Fast and Accurate Genome Anchoring Using Fuzzy Hash Maps

John Healy[1], Desmond Chambers[2]

[1] Department of Computing & Mathematics, Galway-Mayo Institute of Technology, Ireland. [2] Department of Information Technology National University of Ireland Galway, Ireland.

Abstract. Although hash-based approaches to sequence alignment and genome assembly are long established, their utility is predicated on the rapid identification of exact *k*-mers from a hash-map or similar data structure. We describe how a fuzzy hash-map can be applied to quickly and accurately align a prokaryotic genome to the reference genome of a related species. Using this technique, a draft genome of *Mycoplasma genitalium*, sampled at 1X coverage, was accurately anchored against the genome of *Mycoplasma pneumoniae*. The fuzzy approach to alignment, ordered and orientated more than 65% of the reads from the draft genome in under 10 seconds, with an error rate of <1.5%. Without sacrificing execution speed, fuzzy hash-maps also provide a mechanism for error tolerance and variability in *k*-mer centric sequence alignment and assembly applications.

1. Introduction

One of the more enduring approaches for performing an alignment search against a sequence database is the use of hash-tables or hash-maps. Hash-tables are dictionary data structures that use a key and a hashing function to provide rapid access to a set of mapped values [1]. Hash keys may be implemented as *k*-tuples or *k*-mers and are ideal for quickly indexing and detecting exact matches. Established *de facto* standard sequence alignment applications, such as BLAST [2, 3] and FASTA [4] have successfully employed hashing to rapidly seed and then search large databases of biological sequences. Using an approach called “seed and extend”, they employ hash-tables to seed exact matches, before extending an alignment search by attempting to join neighbouring seeds using dynamic programming algorithms. The utility of hash-tables for biological sequence alignment is constrained by the requirement that each key in a hash-table be unique. This uniqueness requirement implies that hash-tables are intolerant of variations in sequence composition, such as the insertions, deletions and polymorphisms that are common in biological sequences. It also implies that genome assemblers that util-

ise this approach require complex error correction mechanisms to deal with sequence trace errors [5, 6].

Many recent hash-based alignment applications and assemblers are notable for their use of spaced seeds [7-10]. Space seeds permit a degree of mismatch at predetermined positions in a sequence. A spaced seed is analogous to a mask and can be constructed using a template that specifies the positions in a *k*-mer that are allowed to contain mismatches. The number of matches defined in the template can also serve as a mechanism for weighting alignments. As spaced seeds inevitably result in a large number of matches, alignment applications that use this approach typically require multiple seeds to match within a region [8, 11].

The advent of second generation sequencing technologies [12-14] has resulted in a reappraisal of existing sequence alignment approaches [15, 16]. Although hash-maps appear to be ideal candidates for use with the *k*-mers of short sequence reads, alternative strategies have emerged that use the Burrows-Wheeler Transform [17] and prefix trees to align sequences [18]. Indeed, some alignment applications based on the use of spaced seeds, such as MAC and SOAP, have recently been refactored to implement this alternative approach [19]. Notwithstanding these developments, the determination of inexact matches between sequences remains an issue, invariably resulting in the application of dynamic programming approaches, which are quadratic in time and space complexity, to compare either divergent sequences or match sequences in the presence of errors.

While the uniqueness requirement of keys in a hash map appears to constrain their alignment use to the exact matching of sequences or *k*-mers, richer object-oriented programming languages offer the capability of permitting the hash key to tolerate some degree of variance. Originally proposed by Topac [20], a Fuzzy Hash Map (FHM) is a data structure that adds fuzzy capabilities to traditional hash-tables and hash-maps, using standard object-oriented techniques such as composition and inheritance. By relaxing the semantics of equality used in determining the uniqueness of a hash key, FHMs not only provide comparable speed in accessing values mapped to keys, but also can accommodate variation and provide a mechanism for error toleration. The remainder of this discussion includes a description of the structure and function of FHMs in the next section. This is followed by a discussion of a *k*-mer centric approach to genome anchor detection and extraction using FHMs and the presentation of results.

2. Fuzzy Hash Maps

Hash-tables and hash-maps are generic data structures that are invaluable for a wide variety of applications. In the Java programming language [21], hash-maps are subtypes of a generic definition for a dictionary structure called a *Map*. Maps are associative arrays in which a unique key functionally determines a value. They provide constant time performance for the basic operations of adding, removing and searching. In contrast with procedural programming languages, hash-maps in

object-oriented languages, such as Java, permit arbitrary objects to act as keys and values in a map. Thus, the notion of hash key uniqueness is not limited to the evaluation of primitive types or memory addresses, but can be customised to vary for each type of object.

