# Approximate *k*-mer Matching using Fuzzy Hash Maps

John Healy and Desmond Chambers

**Abstract**— We present a fuzzy technique for approximate *k*-mer matching that combines the speed of hashing with the sensitivity of dynamic programming. Our approach exploits the collision detection mechanism used by hash maps, unifying the two phases of "seed and extend" into a single operation that executes in close to $O(1)$ average time.

**Index Terms**— Biology and genetics, Fuzzy set.

## 1 Introduction

THE utilisation and exploitation of *k*-mers is a long established motif in bioinformatics, with *k*-mer centric techniques used, among other things, to seed sequence alignments [1], screen sequence errors [2], identify repetitive sequences [3] and model genome assembly [4]. A *k*-mer is a sequence of $k$ consecutive bases, with *k*-mer $s$ adjacent to *k*-mer $t$ if there is a $(k + 1)$-mer in a sequence whose first and last $k$ bases are $s$ and $t$ respectively [5].

Despite their importance in modelling genome assembly, the ubiquity of *k*-mer centric techniques in bioinformatics is motivated primarily by the prohibitive space and time complexity of using dynamic programming to align either long or large numbers of biological sequences. Although the problem of finding an optimal pairwise alignment of two sequences was solved by Smith and Waterman [6], their algorithm has a quadratic space and time complexity, rendering it unfeasible for use with large data sets. Decomposing a sequence into a tiling of *k*-mers is a viable alternative alignment mechanism, as sequences with high similarity must share *k*-mers in their overlapping regions [7]. Moreover, the short length of *k*-mer sequences facilitates their exploitation in efficient hash-based data structures, vastly reducing the computational cost of alignment and assembly.

As *k*-mers represent $k$ consecutive characters in a sequence, their utility is limited to exact string-matching techniques. In the absence of a mechanism for approximate string matching, exact *k*-mer alignment represents a trade-off between speed and sensitivity. This compromise is controlled by the *k*-mer size, with smaller sizes of $k$ increasing the possibility of detecting a local alignment, but also increasing the number of spurious matches. If the size of $k$ is too large, an alignment of two sequences will miss high-scoring matches that do not have $k$ consecutive characters. Furthermore, while larger values of $k$ decrease access time to hash-based dictionary structures, smaller *k*-mer sizes increase the number of hash collisions, resulting in a subsequent escalation in the time complexity of search operations.

To circumvent the loss of sensitivity arising from the constraint of exact *k*-mer matching, hashing techniques can be augmented with dynamic programming algorithms to improve sequence alignment. Known as "seed and extend", this approach uses fast exact-matching data structures to identify regions of $k$ similarity, which can then be extended using approximate string matching algorithms. The "seed and extend" paradigm has a long history in sequence alignment and underlies the alignment strategies used by BLAST [8], BLAT [9] and, more recently, by Mosaik [10].

Despite the longevity and success of the "seed and extend" approach with Sanger sequences, the advent of second generation sequencing (SGS) technologies lead to a reappraisal of *k*-mer alignment techniques. While SGS sequences provide ample information to seed an alignment, their short length precludes the employment of an extension phase to refine and filter seeded regions. In a seminal work, Ma *et al* [11] noted that alignment sensitivity could be increased by seeding regions of high similarity with non-consecutive $k$ matches, called patterns or spaced seeds. Spaced seeds are binary strings, the weight of which is determined by the number of 1's in the seed. Spaced seeds are analogous to masks, with 1's corresponding to a required match and 0's indicating a "don't care" position. The spaced seed model was later extended to support multiple spaced seeds [12], allowing even greater sensitivity during the seeding phase of alignment. Although they provide higher sensitivity without a loss in specificity, the patterns applied in conventional spaced seeds assume only polymorphic mutations between homologous sequences or that indels are widely spaced. These limitations of the spaced seed model were addressed by Noé and Kucherov [13], who suggested the use of transition-constrained seeds to accommodate polymorphisms at 1's positions and later by Mak *et al* [14] who proposed an indel-spaced seed based on a four-character alphabet.

In recent times, there has been a proliferation of alignment tools based on the exploitation of multiple spaced

- *John Healy, Dept. of Computing & Mathematics, Galway-Mayo Institute of Technology, Galway, Ireland. E-mail: john.healy@gmit.ie*
- *Desmond Chambers, Dept. of Information Technology, National University of Ireland Galway, Ireland. E-mail: des.chambers@nuigalway.ie*

seeds [15]. Spaced seed aligners typically use hash-based data structures to build an index from either a set of sequences or a reference genome [16]. Despite the success of this approach with SGS data, spaced seed aligners do not perform well for sequence lengths ≥200bp, and permit very few mismatches, typically ≤ 2 nucleotides [17]. In addition, like exact consecutive *k*-mer matching techniques, the conventional spaced seeds used by most aligners do not permit gaps, limiting the technique in the presence of transpositions and indels.

Fuzzy *k*-mers combine the speed of hashing with the sensitivity of dynamic programming algorithms, permitting rapid and accurate alignments, even in the presence of indels and polymorphisms. This paper provides a more detailed analysis of fuzzy *k*-mers previously described by the authors [18, 19] and expands the discussion to address the impact of the approach on alignment speed, sensitivity and specificity. The remainder of this paper includes a detailed presentation of the fuzzy *k*-mer model in the next section. This is followed by a description of a prototype aligner developed to test the model. The results of tests on the speed, sensitivity and specificity of the approach are then presented and discussed.

