Peer Community In Mathematical & Computational Biology

# Minimal encodings of canonical k-mers for general alphabets and even k-mer sizes

***Paul Medvedev*** *based on peer reviews by 2 anonymous reviewers*

Roland Wittler (2023) General encoding of canonical *k*-mers. bioRxiv, ver. 2, peer-reviewed and recommended by Peer Community in Mathematical and Computational Biology.
https://doi.org/10.1101/2023.03.09.531845

Submitted: 27 March 2023, Recommended: 18 September 2023

**Cite this recommendation as:**
Medvedev, P. (2023) Minimal encodings of canonical k-mers for general alphabets and even k-mer sizes. *Peer Community in Mathematical and Computational Biology*, 100188. 10.24072/pci.mcb.100188

Published: 18 September 2023



---

As part of many bioinformatics tools, one encodes a k-mer, which is a string, into an integer. The natural encoding uses a bijective function to map the k-mers onto the interval [0, s^k - ], where s is the alphabet size. This encoding is **minimal**, in the sense that the encoded integer ranges from 0 to the number of represented k-mers minus 1.

However, often one is only interested in encoding canonical k-mers. One common definition is that a k-mer is **canonical** if it is lexicographically not larger than its reverse complement. In this case, only about half the k-mers from the universe of k-mers are canonical, and the natural encoding is no longer minimal. For the special case of a DNA alphabet and odd k, there exists a ”parity-based” encoding for canonical k-mers which is minimal.

In [1], the author presents a minimal encoding for canonical k-mers that works for general alphabets and both odd and even k. They also give an efficient bit-based representation for the DNA alphabet.

This paper fills a theoretically interesting and often overlooked gap in how to encode k-mers as integers. It is not yet clear what practical applications this encoding will have, as the author readily acknowledges in the manuscript. Neither the author nor the reviewers are aware of any practical situations where the lack of a minimal encoding ”leads to serious limitations.” However, even in an applied field like bioinformatics, it would be short-sighted to only value theoretical work that has an immediate application; often, the application is several hops away and not apparent at the time of the original work.

In fact, I would speculate that there may be significant benefits reaped if there was more theoretical attention paid to the fact that k-mers are often restricted to be canonical. Many papers in the field sweep under the rug the fact that k-mers are made canonical, leaving it as an implementation detail. This may indicate that the theory to describe and analyze this situation is underdeveloped. This paper makes a step forward to develop this theory, and I am hopeful that it may lead to substantial practical impact in the future.

***References:***

[1] Roland Wittler (2023) "General encoding of canonical k-mers. bioRxiv, ver.2, peer-reviewed and recommended by Peer Community in Mathematical and Computational Biology https://doi.org/10.1101/2023.03.09.531845

# Reviews

## Evaluation round #2

### Reviewed by anonymous reviewer 2, 17 September 2023

The author addressed all my concerns. In particular, he revisited the choice of the term "MPHF" and changed the name of the manuscript accordingly as I suggested. He also removed any load balancing consideration that are already solved using standard approaches. He now correctly points out that there are no direct and important practical applications for the proposed method; rather, he aims at filling an interesting theoretical question.

Under this setting, I think the paper can be recommended as a nice theoretical contribution.

## Evaluation round #1

DOI or URL of the preprint: https://doi.org/10.1101/2023.03.09.531845
Version of the preprint: 1

### Authors' reply, 28 August 2023

**Download author's reply**

### Decision by Paul Medvedev, posted 17 May 2023, validated 18 May 2023

**Decision on your PCI MCB preprint**

Dear Dr. Roland Wittler,

Your submission has now been reviewed by two reviewers. They found your preprint to be of interest; however, they raised a number of concerns. Two concerns in particular were raised by both reviewers. The first is the use of the term MPHF, which they found misleading. The second is the lack of convincing applications / usefuleness of your encoding.

I am therefore asking for a revision which addresses the reviewer's concerns. Though all concerns are important, I believe that demonstrating usefuleness will be critical for me to recommend the preprint.

Best,
Paul Medvedev

### Reviewed by anonymous reviewer 2, 30 April 2023

The paper proposes a bijective encoding for canonical k-mers motivated by the fact that such k-mers are sparse in the universe of all k-mers, of size sigma^k where sigma is the alphabet size, and thus, the trivial encoding using klog2(sigma) bits is wasteful. In fact, there are only (sigma^k)/2 canonical k-mers among all the sigma^k k-mers and their encodings are not evenly distributed in the interval [0,sigma^k-1].

In the practical case where sigma=4, e.g. the DNA alphabet, the proposed encoding takes 2k-1 bits per k-mer rather than the naive 2k-bit encoding.
It also shown how to compute the encoding in O(1) time for DNA k-mers.

The paper is written exceptionally well; the prose is precise and concise, helping to easily go thorough the technicalities. There only some minor formatting issues and typos (as noted below). I checked some of the math and found no errors.
I would suggest, however, to add more examples to help the reader to better grasp the result of the various encoding procedures.
It is a very elegant paper overall.

My main criticism regards the usefulness of the proposed encoding. The application where this encoding could be used, as cited in the paper, is to implement a direct-addressing hash table where the encoded canonical k-mer is used as an index into the table.
Since the proposed encoding uses 1 bit less than the simple 2k-bit encoding, then the table can be shrunk by a factor of 2, from 4^k entries to 4^k/2 entries.
This is nice, but I guess only for relatively small values of k, e.g., k < 16 or so, for we expect to observe all the different canonical k-mers from the input.
With larger values of k, instead, e.g. the ”classic” k=31, we expect to only observe a (small) subset of all possible k-mers
making the direct-address table a poor choice in practice.