In the Java language, the semantics of object equality are defined by the behaviour of the *equals()* and *hashCode()* methods, implicitly inherited by every object [21]. The generic implementation of *hashCode()* returns an integer value that is computed by mapping the memory address of an object to an integer value. The *hashCode()* method is used by *Map* implementations in Java as an initial collision detection mechanism during insertion, deletion and retrieval operations. If two objects share the same hash code, the *equals()* method is used to resolve any ambiguity and avoid unnecessary naming collisions.

The semantics of object equality depend on the implementation and form part of the design work for a class. In general, objects that are equal according to the *equals()* method must share the same hash code. Although the default implementation of *equals()* returns true if two objects share the same object ID, it is often desirable to relax the definition of equality to allow some scope for variability.

Using a modification of the approach described by Topac [20], a FHM data structure may be implemented by using the *hashCode()* method to encourage initial collisions during a search of the map. In the FHM structure, the key values are instances of a *FuzzyHashKey* type, each of which map to a collection of *k*-mers. The essence of this approach is to permit collisions based on an exact match of part of a key and confirm a match if the similarity of the remainder of the key is above a specified threshold. Initial collisions are determined by the implementation of *hashCode()* and are conceptually similar to the spaced seed approach described by Ma [11]. The degree of similarity is determined by the implementation of the *equals()* method and can utilise any sequence similarity algorithm that is capable of returning a fuzzy value in the interval [0..1].

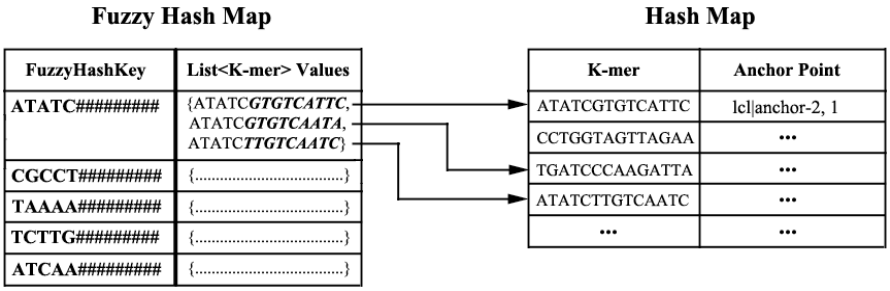

Fig. 1 A *FuzzyHashKey* initialised to cause collisions if the first five bases in a sequence are the same. If the *hashCode()* method returns true for the initial five bases, the *equals()* method is invoked to resolve equality. In practice, *k*-mer sizes of at least 21 bases are used, with collisions only permitted if at least 50% of the search string is an exact match with a hash key.

In the context of genome alignment and anchoring, the application of a FHM structure requires specifying that part of a *k*-mer that may be used to compute the

initial hash code value. A *FuzzyHashKey* can be configured to cause collisions based on exact prefix, suffix or sub-sequence matches. A determination of whether a collision on a hash code equates to a match can be accomplished by computing the edit distance between the remainder of the hash key and the corresponding part of the search string. Edit distance metrics such as Hamming Distance [22] and Fuzzy Hamming Distance [23] are ideal for fixed-length hash keys, such as *k*-mers. Other metrics such as Levenshtein Distance [24] can be used to compute the edit distance between variable-length hash keys.

Consistent with the "seed and extend" approach used by many hash-based aligners [15, 18], FHMs are a compromise between speed and sensitivity. Computing a hash code on too small a part of a *FuzzyHashKey* effectively flattens much of the hash-map into a list, with a reduction in speed proportional to the time complexity of the sequence similarly algorithm used. Hash codes computed on larger portions of the *FuzzyHashKey* will increase the search speed at the expense of sensitivity.

3. Anchoring a Genome

To illustrate the relevance and applicability of FHMs to sequence alignment, the approach was used to anchor a draft genome of *M.genitalium*, at 1X coverage, against the complete genome of *M.pneumoniae*. A measure of the effectiveness of FHMs for intra-species alignment was achieved by anchoring a draft genome of *E.coli 536*, also sampled at 1X coverage, against the complete genome of *E.coli K-12-MG1655*. Given the *k*-mer centric nature of the alignment approach, a *k*-mer centric strategy was also applied to detect and extract anchoring regions from each reference genome. The anchor detection and extraction process consists of three main phases; the construction of a de Bruijn graph to represent overlapping *k*-mers, the transformation of the de Bruijn graph into a sequence graph, and the extraction of unique sequences from the transformed graph.

