Sādhanā (2019) 44:204
https://doi.org/10.1007/s12046-019-1187-9

BREAK, MAKE and TAKE: an information retrieval approach

A PRANAV[1], R RAJESHKANNAN[1,*], V VIJAYARAJAN[1] and V B SURYA PRASATH[2,3,4,5]

[1]School of Computer Science and Engineering, VIT University, Vellore, India
[2]Division of Biomedical Informatics, Cincinnati Children's Hospital Medical Center, Cincinnati, OH 45229, USA
[3]Department of Pediatrics, University of Cincinnati College of Medicine, Cincinnati, OH 45267, USA
[4]Department of Biomedical Informatics, College of Medicine, University of Cincinnati, Cincinnati, OH 45267, USA
[5]Department of Electrical Engineering and Computer Science, University of Cincinnati, Cincinnati, OH 45221, USA
e-mail: cs.pranav.a@gmail.com; rajeshkannan.r@vit.ac.in; vijayarajan.v@vit.ac.in; prasatsa@uc.edu

MS received 18 February 2019; revised 28 June 2019; accepted 9 July 2019

Abstract. Ranking functions used in information retrieval are primarily used in the search engines and they are often adopted for various language processing applications. This paper introduces some novel heuristics combined with probabilistic retrieval functions and are employed in the domain of approximate string similarity problem. Various algorithms have been proposed in the literature to solve approximate string similarity problems; however, none of them makes use of probabilistic retrieval functions. We are the first to explore the intersection between these two areas, that is between string similarity and information retrieval, and propose heuristic designs to resolve this problem. First, we propose chunking heuristic function, called BREAK. We show the variants BREAK-1, -2, -OFF, which split up the terms with the sequential notion. Then we propose BREAK-*n*, which generalizes these variants and scales to larger datasets. In order to relate these split-ups, we propose a graphical error modelling heuristics MAKE over the BREAK variants. Finally, we propose TAKE curve, a novel feature engineering probabilistic distribution, which replaces the prevalent normalization heuristics. Taking the advantage of flexibility over the choice of heuristics, we assess the variants on the cognate detection, mutant identification and problems based on isolated spelling correction. In the extensive evaluation methods, we found that our designs perform better than prevalent heuristics and are robust against database characteristics.

Keywords. Approximate string similarity; cognate detection; information retrieval; mutant identification; isolated-spelling-correction-based problems; probabilistic retrieval functions.

1. Introduction

Approximate string similarity problems deal with returning of a ranked list of closest possible strings against the given query string. These problems require matching patterns, as in string matching, approximately rather than the exact ideal matches. This problem can be posed generally as the following. Given a set of possible strings and a strong example drawn from the set, find a string that approximately matches the drawn one. Some representatives of this problem are

1. **isolated spelling correction**, which involves returning a list of suggestions for a misspelled word [1],
2. **genetic mutant identification**, which is the process of returning a set of possible closest mutations of the given genome sequence [2],
3. **cognate detection**, which deals with identification of the words that have same linguistic derivation [3].

On the other hand, information retrieval (IR) models portray the idea of relevance, so that one can score a document with a given respective query. There are prevailing models like BM25 [4], Dirichlet-prior smoothing [5] and PL2 [6] that are commonly employed mainly in search engine application.

This paper deals with the intersection between these two areas, that is between string similarity and IR, which is largely under-explored in the literature. We show how the notion of retrieval can be incorporated in the approximate string similarity problem by breaking a word into small units. Furthermore, Nguyen *et al* [2] have stated that broken words are more practical to query large databases of

*For correspondence
Published online: 05 September 2019

Table 1. A brief overview of related topics and prevalent methods used in approximate string similarity. Some of these methods are used as baselines in the evaluation section.

Problem domain	Prevalent methods	Source
Isolated spelling correction	Brute-force candidate generation method	[8]
	Weighted longest common subsequence search	[9]
	Usage of subsequences (*s*-grams)	[10] [11]
	Corrections based on finite state automata	[12]
Cognate detection	Using Naive Bayes	[13] [14]
	Using support vector machines (SVM)	[15] [3]
Gene mutant identification	Dot matrix alignment	[16]
	Needleman–Wunsch alignment	[17]
	Smith–Waterman alignment	[18]
	FASTA tool	[19]
	BLAST tool	[20]
	T-COFFEE	[21]

sequences as compared with conventional methods. Additionally, retrieval models provide a variety of alternative heuristics, which can be chosen for the desired application area [7]. Taking advantages of the flexibility of these models, the combination of approximate string similarity operations with IR systems could be beneficial in many cases. The overview of various methods used in approximate string similarity problems is shown in table 1.

1.1 *Related work*

Isolated spelling correction involves returning a list of suggestions for a misspelled word. Some of the prevalent techniques include brute-force candidate generation method [8], weighted longest common subsequence search [9], usage of subsequences [10, 11] and corrections based on finite state automata [12]. Genetic mutant identification is the process of returning a set of possible closest mutations of the given genome sequence. Many biological sequence alignment algorithms such as dot-matrix [16], Needleman–Wunsch [17], Smith–Waterman [18] and tools like FASTA [19], BLAST [20], T-COFFEE [21] are commonly employed for this problem. Cognates are words that exist in different languages but have a common origin. For example, the word *night* in English and *nuit* in French are a pair of cognates, which have a Proto-Germanic origin. Cognate detection is the problem of identification of such pairs within a cross-language corpus. Current approaches based on machine learning include SVM and Naive-Bayes classifiers, which distinguish whether a pair is cognate or not. [3, 13–15].

Probabilistic retrieval functions are used frequently in search engines for returning a ranked list of relevant documents [22]. These functions provide a variety of heuristics, which can be chosen for the desired application area [7]. Taking this advantage of flexibility of these functions, we propose modified versions of these functions for the applications in approximate string similarity problems.

IR deals with the task of *retrieving*, or in simple words, obtaining relevant and necessary information resources from a huge collection of documents. The information obtained is considered relevant according to a piece of information asked about (also known as *query*) [22]. The use of IR has been fundamental for developing search engines. However, there are many other interesting applications such as recommendation systems, spam filtering, plagiarism detection and so on [23]. Ranked IR uses ranking functions that determine the decreasing order of the documents in relevance to the query.