## 2 Fuzzy k-mers

One of the most important facets of *k*-mers, and one that is heavily exploited in sequence alignment, is their suitability as keys in a hash table or hash map. Given a uniform and random distribution of keys in a hash map, operations for insertion, search and deletion execute in $O(1)$ average time [20]. In a chained hash map, a hashing function is used to compute a bucket index from a search key, with each bucket index containing a pointer to a linked list. Determining a bucket index from a hash key is a two-step operation, requiring the computation of an integer value from an arbitrary key type and then transforming that value into an index in the range $[0..m-1]$, where $m$ is the number of buckets in a map. An integer value is typically computed from a string $S$ of length $n$ using the 16-bit Unicode value of each character at each index position in the string:

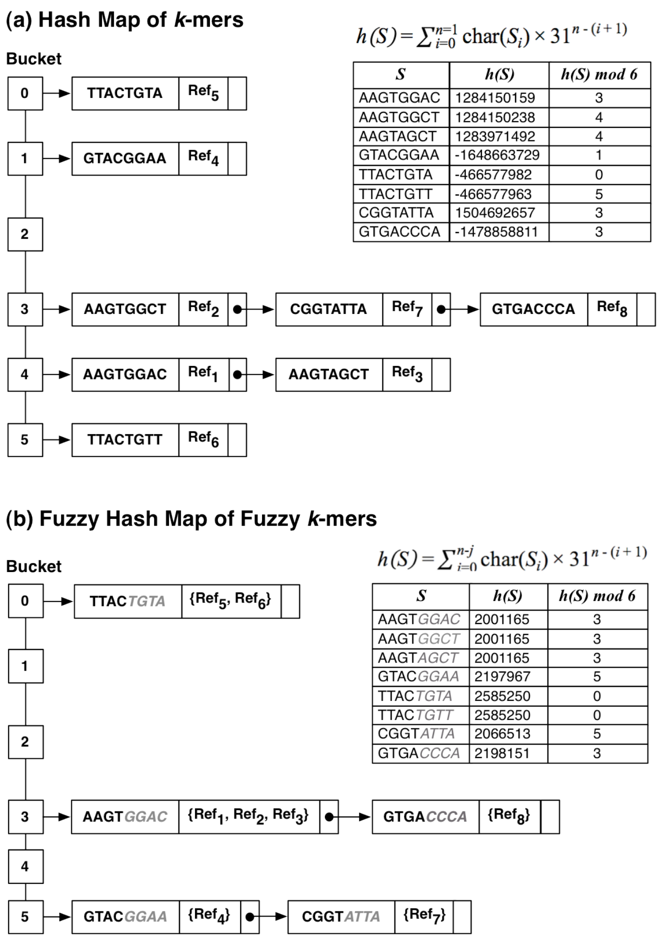
$$h(S) = \sum_{i=0}^{n-1} \mathrm{char}(S_i) \times 31^{n-(i+1)} \quad (1)$$

The 32-bit signed integer returned by (1) can then be translated into a bucket index of a hash map $T$ using the division method, i.e. $T[S] = h(S) \bmod m$.

During an insertion operation, a collision occurs if two keys hash to the same bucket index. Although a good hashing function will minimise their occurrence, collisions can be resolved by adding the new key to the head of the linked list at a bucket index. A search of a hash map operates in a similar manner, with the hashing function providing $O(1)$ time access to a bucket index, followed by the time it takes to search the linked list at that index for a hash key.

The *k*-spectrum for a reference genome $G$ is the set $G^k = \{G[i : i + k - 1] \mid 0 \le i < |G| - k + 1\}$, where $G[i : i + k - 1]$ denotes, for a constant $k$, the substring of consecutive characters in $G$ from index $i$ to index $i + k - 1$. A hash map, $T$, can be constructed from $G$ by inserting each $G[i : i + k - 1]$ in $G^k$ as a key in $T$, along with its positional information as a value (Fig. 1(a)). The average number of *k*-mers stored in the linked list at an index in $T$ is the load factor $\alpha = |G^k|/m$, where $m$ is the capacity or number of buckets in the hash map. In a conventional hash map, under the assumption of simple uniform hashing, a total of $\Theta(1 + \alpha)$ time is required for an insertion or search operation [20]. Clearly therefore, a search at a bucket index that is not empty will require a minimum of two tests for equality – one test based on the hash of a search key and one or more tests against the existing keys in the linked list.

The fuzzy *k*-mer approach is based on the work of Topac [21], who showed how fuzzy operations can be performed on a hash map by manipulating this two-step mechanism. Instead of attempting to design a hashing function to avoid possible collisions, the fuzzy approach deliberately encourages controlled collisions in a hash map, by allowing only part of the search key to be used



$$h(S) = \sum_{i=0}^{n-1} \mathrm{char}(S_i) \times 31^{n-(i+1)}$$

| *S* | *h(S)* | *h(S) mod 6* |
|---|---|---|
| AAGTGGAC | 1284150159 | 3 |
| AAGTGGCT | 1284150238 | 4 |
| AAGTAGCT | 1283971492 | 4 |
| GTACGGAA | -1648663729 | 1 |
| TTACTGTA | -466577982 | 0 |
| TTACTGTT | -466577963 | 5 |
| CGGTATTA | 1504692657 | 3 |
| GTGACCCA | -1478858811 | 3 |

$$h(S) = \sum_{i=0}^{n-j} \mathrm{char}(S_i) \times 31^{n-(i+1)}$$

| *S* | *h(S)* | *h(S) mod 6* |
|---|---|---|
| AAGT*GGAC* | 2001165 | 3 |
| AAGT*GGCT* | 2001165 | 3 |
| AAGT*AGCT* | 2001165 | 3 |
| GTAC*GGAA* | 2197967 | 5 |
| TTAC*TGTA* | 2585250 | 0 |
| TTAC*TGTT* | 2585250 | 0 |
| CGGT*ATTA* | 2066513 | 5 |
| GTGA*CCCA* | 2198151 | 3 |

Fig. 1. (a) A conventional hash table created from a set of 8-mers. Each of the 6 bucket indices available contains a linked list to which *k*-mers are added as hash keys. Each *k*-mer maps to some satellite data, in this case an index position in a reference genome. At buckets 3 and 4, a hash collision has resulted in more than one key being added to the bucket's linked list. (b) A fuzzy hash map created from the same 8-mers. Buckets 1, 2 and 4 are empty. A hash is computed only on the first four bases in the hash key, with the remaining bases allowing variability. The satellite data attached to each fuzzy hash key is a fuzzy set containing approximately matching *k*-mers.