A de Bruijn graph is analogous to the output of a shotgun sequencing experiment that perfectly samples a genome, generating a set of fixed-length reads, with a graph node representing each base position [14]. Nodes in the graph are connected by edges that are weighted to reflect the multiplicity of matches to a given *k*-mer. First described for genome assembly by Pevzner [5], the de Bruijn graph approach has been successfully used for short-read assemblers such as Velvet [6], ALLPATHS [25] and ABySS [26].

In the context of anchor detection and extraction, a de Bruijn graph creates a perfect tiling path through a complete reference genome. Although the memory requirements for a de Bruijn graph are huge [13], this can be reduced by a number of orders of magnitude, by merging together nodes that have an in-degree, out-degree and edge multiplicity of one. As each of these merged nodes represents a unique anchor sequence in a graph, they are easily detected using a depth-first search and can be subsequently extracted with ease. Critically, the exact position

of each anchor in the genome is recorded during the extraction process. The extracted anchors are saved in FASTA format and also as a serialized map.

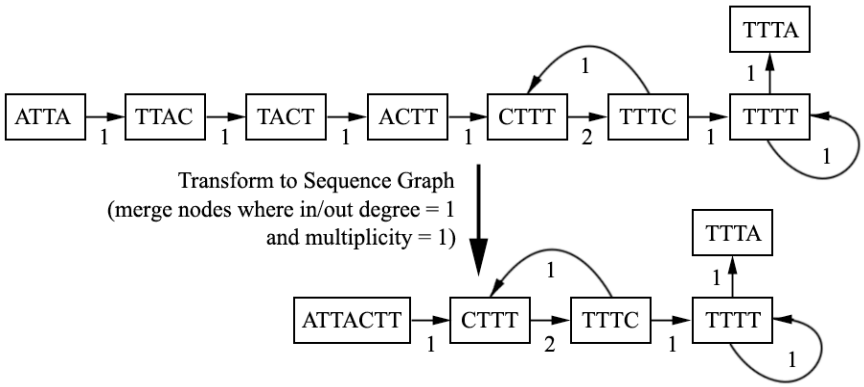

Fig. 2. A 4-mer de Bruijn graph for sequence ATTACTTTCTCTTA. The graph implementation typically decorates the edges with additional information, such as the multiplicity of overlaps between adjacent nodes. Tranforming the de Bruijn graph into a sequence graph is accomplished by merging all adjacent nodes with an in-degree, out-degree and edge multiplicity of 1. All other nodes in the graph represent repetitive sequences and are not extracted.

Before the reads of a draft genome are aligned, the extracted anchors are first read into a FHM structure. The initialisation of the FHM requires the specification of a *FuzzyHashKey* and a threshold value to score the alignment of each draft read. The *FuzzyHashKey* used must also be configured, with an appropriate mask for computing hash codes, an algorithm for determining sequence similarity and a fuzzy threshold. The fuzzy threshold is a value between 0 and 1 and is used to filter out potentially spurious alignments. Thus, a fuzzy threshold of 0.8 will only result in a match, if a given hash code results in a collision in the map and there is a high degree of similarity between the remainder of the search string and the remainder of the hash key.

The alignment of a draft genome requires that each draft read be decomposed into a set of *k*-mers and then added to the FHM. Each time a match is found in the FHM, the name and index of the anchor is recorded, along with the orientation of the read. When each *k*-mer of a read has been aligned with the FHM, a majority count is used to select the best anchor and compute the correct orientation of each read. This process iterates for each read in the draft genome. It is noteworthy that each anchor maintains a list of the name, orientation and starting position of each aligned read. Sorting this list yields the order in which the set of reads align to an anchor. As each anchor knows its own starting position with respect to the reference genome, this process not only orientates each read, but also orders the full set of aligned reads.

4. Evaluation of Results

The 0.58Mb genome of *M.genitalium* was sampled at 1X coverage and aligned against the 0.81Mb genome of *M.pneumoniae*. Using a *k*-mer size of 24 and with fuzzy index and fuzzy threshold values of 11 and 0.8 respectively, the FHM approach anchored 65.56% of the *M.genitalium* reads. A total of 1.47% of the draft reads were erroneously ordered and just 0.84% orientated incorrectly. The execution time for the genome alignment and anchoring was under ten seconds.