1.2 *System architecture*

The set-up of the project is similar to the IR system. Here, the given strings are processed with our proposed chunking methods. The given chunks are stored in the inverted index in the form of the linked lists. This inverted index is processed with the double-barrel-based cache, which is developed according to the term frequency of the chunks. The query is given and with the help of the ranking function's heapsort, the top relevant documents are collected and displayed to the user.

A lot of algorithms have been proposed in the literature to solve approximate string similarity problem; however, none of them makes use of probabilistic retrieval functions. Nyugen *et al* [2] have stated that word split-ups are more practical to query large databases of sequences as compared with conventional methods. Thus, we believe that using probabilistic retrieval functions [24] can improve solutions obtained in solving the approximate string similarity problems. Thus, combination of approximate string similarity operations with IR systems could be beneficial in these cases.

We organize the rest of the paper as follows. Section 2 introduces our BREAK–MAKE–TAKE approach for string similarity along with the generalizations. Section 3 evaluates our proposed tools in cognate detection, mutant identification and isolated spelling correction. Finally, section 4 concludes the paper.

2. BREAK–MAKE–TAKE – proposed IR approach

2.1 *Proposed method 1 – BREAK*

A typical text-based search engine system tokenizes queries and documents into words and uses heuristics to return a

ranked list of relevant documents to the query. This analogy can be extended towards string-related operations, where instead of dividing a sentence into words, one could divide a string into small chunks.

The k-gram splitting technique illustrates the trivial chunking. The word *pizza* is split with $k = 2$ as $\{\langle s\rangle p, pi, iz, zz, za, a\langle/s\rangle\}$. Here, $\langle s\rangle$ is the start token and $\langle/s\rangle$ is the stop token. For the sake of simplicity, we have ignored terminal tokens in the k-gram splits. Hence, the split set looks like $\{p, pi, iz, zz, za, a\}$. We would label this approach as BREAK-0 because it splits the words into smaller k-grams without any sequential information.

2.1a *BREAK-1: sequencing from one end* We argue that BREAK-0 could lead to an extremely generalized matching of tokens since an expansion set could be visualized as a bag-of-words method. Thus, we propose a positional k-gram splitting technique, BREAK-1, which introduces position number in the splits to incorporate the notion of the sequence of the tokens in the word. For example, the word *pizza* could be position-wise split with $k = 2$ as $\{1p, 2pi, 3iz, 4zz, 5za, 6a\}$. Thus, the member *4zz* simply means that it is the fourth member of the set. The motivation behind this tweak is that it gives us a rough amount of sequential-insight for spelling splits when they are used in probabilistic retrieval ranking functions.

2.1b *BREAK-2: sequencing from two ends* The main disadvantage of BREAK-1 is that some misspellings can easily disturb the order of the set, which leads to low similarity. For example, if the misspelling (query) is *ppizza*, the split set would be $\{1p, 2pp, 3pi, 4iz, 5zz, 6za, 7a\}$. The order of the members after *2pp* is misplaced; thus, this would lead to low similarity with the correct spelling (document) $\{1p, 2pi, 3iz, 4zz, 5za, 6a\}$. Only $\{1p\}$ is common between correct and incorrect spell-splits. Hence, we propose BREAK-2, a split algorithm that is robust against such displacements.

We attach position number to the left if the numbering begins from the start, and to the right if the numbering begins from the end. Then the smallest position number would be selected between the two position numbers. If the position numbers are equal, then we select the left position number as a convention. Figure 1 gives an exemplification of this algorithm illustrated with splits of *pizza* and *hearts*.

If the misspelling is *ppizza*, the double-end split set would be

$$\{1p, 2pp, 3pi, 4iz, zz3, za2, a1\}.$$

Thus it gives higher similarity with the correct spelling (document) as compared with positional split, as the set members $\{1p, zz3, za2, a1\}$ would be matched.

2.1c *BREAK-OFF: sequencing with offsets* We propose another variant, BREAK-*X*-OFF, which introduces offsets

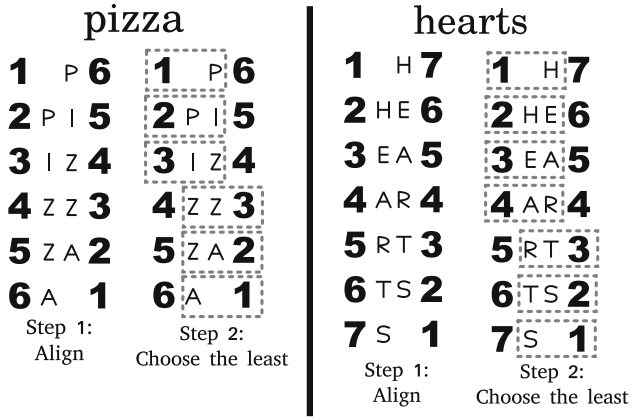

Figure 1. The process of BREAK-2. On the left, algorithm slices pizza into pieces. The output set would be $\{1p, 2pi, 3iz, zz3, za2, a1\}$. On the right, algorithm breaks hearts into pieces. The output set would be $\{1h, 2he, 3ea, 4ar, rt3, ts2, s1\}$.

in the split sets. (Here X stands for the BREAK variant number used). Let the BREAK-1 of pizza be

$$S_0 = \{1p, 2pi, 3iz, 4zz, 5za, 6a\}$$

which has no offsets. An offset of 1 would mean the counts would be displaced by 1, which gives

$$S_1 = \{2p, 3pi, 4iz, 5zz, 6za, 7a\}.$$

An offset of -1 would mean the counts would be displaced by -1, which gives $S_{-1} = \{0p, 1pi, 2iz, 3zz, 4za, 5a\}$. Now the complete split-up would be, thus, $S = S_{-1} \cup S_0 \cup S_1$. Similarly, offsets can be applied to the BREAK-2 splits too.

The motivation behind BREAK-*X*-OFF is that inclusion of offsets would be robust against the incorrect query provided. For example, a spelling typographical error could have been caused by the insertion or deletion of the letter. Such insertions and deletions would displace the position sequences for the chunks. Hence, the key here is to already index these possible errors; thus, for insertion it would have an offset of 1 and for deletion, it would have an offset of -1.