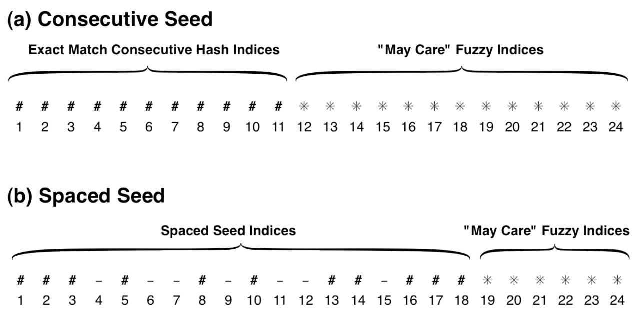


Fig. 2. Fuzzy seeds can extend existing consecutive and spaced-seed models with “may care” indices to augment “must match” or “don't care” positions. The “must match” positions are denoted by the “#” symbol, “don't care” positions by the “-” symbol and fuzzy indices by an asterisk (*). The degree of similarity in “may care” fuzzy indices is computed using an approximate string-matching algorithm.

by a hashing function and permitting a degree of variability in the remainder of the key. Unlike conventional maps, that provide a surjective mapping of keys to values, fuzzy hash maps relate fuzzy hash keys to fuzzy sets, the latter of which are characterised by a fuzzy set membership function μ(). For a *k*-mer $S \in G^k$, we can define a fuzzy hash map $M{:}f(S) \rightarrow A$, where $f(S)$ is a fuzzy hash key operation on $S$ and $A$ is a fuzzy set with a membership function of $\mu_A(S)$. In contrast with the dichotomy implicit in Boolean operations on crisp sets, fuzzy sets are characterised by a continuum of real numbers between 0 and 1 that describe the degree of set membership [22]. For a fuzzy hash map $M$, the degree of membership of the fuzzy set $A$ is governed by the fuzzy set membership function, $\mu_A(S) \rightarrow [0..1]$.

A fuzzy *k*-mer or fuzzy seed may be defined as $f(S) = h(S) \wedge \mu_A(S)$, where $h(S)$ is the subsequence of $S$ used to compute a hash code and $\mu_A(S)$ is a membership function on a substring of $S$ that permits a degree of variability (Fig.1 (b)). The function $h(S)$ maps a fuzzy hash key $f(S)$ to the linked list of keys at a bucket index, with the membership function $\mu_A(S)$ used to evaluate the degree of membership of the fuzzy set associated with a key. The essence of the fuzzy *k*-mer approach is to encourage controlled collisions on a subsequence of a *k*-mer and to use a string similarity algorithm to implement the fuzzy set membership function. This approach has the effect of grouping together approximately matching *k*-mers into a single fuzzy set, allowing the fuzzy set of *k*-mers to be accessed in a single search operation. During an insertion or a search operation, a collision will occur in a fuzzy hash map $M$ when:

$$\exists\, A \in M[h(S)] \mid \#A > 0 \qquad (3)$$

As fuzzy sets permit elements to have a membership degree of zero, a cut-off threshold, $\beta \le \mu_A(S) \le 1$, can be applied to eliminate elements below the threshold limit. By applying a β cut-off threshold, a fuzzy set of matches will be detected for a *k*-mer $S$ during an insertion or search operation where:

$$\exists\, A \in M[h(S)] \mid \mu_A(S) \rightarrow [0, 1],\ \beta \le \mu_A(S) \le 1 \qquad (3)$$

As depicted in Fig. 2, the fuzzy *k*-mer approach permits the use of different seed models to specify the hash indices to use in $h(S)$ and different string similarity algorithms for computing the membership function of the fuzzy set $A$. Specifying a β cut-off threshold of zero for a fuzzy seed will return all exact matching *k*-mers for a given set of hash indices, performing no better than a convention consecutive or spaced seed. This is logically equivalent to extending a seed with an additional set of “don't care” indices. Increasing the β cut-off threshold has the effect of concentrating the fuzzy set $A$, requiring a higher fuzzy membership value from the function $\mu_A(S)$. This alters the semantics of the fuzzy component of a *k*-mer from “don't care” to “may care” indices. At the other extreme, setting the β cut-off threshold to 1 will require an exact match of all indices in a *k*-mer. The β cut-off threshold therefore controls both the sensitivity and specificity of a match, with the speed governed primarily by the number of hash indices.

The fuzzy set membership function, $\mu_A(S)$, can be implemented using any approximate string matching algorithm. For example, for the Levenshtein [23] distance $d$ computed using a match score of +1 and a mismatch score of 0, the membership function $\mu_A(S) = 1 - d/|S_{ij}|$ will return a fuzzy measure of similarity for a substring of the *k*-mer $S$, between indices $i$ and $j$, for a fuzzy set $A$.

Note that no change to the underlying implementation of a regular hash map is required to accommodate the fuzzy behaviour. This enables the fuzzy approach to retain the $O(1)$ average running time of the basic map operations and the space complexity of conventional maps. It also allows fuzzy hash maps to be stored and retrieved like conventional hash maps. Assuming a dynamic programming algorithm is used to compute $\mu_A(S)$, the total running time for a *k*-mer search is $\Theta(1 + \alpha(|S_{ij}|^2))$, comprised of a constant time operation to compute a bucket index using a hash function, followed by the load factor of the map multiplied by the time complexity of the algorithm. Using this technique, the number of hash collisions is controlled by the number of indices in $S$ used by $h(S)$, with the number of fuzzy sets in the linked list at each bucket controlled by the β cut-off threshold. Once a fuzzy hash map has been constructed from $G^k$, the running time required to align a tiling of *k*-mers from a query sequence will be proportional to the length of the query sequence and the size of $k$, but will remain constant with respect to the number of hash keys in the map.

## 3 Prototype Implementation

We developed a prototype sequence aligner in Java that uses the fuzzy *k*-mer approach to align both Sanger and SGS sequences against a reference genome. While the mechanics for manipulating collision detection in hash structures vary between programming languages, collisions are resolved in Java by the semantics of object equality, as implemented in the hashCode() and equals() methods [24] common to all objects in the language. The

