Reference-Based Alignment in Large Sequence Databases

Panagiotis Papapetrou [1], Vassilis Athitsos [2], George Kollios [1], and Dimitrios Gunopulos [3,4]

[1] Computer Science Department, Boston University
[2] Computer Science and Engineering Department, University of Texas at Arlington
[3] Department of Informatics and Telecommunications, University of Athens
[4] Computer Science and Engineering Department, UC Riverside

ABSTRACT

This paper introduces a novel method, called Reference-Based String Alignment (RBSA), that speeds up retrieval of optimal subsequence matches in large databases of sequences under the edit distance and the Smith-Waterman similarity measure. RBSA operates using the assumption that the optimal match deviates by a relatively small amount from the query, an amount that does not exceed a prespecified fraction of the query length. RBSA has an exact version that guarantees no false dismissals and can handle large queries efficiently. An approximate version of RBSA is also described, that achieves significant additional improvements over the exact version, with negligible losses in retrieval accuracy. RBSA performs filtering of candidate matches using precomputed alignment scores between the database sequence and a set of fixed-length reference sequences. At query time, the query sequence is partitioned into segments of length equal to that of the reference sequences. For each of those segments, the alignment scores between the segment and the reference sequences are used to efficiently identify a relatively small number of candidate subsequence matches. An alphabet collapsing technique is employed to improve the pruning power of the filter step. In our experimental evaluation, RBSA significantly outperforms state-of-the-art biological sequence alignment methods, such as q-grams, BLAST, and BWT.

1. INTRODUCTION

There are many applications that require fast searching in sequence databases that consist of collections of strings. Given a query string, the goal is to find the most similar substrings in the database using a distance/similarity measure such as the edit distance (ED) or Smith-Waterman (SW). Applications in this area include 1) spell-checking: given some input text the spell-checker consults its dictionary to find words of high similarity to the text, so as to identify potential typos, 2) data cleaning: data obtained from different sources might contain inconsistencies which can be eliminated by looking for similar entities (strings) in the data, 3) near homology search in biological sequences: given different genomes we want to find regions of high similarity that were the result of a mutation, etc. Being able to efficiently answer such queries is

crucial, especially for online string search applications.

In order to generate and interpret complete genomes of different organisms, various searches need to be performed that 1) involve queries of large length, and 2) only target *near exact* matches [11, 14]. We focus on these two major requirements: we want to be able to retrieve *near-exact matches* of *long query sequences* efficiently. As a motivating example for large query lengths, consider large EST (Expressed Sequence Tag) databases, that contain portions of genes expressed as mature mRNA. In such databases, large scale searches need to be performed against other genomic databases to determine locations of genes [14]. In practice, genes can vary in size from hundreds to millions of nucleotides. Searches can also target whole chromosomes, where the goal is to find chromosome similarities across different organisms. Since chromosomes can be relatively large (e.g. Human Chromosome 1 is approximately 272 million bases), such searches require algorithms that can handle large queries efficiently.

In many applications, database matches are of interest only if their deviation from the query does not exceed a certain, relatively small, fraction of the query length [5, 11, 17]. We denote that fraction as δ and focus on values of δ up to 15%, which is typical in applications, such as shotgun sequencing [30] and mutation analysis [38]. Notice that our focus is on DNA sequences, where the alphabet size is small (4) and the query size can be large (up to $10,000$ bases). In this setting, only near homology search is biologically significant, whereas *remote homology search* is more meaningful and mostly used not for DNA, but for protein sequences.

In this paper, we propose a novel method, called reference-based string alignment (RBSA), for efficient subsequence matching in large databases of strings under the edit distance or the Smith-Waterman similarity measure. RBSA decomposes the subsequence matching problem into two distinct problems:

- **The fixed query length problem:** achieve efficient retrieval assuming that all queries have the same length.

- **The variable query length problem:** using a solution to the fixed query length problem, achieve efficient retrieval for queries of arbitrary length.