For example, the split set for *pizza*, which was originally

$$\{1p, 2pi, 3iz, 4zz, 5za, 6a\}$$

after adding positional offset of 1 becomes

$$\{2p, 3pi, 4iz, 5zz, 6za, 7a\}.$$

Offset with double-ended split set now gives

$$\{2p, 3pi, 4iz, zz4, za3, a2\}.$$

For deletion errors, positional offset of -1 is included. For example, the split set for *pizza* after adding positional offset of -1 is

$$\{0p, 1pi, 2iz, 3zz, 4za, 5a\}$$

Offset with double-ended split set now looks like

$$\{0p, 1pi, 2iz, zz2, za1, a0\}.$$

The complete split set would include offsets of -1, 0 and 1. Thus, complete split set of *pizza* now is

$$\{0p, 1pi, 2iz, zz2, za1, a0, 1p, 2pi, 3iz, zz3, za2, a1, 2p, 3pi, 4iz, zz4, za3, a2\}.$$

However, set members $\{0p, a0, 2p, a2\}$ are unlikely to happen as the numbering always starts from 1. Hence, we can ignore these members. The final set is now

$$C = \{1pi, 2iz, zz2, za1, 1p, 2pi, 3iz, zz3, za2, a1, 3pi, 4iz, zz4, za3\}$$

where C is the correct split set.

Let the misspelling (query) be entered as *piza*. The I, incorrect split set, by this method would be

$$I = \{1pi, 2iz, za1, 1p, 2pi, 3iz, za2, a1, 3pi, 4iz, za3\}.$$

Then the match $C \cap I$ is

$$C \cap I = \{1pi, 2iz, za1, 1p, 2pi, 3iz, za2, a1, 3pi, 4iz, za3\}.$$

Thus, this method received more number of term matches, $|C \cap I|$, than any other methods. Similarly, this method can be visualized for other types of spelling errors like insertion of letters.

2.2 *BREAK-n: sequential splits for longer terms*

The main drawback of BREAK-2 algorithm is that the double-ended counts would be inefficient for the longer words. With elongation, the advantage of having two extremities would fade away, causing a similar drawback like BREAK-1. This is undesirable for the non-natural language processing tasks like genome sequence analysis where a typical genetic sequence can be thousands of base pairs long. In this section, we tackle this problem by proposing BREAK-*n* algorithm, which would generalize BREAK-2 and is a workaround for longer words.

2.2a Anchoring:

Here, we introduce the notion of the anchor points.

Every $t_{e,ij}$ entry will have information about the distance between i^{th} anchor point and its position j, or simply

$$t_{e,ij} = |pos_i - j|. \quad (1)$$

Here, e stands for an entity that can be either a document d or query q. We have illustrated a basic example in figure 2 to exemplify this process.

2.2b Redefining intersection: Comparing the analogy from BREAK-2, we impose conditional intersection here. A term t_q in query and t_d in document are said to be *completely intersected* if any of their relative positions from anchor points are the same. To achieve this, we would investigate their column vectors at the matched positions.

Let the matching position for t_q and t_d be p_1 and p_2, respectively. We would extract p_1 and p_2 column vectors and check their corresponding indices for every anchor point. Afterwards, we extract p_1 and p_2 column vectors $t_{q,:p_1}$ and $t_{d,:p_2}$, respectively. Then we see if any of their corresponding indices are the same. If this happens, we say that they are *completely intersected*. In other words, any of the cells in the absolute difference between $t_{q,:p_1}$ and $t_{d,:p_2}$ should be 0. Mathematically

$$\prod_{i=1}^{n} |t_{q,ip_1} - t_{d,ip_2}| = 0 \quad (2)$$

where n is the length of the column vector (number of the anchor points). If the resulting product around 0 is achieved, we say that the term intersects completely between query and document.

Continuing the example from figure 2, let us assess the term *za*, which is common between t_q ($p_1 = 3$) and t_d ($p_2 = 4$). The t_q column vector at $p_1 = 3$ is $t_{q,:3} = [3\ 1]^T$ and for t_d at $p_2 = 4$ it is $t_{d,:4} = [4\ 1]^T$. Applying Eq. (2), we get $(|3-4|)(|1-1|) = 0$, which means the term *za* is completely intersected. Analytically, this makes sense because position of *za* from the end is the same in the two entities, and hence it should be completely intersected. Although here we have shown the example of BREAK-*n* for a small word with two anchor points, BREAK-*n* can be logically extended to longer words with more anchor points.

2.3 *Proposed Method 2: MAKE error modelling*

Here we extend our proposed split-up heuristics to graphical error models or the algorithm MAKE (figure 3). The motivation behind this creation of MAKE is to aid the identification and suggestion of solutions to the most probabilistic errors. For example, there are many common errors in the spelling typographical errors: *ie* is commonly mistaken with *ei*, adding or deleting of an extra *l* in *lly* and so on.

Figure 3 provides the process of MAKE. Let Q be the query and D be the document. Thus $Q \cap D$ shows number of terms common between them. We are interested in the leftover terms in the sets. For this, we need to infer a certain pattern from leftover sets, which are $Q - \{Q \cap D\}$ and $D - \{Q \cap D\}$. Thus we can draw mappings to gather information of the corrections.

Let *first* and *second* be the *ordered sets* referring to $Q - \{Q \cap D\}$ and $D - \{Q \cap D\}$, respectively. If *first* or *second* is empty, then we insert an empty token ϕ in them (Lines 2–5 of the algorithm).

Query, q = piza

	p	pi	iz	za	a
From pos_1	$t_{q,11} = 0$	$t_{q,12} = 1$	$t_{q,13} = 2$	$t_{q,14} = 3$	$t_{q,15} = 4$
From pos_2	$t_{q,21} = 4$	$t_{q,22} = 3$	$t_{q,23} = 2$	$t_{q,24} = 1$	$t_{q,25} = 0$

Document, d = pizza

	p	pi	iz	zz	za	a
From pos_1	$t_{d,11} = 0$	$t_{d,12} = 1$	$t_{d,13} = 2$	$t_{d,14} = 3$	$t_{d,15} = 4$	$t_{d,16} = 5$
From pos_2	$t_{d,21} = 5$	$t_{d,22} = 4$	$t_{d,23} = 3$	$t_{d,24} = 2$	$t_{d,25} = 1$	$t_{d,26} = 0$

Figure 2. In this example, the words *piza* and *pizza* are given as q and d, respectively, split into 2-grams. If we chose 2 anchor points, the position according to the equation would be at the first and the last (which are shown in arrows). Now we fill every $t_{e,ij}$ entry by calculating relative distance given in Eq. (1).

