before and after combination show the importance of Dempster–Shafer combination in the decrease of false positive and to improve the reliability of prediction. For an overall evaluation we have chosen to present the performance of our approach in comparison with other methods. Our method has the high degree of sensitivity (Sn) and Positive Predictive Value (PPV) with a value of 79 and 48.17, respectively.

References

[1] B. Möller and D. Weijers: “Auxin control of embryo patterning.” *Cold Spring Harb. Perspect. Biol.* vol. 1, no. 5, pp. a001545, 2009.

[2] M.K. Das and H.-K. Dai: “A survey of DNA motif finding algorithms.” *BMC Bioinformatics*. vol. 8, no. Suppl 7, pp. S21, 2007.

[3] I.W. Davis, C. Benninger, P.N. Benfey, and T. Elich: “Powrs: Position-sensitive motif discovery.” *PLoS One*. vol. 7, no. 7, pp. e40373, 2012.

Abstract

Identification of bacterial strains using Oxford Nanopore sequencing

Grégoire Siekaniec[1,2]*

[1]*Univ Rennes, Inria, CNRS, IRISA, Rennes F-35000, France*
[2] *STLO, INRAE, Agrocampus Ouest, Rennes, France*
***Corresponding author**: gregoire.siekaniec@inria.fr

Abstract

The bacterial taxonomic assignation from sequencing data is usually based on few ubiquitous genes (RNA16S, ITS, MLST). However, due to the close genomic proximity of strains of a same species, this approach does not allow to distinguish them. Thanks to the reduction in sequencing costs, it is now possible to consider routine sequencing and identification based on whole bacterial genomes. Most current taxonomic assignation software based on whole genome only support short reads data. In contrast, our project is based on the Oxford Nanopore technology (MinION device) generating long DNA sequences. Using ONT means tackle with high error level (about 6% on raw uncorrected reads). Main existing software dealing with long reads stop at the species level, which are very efficient but do not fully exploit the potential of long reads. We want to show using the *Streptococcus thermophilus* species, that consideration of the whole genome combined with the use of long reads generally enables rapid distinction between one strain and another.

Our method is based on an efficient storing of the known genome sequences of strains. We have chosen spaced seeds [1] for this task, which are more sensitive than standard kmer indexes. Spaced seeds are kmers with defined positions that are not taken into account during matching. Then, with the help of Téo Lemane (PhD student in our lab) we extended to spaced seeds the HowDeSBT structure explained and implemented in [2] in order to obtain a small index allowing to store the genomes. Once the index is built, the second issue is the query part that assigns reads to strains. First, reads are selected based on a quality filter, to limit the error rate. Then, the data structure is requested with the seed matches extracted from the reads, each read is assigned to its potential strains and a presence/absence matrix (strains x reads) is created. Finally, the identification step was performed as an optimization issue, by looking for the minimum number of strains that can explain all the reads from the strains x reads matrix. Our implementation used the answer set programming (ASP) framework.

References

[1] Laurent Noé, Best hits of 11110110111: model-free selection and parameter-free sensitivity calculation of spaced seeds, Algorithms for Molecular Biology, volume 12, issue 1, 2017.

[2] Robert S. Harris, and Paul Medvedev. Improved representation of sequence Bloom trees. Bioinformatics, volume 36, issue 3, pages 721–727, February 2020, btz662

Abstract

SeqBIM

Nanopore MinION long read sequencer: an overview of its error landscape

Clara Delahaye[1]*, Jacques Nicolas[2]

[1,2] *Univ Rennes, Inria, CNRS, IRISA F-35000, Rennes, France*
***Corresponding author**: clara.delahaye@irisa.fr

Abstract

Third generation ONT's (Oxford Nanopore Technologies) sequencer provides longer DNA fragments (mean read length often over 10 kB) than usual second generation sequencers. However, this technology is also more error-prone [1], with currently around 6% of error on raw reads. Many articles worked on read correction methods (there even is a tool to assess error correction methods [2]), while few addressed the detailed characterization of observed errors [3], as the frequent (almost monthly!) updates in ONT chemistry and softwares hinder the task. The MinION sequencer is now getting more stable. We propose here an up-to-date view of its error landscape, using state-of-the-art flowcell and basecaller. We worked on bacterial and human data to get an overview of Nanopore sequencing error biases.

As opposed to usual NGS, Nanopore sequencing does not requires PCR amplification, thus one expects that this technology would not suffer from GC bias. Yet we found that it is actually a decisive factor linked to sequencing errors. In particular, low-GC reads have almost 2% fewer errors than high-GC reads. Nanopore sequencers are also known to struggle sequencing accurately repeated regions (homopolymers or regions with short repeats). Our work highlighted that for these regions, being the source of about half of all sequencing errors, the error profile also depends on the GC content and shows mainly deletions, although there are some reads with long insertions. Another interesting finding is that the quality measure offers valuable information on the error rate as well as the abundance of reads.

Overall we hope this work will help designing more accurate methods for error correction.

References

[1] C. L. Ip, M. Loose, J. R. Tyson, M. de Cesare, B. L. Brown, M. Jain, R. M. Leggett, D. A. Eccles, V. Zalunin, J. M. Urban, et al. Minion analysis and reference consortium: Phase 1 data release and analysis. *F1000Research*, 4, 2015.

[2] C. Marchet, P. Morisse, L. Lecompte, A. Lefebvre, T. Lecroq, P. Peterlongo, and A. Limasset. ELECTOR: evaluator for long reads correction methods. *NAR Genomics and Bioinformatics*, 2(1), 11 2019. lqz015.

[3] R. R. Wick, L. M. Judd, and K. E. Holt. Performance of neural network basecalling tools for oxford nanopore sequencing. *Genome Biology*, 20(1):129, 2019.

Abstract

SeqBIM

Contig error correction based on linked-read sequencing data

Andreea Dréau*, Clément Birbes, Christophe Klopp and Matthias Zytnicki

INRAe, Unité de Mathématiques et Informatique Appliquées de Toulouse, Castanet-Tolosan, France
***Corresponding author**: andreea.dreau@inrae.fr

Abstract

One of the main steps in genome assembly is contig assembly, which consists in reconstructing long and contiguous chromosomal parts based on the overlaps between the reads. The latest sequencing advances allow the construction of longer and more accurate contigs, but misassemblies are still present due to repeat sequences, heterozygosity and read errors. A technique that can be used for identifying these misassemblies is linked read sequencing since it provides long-range and low-error information. This type of sequencing is already used for correcting contigs by Tigmint[1], a tool that splits the contigs in loci with low molecule coverage. However, in case of contigs built from long reads and with the latest assemblers, the coverage drop is no longer sufficient for detecting misassemblies.

In this study we introduce a new correction method based on linked read information and adapted to more accurate contigs. We start by aligning the linked reads to the contigs and identifying the molecules by regrouping reads with the same barcode and aligned in the same region. Then our method computes several metrics for each contig, such as the molecule coverage, the mean read density per molecule and the mean molecule length, per 10kb window. For each metric we identify the outlier values and we split the contig if an interval is considered as outlier for at least two metrics. We tested the method by scaffolding several bovine assemblies with 3d-dna[2] and different Hi-C libraries. 3d-dna was able to connect more contigs into scaffolds and even obtain complete chromosomes when applied on contigs split with our method.

This study is part of the SeqOccIn project (https://get.genotoul.fr/seqoccin/) conducted by Get and Bioinfo Platforms of Genotoul and supported by Region Occitanie and FEDER.

References

[1] Shaun D Jackman, Lauren Coombe, Justin Chu, Rene L Warren, et al. Tigmint: correcting assembly errors using linked reads from large molecules. *BMC Bioinformatics*, 19(1):1–10, 2018.

[2] Olga Dudchenko, Sanjit S Batra, Arina D Omer, et al. De novo assembly of the *Aedes aegypti* genome using Hi-C yields chromosome-length scaffolds. *Science*, 356(6333):92–95, 2017.

Abstract

SeqBIM

kmtricks: modular k-mer count matrix and Bloom filter construction for large read collections

Téo Lemane[1]*, Rayan Chikhi[2], Pierre Peterlongo[1]

[1] *Univ. Rennes, Inria, CNRS, IRISA, Rennes, France*
[2] *Institut Pasteur, CNRS, Paris, France*
***Corresponding author**: teo.lemane@inria.fr*

Abstract

The exponential growth of sequencing data repositories prompts the development of algorithms enabling to query those repositories with any sequence of interest (similarly to Internet search engines). Despite recent intensive developments (see [1] for a detailed review), even the latest tools cannot be used to screen across large collections of sequencing experiments. The fundamental need is a compact data-structure able to assign any queried k-mers, to the list of genomic file(s) (either sequencing experiment or assembled genomes) where this k-mer occurs.

In this context, we present a novel strategy to construct a one-hash Bloom filter which is the basic data structure involved in HowDe-SBT [2], one of the state-of-the-art k-mer indexer. Our method uses minimizers in order to partition and parallelize computations. It directly counts hash values (instead of k-mers) and outputs a matrix, in which each column can be seen as a one-hash Bloom filter corresponding to one data set.

This method improves the efficiency of Bloom filter construction in terms of time and memory footprint. The matrix structure also enables to leverage information across samples in order to recover some rare k-mers usually considered as errors. We will present the algorithmic foundations, current results, and possible future works on query improvements by taking advantage of the better data locality provided by the partitioned hash space.

References

[1] Camille Marchet, Christina Boucher, Simon Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing datasets. *bioRxiv*, page 866756, dec 2019.

[2] Robert S Harris and Paul Medvedev. Improved representation of sequence Bloom trees. *Bioinformatics*, 2019.

Abstract

SeqBIM

Set-min sketch: a probabilistic map for power-law distributions with applications to k-mer annotation

Yoshihiro Shibuya[1]*, Gregory Kucherov[1]

[1]*Laboratoire d'Informatique Gaspard Monge, CNRS & Université Gustave Eiffel, Marne-la-Vallée*
***Corresponding author**: yoshi.itsame@gmail.com

Abstract

Problem: Efficient storage of k-mer counting tables is a crucial part in many bioinformatics pipelines given the ubiquity of alignment-free methods. Common counting tools [1, 2, 3, 4] usually output compressed data structures containing both k-mers and their counter values. These exact representations, albeit more memory efficient than more naive solutions, remain rather large for in-memory usage even on modern commodity computers. For example, counting all 32-mers in the human reference genome with KMC [2] produces a 20 GB file, well above the 8 GB of RAM most computers have today. For a sufficiently large k, the distribution of k-mer frequencies (k-mer spectrum) of most datasets follow a power-law distribution, where most k-mers appear a small number of times and only a few "heavy hitters" have large counter values. Representing power-law distributed counters with fixed-size words can be inefficient because only few k-mers will effectively have a counter using all allocated bits. Recent solutions try to use small counter words for low frequencies allocating additional space only when needed [5]. In many applications, explicitly storing the k-mers alongside their counters can be avoided if the set of k-mers is static. Minimal Perfect Hash Functions (MPHFs) [6, 7, 8, 9, 10] take advantage of this intuition by producing a bijective mapping between keys and integer values from 1 to the size of the input set. Both keys and values are handled by a data structure external to the MPHF, which does not solve the problem of wasting space for small counters, and needs to be rebuilt from scratch for a new key addition or deletion.

Results: Here we present Set-min sketch, a sketching technique for associative tables between keys and labels where the distribution of the labels is power-law. In our work, we focus on the special case where keys are k-mers, labels are multiplicities, the k-mer spectrum is power-law. We show, both theoretically and experimentally, that our sketch can be more space-efficient than MPHFs and provides better error guarantees compared to equally-dimensioned Count-Min sketches [11]

References

[1] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764, March 2011.

[2] Kokot M, Dlugosz M, and Deorowicz S. KMC 3: counting and manipulating k-mer statistics, September 2017.

[3] Rizk G, Lavenier D, and Chikhi R. DSK: k-mer counting with very low memory usage, March 2013.

[4] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. A General-Purpose Counting Filter: Making Every Bit Count. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 775–787, Chicago, Illinois, USA, May 2017. Association for Computing Machinery.

[5] Moustafa Shokrof, C. Titus Brown, and Tamer A. Mansour. MQF and buffered MQF: Quotient filters for efficient storage of k-mers with their counts and metadata. *bioRxiv*, page 2020.08.23.263061, August 2020.

[6] Ye Yu, Djamal Belazzougui, Chen Qian, and Qin Zhang. Memory-efficient and Ultra-fast Network Lookup and Forwarding using Othello Hashing. *arXiv:1608.05699 [cs]*, November 2017. arXiv: 1608.05699.

[7] Ye Yu, Jinpeng Liu, Xinan Liu, Yi Zhang, Eamonn Magner, Erik Lehnert, Chen Qian, and Jinze Liu. SeqOthello: querying RNA-seq experiments at scale. *Genome Biology*, 19(1):167, October 2018.

[8] Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. RecSplit: Minimal Perfect Hashing via Recursive Splitting. *arXiv:1910.06416 [cs]*, November 2019. arXiv: 1910.06416.

[9] Ingo Müller, Peter Sanders, Robert Schulze, and Wei Zhou. Retrieval and Perfect Hashing Using Fingerprinting. In Joachim Gudmundsson and Jyrki Katajainen, editors, *Experimental Algorithms*, Lecture Notes in Computer Science, pages 138–149, Cham, 2014. Springer International Publishing.

[10] Antoine Limasset, Guillaume Rizk, Rayan Chikhi, and Pierre Peterlongo. Fast and scalable minimal perfect hashing for massive key sets. *arXiv:1702.03154 [cs]*, February 2017. arXiv: 1702.03154.

[11] Graham Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. pages 44–55, 2005. 5th SIAM International Conference on Data Mining, SDM 2005 ; Conference date: 21-04-2005 Through 23-04-2005.