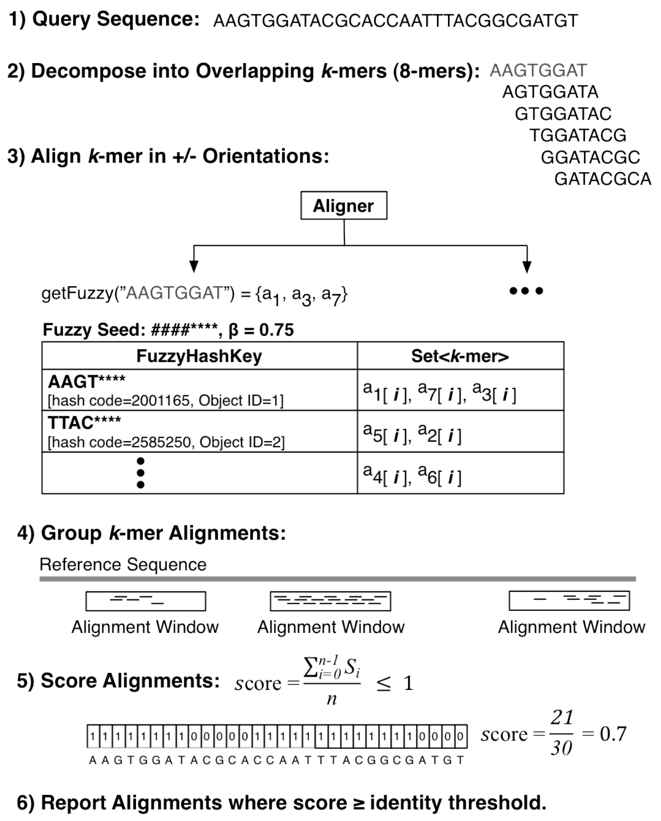

1) Query Sequence: AAGTGGATACGCACCAATTTACGGCGATGT
2) Decompose into Overlapping k-mers (8-mers): AAGTGGAT AGTGGATA GTGGATAC TGGATACG GGATACGC GATACGCA
3) Align k-mer in +/- Orientations:
Aligner
getFuzzy("AAGTGGAT") = {a1, a3, a7}
Fuzzy Seed: ####****, β = 0.75
FuzzyHashKey
Set<k-mer>
AAGT**** [hash code=2001165, Object ID=1]
a1[ i ], a7[ i ], a3[ i ]
TTAC**** [hash code=2585250, Object ID=2]
a5[ i ], a2[ i ]
a4[ i ], a6[ i ]
4) Group k-mer Alignments:
Reference Sequence
Alignment Window
Alignment Window
Alignment Window
5) Score Alignments:
score = (∑ S_i, i=0..n-1) / n ≤ 1
111111110000001111111111111110000
score = 21/30 = 0.7
AAGTGGATACGCACCAATTTACGGCGATGT
6) Report Alignments where score ≥ identity threshold.

Fig. 3. Overview of the alignment process. Query sequences are decomposed into a tiling of *k*-mers and then aligned using one or more fuzzy *k*-mer seeds.

hashCode() method returns a 32-bit signed integer that is translated into a bucket index using the division method [20], i.e. $h(S)$ = hashCode() mod $m$, where $m$ is the capacity or number of buckets in the hash map. When searching a hash map for a given key, if two hashCode() methods return the same integer, a collision is detected if the linked list at that bucket index is not empty. The collision is then resolved by executing the equals() method of the search key against each key in the linked list.

Fuzzy seeds can be defined declaratively, by specifying a seed pattern, a string matching algorithm and the β cut-off threshold for matches. For example, the 24-mer consecutive seed ###########************* computes a bucket index from the first 11 characters in a 24-mer and uses the remaining 13 characters to permit a degree of variability. The hash indices of seeds are not required to be consecutive. The seed ######*************##### generates a hash integer from the 6 prefix and 5 suffix characters of a 24-mer and computes the degree of fuzzy similarity from the middle 13 characters. Fuzzy seeds can also be created from existing spaced-seed patterns. For example, the 33-mer fuzzy spaced-seed ###-#--#-#--##-###************* extends the original spaced-seed used by PatternHunter [11] with 13 fuzzy indices. In this formulation of a fuzzy *k*-mer, the spaced-seed contains both "don't care" and "may care" indices, the latter offering the potential to redress the problem of low specificity in a single spaced-seed. For larger values of *k*, this is effectively a "spaced-seed and extend" approach, combining the two phases of alignment into a single unified operation.

A number of different string similarity algorithms were included in the prototype for computing the membership function $\mu_A(S)$. The Hamming [25] and Levenshtein [23] distance implementations use the formula $\mu_A(S) = 1 - d/|S_{ij}|$ to convert the edit distance $d$ of substring $S_{ij}$ into a real number between 0..1. An implementation of the Smith-Waterman [6] algorithm for local sequence alignment was also used. The implementation computes $\mu_A(S)$ = score$/|S_{ij}| * 2$, using a match score of +2, a mismatch score of -1 and a gap penalty of -1. As the maximum score in the Smith-Waterman dynamic programming matrix is sufficient for the computation of a fuzzy measure of string similarity, the trace-back step of the algorithm was omitted, providing a slight decrease in running time. The prototype can be configured with multiple fuzzy seeds, consecutive, spaced or mixed, for a given *k*-mer size and allows different algorithms and β cutoff thresholds for different seeds.

The prototype aligns two sequences by first generating a tiling of *k*-mers from a reference genome and then inserting each *k*-mer into a fuzzy hash map, along with its positional information, using the specified fuzzy seeds. As *k*-mers are added to the fuzzy hash map, collisions are resolved using the equals() method, resulting in approximately matching *k*-mers being added to the fuzzy set associated with a matching seed. Consequently, as illustrated in Fig. 3, the constructed fuzzy hash map will consist of a set of fuzzy hash keys that map to fuzzy sets of index positions in a reference genome.

After adding a reference genome to the fuzzy hash map, each query sequence is decomposed into a tiling of *k*-mers. The fuzzy hash map is then searched for approximate *k*-mer matches, using each *k*-mer as a search key. Once the result set for an alignment has been created, the set of matching points in the reference genome is further processed to remove spurious or weak matches, by applying an alignment window to the set of matches for a query sequence. The alignment window imposes distance constraints, computed from the alignment indices, the query sequence length and the length of the reference sequence. This has the effect of grouping together both overlapping and non-overlapping *k*-mer matches that fall within the alignment window. As depicted in Fig. 3, each of these groups is represented as a binary array, with 1's corresponding to matching positions and 0's to mismatches. An overall alignment score is then computed for each group by dividing the sum of all the 1's in the binary array by the length of the query sequence. Finally, user-defined fuzzy value, corresponding to a percentage identity constraint, is applied, with alignments scoring under the threshold being discarded.

## 4 Results and Discussion

The results presented in this section were compiled from executing the prototype on a Java HotSpot 1.6 64-bit virtual machine on an OSX 10.6.8 platform, with a single 3.2 GHz Intel Core i3 processor and 16GB of RAM.