Algorithm 1 MAKE algorithm

```
1: procedure MAKE(first, second)
2:     if first.size = 0 then
3:         first.insert(φ)
4:     if second.size = 0 then
5:         second.insert(φ)
6:     while first.size < second.size do
7:         position ← ⌈first.size / 2⌉
8:         first.insert(φ, position)
9:     while first.size > second.size do
10:        position ← ⌊second.size / 2⌋
11:        second.insert(φ, position)
12:    graph = {}
13:    for i ← 0, first.size do
14:        map ← pair(first[i], second[i])
15:        graph.insert(map)
16:    for i ← 0, first.size − 1 do
17:        map ← pair(first[i], second[i + 1])
18:        graph.insert(map)
19:    for i ← 1, first.size do
20:        map ← pair(first[i], second[i − 1])
21:        graph.insert(map)
22:    graph.remove_duplicates()
23:    return graph
```

Figure 3. MAKE algorithm.

If size of *first* is less than size of the *second*, then we keep inserting empty tokens ϕ in the middle of *first*, unless the size of *first* becomes equal to the size of *second* (Lines 6–8). Similarly, if the size of *second* is less than size of the *first*, then we keep inserting empty tokens ϕ in the middle of *second*, unless the size of *second* becomes equal to the size of *first* (Lines 9–11).

We now initialize a hash map *graph* (Line 12). We make pairs from the members of the *first* that correspond to the *second* and insert in the *graph*. We insert pairs of the corresponding same index of *first* and *second* in the *graph* (Lines 13–15). We then insert pairs of the corresponding indexes of *first* and *second* in the *graph*, with one index ahead of others (Lines 16–18). In the same way, we insert pairs of the corresponding indexes of *first* and *second* in the *graph*, with one index before others (Lines 19–21).

After removing the duplicates, we can return the graph as a result. Following examples illustrates the procedure of the proposed approach.

Example 1, deletion error: We visualize *pizza* again. Let the correct split set of *pizza* (ignoring the offsets) be D, which is indexed in the document. Hence

$$D = \{1p, 2pi, 3iz, zz3, za2, a1\}.$$

Let the incorrect split set of *piza* (ignoring the offsets) be Q given as the query. Hence, $Q = \{1p, 2pi, 3iz, za2, a1\}$. Thus the term matches are $Q \cap D = \{1p, 2pi, 3iz, za2, a1\}$.

Now we can analyse the leftover *pizza* sets, which are $D - Q \cap D$ (*first*) and $Q - Q \cap D$ (*second*). Hence, $Q - Q \cap D = \{zz3\}$. Similarly, $D - Q \cap D = \{\phi\}$.

Graph is now constructed from the members of $D - Q \cap D$ to $Q - Q \cap D$. The graph constructed is $\{\phi \rightarrow zz3\}$.

Intuitively, $\phi \rightarrow zz3$ means that the letter z should have been inserted around position 3 from the end.

The query set Q is denoted here as I, which is the incorrect split set, and the document set D is denoted here as C, which is the correct split set.

Example 2, insertion error: For the misspelled word *pizzza*, we have

$$\begin{aligned} C &= \{1p, 2pi, 3iz, zz3, za2, a1\}, \\ I &= \{1p, 2pi, 3iz, 4zz, zz3, za2, a1\}, \\ C \cap I &= \{1p, 2pi, 3iz, zz3, za2, a1\}, \\ C - C \cap I &= \{\phi\}, \\ I - C \cap I &= \{4zz.\} \end{aligned}$$

Thus, the graph constructed is $\{4zz \rightarrow \phi\}$. Intuitively, $4zz \rightarrow \phi$ means that the letter z starting around 4th position should have been deleted.

Example 3, substitution error: For the misspelled word *panc* and the correct word *pant*, we have

$$\begin{aligned} C &= \{1p, 2pa, 3an, nt2, t1\}, \\ I &= \{1p, 2pa, 3an, nc2, c1\}, \\ C \cap I &= \{1p, 2pa, 3an\}, \\ C - C \cap I &= \{nt2, t1\}, \\ I - C \cap I &= \{nc2, c1\}. \end{aligned}$$

Figure 4 shows the set mappings done by the for loop. They correspond to the same index, one index ahead and one index behind from the first set to the second set.

Thus the graph constructed is $\{nc2 \rightarrow nt2, nc2 \rightarrow t1, c1 \rightarrow nt2, c1 \rightarrow t1\}$. Intuitively, this graph depicts that the last letter should have been t.

Example 4, swapping error: For the misspelled word *sieze* and the correct word *seize*, we have

$$\begin{aligned} C &= \{1s, 2se, 3ei, iz3, ze2, e1\}, \\ I &= \{1s, 2si, 3ie, ez3, ze2, e1\}, \\ C \cap I &= \{1s, ze2, e1\}, \\ C - C \cap I &= \{2se, 3ei, iz3\}, \\ I - C \cap I &= \{2si, 3ie, ez3\}. \end{aligned}$$

Thus the graph constructed is $\{2si \rightarrow 2se, 2si \rightarrow 3ei, 3ie \rightarrow 2se, 3ie \rightarrow 3ei, 3ie \rightarrow iz3, ez3 \rightarrow 3ei, ez3 \rightarrow iz3\}$. Thus, the graph neatly captures the idea that the letters i and e should have been swapped, see figure 5.

Example 5, unequal set length: For the misspelled word *pth* and the correct word *path*, we have

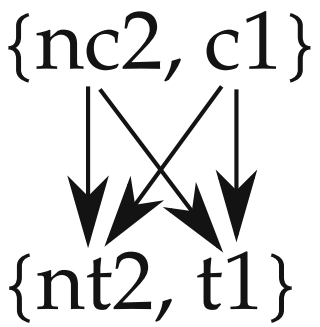

Figure 4. Graphical error model construction from *panc* to *pant*.