SeqBIM

Abstract

Phylo k-mers: constructing phylogenetically-informed k-mers for phylogenetic placement and recombination detection

Nikolai Romashchenko[1]*, Benjamin Linard[1,2], Fabio Pardi[1], Eric Rivals[1]

[1]*LIRMM, University of Montpellier, CNRS, Montpellier, France*
[2]*SPYGEN, 17 Rue du Lac Saint-André, 73370 Le Bourget-du-Lac, France*
Corresponding author: *nromashchenko@lirmm.fr

Abstract

Multiple sequence alignment is an essential preliminary step for a large number of different algorithms in bioinformatics. With the decrease of the sequencing cost, the need to process more and more data increases, making alignment-based approaches computationally expensive in practice. This led to the emergence of many alignment-free algorithms and the widespread adoption of k-mer based approaches.

We describe phylogenetically-informed k-mers, or *phylo k-mers*, a concept recently introduced in the context of phylogenetic placement in metagenomics [1], and successfully applied for viral recombination detection [2]. A phylo k-mer is a k-mer that is present with a non-negligible probability in unknown relatives of the sequences contained in an alignment. While the calculation of these probabilities is computationally heavy and requires the reference alignment as an input, it has to be done only once per alignment. Once calculated, phylo k-mers can be applied in alignment-free algorithms that require a massive input of new query sequences: in RAPPAS [1] for phylogenetic placement, and SHERPAS [2] for viral recombination detection.

We discuss methods of calculation, or *construction* of phylo k-mers, and present `xpas`, the phylo k-mer construction library. It allows for fast and memory-efficient construction of databases of phylo k-mers. Those databases are used by SHERPAS and the new version of RAPPAS, which is currently under development.

Keywords: *alignment-free*, *phylogenetics*, *k-mers*, *phylo k-mers*, *recombination*

References

[1] Benjamin Linard, Krister Swenson, and Fabio Pardi. Rapid alignment-free phylogenetic identification of metagenomic sequences. *Bioinformatics*, 35, 01 2019.

[2] Guillaume E. Scholz, Benjamin Linard, Nikolai Romashchenko, Eric Rivals, and Fabio Pardi. Rapid screening and detection of inter-type viral recombinants using phylo-k-mers. *bioRxiv*, 2020.

Extended abstract

SeqBIM

Correcting Long-Reads with k-mers: A Dream Comes True

Pierre Marijon[1], Philipp Spohr[2] Antoine Limasset[3],

[1]*Heinrich Heine University Düsseldorf Medical Faculty Institute for Medical Biometry and Bioinformatics*
[2]*Heinrich Heine University Düsseldorf, Algorithmic Bioinformatics*
[3]*Univ. Lille, CNRS, UMR 9189 - CRIStAL, F-59000 Lille*
***Corresponding author**: pierre.marijon@hhu.de

Abstract
Long-read sequencing technologies have become widespread for a broad application range. Nevertheless, they still have a high error rate. The correction of such reads is usually time and memory expensive due to the pairwise alignment step. However, previous techniques based on k-mer spectrum analysis performed very well on short-reads and even on long-reads given the availability of short-read sequences. Those techniques were able to be extremely fast and lightweight, relying on a very efficient data structure.

In this presentation, we present two ongoing works showing that such techniques can be adapted to work on noisy long reads directly without relying on precise short read data: `PanCov-Correct` corrects Nanopore reads while keeping low-covered variants, in heterozygous context. `Pcon` & `Br` performs long-read self-correction. These two correction methods reach an error-rate of around 0.1%, faster than other self-correction methods.

Keywords
long-read — correction — variant calling

1. Introduction

Third-generation DNA sequencing is increasingly becoming a standard technology for reference genome construction (de novo assembly), detection of structural variants, long-range variant phasing, and sequencing of GC-rich regions with accurate coverage. However, their higher error rate, which can exceed 10%, and complex error profiles presenting substitutions, deletions, and insertions lead to algorithmic challenges.

Most correction tools use pairwise alignment beforehand to perform the long-read correction. This step is usually the bottleneck in terms of time and memory of most pipelines. We will present preliminary results of two new correction methods based on k-mers from long-reads during this talk. `PanCov-Correct` that corrects reads from COVID-19 while preserving low-covered variants. `Pcon` and `Br` that adapt the well known k-mer spectrum correction to the long-reads error distribution.

2. Materials & Methods

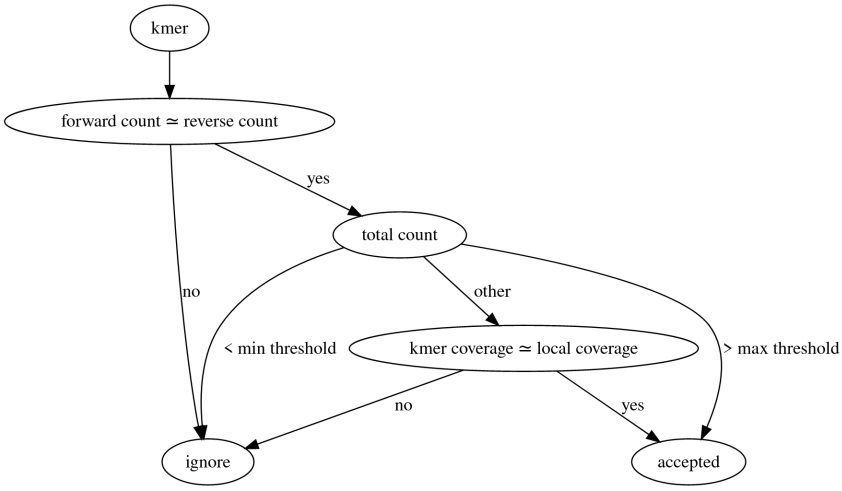


Figure 1. We apply this decision flow to select reads k-mers in the `PanCov-Correct` method i) We only consider k-mers that verify $\min\left(\frac{forwardCount}{totalCount}, \frac{reverseCount}{totalCount}\right) > 0.3$ ii) A k-mer is ignored if its total count is lower than 10 (min threshold) and accepted if upper than 50 (max threshold) iii) For a medium total coverage: we estimate the local coverage of the considered k-mer and accept the k-mer if $kmerCoverage \times 0.7 > estimateLocalCoverage$. All threshold are chosen empirically.

2.1 `PanCov-Correct`

This correction method is part of a larger pipeline `PanCov` (publication in preparation), whose goal is to call variants in COVID-19 samples sequenced with Nanopore technology.

The current state-of-the-art method to call variants of COVID-19 based on Nanopore sequencing, proposed by the ARTIC network [1], can be roughly summarized as (i) reverse transcription from RNA to DNA, (ii) viral genome amplification or enrichment, (iii) generation of the sequencing data (iv) run `Nanopolish` [1] and `Medaka`[2] on this data to produce a consensus sequence and variant calling. This pipeline detects the most abundant viral alleles present in each patient, but exhibits reduced sensitivity for alleles present at lower frequencies and often fails to detect mixed strands in samples.

The sequencing pipeline produces Nanopore reads with a 300 base pairs length, 9% error rate 300x of coverage (with some region's coverage as low as 20x), and some strand bias. Our goal is to provide a correction tool that removes the majority of sequencing errors as well as the strand bias while keeping variants with a low allele frequency.

The correction performed by `PanCov-Correct` is based on `GraphAligner`'s[2] hybrid correction method. `GraphAligner` runs `bcalm` [3] on short-reads to build a *DeBruijn* graph, maps long-reads on this graph, and uses graph information as a ground truth to correct long-reads; this method helps preserving variants if they are

[1] https://artic.network/
[2] https://github.com/nanoporetech/medaka

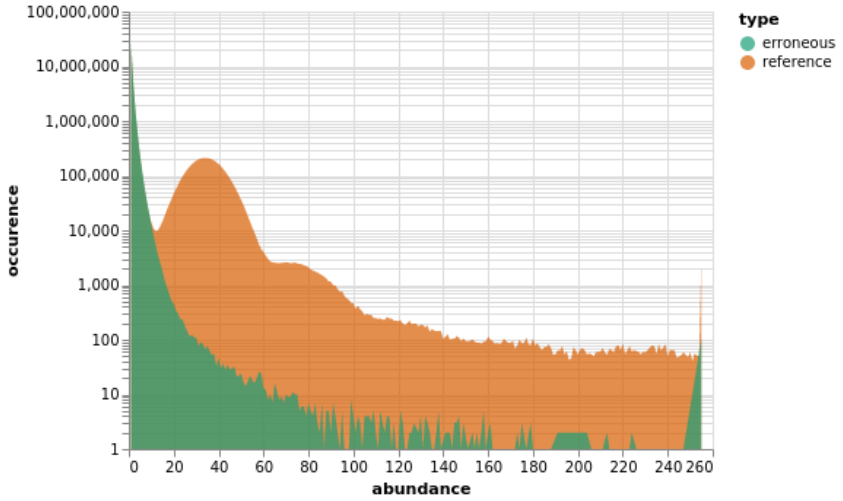


Figure 2. We can see erroneous k-mers generally have a low abundance, the majority of reference k-mers have an abundance of around 40 for this dataset. By replacing low abundance k-mers with high abundant ones we correct the errors.

present in the *DeBruijn* graph.

To replace k-mers from short-reads, we use the reference k-mers, and additional k-mers from Nanopore reads. After a count of k-mers in both strand (with `jellyfish` [4]) we select a subset with the decision tree presented in Figure 1.

The first filter removes erroneous k-mers due to strand bias, and the two other filters erroneous k-mers while keeping k-mers in low coverage regions.

With all accepted k-mers, as well as the k-mers from the reference, we build a *DeBruijn* graph with `bcalm` and use `GraphAligner` to correct reads. We execute this pipeline iteratively with an increasing value for the k-mer size.

2.2 `Pcon` & `Br`

`Pcon` & `Br` correction method is based on the k-mer spectrum and the idea that erroneous k-mers are observed less frequently than correct k-mers in the dataset. Figure 2 shows a k-mer spectrum with reference and erroneous k-mers in 50x *E. coli* Nanopore R10.3 with an error rate of around 5%. We can see that most erroneous k-mers have low coverage.

With `Pcon` and `Br`, we try to apply this method to long-reads. `Pcon` is a k-mer counter designed to count short k-mers quickly. `Br` scans reads and tries to replace low count k-mers, designated as *weak*, by k-mers with high coverage, design now as *solid*, as Musket[5] and Lighter[6].

2.2.1 `Pcon`

`Pcon` is based on a simple reversible hash function that maps all possible canonical k-mer to the range between 0 to $2^{(2\times k)-1}$. By allocating a table of size $2^{(2\times k)-1}$ `Pcon` can store the counts of all possible canonical k-mers.

Each k-mer exists in two versions, forward and reverse, that are considered indistinguishable. Therefore, to reduce memory usage, we store only one version of each k-mer. The chosen form is dubbed canonical. Most of the time, the smallest integer

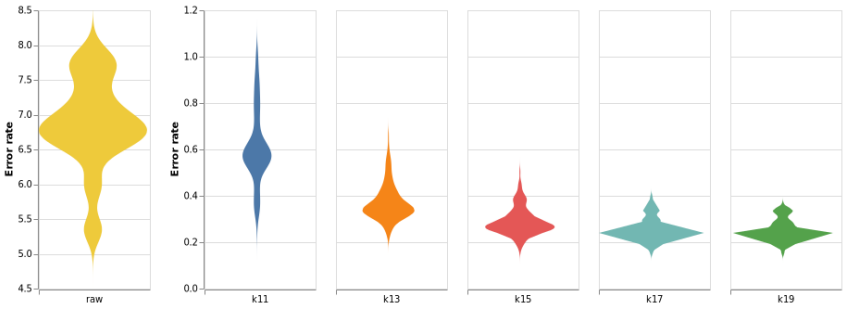


Figure 3. We present the mean error rate of each sample as a violin plot, for the raw reads and after each correction round. The first round with k=11 divides the error rate by more than 10 and further iterations allow the reads to reach an average error of 0.24%. The correction method also reduces the variance of mean error.

value is chosen as the canonical. The `Pcon` hash function is designed to avoid the computation of two versions to get the canonical one.

First, we define a function `popcount`, which returns the number of bits equal to 1 in a k-mer binary representation. The hash function converts the nucleotides of a k-mer in a two bits representation following this encoding: $A \leftarrow 00$, $C \leftarrow 01$, $G \leftarrow 11$ and $T \leftarrow 10$. This encoding's main property is that, given that the k-mer length is odd, `popcount`$(forward)$ is odd iff `popcount`$(reverse)$ is even. Thus, we define the canonical version of a k-mer as the one with even `popcount`. This property helps to determine the canonical k-mer without computing its reverse complement. Moreover, if we work only with canonical k-mers, we can remove one bit for each k-mer and reconstruct this bit by adding a one or a zero to get an even `popcount`.

`Pcon` can convert its count in a bitfield. If a k-mer count is larger than a threshold, the bit corresponding to the k-mer value is set to 1 else to 0.

`Pcon` can write its result in different formats: pcon (a gzip-compressed dump of the count table), CSV, solid (a gzip-compressed dump of bitfield), and it can also produce a k-mer spectrum. `Pcon` is usable as a standalone tool and as a Rust library with C and Python bindings.

2.2.2 `Br`

`Br` screens the read sequence uses `Pcon`'s solid bitfield to know if each k-mer is *solid* or not. When `Br` detects a *weak* k-mer, it can apply four different algorithms to correct the faulty nucleotides. Here we describe those four algorithms dubbed: **One**, **GapLength**, **Graph**, and **Greedy**.

One This algorithm is the simplest one. It supposes that a single isolated error produced the *weak* k-mer. This type of error generates a succession of k *weak* k-mers. This algorithm tries to replace the last base of the first *weak* k-mer to convert it in *solid* k-mer. This base is considered the correct one. If the N following k-mers are *solid* with this correction, `Br` validates this correction. If two or more corrections are possible, `Br` does not try to correct this error.