To illustrate the utility of the approach for intra-species comparison, the 4.9Mb genome of *E.coli 536* was also sampled at 1X coverage and anchored against the 4.6Mb complete genome of *E.coli K-12-MG1655*. Using the same *k*-mer size and fuzzy parameters described above, 80.88% of the draft genome was anchored, with 1.5% and 1.18% of reads incorrectly ordered and orientated. The execution time required to anchor *E.coli 536*, whose genome is approximately ten times the size of that of *M.genitalium*, was less than one minute.

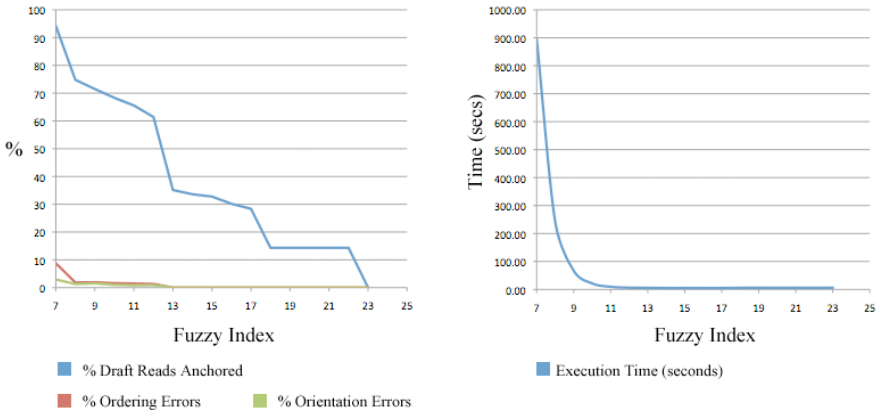

Fig. 3. The choice of fuzzy index has a major impact on the performance of FHMs and requires a balance between speed and sensitivity.

In common with most sequence alignment applications, FHMs involve a trade-off between execution speed and sensitivity. In the context of FHMs, the most important consideration in this respect is the design of the *FuzzyHashKey*. Execution speed is improved by discouraging collisions, through the specification of more rigorous exact matching criteria in the *hashCode()* method. Improved sensitivity requires a larger part of the hash key be used to compute edit distance.

In this study, the *FuzzyHashKey* was configured to cause an initial collision if the first *n* characters in the prefix of a *k*-mer matched the prefix of a *FuzzyHashKey* already in the FHM. A full collision and subsequent match was therefore predicated on the remainder of the *k*-mer matching the existing *FuzzyHashKey*, with a fuzzy value at or above a specified threshold. Constructing a *FuzzyHashKey* in such a manner is valid for this approach, as the *k*-mers in question repre-

sent a tiling path through a de Bruijn graph, with a logical graph existing in the FHM for each anchoring sequence. Empirical evidence, using the genomes of *M.genitalium* and *M.pneumoniae*, demonstrates that, for a *k*-mer size of 24, once the fuzzy index falls below 11, the percentage of the draft reads anchored increases at the expense of larger ordering and orientation errors. Moreover, the execution time begins to increase exponentially, as the variance permitted in the hash key increases. Using too small a part of the *FuzzyHashKey* to compute an exact match dramatically increases collisions, and has the effect of collapsing much of the FHM into a list, compromising execution speed.

Another FHM property worthy of discussion is the fuzzy threshold. This value is used by the edit distance algorithm to make a determination on object equality and to decide whether or not a match is permitted. Fuzzy threshold values of 0.8 were applied for both comparisons in this study. For more divergent species, this value should be relaxed, permitting more variability to be entertained by the collision detection mechanism. Low fuzzy thresholds reduce the criteria for determining a match and increase the potential for detecting an alignment. However, too low a value may result in spurious matches and increase the number of ordering and orientation errors.

6. Conclusions

FHMs combine the speed of data structures based on hashing functions with the sensitivity of sequence similarity algorithms, such as dynamic programming techniques. Although not constrained to fixed-length keys, their application to *k*-mer centric approaches to sequence alignment offers an alternative to existing implementations of the successful “seed and extend” strategy used by many applications. Moreover, due to the object-oriented nature of the approach, FHMs are easily configured with alternative keys and provide a fast, robust, flexible and extensible mechanism for sequence alignment.