Figure 5. Graphical error model construction from *sieze* to *seize*.

$$\begin{aligned} C &= \{1p, 2pa, 3at, th2, h1\}, \\ I &= \{1p, 2pt, th2, h1\}, \\ C \cap I &= \{1p, th2, h1\}, \\ C - C \cap I &= \{2pa, 3at, th2\}, \\ I - C \cap I &= \{2pt\}. \end{aligned}$$

Since the size of topmost set was not equal to that of the bottom one, we inserted empty tokens in the top one until they became of equal sizes. Thus, the graph constructed is $\{2pt \rightarrow 2pa, 2pt \rightarrow 3at, \phi \rightarrow 2pa, \phi \rightarrow 3at\}$. This graph roughly conveys the information that the letter t should be second last, see figure 6.

Example 6, unequal set length: For the misspelled word *patthhs* and the correct word *paths*, we have

$$\begin{aligned} C &= \{1p, 2pa, 3at, th3, hs2, s1\}, \\ I &= \{1p, 2pa, 3at, 4tt, th4, hh3, hs2, s1\}, \\ C \cap I &= \{1p, 2pa, 3at, hs2, s1\} \\ C - C \cap I &= \{th3\}, \\ I - C \cap I &= \{4tt, th4, hh3\}. \end{aligned}$$

Since the size of bottom set was not equal to the top one, we inserted empty tokens in the bottom set at the middle, until they became of equal sizes. Thus, the graph constructed is $\{4tt \rightarrow \phi, th4 \rightarrow \phi, th4 \rightarrow th3, hh3 \rightarrow \phi, hh3 \rightarrow th3\}$. This graph roughly conveys the information that the letter t and letter h should have been deleted, see figure 7.

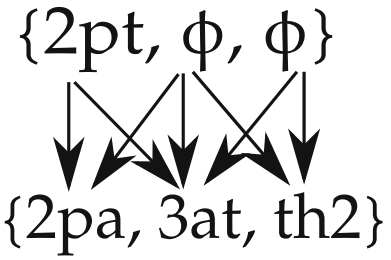

Figure 6. Directed graph guiding from the incorrect *pth* to the correct *path*.

Figure 7. Graphical error model construction.

2.4 *Proposed Method 3: TAKE curve*

The lengths of the misspelled word and correctly spelled word should be almost similar. Usually edit distance between misspelled and correctly spelled word is around 1 or 2. The idea here is to reward the documents more in which document length and query length are closer to each other, and similarly penalize the extremely dissimilar lengths.

Let $|q|$ be the query length and $|d|$ be the document length. Let $h(|q|, |d|)$ be the heuristic function to calculate length similarity. Here we have used two variants, namely power penalization and bucket-shaped sigmoidal function.

- Power penalization
  This uses hyperparameter γ that varies in [0, 1]:

$$h(|q|, |d|) = (||q| - |d|| + 1)^{\gamma}. \tag{3}$$

  The curve is plotted in figure 8.
- Bucket-shaped sigmoidal penalization
  Here, the function would be modelled using Richard's curve, also known as generalized sigmoidal curve [25]. The curve is defined by

$$g(x) = l + \frac{u - l}{(A + e^{-B(x-M)})^{1/v}}. \tag{4}$$

  Here, l is lower limit, u is upper limit and M, v, A, B are free parameters
  As compared with the normal sigmoid function, this function allows more flexibility in the choice of the parameters. Since, we have to model a penalization function, the aim would be to model a "trough" in the curve as $|q|$ approaches $|d|$. We divide this function into three parts:

1. monotonically decreasing Richard's curve $g(x_1)$ when $|d| < |q|$, or $\frac{\partial g(x_1)}{\partial x_1} < 0$;
2. monotonically increasing Richard's curve $g(x_2)$ when $|d| > |q|$, or $\frac{\partial g(x_2)}{\partial x_2} > 0$;
3. value of heuristic function $h(|d|, |q|) = 1$ when $|d| = |q|$
   where $x_1 \in [0, |d|]$, $x_2 \in [|d|, \infty]$ and $x_1, x_2 \subset |q|$. If we solve the limiting derivatives, use binomial approximations and remodel our parameters, we get the following penalization function:

$$h(|d|, |q|) = \begin{cases} 1 + \dfrac{b_1 - 1}{1 + e^{B_1(|d| - c|q|)}} & \text{if } |d| < |q| \\ 1 & \text{if } |d| = |q| \\ 1 + \dfrac{b_2 - 1}{1 + e^{-B_2(|d| - (1+c)|q|)}} & \text{if } |d| > |q| \end{cases} \tag{5}$$

where

- b_1 is the upper limit on the left side,
- b_2 is the upper limit on the right side,
- B_1 and B_2 are growth parameters, which can be typically set to 1,
- c is the trough curvature, where $c \in (0, 1)$.

This feature can be plotted as shown in figure 9 against various parameters.
If we observe the graph we see a neat bucket-shaped trough when $|d|$ approaches $|q|$. It does not penalize when the lengths are almost similar but starts

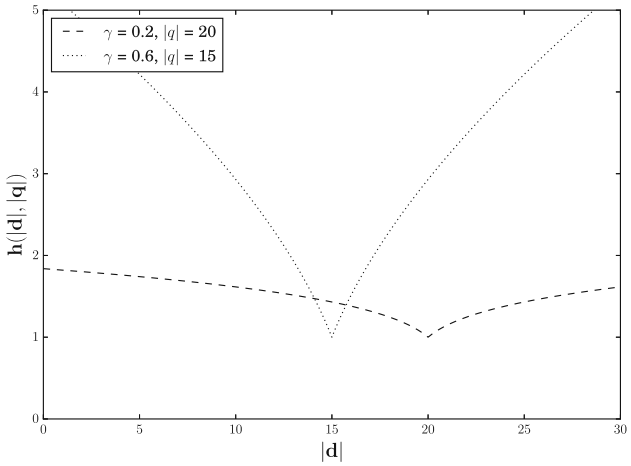

Figure 8. The power penalization $h(|d|, |q|)$ curve. The value of $h(|d|, |q|)$ dips when $|d|$ approaches $|q|$. As γ increases, penalization would become harsher.