## 4.1 Speed Comparison with Exact-match Seeds

To evaluate the overall impact on running time, of using fuzzy *k*-mers in place of exact matching seeds, we measured the load times and the multiplicity of hash keys generated from adding a set of whole genomes to a fuzzy hash map, a regular hash map and a tree map. In contrast with a fuzzy map, a hash map and a tree map use exact matching *k*-mers for insertion and search operations. While the running times of these operations averages $O(1)$ in a regular hash map, a tree map guarantees $O(\log(n))$ time for insertion and search operations, by storing the set of map keys in a binary search tree [26]. The results of tests using the 1.95Mbp genome of B.*suis* ATCC are shown in Table 1 and depicted in Fig 4.

Using an exact matching 24-mer seed resulted in the generation of $1.91\text{x}10^6$ keys in both a hash map and a tree map. This figure represents an approximation of the *k*-spectrum set of the B.*suis* ATCC genome, as hash collisions will result in the bucket address of a hash key containing more than one *k*-mer. As illustrated in the Fig 4, a 24-mer fuzzy seed generates fewer keys, with the number decreasing rapidly as the hash size is reduced. The shaded area above the fuzzy seed in Fig 4 represents the increased sensitivity of a fuzzy 24-mer seed over an exact 24-mer seed, in the window of hash sizes from 10 to 13 bases. The increased sensitivity is due to approximately matching *k*-mers being added to the fuzzy set of an existing fuzzy hash key. Consistent with exact matching seeds, fuzzy *k*-mer operations are performed on discrete chunks of *k*-sized sequences, with the fuzzy approach permitting some variability in *k*-mer content. The shaded area below the fuzzy seed corresponds to the level of increased specificity over an exact match seed with the same number of hash indices. As the complexity of a *k*-mer increases in the order of $4^k$, an exact matching *k*-mer at small hash sizes will generate a high number of collisions and an accompanying high number of candidate matches. It is precisely because it is more specific, that the fuzzy approach produces more keys than an exact-matching seed with the same number of hash indices.

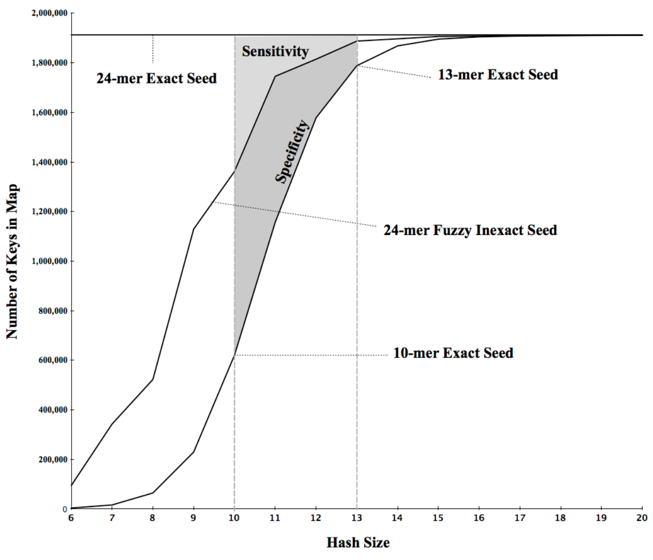


Fig. 4. The multiplicity of hash keys produced for the genome of B.suis ATCC at different hash sizes, using a β-cutoff threshold of 0.5. Fuzzy seeds produce a larger number of hash keys than an exact seed with the same number of hash indices. This enables fuzzy *k*-mers to achieve a higher level of specificity than an exact seed without reducing sensitivity.

The load times listed in Table 1 show a large disparity in running times between fuzzy and exact *k*-mer keys when the number of hash indices is small. At low hash sizes, the increased load time of the fuzzy approach is due to the necessity of executing an approximate string-matching algorithm to determine the equality between keys after an initial hash collision. At very low hash sizes, the linked list at each bucket index will contain a large number of fuzzy hash keys. A subsequent insertion into a fuzzy hash map will therefore require the insertion key to be compared to a relatively large number of existing keys, with an accompanying degradation in running time. This effect is clearly accentuated at small hash sizes, as the running time recovers rapidly when the hash size is increased, matching the $O(\log(n))$ time of a tree map.

TABLE 1
COMPARISON OF LOAD TIMES AND KEY NUMBER FOR B. *SUIS* ATCC

| | Fuzzy Map | | | | Hash Map | | Tree Map | |
|---|---|---|---|---|---|---|---|---|
| Hash | Fuzzy Seed | Keys | Time (s) | Exact Seed | Time (s) | Keys | Time (s) | Keys |
| 20 | ####################**** | 1911513 | 5 | #################### | 2 | 1911433 | 7 | 1911433 |
| 19 | ###################***** | 1911248 | 5 | ################### | 2 | 1910797 | 7 | 1910797 |
| 18 | ##################****** | 1910430 | 5 | ################## | 2 | 1909953 | 7 | 1909953 |
| 17 | #################******* | 1909981 | 5 | ################# | 2 | 1908451 | 7 | 1908451 |
| 16 | ################******** | 1908198 | 5 | ################ | 2 | 1905056 | 7 | 1905056 |
| 15 | ###############********* | 1906894 | 5 | ############### | 2 | 1895820 | 7 | 1895820 |
| 14 | ##############********** | 1898160 | 5 | ############## | 2 | 1868607 | 7 | 1868607 |
| 13 | #############*********** | 1890294 | 5 | ############# | 2 | 1788643 | 7 | 1788643 |
| 12 | ############************ | 1823537 | 7 | ############ | 2 | 1578407 | 7 | 1578407 |
| 11 | ###########************* | 1763727 | 13 | ########### | 2 | 1157492 | 6 | 1157492 |
| 10 | ##########************** | 1400260 | 23 | ########## | 2 | 620488 | 5 | 620488 |
| 9 | #########*************** | 1183447 | 50 | ######### | 1 | 229749 | 4 | 229749 |
| 8 | ########**************** | 562460 | 62 | ######## | 1 | 64658 | 3 | 64658 |
| 7 | #######***************** | 378694 | 125 | ####### | 1 | 16379 | 2 | 16379 |
| 6 | ######****************** | 105895 | 107 | ###### | 1 | 4096 | 1 | 4096 |

*The Levenshtein distance algorithm, with a β-cutoff threshold of 0.5, was used to compute $\mu_A(S)$ for the fuzzy alignments.*