Graph This algorithm assumes an error generates a succession of *weak* k-mers bordered by two *solid* k-mers. **Graph** algorithm considers the set of *solid* k-mers as

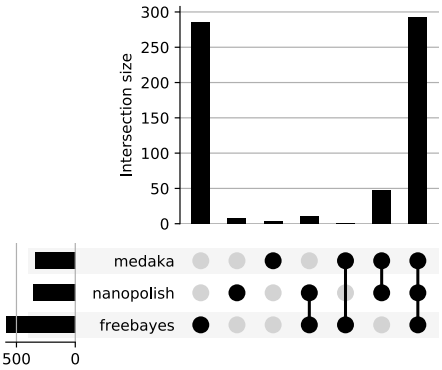


Figure 4. We present the discovered variants with Nanopore reads compared between different methods, `freebayes` corresponds to `freebayes` on the corrected read, `nanopolish` and `medaka` correspond to variants called by the ARTIC network pipeline. We observe that the amount of variant discovered exclusively by freebayes (first bar) is comparable to the amount of variant discovered by all tools (last bar)

a *DeBruijn* graph and searches a simple path between the two border *solid* k-mers. **Graph** starts from the last *solid* k-mer before the error and progresses in the *DeBruijn* graph, if we reach the first *solid* k-mer after error, we replace the erroneous section of the read by the simple path. If we reach a fork, a dead-end, or a cycle during *DeBruijn* graph exploration, `Br` does not apply any correction.

GapLength Inspired by `MindTheGap` [7], the length of *weak* k-mers allows the determination of the error type:

- If $length < k$, it is a deletion
- If $length == k$, it is an error generated by a single nucleotide change
- If $length > k$, it is a substitution or insertion with length equal to the number of *weak* k-mers minus size of k-mer

In practice, we apply the **Graph** algorithm for deletion and the **One** algorithm for isolated errors. For the last case, we can determine the number of k-mers required to replace the last erroneous base. We apply a very similar algorithm as **Graph**, but we can stop the graph exploration earlier.

Greedy Contrary to **Graph** and **GapLength**, **Greedy** does not try to analyze the errors to correct them but directly tries to replace the erroneous sequence by a path in the *DeBruijn*. This algorithm explores the *DeBruijn* graph and tries to find when the graph path matches the read sequence.

3. Result

3.1 PanCov-correct

To evaluate the `PanCov-Correct` method, we run it iteratively with a k-mer size ranging from 11 to 19 and a step size of 2. We evaluate the error rate by mapping

Dataset	Organisme	Technology	Error rate	Coverage
bacteria	*E. coli*	Nanopore R10.0	14.7%	$\approx 127x$
bacteria5	*E. coli*	Nanopore R10.3	5.9%	$\approx 54x$
bacteria7	*E. coli*	Nanopore R10.3	7.7%	$\approx 127x$
celegans	*C. elegans*	Badreads	5 %	16x, 20x, 50x to 400x per 50x step
metagenome	metagenome[3]	Nanopore R10.3	10.8%	
synthetic	*E. coli*	Badreads	1% to 10% per 1% step	$\approx 50x$
yeast	*C. elegans*	Nanopore R10.3	5 %	$\approx 283x$

Table 1. Main characteristic of the data sets used to evaluate `Pcon` & `Br` against other tools.

reads against the reference with `minimap2` and compute the mean error rate with `samtools stats`. The result is presented in Figure 3. By correcting reads with k equals 11, we reduce the error rate by ten. Another iteration improves read quality, just above 0.2% of error. We start with a k-mer size equal to 11 because most *DeBruijn* graphs built from reads are composed of one connected component with this k-mer size.

To evaluate the effect of correction on downstream analysis, we call variants on the corrected reads with `freebayes`. We compare the set of variants found in any dataset between `freebayes`, `nanopolish` and `medaka`. The result is presented in Figure 4. We can notice an important number of variants common with all variant callers, but half of the variants found by `freebayes` on the corrected reads are found exclusively by `freebayes`.

This comparison is not straightforward because we compare variant calling on raw reads and corrected reads. Moreover, we did not have a ground truth like manual curation or variant calling with second-generation reads to evaluate if the variants found only by `freebayes` are genuine variants. However, we plan to do this analysis in the future.

3.2 `Pcon` & `Br`

To evaluate `Pcon` & `Br`, we use some real and synthetic datasets. A resume of the main properties of those datasets are present in Table 1

To evaluate `Pcon`'s performance we compare wall clock time and memory usage against two other tools `jellyfish` [4] and `kmc` [8] (in ram mode), on `metagenomic` dataset reads. The results are presented in Figure 5. We observe that `Pcon` is the tool that has the best wall clock computation and memory usage (except for $k = 19$.

To evaluate `Br`'s performance we computing the error rate with the same method used previously on `PanCov-Correct` and compare our results with other self-correction tools: `CONSENT` [9], `NECAT` [10] and `Canu` [11] correction module. Results on our dataset are shown in Figure 6.

We observe that the runtime of `Br` grows slower than that of other tools. `Br` memory usage with $k = 19$ is large (140 Gb), but constant. The initial error rate impacts the corrected error rate for all tools, but this impact is larger for `Br`. On bacterial datasets with relatively small error rates, around 6%, `Br` performs similarly to other tools. Nevertheless, on more complex datasets like yeast and C. elegans, `Br`] has room for improvement.

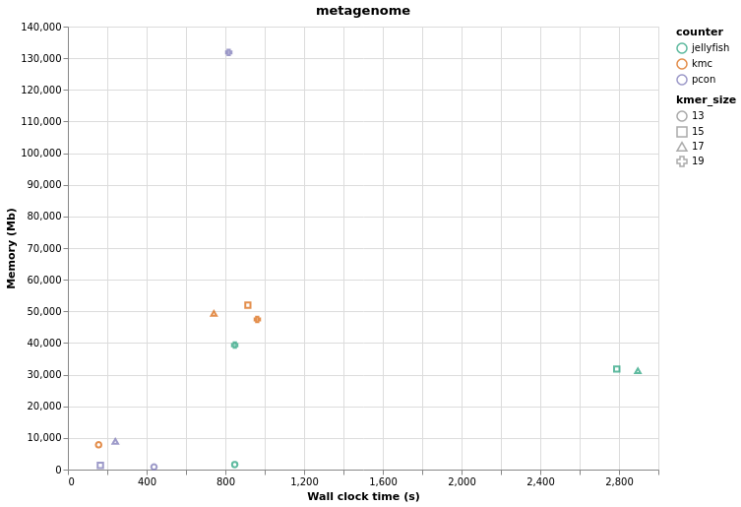


Figure 5. Comparison of `Pcon` computation time and memory usage (in purple) against different k-mers counter on different k-mer size.

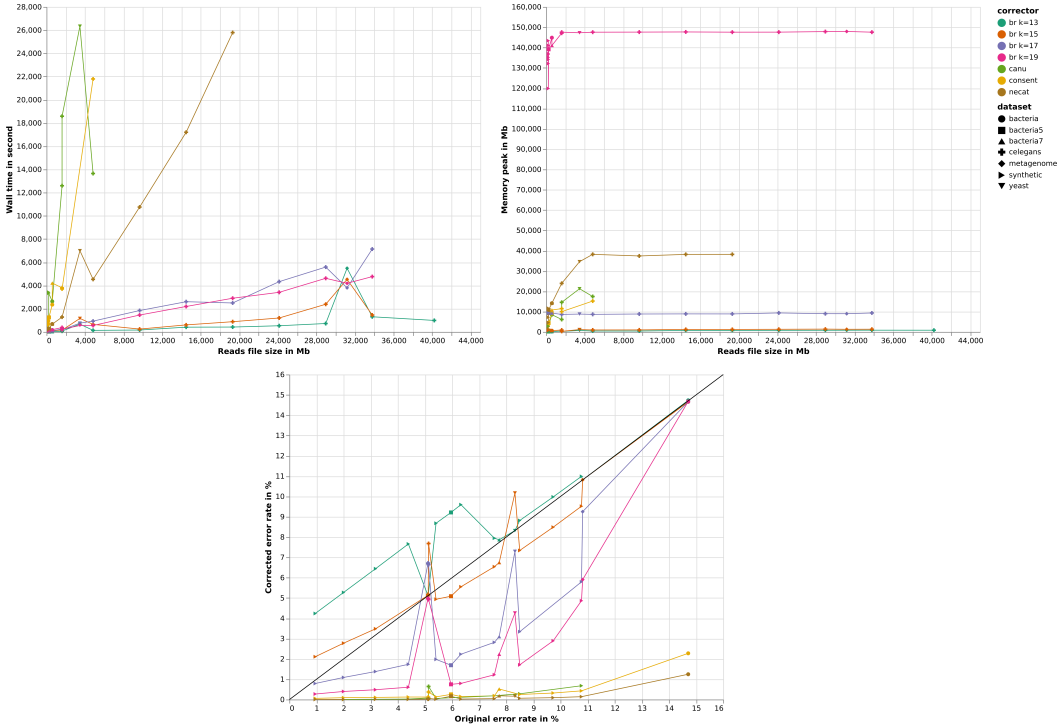


Figure 6. Runtime, memory usage and error rate, of `Br`, `CONSENT`, `NECAT` and `Canu` each point corresponds to a dataset. `Canu`, `CONSENT` and `NECAT` didn't finish in less than 9 hour on all dataset. The black line correspond to identity between original and corrected error rate.

4. Conclusion

`PanCov-Correct` produces good results in terms of correction quality, and downstream analysis allows the discovery of new variants in the COVID-19 sample. We plan to create a standalone tool with a similar method to apply it easily to other organisms.

`Pcon` for its specific target, k-mers with an abundance lower than 20, is faster than other k-mer counters. Preliminary results show that `Br` performs good correction faster than other tools, and recent improvement of raw long-read quality could help `Br` to perform a better correction. Another improvement could be to use another set structure than the one provided by `Pcon`, to reduce the memory impact and use larger k-mer.

These results on two different correction methods demonstrate that k-mer-based correction can be applied to long-read sequences. Our ongoing focus is to improve correction quality, but our first and naive approaches show those strategies' potential.

Acknowledgement

The Centre for Information and Media Technology at Heinrich Heine University Düsseldorf provided computational infrastructure and support.

References

[1] Nicholas J Loman, Joshua Quick, and Jared T Simpson. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature Methods*, 12(8):733–735, jun 2015.

[2] Mikko Rautiainen and Tobias Marschall. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome Biology*, 21(1), sep 2020.

[3] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, jun 2016.

[4] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, jan 2011.

[5] Yongchao Liu, Jan Schröder, and Bertil Schmidt. Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data. *Bioinformatics*, 29(3):308–315, nov 2012.

[6] Li Song, Liliana Florea, and Ben Langmead. Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biology*, 15(11), nov 2014.

[7] G. Rizk, A. Gouin, R. Chikhi, and C. Lemaitre. MindTheGap: integrated detection and assembly of short and long insertions. *Bioinformatics*, 30(24):3451–3457, aug 2014.

[8] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, may 2017.

[9] Pierre Morisse, Camille Marchet, Antoine Limasset, Thierry Lecroq, and Arnaud Lefebvre. CONSENT: Scalable long read self-correction and assembly polishing with multiple sequence alignment. feb 2019.

[10] Chuan-Le Xiao, Ying Chen, Shang-Qian Xie, Kai-Ning Chen, Yan Wang, Yue Han, Feng Luo, and Zhi Xie. MECAT: fast mapping, error correction, and de novo assembly for single-molecule sequencing reads. *Nature Methods*, 14(11):1072–1074, sep 2017.

[11] 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 adaptivek-mer weighting and repeat separation. *Genome Research*, 27(5):722–736, mar 2017.

Abstract

SeqBIM

srnaMapper: an optimal mapping tool for sRNA-Seq reads

Matthias Zytnicki *, Christine Gaspin

Unité de Mathématiques et Informatique Appliquées, INRAE, France
***Corresponding author**: matthias.zytnicki@inrae.fr

Abstract

Motivation Sequencing is the key method to study the impact of short RNAs, which include micro RNAs, tRNA-derived RNAs, and piwi-interacting RNA, among other. The first step to make use of these reads is to map them to a genome. Existing mapping tools have been developed for the long RNAs in mind, and, so far, no tool has been conceived for short RNAs. However, short RNAs have several distinctive features which make them different from messenger RNAs: they are shorter (not greater than 200bp), they often redundant, they can be produced by duplicated *loci*, and they may be edited at their ends.

Results In this work, we present a new tool, srnaMapper, that maps these reads with all these objectives in mind. We show on two data sets that srnaMapper is more efficient considering computation time and edition error handling: it quickly retrieves all the hits, with arbitrary number of errors.

Availability srnaMapper source code is available at https://github.com/mzytnicki/srnaMapper.

1. Introduction

Eukaryotic small RNAs (sRNAs) are defined as <200-bp long, usually untranslated, RNAs. They have been shown to participate in many aspects of cell life [1, 2].

They are generally classified according to their specific size range, biogenesis, and functional pathway. Among them, microRNAs (miRNAs) are certainly the most studied, but many other small RNAs have been shown to have a key role in regulation: tRNA-derived small RNAs (tsRNAs), small interfering RNAs (siRNAs), piwi-associated RNAs (piRNAs) and repeat-associated siRNAs (rasiRNAs), to name a few.

After the sequencing, the first task is usually to map the reads to the genome, *i.e.* predict the putative *loci* which may have produced the reads. Many mapping tools have been created so far, but none has been developed especially for sRNAs. User then resort to DNA mapping tools such as bowtie [1], bowtie2 [2], or bwa [3], with tuned parameters. Downstream tools may then be applied to filter the results.