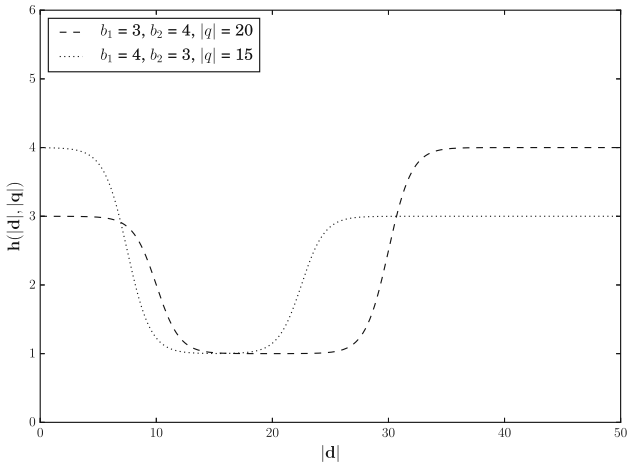

Figure 9. The nature of sigmoidal curve $h(|d|, |q|)$ with $B_1 = 1$, $B_2 = 1$ and $c = 0.5$.

penalizing when they are extremely dissimilar. This is feasible as a misspelling (query length) would be around the length of the correct spelling (document length). Using harsh penalization when lengths are dissimilar can prune unnecessary spelling matches with this heuristic.

Clearly, these two penalization heuristics behave and achieve the stated objectives. We now demonstrate how we can use this penalization heuristics in our ranking functions. We know that BM25 ranking function is given by

$$score(q,d) = \sum_{t \in d \cap q} f(t,q) \log \frac{M+1}{df(t)} \times \frac{(k+1)f(t,d)}{f(t,d) + k\left(1 - b + b\frac{|d|}{avgdl}\right)},$$

where $f(t, q)$ is the frequency of terms in queries, M is the number of documents, k is a hyperparameter that sets the bound, $f(t, d)$ is the frequency of terms in documents, $df(t)$ is the document frequency, b is the normalization parameter, $|d|$ is the document length and $avgdl$ is the average document length. Here $\left(1 - b + b\frac{|d|}{avgdl}\right)$ is the normalization heuristic. If we replace this heuristic with $h(|d|, |q|)$ we get

$$score(q,d) = \sum_{t \in d \cap q} f(t,q) \log \frac{M+1}{df(t)} \times \frac{(k+1)f(t,d)}{f(t,d) + k(h(|d|,|q|))}. \quad (6)$$

If the value of $h(|d|, |q|)$ decreases, that is if $|d|$ and $|q|$ are similar, then the value of $score(q, d)$ increases. This means similar document and query word lengths are rewarded. Vice versa can be said if the value of $h(|d|, |q|)$ increases. Equation (6) is mentioned as the TAKE curve, which would be involved in our experimental set-ups.

3. Experimental evaluation

We evaluate our heuristics over three problems: isolated spelling correction, cognate detection and SNP detection. All of the hyperparameters of baselines presented in this section are tuned in the mentioned datasets according to their best performances. MeTA toolkit [26] is employed to build the search engine. We performed these experiments on a MacBook Pro Laptop with an Intel Core i7 2.2GHz CPU, 16 GB of RAM and a 512 GB disk. The time complexity of our proposed BREAK, MAKE and TAKE approach depends on the dataset size and the three problems considered. However, our models are faster than the compared methods; for example, in the cognate detection problem, SVM classifier took 2.54 s whereas our approach took 0.648 s.

3.1 *Spelling correction*

This problem is tested using four different datasets:

1. *Conventional English Spelling Evaluation (CESE)*: A collection of 4000 misspellings and their corrections were organized from Fawthrop's contribution in Birkbeck and Wikipedia spelling error corpus [27, 28]. These datasets are considered as the gold standard and are commonly tested by other researchers in the spelling correction. The dataset is divided into two parts: training set (comprises 3000 words) and test set (comprises 1000 words).
2. *Low Training Set Evaluation (LTSE)*: We simulated 50000 English misspellings probabilistically. This dataset was then divided into two parts: training set (comprises 5 words) and test set (comprises 49995 words). This dataset is constructed to assess how algorithm behaves with less training data (LT).
3. *Corrupted Label Evaluation (CLE)*: We simulated 50000 English misspellings probabilistically. This dataset was then divided into two parts: training set (comprises 40000 words) and test set (comprises 10000 words). From the 40000 words, we mislabelled 20000 of them randomly. This dataset is constructed to check the robustness of the algorithms.
4. *Hindi Spelling Evaluation (HSE)*: A collection of 5000 Hindi errors were taken (error corpus constructed by ourselves). The dataset is divided into two parts: training set (comprises 4000 words) and test set (comprises 1000 words). We used the technique called *varn-viched* to tokenize the split of the Hindi words into individual k-grams, which is analogous to letter splitting technique in Roman languages.

We chose five baselines for the evaluation purposes: a brute-force looped method [8], weighted longest common subsequence [9], finite state transducer [12], basic split-up (BREAK-0) with Jaccard similarity and skip-grams [10].

We used MRR (Mean Reciprocal Rank) here as our evaluation metric, since each misspelling had only one solution in our dataset. The MRR is given by

$$\text{MRR} = \frac{1}{Q}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i},$$

where rank_i refers to the rank position for a sample of queries Q. Aspell [29] with the dictionary size of 60 has been used to create the spell-check dictionary for the English evaluation.

Table 2 shows that BREAK-2 with BM25-based length penalty (TAKE) and graphical error models (MAKE) performed the best in conventional English and Hindi spelling evaluation. BREAK-2 combined with the BM25-based length penalty (MAKE) performed the best in the corrupted label and low training dataset size evaluation, as these techniques are robust against data abnormalities.

3.2 *Cognate detection*

This experiment was performed on the dataset and evaluation scheme proposed by Rama [3]. In this dataset, the word pairs are organized into cognate class numbers. Positive and negative labels are assigned to the same and different class numbers, respectively. For comparisons, we chose SVM classifier for cognates detection as proposed by Kondrak and Sherif [13] and Naive-Bayes-based features as proposed by Ciobanu and Dinu [15]. The dataset was divided in the ratio of 3:1 for the training and testing purposes.