TABLE 2
COMPARISON OF STRING SIMILARITY ALGORITHMS FOR FOR B. *SUIS* ATCC

| | | Hamming | | | Levenshtein | | | Smith Waterman | | | Exact Seed | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Hash** | **Seed** | **Keys** | **α'** | **T (s)** | **Keys** | **α'** | **T(s)** | **Keys** | **α'** | **T (s)** | **Keys** | **α'** | **T (s)** |
| 20 | ####################**** | 1911557 | 1.01 | 4 | 1911513 | 1.01 | 6 | 1911610 | 1.01 | 6 | 1911433 | 1.01 | 2 |
| 19 | ###################***** | 1911298 | 1.01 | 4 | 1911248 | 1.01 | 7 | 1911218 | 1.01 | 7 | 1910797 | 1.01 | 2 |
| 18 | ##################****** | 1910559 | 1.01 | 4 | 1910430 | 1.01 | 7 | 1910494 | 1.01 | 7 | 1909953 | 1.01 | 2 |
| 17 | #################******* | 1910157 | 1.01 | 4 | 1909981 | 1.01 | 8 | 1909878 | 1.01 | 8 | 1908451 | 1.01 | 2 |
| 16 | ################******** | 1908783 | 1.01 | 4 | 1908198 | 1.01 | 8 | 1908703 | 1.01 | 9 | 1905056 | 1.01 | 2 |
| 15 | ###############********* | 1907896 | 1.01 | 4 | 1906894 | 1.01 | 9 | 1905962 | 1.01 | 9 | 1895820 | 1.01 | 2 |
| 14 | ##############********** | 1902996 | 1.01 | 4 | 1898160 | 1.01 | 10 | 1900685 | 1.01 | 10 | 1868607 | 1.03 | 2 |
| 13 | #############*********** | 1899385 | 1.01 | 5 | 1890294 | 1.02 | 11 | 1886874 | 1.02 | 12 | 1788643 | 1.08 | 2 |
| 12 | ############************ | 1870331 | 1.03 | 5 | 1823537 | 1.05 | 16 | 1844425 | 1.04 | 16 | 1578407 | 1.22 | 2 |
| 11 | ###########************* | 1847924 | 1.04 | 7 | 1763727 | 1.09 | 25 | 1747961 | 1.10 | 26 | 1157492 | 1.66 | 3 |
| 10 | ##########************** | 1678528 | 1.15 | 12 | 1400260 | 1.37 | 46 | 1546880 | 1.24 | 55 | 620488 | 3.10 | 4 |
| 9 | #########*************** | 1570251 | 1.23 | 25 | 1183447 | 1.63 | 105 | 1187530 | 1.62 | 110 | 229749 | 8.37 | 9 |
| 8 | ########**************** | 1055447 | 1.82 | 38 | 562460 | 3.42 | 139 | 782960 | 2.46 | 223 | 64658 | 29.75 | 19 |
| 7 | #######***************** | 852880 | 2.26 | 106 | 378694 | 5.08 | 298 | 437661 | 4.40 | 340 | 16379 | 117.45 | 37 |
| 6 | ######****************** | 344437 | 5.59 | 104 | 105895 | 18.17 | 300 | 200846 | 9.58 | 485 | 4096 | 469.67 | 77 |

*$\alpha'$ = effective load factor, T = time. A $\beta$-cutoff threshold of 0.5 was used for all fuzzy alignments.*

## 4.2 Computation Burden of Approximate String Matching

Given its centrality to the fuzzy *k*-mer approach, an evaluation of the computational overhead associated with the use of an approximate string-matching algorithm for a *k*-mer search is essential. The most significant factor that impacts the running time of map operations is the effective load factor of a map, $\alpha' = |G^k|/m$, where $m$ is the number of keys. The effective load factor, $\alpha'$, represents the average number of *k*-mers in the linked list at a non-empty bucket index. An increase in the effective load factor corresponds to an increase in the number of hash collisions in a map and degrades the average $O(1)$ running time of map operations.

To assess this aspect of fuzzy *k*-mers, we aligned a set of whole genomes and computed the number of keys generated, along with the load and alignment times using different string similarity algorithms. The prototype was configured with different approximate string-matching algorithms and a 24-mer seed, for a range of hash sizes decreasing from 20 to 6 hash indices. The β cut-off threshold was held constant at 0.5 for all alignments. For the test with an exact-matching seed, the implementation of the equals() method of the fuzzy hash key was held constant, returning a Boolean false for every invocation, regardless of the content of the fuzzy component of a seed. This had the effect of forcing the collision detection mechanism of the hash map to compare each *k*-mer search key against all of the existing keys in the linked list at a bucket index, allowing a measure of the relative running time cost of using different similarity algorithms against a guarenteed $O(1)$ operation. The results of tests for the alignment of B.*suis* 1330 against the genome of B.*suis* ATCC are shown in Table 2.

As the two B.*suis* genomes are highly homologous, the strong sequence similarity should induce a high number of hash collisions for a *k*-mer search, with an accompanying escalation in the invocation of the string similarity algorithm. The tests demonstrate that, regardless of the string similarity algorithm used, the fuzzy approach generates more hash keys than an exact-matching seed, resulting in a reduction in the effective load factor of the map. This effect is accentuated when the number of hash indices in a seed is reduced, as approximately matching *k*-mers will be grouped together as satellite data of a fuzzy hash key. This reduction in load factor has a direct impact on the running time of map operations, as the increase in the number of hash keys reduces the overall number of hash collisions. The reduced load factor also contributes to mitigating the computational overhead of executing a string similarity algorithm for each *k*-mer search operation on the map, as the $\Theta(1 + \alpha'(|S_{ij}|^n))$ running time is controlled by the effective load factor ($\alpha'$), with both the fuzzy subsequence ($S$) of a *k*-mer and the running time ($n$) of the algorithm remaining constant for a given seed.

For hash sizes > 13, there is no significant variation in the running times of any of the dynamic programming algorithms used in the test. Although the alignment running times converge as the hash size increases, the likelihood of locating an exact or approximate *k*-mer match rapidly decreases. The opposite effect can be observed as the hash size is reduced below 11, indicating that, consistent with established best practice, the window from 10 to 13 indices contains the optimal hash size for a *k*-mer seed. The superior alignment time for Hamming distance is not surprising, as the algorithm has a time complexity of $O(n)$ [25], in contrast with the $O(n^2)$ running time of the other measures of string similarity used. While the divergence in running time from that achieved by an exact-matching seed is significant, the use of dynamic programming in a fuzzy *k*-mer does not place an excessive or punitive computation burden on sequence alignment.