Here, we present a new tool for efficiently mapping sRNA reads. It addresses all the particularities of these reads efficiently.

First, sRNA-producing *loci* are often duplicated: miRNAs are sometimes grouped into families, which generate highly similar or identical RNAs, and piRNAs are

produced in interaction with transposable elements, which are known to be duplicated. Our tools provides all the hits (up to user given threshold) for each read.

Second, some sRNAs, such as miRNAs, undergo editing at their ends. Both the 5' or the 3' can be shrinked, extended with a template, or both. Contrary to other tools, such as bwa or bowtie, we did not set any "seed" at the ends of the reads. Moreover, the tool requires a maximum number of errors (which can be mismatches or indels), and not a percentage, since the editing is, as far as we know, not dependent of the size of the read.

Third, sRNAs are short, usually <30bp long, and their are highly redundant (the same sRNA may be sequenced thousands times). As a result, we can store all the reads in a tree, which fits into memory, and substantially accelerate the mapping process, since the same sRNA is mapped only once, even though it is sequenced several times.

Last, our experience in sRNA-Seq showed us that the users usually want all the hits that map with the lowest number of errors. These feature is usually implemented with the option `--best --strata` in bowtie1, but is not available in every mapping tool.

2. Methods

First, a method based on q-gram filtering cannot used here, since 21bp long miRNAs, with 2 errors, should be split into q-grams that are too short to be useful.

In our implementation, the genome is indexed using the bwa suite, which creates a suffix array, together with the BW transform and the FM index. Since we will manipulate this structure like a tree, for the clarity of the discussion, we will refer to this structure as the genome tree, even though it is, *stricto sensu*, an array. The tools then stores the reads into a radix tree, where each path from the root to a terminal node stores a sequence. The number of reads for the corresponding sequence, as well as the read quality, is also stored into the terminal node.

Given a threshold k and a terminal node in the reads tree, the aim is then to find all the nodes in the genome tree with the minimal edit distance, not greater than k (when they exist). If $k = 0$, the problem reduces to finding the common sub-tree of the genome tree and the reads tree. If $k > 1$, the problem could be described as an "approximate" sub-tree search. To the best of our knowledge, the latter problem has never been described so far.

To map the reads, we first map the reads root node to the genome tree with at most k errors. We thus have a list of corresponding genome nodes. Then, we add a nucleotide from the reads tree: we try the new corresponding genome nodes using the previously computed list. We recursively traverse the reads tree this way to find all the matching genome nodes, and report the results when we find a terminal node.

We implemented several optimizations. First, we do not store one genome tree, but 4^8 trees, which start with all the 8-mers. This saves time and space, since, in our data, virtually all the 8-mers were seen in the first 8bp of the reads. Second, we first try to map a read with 0 error. If it fails, the search is backtracked to the last point where a mapping with 1 error was done, etc. Third, almost the search (except loading the genome tree) can performed in parallel. Fourth, several fastq files

can be given to the tool, and they will be mapped simultaneously. We tried other optimizations, but they did not given significant improvements.

3. Results

We compared our approach with several different tools. First, we use the widely used tool bowtie [1], bowtie2 [2], and bwa [3], with the parameters suggested by the review [4]. We also tried several "all mapper", such as Yara [5], which are tools designed to quickly retrieve all hits. Note that Yara does not make it possible to specify a fixed edit distance. Instead, the user can specify an error rate, which is the percentage of errors, given the read size. We choose an error rate of 10 (which is two errors at most for a read or size 20), and discarded reads with more that 2 erros. Other all mappers, such as FEM [6], Hobbes [7], and BitMapper2 [8], could not be used, because the reads were too short for an edit distance of 2.

Results on the two datasets show that srnaMapper mapps all the reads that the other map (Fig. 1), and maps several reads that other do not (see Supplementary Data). It also can map read with fewer errors, and find more *loci* per read. Time-wise, srnaMapper is slower than methods that map significantly less reads, but comparable with the tools that map almost the same number of reads.

We believe that srnaMapper could be the tool of choice for mapping short-RNA reads, since it maps more reads, at more locations, with a modest time difference when compared to other best tools.

References

[1] B Langmead, C Trapnell, M Pop, and S Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10:R25, 2009.

[2] B Langmead and S Salzberg. Fast gapped-read alignment with Bowtie 2. *Nat Methods*, 9:357–359, 2012.

[3] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[4] M Ziemann, A Kaspi, and A El-Osta. Evaluation of microRNA alignment techniques. *RNA*, 22:1120–1138, 2016.

[5] Enrico Siragusa, David Weese, and Knut Reinert. Fast and accurate read mapping with approximate seeds and multiple backtracking. *Nucleic Acids Research*, 41(7):e78–e78, 2013.

[6] Haowen Zhang, Yuandong Chan, Kaichao Fan, Bertil Schmidt, and Weiguo Liu. Fast and efficient short read mapping based on a succinct hash index. *BMC Bioinformatics*, 19:92, 2018.

[7] Athena Ahmadi, Alexander Behm, Nagesh Honnalli, Chen Li, Lingjie Weng, and Xiaohui Xie. Hobbes: optimized gram-based methods for efficient read alignment. *Nucleic Acids Research*, 40:e41, 2011.

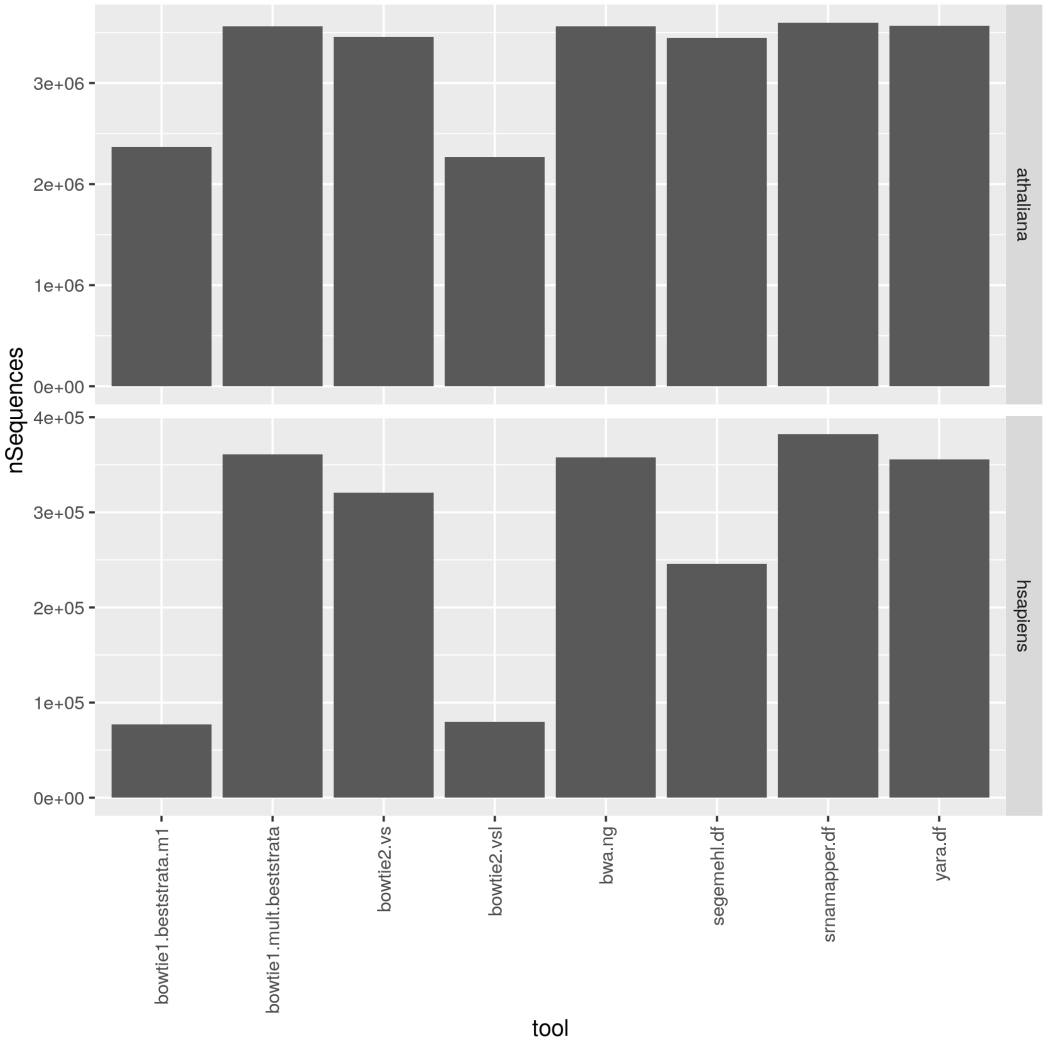


Figure 1. Number of reads mapped.

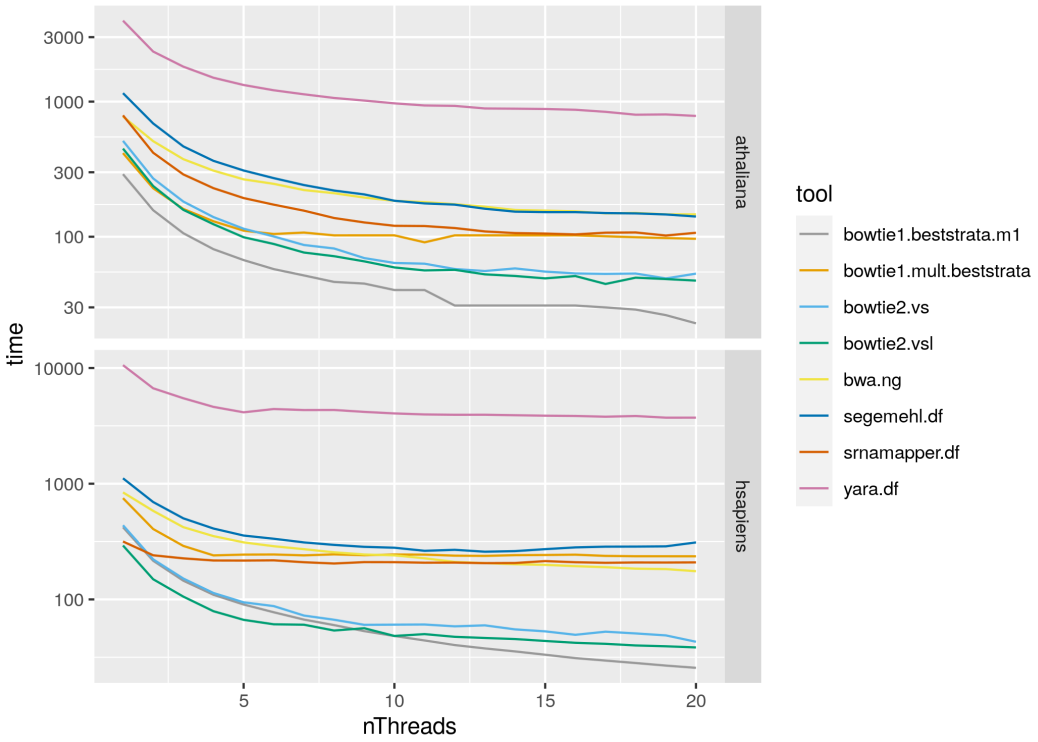


Figure 2. Time (in seconds) needed to map the reads. The y-axis is in log scale.

[8] H. Cheng, Y. Zhang, and Y. Xu. Bitmapper2: A gpu-accelerated all-mapper based on the sparse q-gram index. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(3):886–897, 2019.

Abstract

SeqBIM

Efficient enumeration of regex matches

Antoine Amarilli

LTCI, Télécom Paris, Institut Polytechnique de Paris

Abstract

Database theory research has recently studied the task of *declarative information extraction*, i.e., extracting structured information by searching for relevant patterns in unstructured textual documents. This was initially motivated by IBM's SystemT tool [1]. The task is formalized using so-called *regex-formulas*, which are regular expressions with capture variables. For instance, if we have text containing names and email addresses of the form "`John Doe <john.doe@example.com>`", the following regex formula applied to the text will extract all pairs (x, y) of a name (mapped to variable x) and its corresponding email address (mapped to variable y):

$$x\{\texttt{[A-Z][a-z]* [A-Z][a-z]*}\} \texttt{<}y\{\texttt{[a-z.]+@[a-z.]+}\}\texttt{>}$$

Recent research on this topic has studied how to efficiently perform this extraction task: given a regex formula ϕ with variables X and a textual document D, find all possible assignments of the variables X to spans (i.e., intervals of positions) of D such that ϕ is satisfied. As the number of results may be huge, we are looking for an *enumeration algorithm* [2], which produces results one after the other in an anytime fashion. Specifically, we measure the *preprocessing* time before the first answer is found, and then measure the *delay* between any two successive answers.

The proposed talk will present this research area, recent results, and ongoing research directions. It will focus on our recent work with Pierre Bourhis, Stefan Mengel, and Matthias Niewerth, published at ICDT'19 [3] and distinguished at SIGMOD Research Highlights[1]. In this work, we showed that the matches of a regex-formula in a textual document can be enumerated with linear preprocessing in the input document and constant delay between each answer, with the complexity in the regex-formula being polynomial. Our work is supported by an implementation[2] which is benchmarked in our upcoming journal article [4].

The goal is to explore how these methods could relate to those of the SeqBIM community and identify use cases, e.g., efficiently locating patterns in genomic data.

References

[1] IBM Research. SystemT, 2018.

[2] Kunihiro Wasa. Enumeration of enumeration algorithms. *CoRR*, abs/1605.05102, 2016.

[3] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT*, 2019.

[4] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. Under review, 2020.