Perhaps the most cogent property of FHMs is that they permit a degree of error tolerance. This property is potentially invaluable to *k*-mer centric sequence alignment and genome assembly applications. The application of FHMs to de Bruijn graphs is particularly worthy of investigation, as these graph structures are intolerant of errors, requiring complex algorithms to detect and correct the tips and bubbles caused by polymorphisms and sequence errors.

References

[1] M. Goodrich and R. Tamassia, "Data Structures and Algorithms in Java," John Wiley & Sons, 2001.

[2] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology,* vol. 215, pp. 403-410, 1990.
[3] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research,* vol. 25, p. 3389, 1997.
[4] W. Pearson and D. Lipman, "Improved tools for biological sequence comparison," *Proceedings of the National Academy of Sciences,* vol. 85, p. 2444, 1988.
[5] P. Pevzner, H. Tang, and M. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proceedings of the National Academy of Sciences of the United States of America,* vol. 98, p. 9748, 2001.
[6] D. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de Bruijn graphs," *Genome Research,* vol. 18, p. 821, 2008.
[7] H. Li, J. Ruan, and R. Durbin, "Mapping short DNA sequencing reads and calling variants using mapping quality scores," *Genome Research,* vol. 18, p. 1851, 2008.
[8] S. Rumble, P. Lacroute, A. Dalca, M. Fiume, A. Sidow, and M. Brudno, "SHRiMP: accurate mapping of short color-space reads," *PLoS computational biology,* vol. 5, 2009.
[9] R. Li, Y. Li, K. Kristiansen, and J. Wang, "SOAP: short oligonucleotide alignment program," *Bioinformatics,* vol. 24, p. 713, 2008.
[10] H. Lin, Z. Zhang, M. Zhang, B. Ma, and M. Li, "ZOOM! Zillions of oligos mapped," *Bioinformatics,* vol. 24, p. 2431, 2008.
[11] B. Ma, J. Tromp, and M. Li, "PatternHunter: faster and more sensitive homology search," *Bioinformatics,* vol. 18, p. 440, 2002.
[12] N. Hall, "Advanced sequencing technologies and their wider impact in microbiology," *Journal of Experimental Biology,* vol. 210, p. 1518, 2007.
[13] M. Schatz, A. Delcher, and S. Salzberg, "Assembly of large genomes using second-generation sequencing," *Genome Research,* vol. 20, p. 1165, 2010.
[14] M. Pop, "Genome assembly reborn: recent computational challenges," *Briefings in bioinformatics,* vol. 10, p. 354, 2009.
[15] S. Batzoglou, "The many faces of sequence alignment," *Briefings in bioinformatics,* vol. 6, p. 6, 2005.
[16] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Brief Bioinform,* p. bbq015, 2010.
[17] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm," *Digital SRC Research Report,* 1994.
[18] P. Flicek and E. Birney, "Sense from sequence reads: methods for alignment and assembly," *Nature Methods,* vol. 6, pp. S6-S12, 2009.
[19] R. Li, C. Yu, Y. Li, T. Lam, S. Yiu, K. Kristiansen, and J. Wang, "SOAP2: an improved ultrafast tool for short read alignment," *Bioinformatics,* vol. 25, p. 1966, 2009.
[20] V. Topac, "Efficient fuzzy search enabled hash map," 2010, pp. 39-44.
[21] J. Gosling, B. Joy, G. Steele, and G. Bracha, *Java (TM) Language Specification, The (Java (Addison-Wesley))*: Addison-Wesley Professional, 2005.
[22] R. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal,* vol. 29, pp. 147-160, 1950.
[23] A. Bookstein, S. Tomi Klein, and T. Raita, "Fuzzy Hamming Distance: A New Dissimilarity Measure (Extended Abstract)," 2001, pp. 86-97.
[24] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," 1966.
[25] J. Butler, I. MacCallum, M. Kleber, I. Shlyakhter, M. Belmonte, E. Lander, C. Nusbaum, and D. Jaffe, "ALLPATHS: De novo assembly of whole-genome shotgun microreads," *Genome Research,* vol. 18, p. 810, 2008.
[26] J. Simpson, K. Wong, S. Jackman, J. Schein, S. Jones, and Birol, "ABySS: A parallel assembler for short read sequence data," *Genome Research,* vol. 19, p. 1117, 2009.