## 4.3 Sensitivity and Selectivity

To evaluate the sensitivity and specificity of fuzzy *k*-mers, we extracted 800bp sequences at 10X coverage from a set of bacterial genomes and recorded the index position of each read. We then induced random indels into the synthetic sequences at a rate of 10% and aligned each set of

TABLE 3
EFFECT OF β-CUTOFF THRESHOLD ON B.*SUIS* 1330 ALIGNMENT

| β | Align Time (s) | Sensitivity % | Specificity % | Keys | α' |
|---|---|---|---|---|---|
| 0.9 | 176 | 0.03 | 100.00 | 2089955 | 1.01 |
| 0.8 | 173 | 99.74 | 99.76 | 2087890 | 1.01 |
| 0.7 | 180 | 99.76 | 99.76 | 2085383 | 1.01 |
| 0.6 | 175 | 99.76 | 99.76 | 2077917 | 1.01 |
| 0.5 | 172 | 99.72 | 99.72 | 1984896 | 1.06 |
| 0.4 | 212 | 99.71 | 99.72 | 1880955 | 1.12 |
| 0.3 | 223 | 99.66 | 99.67 | 1778993 | 1.18 |
| 0.2 | 852 | 99.63 | 99.63 | 1717524 | 1.23 |
| 0.1 | 911 | 99.45 | 99.45 | 1698438 | 1.24 |
| Exact 12-mer | 90 | 92.73 | 92.74 | 1696065 | 1.24 |
| Exact 24-mer | 20 | 0.003 | 100.00 | 2092274 | 1.01 |
| BLAT | 302 | 99.38 | 98.67 | | |
| Mosaik | 342 | 99.96 | 100.00 | | |

*α′ = effective load factor. The fuzzy seed ############************ with 12 hash positions and an overall sequence percentage identity of 60% was used. BLAT was configured with the parameters -tileSize=12 -oneOff=1 -minIdentity=60. Mosaik was executed with the switches -hs 12 -mm 80 -mmp 0.40 -mhp 100 -act 24 -bw 51.*

sequences against its original genome. With *a priori* knowledge of the correct index of each read, we were able to accurately evaluate the approach by comparing alignments with their correct position in the original genome. For these tests, we selected only the highest scoring alignment for each sequence. A read alignment was considered a true positive (TP) if its alignment indices mapped exactly to their original position and a false positive (FP) if not. We then computed the level of accuracy as *sensitivity = TP/(TP + FN)* and *specificity = TP/(TP + FP)*.

We selected BLAT [9] and Mosaik [10] to evaluate the fuzzy *k*-mer approach against the "seed and extend" and "spaced-seed and extend" models respectively. The implementation of the "seed and extend" model used in BLAT aligns *k*-mers extracted from sequence reads against a set of non-overlapping *k*-mers from a reference genome. BLAT can accommodate a single polymorphism at the end of a *k*-mer by making separate lookups of a hash table for each terminating base. The Mosaik aligner clusters *k*-mer matches of reads against a hash table of reference positions using spaced seeds, before evaluating each cluster with a full Smith-Waterman alignment.

Table 3 shows the results of the test with sequences from the B.*suis* 1313 genome, using the single 24-mer fuzzy seed, ############************, with 12 hash positions and a requirement for an overall percentage identity match of 60%. The fuzzy set membership function, $\mu_A(S)$, was computed using a variation of the Levenshtein distance algorithm. For this comparison, both BLAT and Mosaik were configured to use 12 hash positions per seed and a similar requirement for a 60% identity match along a query sequence. In general, an increase in the β cut-off threshold results in a corresponding increase in the level of specificity, with no loss of sensitivity below 0.8. Because it permits a full 24-mer approximate match, the fuzzy *k*-mer seed is significantly more sensitive and more specific than an exact matching 12-mer consecutive seed. Once the β cut-off threshold is increased above 0.8, the sensitivity of the fuzzy approach is reduced to that of a full 24-mer exact-matching seed. Despite using a single 24-mer seed, the fuzzy approach is significantly both more sensitive and more specific than the traditional "seed and extend" approach used by BLAT, but less specific than the multiple spaced-seed strategy used by Mosaik.

A further comparison of fuzzy *k*-mer alignments with those produced by BLAT and Mosaik is shown in Table 4. The results confirm the superiority of fuzzy *k*-mer seeds over the exact-matching consecutive seeds used by BLAT and the greater specificity of the multiple spaced-seed approach used by Mosaik. The alignment times also confirm that the execution of a string-similarity algorithm for every fuzzy *k*-mer search does not adversely impact overall running time. The results further demonstrate that the fuzzy approach reduces the effective load factor, enabling faster fuzzy search and insertion operations on a hash map. This is evidenced by the tests with Y.*pestis*, where the average load factor was reduced from 1.31 to 1.09, enabling the overall alignment using fuzzy *k*-mers to be completed in 212 seconds, against the 1123 and 1634 se-

TABLE 4
COMPARISON WITH BLAT AND MOSAIK

| | | | | Fuzzy k-mers | | | | | BLAT | | | Mosaik | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Organism | Size (Mbp) | 12-mer Keys | Load Factor | Fuzzy Keys | Load Factor | Align Time (s) | Sn % | Sp % | Align Time (s) | Sn % | Sp % | Align Time (s) | Sn % | Sp % |
| S.*epidermidis* RP62A | 2.61 | 1889928 | 1.38 | 2324358 | 1.13 | 151 | 98.84 | 98.97 | 431 | 85.62 | 95.04 | 442 | 99.86 | 100.00 |
| E.*coli* 536 | 4.93 | 3678092 | 1.34 | 4583586 | 1.08 | 254 | 99.29 | 99.55 | 546 | 84.56 | 97.11 | 697 | 99.82 | 100.00 |
| S.*aureus* COL | 2.80 | 2021176 | 1.39 | 2514925 | 1.12 | 162 | 99.40 | 99.51 | 412 | 86.43 | 96.58 | 412 | 99.88 | 100.00 |
| Synechococcus sp. WH8109 | 2.11 | 1732851 | 1.22 | 2001675 | 1.06 | 86 | 99.16 | 99.72 | 211 | 83.94 | 97.90 | 310 | 99.67 | 100.00 |
| Y.*pestis* CO92 | 4.65 | 3542946 | 1.31 | 4250475 | 1.09 | 212 | 97.57 | 98.10 | 1123 | 84.79 | 79.78 | 1634 | 99.86 | 100.00 |
| K.*pneumoniae* MGH-78578 | 5.31 | 3532226 | 1.50 | 4678125 | 1.14 | 317 | 99.52 | 99.63 | 1147 | 84.10 | 98.20 | 810 | 99.82 | 100.00 |