[1] https://sigmodrecord.org/2020/07/31/constant-delay-enumeration-for-nondeterministic-document-spanners/

[2] https://github.com/PoDMR/enum-spanner-rs

Hide and Mine in Strings: Hardness and Algorithms

Giulia Bernardini[1], Alessio Conte[2], Garance Gourdel[3], Roberto Grossi[2,4], Grigorios Loukides[5]
Nadia Pisanti[2,4], Solon P. Pissis[4,6,7], Giulia Punzi[2], Leen Stougie[4,6,7], Michelle Sweering[6]
[1]University of Milano - Bicocca, Italy
[2]Università di Pisa, Italy
[3]Inria Rennes, École normale supérieure, ENS Paris-Saclay, France
[4]ERABLE Team, France
[5]King's College London, United Kingdom
[6]CWI, The Netherlands
[7]Vrije Universiteit, The Netherlands
[1]giulia.bernardini@unimib.it, [2]{alessio.conte,roberto.grossi,nadia.pisanti}@unipi.it, giulia.punzi@phd.unipi.it
[3]garance.gourdel@ens-paris-saclay.fr, [5]grigorios.loukides@kcl.ac.uk, [6]{solon.pissis,leen.stougie,michelle.sweering}@cwi.nl

***Abstract*—We initiate a study on the fundamental relation between data sanitization (i.e., the process of hiding confidential information in a given dataset) and frequent pattern mining, in the context of sequential (string) data. Current methods for string sanitization hide confidential patterns introducing, however, a number of spurious patterns that may harm the utility of frequent pattern mining. The main computational problem is to minimize this harm. Our contribution here is twofold. First, we present several hardness results, for different variants of this problem, essentially showing that these variants cannot be solved or even be approximated in polynomial time. Second, we propose integer linear programming formulations for these variants and algorithms to solve them, which work in polynomial time under certain realistic assumptions on the problem parameters.**

***Index Terms*—data privacy, data sanitization, knowledge hiding, frequent pattern mining, string algorithms**

I. Introduction

A string is a sequence of letters over some alphabet Σ. Strings are commonly used to represent individuals' data in domains ranging from transportation to web analytics and bioinformatics. For example, a string can represent a user's location profile, with each letter corresponding to a visited location [28], a user's purchasing history, with each letter corresponding to a purchased product [2], or a patient's genome sequence, with each letter corresponding to a DNA base [20]. Mining patterns from such strings is thus useful in a gamut of applications, including route planning [8], marketing [2], and clinical diagnostics [20]. To support these applications while preserving privacy, strings representing individuals' data are often being disseminated after sanitization [1], [27].

In this paper, we study the fundamental relation between *data sanitization* [1], [4], [27] (also known as *knowledge hiding*) and *frequent pattern mining* [19], [22], [25]. The objective of frequent pattern mining in strings is to obtain all patterns occurring frequently enough in a string, or in a collection of strings. There may also be constraints for the mined strings (e.g., to be of fixed length k [3], [9]). In string sanitization, the privacy objective is to transform a string to ensure that a given set of *sensitive patterns*, modeling confidential knowledge, does not occur in the sanitized version of the string; sensitive patterns are selected based on domain expertise [4], [15], [27]. This transformation may incur some utility loss that should be minimized. Recent methods achieve this using combinatorial algorithms [4], [5]. Let W be the input string over Σ, $k > 0$ be an integer, and $\mathcal{S}$ be the set of sensitive length-k substrings. These methods construct a string X such that: (I) X contains no element of $\mathcal{S}$ as a substring; (II) the total order and thus the frequency of all non-sensitive length-k substrings of W is preserved in X; and (III) the length of X is minimized [4], or the edit distance between W and X is minimized [5]. These methods work by copying carefully selected substrings of W into X and separating them by a special letter $\# \notin \Sigma$.

Example 1. Let $W = \texttt{GACAAAAACCCAT}$, $k = 3$, and the set of sensitive patterns $\mathcal{S} = \{\texttt{ACA}, \texttt{CAA}, \texttt{AAA}, \texttt{AAC}, \texttt{CCA}\}$. Further, let $X_{\mathrm{TR}} = \texttt{GAC\#ACC\#CCC\#CAT}$, $X_{\mathrm{MIN}} = \texttt{GACCC\#CAT}$ and $X_{\mathrm{ED}} = \texttt{GAC\#AA\#ACCC\#CAT}$ be three sanitized strings. All three strings contain *no sensitive pattern* and preserve the *total order* and thus the *frequency* of all non-sensitive length-3 patterns of W: X_{TR} is the trivial solution of interleaving the non-sensitive length-3 patterns of W with $\#$; X_{MIN} is the *shortest* possible such string [4]; and X_{ED} is a string *closest* to W in terms of edit distance [5].

Unfortunately, as noted in [4], the occurrences of $\#$ reveal the *locations* of sensitive patterns and thus must be ultimately replaced by letters of the original alphabet Σ. This replacement gives rise to another string over Σ, which we denote by Z. However, this replacement may create spurious patterns that could not be mined from X at a minimum frequency threshold τ but would be mined from Z at the same frequency threshold. These patterns are referred to as τ-*ghosts*.

We investigate the crucial interplay between $\#$ replacements and τ-ghosts, posing here the following question that, to the best of our knowledge, has not been addressed: *Given a string X containing $\#$'s, a positive integer k, and a positive integer τ, how should we replace the $\#$'s in X with letters in Σ, so that the number of length-k τ-ghosts in the resulting string Z is minimized?* This question helps preserving the

accuracy of frequent pattern mining and tasks based on it (e.g., pattern-based clustering [17] and classification [24], as well as sequential rule mining [26]) that we may not know a priori.

The above question is also of quite general interest, as it applies to sequential datasets that may have occurrences of a special letter for a variety of reasons beyond data sanitization. This special letter, denoted here by # for consistency, represents some information that is *missing* from these datasets. For instance, in genome sequencing data, # corresponds to an unknown DNA base [18]; in databases, # represents a value that has not been recorded [7], [12]; and in masked data outputted by other privacy-preserving methods [6], # is introduced deliberately to achieve their privacy goal.

Like in data outputted by sanitization methods, the occurrences of # in other string datasets often have to be replaced. For example, since the DNA alphabet consists of four letters (A, C, G, and T), off-the-shelf algorithms for processing DNA data use a two-bits-per-base encoding to represent the DNA alphabet. In order to use these algorithms with input strings containing unknown bases, we would have to amend them to work on the extended alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}, \#\}$. This solution may have a negative impact on the time efficiency of the algorithms or the space efficiency of the data structures they use. Thus, instead, in several state-of-the-art DNA data processing tools (e.g., [21]), the occurrences of # are replaced by an arbitrarily chosen letter from the DNA alphabet, so that off-the-shelf algorithms can be directly employed. This, however, may introduce a large number of spurious patterns, negatively affecting the accuracy of DNA analyses.

Replacing the occurrences of # in a database is often needed to be able to perform frequent pattern mining with off-the-shelf algorithms [12]. To this end, the occurrences of # are commonly replaced by some statistical estimate, such as the most frequent value [12], [16]. However, such a replacement does not generally maintain the accuracy of frequent pattern mining, since it may introduce many spurious patterns [12].

Example 2. Let again $W = \mathtt{GACAAAAACCCAT}$, $k = 3$, and $\mathcal{S} = \{\mathtt{ACA}, \mathtt{CAA}, \mathtt{AAA}, \mathtt{AAC}, \mathtt{CCA}\}$. Further, let the frequency threshold be $\tau = 2$. Note that the frequency of all non-sensitive length-3 patterns in W is preserved in all three sanitized strings $X_{\mathrm{TR}} = \mathtt{GAC\#ACC\#CCC\#CAT}$, $X_{\mathrm{MIN}} = \mathtt{GACCC\#CAT}$, and $X_{\mathrm{ED}} = \mathtt{GAC\#AA\#ACCC\#CAT}$. Replacing, however, all #'s with G would create τ-ghost GAC both in X_{TR} and in X_{ED}.

Contributions. To our knowledge, there does not exist a general solution to the question we pose here that simultaneously guarantees effectiveness and efficiency. In this work, we provide compelling evidence as to why this is the case. Within the string sanitization context, we also provide algorithms for answering this question. Specifically:

1) We embark on a theoretical study to understand the relation between replacing #'s and creating τ-ghosts. In particular, we define the following problems and examine their hardness:

- HMD (Hide and Mine decision): This is the core decision version of the problem asking whether or not we can replace all #'s in X, so that no sensitive pattern and no τ-ghost occurs in Z. Deciding this may allow for sanitizing X with no utility loss in frequent pattern mining. We show that HMD is *strongly NP-complete* via a reduction from a variant of the well-known Bin Packing problem [14] (see Section III). This is the most technically involved part of the paper, as the provided reduction is highly non-trivial.
- HM (Hide and Mine): This is the optimization version of HMD asking how we can replace all #'s, while ensuring that no sensitive patterns and a minimal number of τ-ghosts occur in Z. This would minimize the utility loss in frequent pattern mining. HM is clearly NP-hard as a consequence of HMD being NP-complete, but we also show that it is *hard to approximate*.
- HMMT (Hide and Mine minimum threshold): Given a parameter τ, this problem asks for the minimum frequency threshold $\tau_1 \geq \tau$ for which no sensitive pattern and no τ_1-ghost occurs in Z. Solving HMMT would imply no utility loss in frequent pattern mining at a higher frequency threshold τ_1 that is as close as possible to τ. We show that HMMT is (*NP-hard* and) *hard to approximate*.

The hardness (see Section III) and inapproximabilty (see Section IV) results for our problems provide solid evidence for the lack of polynomial-time exact or approximation algorithms for these problems and motivate our next contribution.

2) We develop *exact algorithms* for HMD and HM (see Section V) that require polynomial time, under certain realistic assumptions on the problem parameters:

- Exact algorithms based on an Integer Linear Programming (ILP) formulation of HMD. The main idea is to identify all length-k strings over Σ in X that may potentially become τ-ghosts in Z, and then decide whether each of the #'s can be replaced by a letter in Σ without creating any sensitive pattern or any τ-ghost pattern in Z. We prove that HMD is *fixed-parameter tractable* [11] in most cases encountered in practice (e.g., when the number of distinct letters in the string and the length k of sensitive patterns are upper bounded by a constant).
- Exact algorithms based on an ILP formulation of HM. This ILP formulation differs from the HMD formulation in that it takes into account the number of τ-ghosts created by replacing #'s, so as to minimize their number. We prove that HM is fixed-parameter tractable in many cases encountered in practice (e.g., when the length k of sensitive patterns and the number of distinct patterns that may become τ-ghosts are upper bounded by a constant).

II. Preliminaries and Problem Statement

An *alphabet* Σ is a finite nonempty set whose elements are called *letters*. We also consider an alphabet $\Sigma_{\#} = \Sigma \cup \{\#\}$, where # is a special letter not in Σ. We fix a *string* $X = X[0] \cdots X[n-1]$ of length $|X| = n$ over $\Sigma_{\#}$. The set of length-k strings over Σ is denoted by Σ^k. For two indices $0 \leq i \leq j < n$, $X[i\,..\,j] = X[i] \cdots X[j]$ is the *substring* of X that starts at position i and ends at position j of W. $\mathrm{Freq}_X(U)$ denotes the number of occurrences (starting positions) of string

U as a substring of X. A *prefix* of X is a substring of X of the form $X[0\,..\,j]$, and a *suffix* of X is a substring of X of the form $X[i\,..\,n{-}1]$. A *dictionary* over Σ is a set of strings over Σ. The dictionary used in our work is a set of length-k strings that do not occur in X; we refer to these strings as *sensitive patterns*. Any element of Σ^k that is not in this dictionary is referred to as a *non-sensitive pattern*. In combinatorics on words, such a dictionary is known as *antidictionary* and the sensitive patterns are known as *forbidden patterns* (e.g., see [10]).

Problem 1 (HIDE & MINE (HM))**.** Given an integer $k > 0$, a string $X = X_0\#X_1\#\cdots\#X_\delta$ of length n over an alphabet $\Sigma_\#$, with $|X_i| \geq k-1$, for all $i \in [0, \delta]$, a dictionary $\mathcal{S} \subseteq \Sigma^k$ such that no $S \in \mathcal{S}$ occurs in X, and an integer $\tau > 0$, compute a function $g : [\delta] \to \Sigma$ such that the following hold for string $Z = X_0 g(1) X_1 g(2) \cdots g(\delta) X_\delta$:

I The number of strings $U \in \Sigma^k$, with $\mathrm{Freq}_X(U) < \tau$ and $\mathrm{Freq}_Z(U) \geq \tau$ in Z, is minimized.
II No $S \in \mathcal{S}$ occurs in Z.

Note that function g replaces each $\#$ by exactly one letter from Σ. Condition $|X_i| \geq k-1$ means that any two $\#$'s in X are at least k positions apart. Thus, any length-k substring $X[i\,..\,i{+}k{-}1]$ of X is affected by at most one $\#$ replacement. The sanitization method of [4, Lemma 1] produces an X satisfying this condition, for *any* set $\mathcal{S} \subseteq \Sigma^k$, to guarantee that the frequency of every non-sensitive pattern is preserved in X. Thus, HM is directly applicable to the output of [4].

A string $U \in \Sigma^k$ with $\mathrm{Freq}_X(U) < \tau$ and $\mathrm{Freq}_Z(U) \geq \tau$ is referred to as τ-*ghost*. To prove NP-completeness, we consider the decision variant HMD of HM, which asks to decide if there exists any function $g : [\delta] \to \Sigma$ such that the following hold:

I No τ-ghost occurs in Z.
II No $S \in \mathcal{S}$ occurs in Z.

III. HMD IS NP-COMPLETE

Problem HMD is clearly in NP. In this section, we show it to be strongly NP-complete via a reduction from a variant of Bin Packing [14].