I am not sure what is meant by ”a MPHF allows/ensures an even distribution of k-mers to buckets of equal size”. This appears multiple times in the paper, so I wanted to note this down. Is it meant that all slots in the table are used, with no holes (where a slot is a bucket of size 1)?
To my knowledge, MPHFs are not used to ensure load balancing, rather they are used to address an array with (almost) no space overhead.
In fact, we can always hash the canonical k-mers using a pseudo random hash function (like, Murmurhash or xxhash, etc.) and take the mod of the hash code to achieve almost perfect balancing. This is also noted in the paper.
I do not know what the advantage of the presented method is compared to this; better balancing and speed, perhaps?

Lastly, I find it a bit misleading the choice of the term ”MPHF”. A MPHF is a data structure implementing a bijection for the keys of a set S, which usually is a subset of another, larger universe U. Clearly, S can be U but usually it is not and the challenge is to design a data structure implementing the bijection knowing that we will only map a subset of the keys of U.
(There is a large body of research on compressed MPHFs – see, e.g. BBHash, FHC, CHD, Recsplit, and the more recent PTHash and LPHash.)
My point is that the presented method does not work for a subset of canonical k-mers, but for the whole universe C_k^\sigma\.
Mine is not a criticism – it is stated that the work focuses on hashing the entire universe (top of page 2) – but I’m afraid most people have a different notion of minimal perfect hash function in mind and would expect to be able to build a MPHF for any wanted set of keys.
Therefore, I would suggest to use the term ”bijective encoding” which seems more appropriate to me.

Minor:

- There are several sentences terminating with a formula, for example: there

are two on page 4 (or on page 7).
Since the formula is part of a sentence, a point should grammatically terminate
the sentence. If you do not want a '.' after a formula, just rephrase in a way
that the formula does not appear at the end of the sentence. That's what I would do.

- End of page 4: "[0,|C_k|-1]" –> "[0,|C_k^\\sigma\|-1]" ?

- Page 5: notation "K(l,r)" hasn't been properly introduced. I guess "l" is a row index
and "r" is a column index into the table K.
Also T_\\sigma-1\...T_1 do not seem to have been introduce (but perhaps I missed the
point where they have!).

- Section 4.1: "|K_\16\^4|" and "|R_\16\^4|" should be "|K_\15\^4|" and "|R_\15\^4|"
since k=15?

## Reviewed by anonymous reviewer 1, 19 April 2023

The paper proposes to create a function named enc_c able to map a sequence of size k from an arbitrary alphabet of size A, to 1/2(A^k + A^(ceil(k/2)). The function is bijective. The size of the encoding is lower than A^k as enc_c has encodes only canonical kmers (when adapted to DNA-like alphabets taking into account the complement of characters, the function is called enc_c^r)

A simple implementation is available on GitLab: https://gitlab.ub.uni-bielefeld.de/gi/mphfcan/. The code is perfectly clear and the README file provides the necessary information

Despite the paper could be improved with more examples or figures, it is globally easy to follow and understand. Nevertheless, I'd say that formalism may appear a bit heavy.

A major advantage offered by this approach is that it offers a uniform distribution of canonical kmer encoding, while this is not the case when considering the usually applied canonical definition based on alphabetic order (in which kmers starting by AAA are much more often canonical than those starting by TTT for instance.)

I've several major remarks, and major questions, that make me uncomfortable to accept this work to be published in PCI.

**Terminology**.
The title and the whole text use the term "MPHF". I agree that enc_c and enc_c^r are technically speaking MPHFs. However, they can apply only on a full universe of ALL words of fixed size k on a given alphabet. SO the encodable set is of size A^k. This is not possible to encode a set of say a million kmers of size 31 on the ACGT alphabet (for instance).
This does not devaluates the work, but I really think that you should modify the name that can lead disappointed readers to stop the reading at the end of the abstract.
I may suggest something as "general kmer encoding".

**Concrete usage.**
Certainly I missed something. I tried to imagine practical use cases in which enc_c[^r] would be beneficial, but I failed.
In practice (on DNA alphabet), when I want to MPHF a set of kmers I use the following steps: 1/ transform canonical (based on lexicographic order) ASCII kmer to binary kmers using the "enc" function you describe in the manuscript, 2/ create an MPHF on the encoded set. Note that during step 1, the distribution of kmers is uneven as much canonical kmers will appear in the "smaller binary values", but this has no impact on the MPHF construction in 2.

Using enc_c, step 1 would generate uniform distribution of binary values (on a smaller space, roughly of size 1/2A^k) but finally the final step would ends up with the same result.

**Evaluation**
The evaluation somehow reflects the previous remark. The only result is about the distribution of encoded

kmers to 16 buckets. Why 16? And, more importantly, I'd be nice to also compare with any bijective hash function (such as reversible xorshift hash function for instance provided by Heng Li https://github.com/lh3/bfc/blob/master/kmer.h).

So what are the concrete advantages ? Sorry if I missed this, but the message is not clear to me.

**More minor remarks**

* Fig1: you may give a few named squares (eg first one is AAAAA, last one from first line is AAATTA if I'm correct, ...)
* Some definitions are generic while the differ when applied to completed sequences or not (eg canonical kmer definition in the preliminaries, number of palindromes, ...)
* C_k (end of page 4 undefined)
* Preliminaries: precise that sequences start at index 1 on your framework
* Page 5: avoid using the name 'K' for the function rank as this notation is already used.
* I did not understand the triangle numbers from which K is "derived" -> also what do you exactly mean by "derived"?
* Did you implement the rolling approach you propose for computing enc_c^r of successive kmers on DNA sequences?