Table 2. Test dataset mean reciprocal rank (MRR) results for spelling correction evaluation.

Algorithm applied	CESE	LTSE	CLE	HSE
Brute-force [8]	68%	70%	71%	38%
LCS [9]	61%	59%	52%	23%
FST [12]	81%	22%	19%	31%
BREAK-0 + Jaccard	63%	68%	67%	35%
Skip-grams [10]	65%	70%	68%	29%
BREAK-0 + Dirichlet	70%	65%	69%	66%
BREAK-0 + BM25	69%	70%	68%	64%
BREAK-2 + Dirichlet	72%	72%	69%	68%
BREAK-2-OFF + Dirichlet	75%	76%	71%	70%
BREAK-2 + TAKE + BM25	78%	**79%**	**74%**	75%
BREAK-2 + TAKE + MAKE + BM25	**86%**	38%	35%	**84%**

Precision at $k = 1$ and $F1$ scores are used to evaluate the algorithms used in this experiment. Precision and $F1$ scores are given by

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}},$$
$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}},$$

with $\text{recall} = \frac{\text{true positive}}{\text{true positive+false negative}}$, respectively. The results in table 3 show that BREAK-2 with MAKE and TAKE performed slightly better than the others.

3.3 *SNP detection*

This is a new problem for detecting SNPs (single-nucleotide polymorphism), which corresponds to genetic mutant identification suggested by Nguyen *et al* [2]. For this experiment, we simulated two datasets. The dataset D1 contained 10000 genomes. Each genome sequence comprises length of 500–600 base pairs for the artificial DNA sequences. Ten distinct mutations were created for each genome sequence. Later the dataset is populated with 99% noise, resulting in total size of D1 as 1 million genome sequences. The dataset D2 has 50000 protein sequences with length around 1000–1500 base pairs, which is populated with 99.9% noise, resulting in total size of D2 as 1.5 million protein sequences. The D2 dataset was artificially created by simulation of the random patterns and mutations caused by the 21 proteins. The BREAK-0 with Jaccard similarity is chosen as a baseline and we use MinDist proximity heuristic clubbed with BM25 and BREAK-0 [30], to compare our results.

Along with the MRR we also utilized the Normalized Discounted Cumulative Gain (NDCG) as metrics for this experiment. The NDCG is given by

$$\text{NDCG}_p = \frac{\text{DCG}_p}{\text{IDCG}_p},$$

where $\text{IDCG}_p = \sum_{i=1}^{|\text{rel}_i|}\frac{2^{\text{rel}_i}-1}{\log_2(i+1)}$ is the ideal discounted cumulative gain, rel_i represents the list of relevant documents up to position p and $\text{DCG}_p = \sum_{i=1}^{p}\frac{2^{\text{rel}_i}-1}{\log_2(i+1)}$. Table 4

Table 3. Test dataset results for cognate detection experiment.

Algorithm applied	$P@1$	$F1$
Naive Bayes [15]	0.75	0.4
SVM [13]	0.78	0.46
BREAK-2-OFF + Dirichlet	0.76	0.42
BREAK-2 + MAKE + TAKE + BM25	**0.80**	**0.48**

Table 4. Results for gene mutant detection simulation experiment.

Algorithm	*D*1 (MRR)	*D*1 (NDCG)	*D*2 (MRR)	*D*2 (NDCG)
BREAK-0 + Jaccard	0.33	0.16	0.31	0.12
MinDist + BM25 + BREAK-0	0.55	0.40	0.49	0.38
BREAK-*n* + Dirichlet	0.99	0.98	0.95	0.94

shows that BREAK-*n* with Dirichlet outperforms others. We used 41 and 83 anchor points for D1 and D2, respectively, which were obtained by experimentation on cross-validation dataset. This shows that our heuristics is scalable and works well with larger datasets.

3.4 *Analysis of results*

Before indexing the documents, the dataset must be split according to the desired BREAK variant. For the usage of BREAK-*n*, note that one can store the positional vectors while indexing the corresponding term, or generate the vector directly during evaluation. This choice highly depends on the time–space tradeoff. However, in our experiments we have stored the vectors during indexing, prioritizing faster evaluation.

The common trend in the experimental set-ups is that Dirichlet tends to perform better than Okapi BM25. This is due to the fact that Dirichlet prior smoothing factor takes account of the language modelling, which in turn makes it more successful than vanilla BM25. We investigate the effect of Dirichlet prior with BREAK-2 μ_1 and with MAKE in μ_2. Figure 10 shows that both reach the maximum MAP equally; however μ_1 reaches earlier.

Now we analyse the effect of weighted average equation of MAKE, the hyperparameter tuning of the factor λ. We

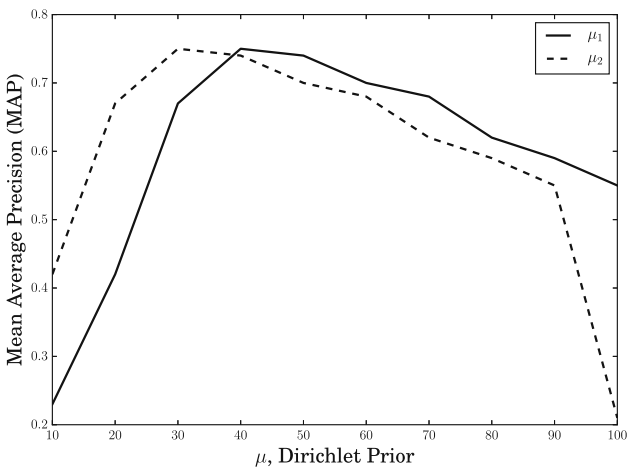

Figure 10. Effect of Dirichlet prior against MAP with BREAK-2 (μ_1) and with MAKE (μ_2).