A. The UNIQUE-WEIGHTS BIN PACKING problem

The BIN PACKING (BP) problem is defined as follows. Given three positive integers, M (number of bins), B (capacity of every bin), and N (number of items), and a vector $[w_1, \ldots, w_N]$ of positive integers (the weights of the items), BP asks whether we can partition the items into M subsets (bins) without exceeding the capacity of any bin.

BP is *strongly* NP-complete [14], i.e., it is NP-complete even when weights and bin capacities are bounded by a polynomial function of N and M. We can thus use gadgets whose size is proportional to the numerical values in the instance $\mathcal{I}_{\mathrm{BP}}$ of BP, as if we were representing those numbers in unary notation. To simplify the reduction, we assume there are no items of weight 1 (they can be added at the end where capacity is left), and that no two items have the same weight. We refer to this variant as UNIQUE-WEIGHTS BIN PACKING (UWBP). UWBP is also strongly NP-complete; we defer the proof of this claim to the full version of the paper.

Lemma 1. UWBP *is strongly NP-complete.*

B. Overview of the Reduction from UWBP to HMD

For any UWBP instance, we construct in polynomial time an instance of HMD that has positive answer if and only if UWBP has positive answer. To this end, we will introduce several gadgets which will serve to model the different constraints of UWBP. Each gadget consists of a string of length $2k-1$ over a specific alphabet: $\#$, x, y, \$, and a letter b_i for each $i \in [M]$. We will explain how all UWBP constraints are linked to the gadgets. The gadget t_{ij} models whether item $j \in [N]$ is placed in bin $i \in [M]$:

$$t_{ij} = b_i \underbrace{x \ldots x}_{k-w_j-1} \underbrace{b_i \ldots b_i}_{w_j-1} \# \underbrace{b_i \ldots b_i}_{k-1}$$

The structure models the weight of items placed in bin i: when we replace the $\#$ with b_i, we introduce w_j occurrences of b_i^k. The gadget u_{ij}, together with t_{ij} and the set of forbidden patterns, ensures that each item is placed in some bin:

$$u_{ij} = b_i \underbrace{x \ldots x}_{k-w_j-1} \underbrace{b_i \ldots b_i}_{w_j-1} \# \underbrace{y \ldots y}_{w_j} \underbrace{x \ldots x}_{k-w_j-2} y$$

We link the filling of the ith bin with the number of occurrences of b_i^k. To limit the other non-sensitive patterns flexibly, we then choose a value τ high enough, and lower the available occurrences of each pattern by adding extra copies of them at the end. Namely, we have $k = \max_j w_j + 3$ and $\tau = \max\{M, B\} + 1$.

The final instance of HMD is the concatenation of the following patterns separated by the string \$\$:

1) t_{ij}, $\forall i, j$.
2) u_{ij}, $\forall i, j$.
3) $\tau - B - 1$ occurrences of b_i^k, $\forall i$ (allowed occurrences of b_i^k model the capacity of bin i).
4) $\tau - 2$ occurrences of $b_i x^{k-w_j-1} b_i^{w_j-1} x$, $\forall i, j$ (only one more occurrence of this pattern is allowed, and one is created by replacing the $\#$ in t_{ij} or u_{ij} with x).
5) $\tau - M$ occurrences of $y^{w_j+1} x^{k-w_j-2} y$, $\forall j$ (allowed occurrences force us to replace at least one $u_{\cdot j}$ with x for each j, thus forcing us to use b_{ij} in the corresponding t_{ij} gadget, i.e., placing each item in a bin).

The set $\mathcal{S}$ of sensitive patterns is carefully chosen to link these gadgets, and consists of the union of the following sets:

1) $\{b_{i'} b_i^{k-1} \mid i, i' \in [M], i' \neq i\}$, which forbids putting a $b_{i'}$ to replace the $\#$ in any t_{ij}, if $i' \neq i$.
2) $\{b_i y b_i^{k-2} \mid i \in [M]\}$, which forbids putting a y to replace the $\#$ in a t_{ij}.
3) $\{b_i \$ b_i^{k-2} \mid i \in [M]\}$, which forbids putting a \$ to replace the $\#$ in a t_{ij}.
4) $\{b_i y^{w_j} x^{k-w_j-2} y \mid i \in [M], j \in [N]\}$, which forbids putting any b_i to replace the $\#$ in a u_{ij}.

5) $\{b_i\$y^{w_j}x^{k-w_j-2} \mid i \in [M], j \in [N]\}$, which forbids putting a \$ to replace the # in a u_{ij}.

It can be shown that this instance of HMD has positive answer if and only if the original UWBP does, thus proving our claim. We defer the details to the full version of the paper.

Theorem 1. HMD *is strongly NP-complete.*

IV. HM is Hard to Approximate

Given the hardness of HMD, we now shift our focus on checking whether an approximately optimal solution of HM can be obtained instead. Given any instance $\mathcal{I}_M$ of a minimization problem M, an algorithm is called an α-*approximation*, for some $\alpha \geq 1$, if it runs in polynomial time in the size of $\mathcal{I}_M$ and always outputs a solution value $\Gamma \leq \alpha \cdot \text{OPT}$, where OPT denotes the optimal value for $\mathcal{I}_M$. We start with the following:

Theorem 2. *There is no α-approximation algorithm for* HM*, for any $\alpha \geq 1$, unless P=NP.*

Proof. Suppose by contradiction that an α-approximation algorithm A existed for minimizing the number of τ-ghosts in HM. We could then use A to solve HMD: the answer to HMD would be positive (i.e., there would exist a function g that creates 0 τ-ghosts) if and only if the answer of A was $\Gamma = 0 \leq \alpha \cdot \text{OPT} = 0$, which contradicts Theorem 1. □

The reader may now wonder whether the problem becomes easier should one relax the requirement for a *fixed threshold* τ. Thus, the following problem arises naturally.

Problem 2 (HMMT)**.** Given an integer $k > 0$, a string $X = X_0\#X_1\#\cdots\#X_\delta$ of length n over alphabet $\Sigma_\#$, with $|X_i| \geq k-1$ for all $i \in [0, \delta]$, a dictionary $\mathcal{S} \subseteq \Sigma^k$ such that no $S \in \mathcal{S}$ occurs in X, and an integer $\tau_0 > 0$, compute the smallest integer $\tau_1 \geq \tau_0$ so that there exists a function $g : [\delta] \to \Sigma$, such that the following hold for string $Z = X_0 g(1) X_1 g(2) \cdots g(\delta) X_\delta$:

I No $U \in \Sigma^k$, with $\text{Freq}_X(U) < \tau_1$ and $\text{Freq}_Z(U) \geq \tau_1$ occurs in Z.

II No $S \in \mathcal{S}$ occurs in Z.

The practical rationale for considering HMMT is that it could be useful if, for instance, τ_1 is only slightly larger than τ in a given HM instance. Unfortunately, we show that HMMT is NP-hard, and it is even hard to approximate.

Theorem 3. HMMT *is NP-hard.*

Proof. We reduce HMD to HMMT as follows. Let $\mathcal{I}_{\text{HMD}}$ be the instance of HMD we would like to solve for some threshold τ. We construct an instance of HMMT consisting of the X, k, and $\mathcal{S}$ from $\mathcal{I}_{\text{HMD}}$, and we also set $\tau_0 = \tau$. We denote this instance by $\mathcal{I}_{\text{HMMT}}$. The reduction takes linear time in the size of HMD. We seek to find the minimum threshold $\tau_1 \geq \tau_0$ such that no length-k substring of Z is a τ_1-ghost. Then $\mathcal{I}_{\text{HMD}}$ has a positive answer if and only if the answer τ_1 of $\mathcal{I}_{\text{HMMT}}$ is equal to $\tau_0 = \tau$. The statement thus follows. □

Observe that a pattern U is a τ-ghost if and only if $\tau \in (\text{Freq}_X(U), \text{Freq}_Z(U)]$. Therefore, the minimal number of τ-ghosts is not monotonous in τ. On the contrary, the minimal number of τ-ghosts is zero when $\tau = 0$ and all patterns are already frequent (i.e., they appear at least τ times), or when $\tau > n$ and the threshold is so high that no pattern can ever become a τ-ghost. In between, the minimal number of τ-ghosts increases whenever τ equals the frequency of some patterns in X, and then slowly decreases again. We will use this behavior, and the fact that HMD is NP-hard, to construct a string for which we cannot determine in polynomial time whether $\tau_1 = \tau_0$ or $\tau_1 > \alpha\tau_0$ (and for which we can prove that $\tau_1 \notin [\tau_0 + 1, \alpha\tau_0]$), implying inapproximability.

Theorem 4. *There is no α-approximation algorithm for* HMMT*, for any $\alpha \geq 1$, unless P=NP.*

Proof. Let X be an arbitrary string and $\mathcal{S}$ be the set of sensitive patterns as defined in HMD. Further, let T be the length-$(k-2)$ suffix of X and Z be a string obtained by replacing the #'s of X. From this instance of HMD, we will construct an instance of HMMT consisting of a string Y and a set $\mathcal{S}'$ of sensitive patterns, so that if an α-approximation algorithm existed for HMMT, we could decide HMD in polynomial time. We define Y over $\Sigma \cup \{\#, \&\}$ to be

$$Y = X(\&\&T)^{\tau_0}\&(\#T\&)^{\lceil(\alpha-1)\tau_0\rceil}.$$

Let $\mathcal{R}$ be the set of all strings $\&sT$, with $s \in \Sigma$. We define the dictionary of sensitive patterns be $\mathcal{S}' = \mathcal{S} \cup \mathcal{R}$. Note that we need to replace all #'s in $(\#T\&)^{\lceil(\alpha-1)\tau_0\rceil}$ by &'s in order not to introduce any sensitive patterns. However, doing so increases the number of $\&T\&$ patterns (and all other newly created patterns) from τ_0 to $\lceil\alpha\tau_0\rceil$. Therefore, if $\tau = \tau_0$, then the number of τ-ghosts in Z equals that in $Z(\&\&T)^{\tau_0}\&(\&T\&)^{\lceil(\alpha-1)\tau_0\rceil}$, because the additional new patterns were already occurring at least τ times in Y. However if $\tau_0 < \tau \leq \lceil\alpha\tau_0\rceil$, then there will always be at least one τ-ghost, namely $\&T\&$. Recall that deciding HMD is NP-complete. Therefore it is NP-complete to decide whether or not $\tau_1 = \tau_0$ or $\tau_1 > \lceil\alpha\tau_0\rceil$. We conclude that there exists no α-approximation algorithm for HMMT, unless P=NP. □

V. Exact Algorithms for HM

We resort to ILP to design exact algorithms for HMD and HM. In particular, we show that both problems are fixed-parameter tractable for several combinations of parameters.

We say that the length-$(k-1)$ substring U preceding an occurrence of # in X, and the length-$(k-1)$ substring V following it, form its *context* UV. Recall that there are δ occurrences of # in X, and that any two occurrences are at least k letters apart, so UV is in Σ^{2k-2}. We assign to every context UV a unique identifier (id). We write $\#_i$ for # in X if its context UV has id i. A string $N \in \Sigma^k$ is *critical* if it may become a τ-ghost, i.e., if an additional occurrence of N can be created by replacing some # by a letter in Σ and $\text{Freq}_X(N) \in [\tau - k\delta, \tau - 1]$. This is because the frequency of N cannot increase by more than $k\delta$, and the frequency of N

in X must be less than τ for N to become τ-ghost. We assign to each critical string N a unique id ℓ, and denote it by N_ℓ. We introduce the following parameters:

- $\boldsymbol{\gamma}$ number of distinct contexts present in X;
- $\boldsymbol{\delta_i}$ number of occurrences of letter $\#_i$ in X, for $i \in [\gamma]$;
- $\boldsymbol{\lambda}$ number of distinct critical length-k strings;
- $\boldsymbol{\alpha^i_{\ell,j}}$ additional number of occurrences of N_ℓ introduced by replacing a $\#_i$ with a letter $j \in \Sigma$, for $\ell \in [\lambda]$;
- $\boldsymbol{e_\ell}$ difference $(\tau - 1) - \mathrm{Freq}_X(N_\ell)$, for $\ell \in [\lambda]$.

Intuitively, $\boldsymbol{e_\ell}$ is the *budget* we have for N_ℓ: the number of its additional occurrences we can afford. Since replacing an occurrence of $\#_i$ by $j \in \Sigma$ adds k new strings in Σ^k, $\boldsymbol{\alpha^i_{\ell,j}}$ counts how many of them are equal to N_ℓ. Let $\boldsymbol{x_{i,j}}$ be the number of times we replace $\#_i$ by $j \in \Sigma$, and let $\mathcal{F} \subseteq [\gamma] \times \Sigma$ be the set of *forbidden* replacements: $(i,j) \in \mathcal{F}$ if and only if replacing $\#_i$ by j introduces a sensitive pattern. To determine whether there exists a way of replacing all $\#$'s with letters without introducing any sensitive patterns nor τ-ghosts, we need to find a solution $x \in \mathbb{Z}^{\gamma \times |\Sigma|}$ to the following problem:

$$\begin{cases} x_{i,j} \geq 0 & \forall (i,j) \in [\gamma] \times \Sigma \\ x_{i,j} = 0 & \forall (i,j) \in \mathcal{F} \\ \sum_{i \in [\gamma], j \in \Sigma} \alpha^i_{\ell,j} x_{i,j} \leq e_\ell & \forall \ell \in [\lambda] \\ \sum_{j \in \Sigma} x_{i,j} = \delta_i & \forall i \in [\gamma] \end{cases} \tag{1}$$

The first and fourth constraints ensure that each $\#$ is replaced by exactly one letter, the second constraint that we do not reinstate any sensitive patterns, and the third constraint that we do not introduce any τ-ghosts. This is clearly an ILP with $m = \gamma|\Sigma|$ variables and at most $2m+\lambda+\gamma$ constraints. The well-known algorithm by Megiddo [23] solves the ILP problem in linear time in the number constraints (resp. variables) when the number of variables (resp. constraints) is upper bounded by a constant. Hence, although HMD is NP-complete in general, if appropriate subsets of parameters are bounded by a constant, we can count on polynomial-time solutions.