*Sn = sensitivity, Sp = specificity. The fuzzy seed ############************ with 12 hash positions, a β-cutoff threshold of 0.5 and an overall sequence percentage identity of 90% was used for all fuzzy alignments. BLAT was configured with -tileSize=12 -oneOff=1 -minIdentity=90 and Mosaik with the parameters -hs 12 -mm 80 -mmp 0.10 -mhp 100 -act 24 -bw 51.*

conds required by BLAT and Mosaik respectively. A similar effect can be observed for the alignment of Synechococcus, where a decrease in the effective load factor from 1.22 to 1.06 reduced the overall alignment time to 86 seconds. The alignment times confirm that the computational burden required for an approximate *k*-mer match is not excessive or punitive, as the difference in running times of the various fuzzy alignments is primarily due to the overhead of processing a larger number of query sequences. This is borne out by a comparison of the results for K.*pneumoniae* and S.*epidermidis*, where a similar effective load factor enabled more than twice the number of query sequences in the former to be aligned in approximately double the alignment time for the latter.

## 5 Conclusion

We have demonstrated a fuzzy approach that enables approximate *k*-mer matching, without significantly impacting on the overall running time of sequence alignment. By manipulating the collision detection mechanism and reducing the effective load factor of a hash map, fuzzy k-mers combine the sensitivity of small *k*-mer seeds with the speed and specificity of larger seeds. The fuzzy *k*-mer approach is flexible and extensible, allowing different seed patterns to be defined with different string similarity algorithms. Given the ubiquity of *k*-mer centric techniques in sequence alignment and genome assembly, the techniques described in this paper have the potential to be exploited for a variety of alternative uses, including read error correction and the accommodation of variability in assembly graphs.

## References

[1] W. Pearson and D. Lipman, "Improved tools for biological sequence comparison," Proceedings of the National Academy of Sciences, vol. 85, p. 2444, 1988.

[2] X. Yang, K. S. Dorman, and S. Aluru, "Reptile: representative tiling for short read error correction," Bioinformatics, vol. 26, pp. 2526-2533, 2010.

[3] X. Li and M. S. Waterman, "Estimating the Repeat Structure and Length of DNA Sequences Using l-Tuples," Genome research, vol. 13, pp. 1916-1922, 2003.

[4] R. Idury and M. Waterman, "A new algorithm for DNA sequence assembly," Journal of Computational Biology, vol. 2, pp. 291-306, 1995.

[5] 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.

[6] T. Smith and M. Waterman, "Identification of common molecular subsequences," Journal of Molecular Biology, vol. 147, pp. 195-197, 1981.

[7] J. R. Miller, S. Koren, and G. Sutton, "Assembly algorithms for next-generation sequencing data," Genomics, vol. 95, pp. 315-327, 2010.

[8] 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.

[9] W. Kent, "BLAT - the BLAST-like alignment tool," Genome Research, vol. 12, p. 656, 2002.

[10] Lee W-P, Stromberg MP, Ward A, Stewart C, Garrison EP, et al. (2014) MOSAIK: A Hash-Based Algorithm for Accurate Next-Generation Sequencing Short-Read Mapping. PLoS ONE 9(3): e90581. doi:10.1371/journal.pone.0090581.

[11] B. Ma, J. Tromp, and M. Li, "PatternHunter: faster and more sensitive homology search," Bioinformatics, vol. 18, p. 440, 2002.

[12] M. Li, B. Ma, D. Kisman, and J. Tromp, "Patternhunter II: highly sensitive and fast homology search," Journal of Bioinformatics and Computational Biology, vol. 2, p. 417, 2004.

[13] L. Noé and G. Kucherov, "YASS: enhancing the sensitivity of DNA similarity search," Nucleic acids research, vol. 33, pp. W540-W543, 2005.

[14] D. Mak, Y. Gelfand, and G. Benson, "Indel seeds for homology search," Bioinformatics, vol. 22, pp. e341-e349, 2006.

[15] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," Briefings in bioinformatics, vol. 11, pp. 473-483, 2010.

[16] P. Flicek and E. Birney, "Sense from sequence reads: methods for alignment and assembly," Nature Methods, vol. 6, pp. S6-S12, 2009.

[17] S. Misra, A. Agrawal, W. Liao, and A. Choudhary, "Anatomy of a hash-based long read sequence mapping algorithm for next generation DNA sequencing," Bioinformatics, vol. 27, pp. 189-195, 2011.

[18] J. Healy and D. Chambers, "Fast and Accurate Genome Anchoring Using Fuzzy Hash Maps," in 5th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2011), 2011, pp. 149-156.

[19] J. Healy and D. Chambers, "De Novo Draft Genome Assembly Using Fuzzy K-mers," in BIOTECHNO 2011, The Third International Conference on Bioinformatics, Biocomputational Systems and Biotechnologies, 2011, pp. 104-109.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms: MIT press, 2001.

[21] V. Topac, "Efficient fuzzy search enabled hash map," 2010, pp. 39-44.

[22] K. Tanaka and T. Niimura, An introduction to fuzzy logic for practical applications: Springer, 1997.

[23] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," 1966.

[24] J. Gosling, B. Joy, G. Steele, and G. Bracha, Java (TM) Language Specification, The (Java (Addison-Wesley)): Addison-Wesley Professional, 2005.

[25] R. Hamming, "Error detecting and error correcting codes," Bell System Technical Journal, vol. 29, pp. 147-160, 1950.

[26] Oracle, "The Java Collections Framework," Java Language Application Programming Interface, 2011.

**John Healy** has been a faculty member at the Department of Mathematics & Computing at the Galway-Mayo Institute of Technology since 2000. He holds a B.Sc., a M.Sc. and a Ph.D. from the National University of Ireland, Galway.

**Desmond Chambers** is a faculty member at the Department of Information Technology, National University of Ireland, Galway. He holds a B.Eng. in Electronic Engineering from the University of Limerick, a M.Sc. in Computer Systems Design from Trinity College Dublin and a Ph.D. from the National University of Ireland, Galway.