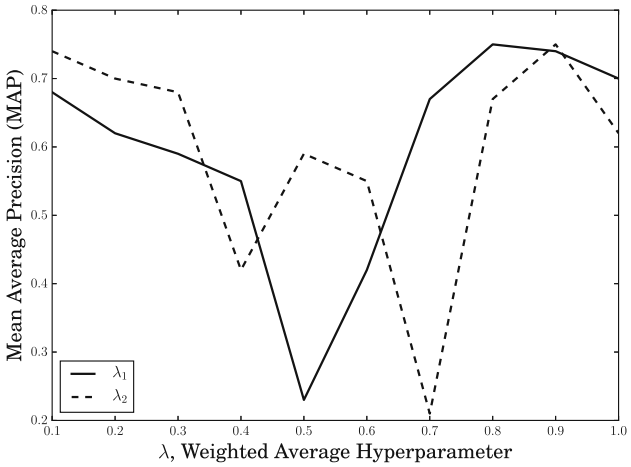

Figure 11. The effect of hyperparameter sensitivity in isolated spelling correction λ_1 and with cognate detection in λ_2.

investigate the effect of hyperparameter sensitivity in isolated spelling correction λ_1 and with cognate detection in λ_2. Figure 11 describes the optimal conditions of the suitable hyperparameters. The late reach of λ_2 suggests that cognate detection experiments depend more on graphical error modelling MAKE algorithm than the BREAK heuristic.

One of the limitations of the presented experimental results is the amount of noise robustness. When tested for a relatively small noise level (up to 10%) our approach gave good results; however a deeper analysis for higher levels (>10%) requires elaborate experiments, which constitutes one of the current works.

4. Conclusion

The central theme of this paper is to bring retrieval models and approximate string similarity algorithms together. We proposed sequential splitting variants like BREAK-1, -2, -OFF, -*n*. To relate these sequential splits, we propose MAKE algorithm, which constructs the probabilistic distribution between the query and document split sets. Finally we designed the TAKE curve, to fit these heuristics in the normalization function of the ranking functions. In the experiments, we see that our heuristics is robust against the datasets characteristics like mislabelling and low training dataset size, and scales well in larger datasets. We plan to improve BREAK-*n* and apply SNP detection problem to real datasets and complex bioinformatics applications in the future.

Acknowledgements

The authors sincerely thank the reviewers for their comments that helped improve the presentation of the paper.

Compliance with ethical standards

Conflicts of interest None.

References

[1] Kukich K 1992 Techniques for automatically correcting words in text. *ACM Comput. Surv.* 24: 377–439

[2] Nguyen K, Guo X and Pan Y 2016 *Multiple biological sequence alignment: scoring functions, algorithms and evaluation*. Hoboken: Wiley

[3] Rama T 2015 Automatic cognate identification with gap-weighted string subsequences. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1227–1231

[4] Robertson S E and Walker S 1994 Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, USA, pp. 232–241

[5] Zhai C and Lafferty J 2004 A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.* 22(2): 179–214

[6] Amati G and Van Rijsbergen C J 2002 Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. Inf. Syst.* 20: 357–389

[7] Fang H, Tao T and Zhai C 2011 Diagnostic evaluation of information retrieval models. *ACM Trans. Inf. Syst.* 29: 7:1–7:42.

[8] Norvig P 2007 How to write a spelling corrector. http://norvig.com/spell-correct.html

[9] Islam A and Inkpen D 2009 Real-word spelling correction using Google web IT 3-grams. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, vol. 3, Association for Computational Linguistics, pp. 1241–1249

[10] Järvelin A, Järvelin A and Järvelin K 2007 s-grams: defining generalized n-grams for information retrieval. *Inf. Process. Manag.* 43: 1005–1019

[11] Keskustalo H, Pirkola A, Visala K, Leppänen E and Järvelin K 2003 Non-adjacent diagrams improve matching of cross-lingual spelling variants. In: *Proceedings of the International Symposium on String Processing and Information Retrieval*, pp. 252–265

[12] Pirinen T and Lindén K 2010 Finite-state spell-checking with weighted language and error models. In: *Proceedings of LREC 2010 Workshop on Creation and Use of Basic Lexical Resources for Less-resourced Languages*, Malta, pp. 1–6

[13] Kondrak G and Sherif T 2006 Evaluation of several phonetic similarity algorithms on the task of cognate identification. In: *Proceedings of the Workshop on Linguistic Distances*. Association for Computational Linguistics, pp. 43–50

[14] Inkpen D, Frunza O and Kondrak G 2005 Automatic identification of cognates and false friends in French and English. In: *Proceedings of the International Conference on Recent Advances in Natural Language Processing*, pp. 251–257

[15] Ciobanu A M and Dinu L P 2014 Automatic detection of cognates using orthographic alignment. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pp. 99–105

[16] Huang Y and Zhang L 2004 Rapid and sensitive dot-matrix methods for genome analysis. *Bioinformatics* 20: 460–466

[17] Needleman S B and Wunsch C D 1970 A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 48(3): 443–453

[18] Smith T F and Waterman M S 1981 Identification of common molecular subsequences. *J. Mol. Biol.* 147: 195–197

[19] Pearson W R 1990 Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods Enzymol.* 183: 63–98

[20] Kent W J 2002 BLAT—the BLAST-like alignment tool. *Genome Res.* 12: 656–664

[21] Notredame C, Higgins D G and Heringa J 2000 T-coffee: a novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.* 302: 205–217

[22] Zhai C and Massung S 2016 *Text data management and analysis: a practical introduction to information retrieval and text mining*. USA: Morgan & Claypool

[23] Manning C D, Raghavan P and Schütze H 2008 *Introduction to information retrieval*. New York, NY, USA: Cambridge University Press

[24] Robertson S and Zaragoza H 2009 The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.* 17: 333–389

[25] Birch C P 1999 A new generalized logistic sigmoid growth equation compared with the Richards growth equation. *Ann. Bot.* 83: 713–723

[26] Massung S, Geigle C and Zhai C 2016 MeTA: a unified toolkit for text retrieval and analysis. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 91–96

[27] Mitton R 1985 *Birkbeck spelling error corpus*. Retrieved from The Oxford Text Archive. http://ota.ahds.ac.uk

[28] Wikipedia 2017 *Commonly misspelled English words*. https://en.wikipedia.org/wiki/Commonly_misspelled_English_words

[29] Atkinson K *Spell checking oriented word lists*. GNU Aspell. http://aspell.net/

[30] Tao T and Zhai C 2007 An exploration of proximity measures in information retrieval. In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 295–302