To show that HMD takes polynomial time in certain cases, let us start with a general preprocessing step. We construct a static dictionary with $\mathcal{O}(1)$ access time of the letters in X and the letters in strings of $\mathcal{S}$. The value (id) of each key (letter) is chosen from $\{1, \ldots, k|\mathcal{S}| + n\}$. This construction can be done in $\mathcal{O}(n + k|\mathcal{S}|)$ time using perfect hashing [13]. We can thus lexicographically sort all length-k substrings of X and all length-k strings in $\mathcal{S}$ (viewed as strings over letter id's) using radix sort in $\mathcal{O}(nk+|\mathcal{S}|k)$ time, and construct two dictionaries, one for X and one for $\mathcal{S}$, as follows. For X, we construct a trie of all its non-sensitive length-k substrings. The value of each key (non-sensitive pattern) is its multiplicity in X. We also construct a trie of all strings in $\mathcal{S}$ in a similar fashion (no multiplicities are relevant here, so no values are stored). We store in both tries, for every node, the first letter on each of its outgoing edges in a static dictionary with $\mathcal{O}(1)$ access time [13]. Thus both trie dictionaries support $\mathcal{O}(k)$ access time: if a length-k string Q is given as a query, we first convert it to a string $I(Q)$ of id's in $\mathcal{O}(k)$ time using the letter dictionary, and then search for $I(Q)$ from the root of the tries in $\mathcal{O}(k)$ time. The total construction time is $\mathcal{O}(nk+|\mathcal{S}|k)$.

When $\delta = \mathcal{O}(1)$, the brute-force algorithm checking all possible ways to replace the $\#$'s with letters of Σ runs in polynomial time. There are $|\Sigma|^\delta$ ways to replace the $\#$'s. Each of these ways generates δk new length-k strings for which we have to check if they are sensitive or create a τ-ghost. Checking if they are sensitive can be done using the trie of $\mathcal{S}$ in $\mathcal{O}(k)$ time per each length-k string. Counting the additional number of occurrences of each length-k substring of X can be done using the trie of X in $\mathcal{O}(k)$ time. Counting the number of occurrences of each length-k string that does not occur in X can be done by constructing a trie of all such strings (we have at most δk of them per way), similar to the preprocessing step. This gives $\mathcal{O}(nk + |\mathcal{S}|k + |\Sigma|^\delta \delta k^2)$ time in total.

A problem with parameters p and q is *fixed-parameter tractable* (FPT) in p if there exists a function f and a polynomial P such that the problem has time complexity $\mathcal{O}(f(p) \cdot P(q))$ [11]. The following theorem shows three scenarios where an FPT algorithm exists for HMD.

Theorem 5. HMD *is fixed-parameter tractable if*

- (a) $|\Sigma| = \mathcal{O}(1)$ *and* $\gamma = \mathcal{O}(1)$*; or*
- (b) $|\Sigma| = \mathcal{O}(1)$ *and* $k = \mathcal{O}(1)$*; or*
- (c) $k = \mathcal{O}(1)$ *and* $\lambda = \mathcal{O}(1)$.

Proof. We first perform the above-mentioned preprocessing. (a) We will solve this case by constructing and solving the ILP in Eq. 1. We can count the number of occurrences of each length-k substring of X using the trie of X (and thus determine e_ℓ for these strings) in $\mathcal{O}(nk)$ time. The id i of each context $\#_i$ and its number δ_i of occurrences can be determined within the same complexity using a similar preprocessing: this is possible because the length of every context is $2k - 2 = \mathcal{O}(k)$. Finally, the $\alpha^i_{\ell,j}$'s and $\mathcal{F}$ can be computed in $\mathcal{O}(\gamma|\Sigma|k^2)$ total time as follows. For a context $\#_i$ and a letter $j \in \Sigma$, we create k new length-k strings when replacing $\#_i$ with j, each of which is either sensitive (in which event we add (i,j) to $\mathcal{F}$) or non-sensitive (we increase $\alpha^i_{\ell,j}$ by 1). Checking if they are sensitive can be done using the trie of $\mathcal{S}$ in $\mathcal{O}(k)$ time per length-k string. Counting the additional number of occurrences of a critical length-k substring of X can be done using the trie of X in $\mathcal{O}(k)$ time. Counting the number of occurrences of a critical length-k string that does not occur in X (note that $e_\ell = \tau - 1$ for these strings) can be done by constructing a trie of all such strings, similar to the preprocessing step. The ILP is thus constructed in $\mathcal{O}(nk + |\mathcal{S}|k + \gamma|\Sigma|k^2)$ total time. Since the number of variables in the ILP is $m = \gamma|\Sigma| = \mathcal{O}(1)$ and solving ILP's is fixed-parameter linear in the number of variables [23], HMD is FPT if γ and $|\Sigma|$ are fixed.

(b) Since every context has length $2k - 2$ and also $|\Sigma| = \mathcal{O}(1)$ and $k = \mathcal{O}(1)$, we have that $\gamma \leq |\Sigma|^{2k-2} = \mathcal{O}(1)$. Thus, if k and $|\Sigma|$ are fixed, we are in case (a), and HMD is FPT.

(c) If $k = \mathcal{O}(1)$ and $\lambda = \mathcal{O}(1)$, the numbers of constraints and variables in the ILP are not necessarily upper bounded by a constant. Therefore, we cannot directly solve the ILP in polynomial time. However, since the λ critical length-k strings contain overall at most λk different letters, we actually only need to distinguish among a bounded number of letters.

Since we do not need to consider explicitly the remaining letters, we rather represent them by a single special letter. Let $\sigma \subseteq \Sigma$ denote the set of letters contained in critical length-k strings. Note that critical length-k strings can be determined as described in part (a). Thus σ can be specified and indexed using perfect hashing [13] within the same time complexity. We introduce a new letter \$ representing all the letters in $\Sigma \setminus \sigma$, and we denote by $\mathcal{F}|_{\$}$ the set of forbidden replacements where all pairs $(i,j) \in \mathcal{F}$ with $j \in \Sigma \setminus \sigma$ are collapsed in a single pair $(i, \$)$. We thus need to find a solution $x \in \mathbb{Z}^{\gamma \times (|\sigma|+1)}$ for:

$$\begin{cases} x_{i,j} \geq 0 & \forall i \in [\gamma], j \in \sigma \cup \{\$\} \\ x_{i,j} = 0 & \forall (i,j) \in \mathcal{F}|_{\$} \\ \sum_{i \in [\gamma], j \in \sigma} \ \alpha^i_{\ell,j} x_{i,j} \leq e_\ell & \forall \ell \in [\lambda] \\ \sum_{j \in \sigma \cup \{\$\}} \ x_{i,j} = \delta_i & \forall i \in [\gamma] \end{cases} \tag{2}$$

This new ILP can be constructed in $\mathcal{O}(nk + |\mathcal{S}|k + \gamma|\Sigma|k^2)$ time, like Eq. 1. Since the ILP has only $\gamma(|\sigma|+1) = \mathcal{O}(1)$ variables, HMD is FPT for fixed k and λ [23]. We can obtain a solution to the original problem by replacing \$ by any letter in $\Sigma \setminus \sigma$ that does not create a sensitive pattern. □

We can decide in polynomial time if HM has a solution: we check all $|\Sigma|$ letter replacements at each of the δ positions where a # occurs. If, at each position, there exists at least one letter replacement that does not create a sensitive pattern, then HM has a solution. Thus, without loss of generality we assume that HM always has a solution. To minimize τ-ghosts in Z, we define a binary variable z_ℓ, $\ell \in [\lambda]$, which is equal to 1 (resp. 0) when N_ℓ has (resp. has not) become τ-ghost. The ILP formulation for HM is to find $x \in \mathbb{Z}^{\gamma \times |\Sigma|}$ so as to: Minimize $\sum_{\ell=1}^{\lambda} z_\ell$ subject to

$$\begin{cases} x_{i,j} \geq 0 & \forall (i,j) \in [\gamma] \times \Sigma \\ x_{i,j} = 0 & \forall (i,j) \in \mathcal{F} \\ z_\ell \geq 0 & \forall \ell \in [\lambda] \\ \sum_{i \in [\gamma], j \in \Sigma} \ \alpha^i_{\ell,j} x_{i,j} - k\delta z_\ell \leq e_\ell & \forall \ell \in [\lambda] \\ \sum_{j \in \Sigma} \ x_{i,j} = \delta_i & \forall i \in [\gamma] \end{cases} \tag{3}$$

Note that, in the ILP of Eq. 3, $\sum_{i \in [\gamma], j \in \Sigma} \alpha^i_{\ell,j} x_{i,j} - k\delta z_\ell \leq e_\ell$ if and only if N_ℓ is not a τ-ghost or $z_\ell = 1$.

Theorem 6. HM *is fixed-parameter tractable if*

(a) $|\Sigma| = \mathcal{O}(1)$, $\gamma = \mathcal{O}(1)$, *and* $\lambda = \mathcal{O}(1)$*; or*

(b) $k = \mathcal{O}(1)$ *and* $\lambda = \mathcal{O}(1)$.

Proof. (a) We can obtain the ILP of Eq. 3 in $\mathcal{O}(\lambda)$ time from the ILP of Eq. 1, which can be constructed in $\mathcal{O}(nk + |\mathcal{S}|k + \gamma|\Sigma|k^2)$ time; see the proof of Theorem 5(a). The ILP of Eq. 3 has at most $2m + 2\lambda + \gamma$ constraints and $m + \lambda = |\Sigma|\gamma + \lambda$ variables. Therefore HM is FPT if $|\Sigma|$, γ and λ are fixed [23].

(b) Similar to the ILP of Eq. 2 (see Theorem 5(c)), we can reduce the alphabet Σ to the letters of the critical length-k strings and a special letter \$. This new minimization ILP has $\gamma(|\sigma|+1)+\lambda \leq (k\lambda+1)^{2k-1}+\lambda = \mathcal{O}(1)$ variables. Therefore HM is FPT if k and λ are fixed [23]. □

Acknowledgments. MIUR Grant 20174LF3T8 AHeAD; University of Pisa "PRA – Progetti di Ricerca di Ateneo" (Institutional Research Grants) Grant PRA_20202021_26 "Metodi Informatici Integrati per la Biomedica"; and NWO Gravitation-grant NETWORKS-024.002.003.

REFERENCES

[1] O. Abul, F. Bonchi, and F. Giannotti. Hiding sequential and spatiotemporal patterns. *TKDE*, 22(12):1709–1723, 2010.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pages 3–14, 1995.

[3] H. Arimura and T. Uno. An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. *J. Comb. Optim.*, 13(3):243–262, 2007.

[4] G. Bernardini, H. Chen, A. Conte, R. Grossi, G. Loukides, N. Pisanti, S. P. Pissis, and G. Rosone. String sanitization: A combinatorial approach. In *ECML/PKDD*, pages 627–644, 2019.

[5] G. Bernardini, H. Chen, G. Loukides, N. Pisanti, S. P. Pissis, L. Stougie, and M. Sweering. String sanitization under edit distance. In *CPM*, pages 7:1–7:14, 2020.

[6] E. Bier, R. Chow, P. Golle, T. H. King, and J. Staddon. The rules of redaction: Identify, protect, review (and repeat). *IEEE Secur. Priv.*, 7(6):46–53, 2009.

[7] F. Biessmann, D. Salinas, S. Schelter, P. Schmidt, and D. Lange. "deep" learning for missing value imputationin tables with non-numerical data. In *CIKM*, pages 2017–2025, 2018.

[8] M. Chen, X. Yu, and Y. Liu. Mining moving patterns for predicting next location. *Inf. Syst.*, 54(C):156–168, 2015.

[9] N. Cristianini and M. W. Hahn. *Introduction to computational genomics - a case studies approach*. Cambridge University Press, 2007.

[10] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. Text compression using antidictionaries. In *ICALP*, pages 261–270, 1999.

[11] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[12] C. Fiot, A. Laurent, and M. Teisseire. Approximate sequential patterns for incomplete sequence database mining. In *FUZZ*, pages 1–6, 2007.

[13] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with O(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[14] M. R. Garey and D. S. Johnson. "Strong" NP-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, 1978.

[15] A. Gkoulalas-Divanis and G. Loukides. Revisiting sequential pattern hiding to enhance utility. In *KDD*, pages 1316–1324, 2011.

[16] J. W. Grzymala-Busse and M. Hu. A comparison of several approaches to missing attribute values in data mining. In *Rough Sets and Current Trends in Computing*, pages 378–385, 2001.

[17] V. Guralnik and G. Karypis. A scalable algorithm for clustering sequential data. In *ICDM*, pages 179–186, 2001.

[18] IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 9(20):4022–4027, 1970.

[19] U. Keich and P. A. Pevzner. Finding motifs in the twilight zone. *Bioinformatics*, 18(10):1374–1381, 2002.

[20] D. C. Koboldt, K. M. Steinberg, David E. Larson, Richard K. Wilson, and Elaine R. Mardis. The next-generation sequencing revolution and its impact on genomics. *Cell*, 155(1):27–38, 2013.

[21] R. Li, C. Yu, Y. Li, T. Wah Lam, S. Yiu, K. Kristiansen, and J. Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[22] H. M. Martinez. An efficient method for finding repeats in molecular sequences. *Nucleic Acids Research*, 11(13):4629–4634, 1983.

[23] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, 1984.

[24] S. Rangavittal, R. S. Harris, M. Cechova, M. Tomaszkiewicz, R. Chikhi, K. D. Makova, and P. Medvedev. RecoverY: k-mer-based read classification for Y-chromosome-specific sequencing and assembly. *Bioinformatics*, 34(7):1125–1131, 2017.

[25] W. Shen, J. Wang, and J. Han. Sequential pattern mining. In C. C. Aggarwal and J. Han, editors, *Frequent Pattern Mining*, pages 261–282. 2014.

[26] M. Spiliopoulou. Managing interesting rules in sequence mining. In *PKDD*, pages 554–560, 1999.

[27] Y. Wu, C. Chiang, and A. L. P. Chen. Hiding sensitive association rules with limited side effects. *TKDE*, 19(1):29–42, 2007.

[28] J. J. Ying, W. Lee, T. Weng, and V. S. Tseng. Semantic trajectory mining for location prediction. In *SIGSPATIAL*, pages 34–43, 2011.

Abstract

SeqBIM

Efficient Construction of Hierarchical Overlap Graphs

Sung Gwan Park[1], Bastien Cazaux[2], Kunsoo Park[1], Eric Rivals[2]

[1]*Department of Computer Science, Seoul National University, Korea*
[2]*LIRMM, Montpellier University, Montpellier, France*
***Corresponding author**: rivals@lirmm.fr

Abstract

The hierarchical overlap graph (HOG for short) is an overlap encoding graph that efficiently represents overlaps from a given set P of n strings. An existing algorithm constructs the HOG in $O(||P|| + n^2)$ time and $O(||P|| + n \times \min(n, \max\{|s| : s \in P\})$ space, where $||P||$ is the sum of lengths of the n strings in P. We present a new algorithm of $O(||P|| \log n)$ time and $O(||P||)$ space to compute the HOG, which exploits the segment tree data structure. We also propose an alternative algorithm using $O(||P|| \frac{\log n}{\log \log n})$ time and $O(||P||)$ space in the standard word RAM model of computation.

Work published in SPIRE 2020 conference: [1].

References

[1] Sung Gwan Park, Bastien Cazaux, Kunsoo Park, and Eric Rivals. Efficient construction of hierarchical overlap graphs. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval*, pages 277–290, Orlando, FL, Oct. 2020. Springer International Publishing.

Extended abstract

SeqBIM

On the realizations of sequence graphs

Sammy Khalife[1]*, Yann Ponty[1], Laurent Bulteau[2]

[1] *LIX, CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau, France*
[2] *LIGM, CNRS, Université Gustave Eiffel, 77454 Marne-la-Vallée, France*
***Corresponding author**: khalife@lix.polytechnique.fr

Abstract

Several language models rely on an assumption modeling each local context as a (potentially oriented) bag of words, and have proven to be very efficient baselines. Sequence graphs are the natural structures encoding their information. However, a sequence graph may have several realizations as a sequence, leading to a degree of ambiguity. Several combinatorial problems are presented, depending on three levels of generalisation (window size, graph orientation, and weights). We present some complexity results and a dynamic programming algorithm to measure this level of ambiguity.

Keywords

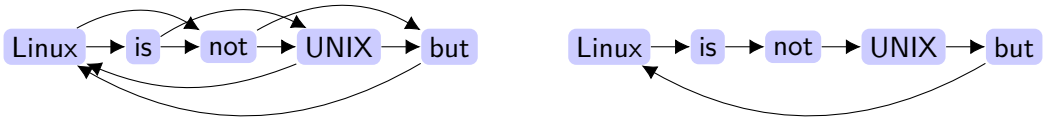
Sequence Algorithms — Graphs — Natural Language Models — Inverse problem

1. Introduction

The automated treatment of familiar objects, either natural or artifacts, always relies on a translation into entities manageable by computer programs. However, the correspondence between the object to be treated and "its" representation is not necessarily one-to-one. The representations used for learning algorithms are no exception to this rule. In particular, natural language words and textual documents representations are essential for several tasks, including document classification [1], role labelling [2], and named entity recognition [3]. The traditional models based on pointwise mutual information, or graph-of-words (GOW), [4, 5, 6], supplement the content of bag-of-words (TF, TFIDF) with statistics of co-occurrences within a **window** of fixed size w, introduced to mitigate the degree of ambiguity. Several models [7, 8, 9, 10] also use the same type of information and constitute strong baselines for natural language processing. While these representations are more precise than the traditional bag-of-words (e.g Parikh vectors), they still induce some level of ambiguity, *i.e.* a given graph can represent several sequences. Our study is thus motivated by a quantification of the level of ambiguity, seen as an algorithmic problem, coupled with an empirical assessment of the consequences of ambiguity for the representations.

2. Definitions and problem statement

Let $x = x_1, x_2, ..., x_p$ be a finite sequence of discrete elements among a finite vocabulary X. Without loss of generality, we can suppose that $X = \{1, ..., n\}$. In the following, let $I_p = \{1, ..., p\}$. This motivates the following definition:



(a) No ambiguity ($w = 3$) **(b)** Ambiguity ($w = 2$)

Figure 1. Sequence graphs (or *graphs-of-words*) built for the sentence "Linux is not UNIX but Linux" using window sizes 3 (a) and 2 respectively (b). In the second case, the sequence graph is ambiguous, since any circular permutation of the words admits the same representation.

Definition 1 *$G = (V, E)$ is the graph of the sequence x with window size $w \in \mathbb{N}^*$ if and only if $V = \{x_i \mid i \in I_p\}$, and*

$$(i, j) \in E \iff \exists (k, k') \in I_p^2,\ |k - k'| \leq w - 1\ x_k = i \text{ and } x_{k'} = j \tag{1}$$

For digraphs, Eq. (1) is replaced with

$$(i, j) \in E \iff \exists (k, k') \in I_p^2,\ k \leq k' \leq k + w - 1, x_k = i \text{ and } x_{k'} = j. \tag{2}$$

Finally, a weighted sequence graph G is endowed with a matrix $\Pi(G) = (\pi_{ij})$ such that

$$\pi_{ij} = \mathsf{Card}\ \{(k, k') \in I_p^2 \mid k \leq k' \leq k + w - 1,\ x_k = i \text{ and } x_{k'} = j\} \tag{3}$$

We say that x is a w-admissible sequence for G (or a realization of G), if G is the graph of sequence x with window size w.

The natural integers π_{ij} represent the number of co-occurrences of i and j in a window of size w. Hence, the graph of sequence is unique. An linear time algorithm to construct a weighted sequence digraph is obtained by sliding a window of size w over the sequence and incrementing the counter of presence of two elements in the window. This construction defines a correspondence between the sequence set $X^\star$ into the graph set $\mathcal{G} : \phi_w \colon X^\star \to \mathcal{G}, x \mapsto G_w(x)$. Based on these definitions, we consider the following problems:

Problem 1 (Weighted-Realizable **(W-**Realizable**))**
Input: *Possibly directed graph G, matrix weights Π, window size w*
Output: *True if (G, Π) is the w-sequence graph of some sequence x, False otherwise.*

Problem 2 (Unweighted-Realizable **(U-**Realizable**))**
Input: *Possibly directed graph G, window size w*
Output: *True if G is the w-sequence graph of some sequence x, False otherwise.*

We denote D-Realizable (resp. G-) the restricted version of Realizable where the input graph G is directed (resp. undirected), and W-Realizable (resp. U-) the restricted version of Realizable where the input graph G is weighted (resp. unweighted), possibly in combination with the D- or G- variants. We write Realizable$_w$ for the case where w is a fixed (given) constant. We also consider the variants of W-Realizable, denoted WG-Realizable and WD-Realizable where

the input graph is restricted to be respectively undirected and directed. We define UG-Realizable and UD-Realizable similarly. Finally, we write (WG-, WD-, ...)Realizable$_w$ for the case where w is a fixed strictly positive integer.

Problem 3 (Unweighted-NumRealizations (U-NumRealizations))
Input: *Possibly directed graph G, window size w*
Output: *The number of **realizations** of G,* i.e. *preimages of G through ϕ_w such that $|\{x \in X^\star \mid \phi_w(x) = G\}|$ if finite, or $+\infty$ otherwise.*

Problem 4 (Weighted-NumRealizations (W-NumRealizations))
Input: *Possibly directed graph G, matrix weights Π, window size w*
Output: *The number of **realizations** of G in the weighted sense.*

Similarly, we use the same prefix for the directed or undirected versions of (D-, G-, i.e. DU- for directed and unweighted):

DW Directed weighted	**DU** Directed unweighted
GW Undirected weighted	**GU** Undirected unweighted

We also denote NumRealizations$_w$ for the case where w is a fixed strictly positive integer. Note that NumRealizations strictly generalizes the previous one, as Realizable can be solved by testing the nullity of the number of suitable realization computed by NumRealizations.

3. Main theoretical results

3.1 Complete characterization of 2-sequence graphs

Table 1. Complexity for various instances of our problems ($w = 2$)

	NumRealizations$_2$		Realizable$_2$	
Variation	Complexity	#Sequences	Complexity	Characterization
GU	P	$\{0, +\infty\}$	P	G connected
GW	#P-hard	$\{0, 1\} \cup 2\mathbb{N}^*$	P	$\psi(G)$ (semi) Eulerian
DU	P	$\{0, 1, +\infty\}$	P	Theorem 1
DW	P	$\mathbb{N}$	P	$\psi(G)$ (semi) Eulerian

Definition 2 *Let G be a digraph, and $R^+(G)$ be the weighted DAG obtained from $R(G)$, such that the weight of an edge is attributed the number of distinct arcs from two strongly connected components in G.*

Theorem 1 *Let $G = (V, E)$ be an unweighted digraph. G is a 2-sequence graph if and only if $R^+(G)$ is a directed path and its weights are all equal to 1.*

3.2 General case: main complexity results

Table 2. Complexity for various instances of our problems ($w \geq 3$)

Variation	NumRealizations$_w$ Complexity	Realizable$_w$ Complexity	NumRealizations Complexity	Realizable Complexity
GU	P	P	W[1]-hard	W[1]-hard
GW	#P-hard $\forall w \geq 3$	NP-hard $\forall w \geq 3$	#P-hard	NP-hard
DU	Open	Open	W[1]-hard	W[1]-hard
DW	#P-hard	NP-hard	#P-hard	NP-hard

4. Dynamic programming formulation for NumRealizations$_w$

The recursion proceeds by extending a partial sequence, initially set to be empty, keeping track of for represented edges along the way. Namely, consider $N_w[\Pi, p, \mathbf{u}]$ to be the number of w-admissible sequences of length p for the graph $G = (V, E)$, respecting a weight matrix $\Pi = (\pi_{ij})_{i,j \in V^2}$, preceded by a sequence of nodes $\mathbf{u} := (u_1, \ldots, u_{|\mathbf{u}|}) \in V^\star$. It can be shown that, for all $\forall p \geq 1$, $\Pi \in \mathbb{N}^{|V^2|}$ and $\mathbf{u} \in V^{\leq w}$, $N_w[\Pi, p, \mathbf{u}]$ obeys the following formula:

$$N_w\left[\Pi, p, \mathbf{u}\right] = \sum_{v \in V} \begin{cases} N_w\left[\Pi'_{(\mathbf{u},v)}, p-1, (u_1, ..., u_{|u|}, v)\right] & \text{if } |\mathbf{u}| < w-1 \\ N_w\left[\Pi'_{(\mathbf{u},v)}, p-1, (u_2, ..., u_{w-1}, v)\right] & \text{if } |\mathbf{u}| = w-1 \end{cases} \tag{4}$$

with $\Pi'_{(\mathbf{u},v)} := (\pi_{ij} - |\{k \in [1, |\mathbf{u}|] \mid (u_k, v) = (i, j)\}|)_{(i,j) \in V^2}$. The base case of this recurrence corresponds to $p = 0$, and is defined as

$$\forall\, \Pi,\ N_w[\Pi, 0, \mathbf{u}] = \begin{cases} 1 & \text{if } \Pi = (0)_{(i,j) \in V^2} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The total number of admissible sequences is then found in $N_w[\Pi, p, \varepsilon]$, *i.e.* setting $\mathbf{u}$ to the empty prefix ε, allowing the sequence to start from any node.

The recurrence can be computed in $\mathcal{O}(|V|^w \times \prod_{i,j \in V^2}(\pi_{i,j} + 1))$ time using memoization, for p the sequence length. The complexity can be refined by noting that:

$$\sum_{i,j \in V^2} \pi_{i,j} \leq w \times p$$

It follows that, in the worst-case scenario, $\prod_{i,j \in V^2}(\pi_{i,j} + 1) \in \mathcal{O}(2^{w\,p})$. Thus, it is still possible to compute $N_w[\Pi, p, u_{1:w}]$ for "reasonable" values of p and w such as $p \leq 500$ and $w \leq 10$.

Acknowledgments

We thank Guillaume Fertin for his suggestions and questions which helped to orientate this work in the right direction.

References

[1] Konstantinos Skianis, Fragkiskos Malliaros, and Michalis Vazirgiannis. Fusing document, collection and label graph-based representations with word embeddings for text classification. In *Proceedings of the Twelfth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-12)*, pages 49–58, 2018.

[2] Michael Roth and Kristian Woodsend. Composition of word representations improves semantic role labelling. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 407–413, 2014.

[3] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[4] Jaume Gibert, Ernest Valveny, and Horst Bunke. Dimensionality reduction for graph of words embedding. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 22–31. Springer, 2011.

[5] François Rousseau, Emmanouil Kiagias, and Michalis Vazirgiannis. Text categorization as a graph classification problem. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1702–1712, 2015.

[6] Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 World Wide Web Conference*, pages 1063–1072, 2018.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[8] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[9] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. A latent variable model approach to pmi-based word embeddings. *Transactions of the Association for Computational Linguistics*, 4:385–399, 2016.

[10] Arora Sanjeev, Liang Yingyu, and Ma Tengyu. A simple but tough-to-beat baseline for sentence embeddings. *Proceedings of ICLR*, 2017.