To solve the fixed query length problem, RBSA precomputes, for each position of every database string, alignment scores corresponding to different reference sequences. These alignment scores are based on the edit distance. Given a query, alignment scores between the query and all reference sequences are computed online and are used to prune away large portions of the database, so as to leave a relatively small number of candidate matches. We can guarantee that the *optimal subsequence match* will be included among the candidates. Exact alignment scores are then computed to identify the optimal match among the remaining candidates. Notice that

the term *optimal subsequence match*, refers to the database position that gives the best alignment score for an input query using the exact and full dynamic programming algorithm. In the case of edit distance, this refers to the database position with the lowest score and for the case of Smith-Waterman this is the position with the highest similarity score.

To solve the variable query length problem, RBSA first breaks up that problem into multiple fixed query length problems, by partitioning the query sequence into segments of fixed length. In the exact version of RBSA, all query segments are considered, and subsequence matches found for those segments are used to identify candidate subsequence matches for the entire query. In the approximate version of RBSA, only a subset of query segments is considered. Another contribution in our paper consists in showing that the probability of failing to find the optimal match drops very fast (exponentially) as we increase the number of query segments that we consider, and thus we can achieve both significantly improved efficiency and very high accuracy rates by considering only a relatively small number of segments.

The main contributions of this paper are summarized below:

1. We present RSBA, the first reference-based method for subsequent matching in string databases that both guarantees correct results and performs well for large queries. RBSA produces lower bounds of the edit distance and upper bounds of the Smith-Waterman similarity between the query and database subsequences using precomputed alignment scores with reference sequences. In prior work, such bounds have only been derived for full sequence matching [42].

2. We present an exact method for decomposing the variable-length query problem into multiple fixed-length queries, so that we can achieve state-of-the-art retrieval runtimes for long queries, while still guaranteeing correct results.

3. We present an approximate method for decomposing the variable-length query problem into multiple fixed-length queries. The approximate variant achieves speedups of over an order of magnitude, compared to exact RBSA and other competitors, in experiments with queries of length 10,000. At the same time, the probability of missing the correct result in approximate RBSA drops exponentially with the number of query segments that we consider, and thus can easily be reduced to a negligible quantity.

4. The experimental evaluation shows that, for query lengths ≥ 200, RBSM outperforms current state-of-the-art sequence alignment methods: BLAST2[2], BWT-SW[22] and q-grams. Speedups of one to two orders of magnitude over the current state of the art are demonstrated for query sizes $\geq 2,000$.

2. RELATED WORK

A preeminent group of methods for string subsequence matching are based on dynamic programming [23]. In [33], a global alignment method is described, where both query and database sequences are aligned along their entire lengths, using *match*, *mismatch* and *gap* scores. A similar, but generalized algorithm [13] for global alignment, handles sequences of intermittent similarities. Smith and Waterman [39] developed a dynamic programming approach for local alignment, where a subsequence of the query is matched to a subsequence of the database. [40] exploits the fact that in approximate string searching we are looking for patterns that match with substrings of the text with at most k errors. Thus, it

speeds up the dynamic programming (DP) computation by pruning cells in the DP matrix with values larger than k.

Several q-gram-based methods [5, 6, 7, 18, 24, 25, 27, 31, 43] have been developed to solve the problem of exact and approximate string matching in large sequence databases. Their main characteristic is that they build a dictionary of words on a given database of sequences. At query time the query is broken into a set of overlapping q-grams and the index is searched for exact matches of those q-grams. These matches provide candidate hits that are later refined to remove false positives.

QUASAR [5] is a subsequence matching method that performs q-gram based filtering on a sequence database. QUASAR is limited to relatively short queries (the maximum query length on which the performance of QUASAR was evaluated was 393 characters) of high similarity to the database. A generalization of QUASAR, which uses gapped instead of contiguous q-grams is described in [6]. Similar q-gram based methods for approximate full string matching are described in [24, 25, 43].

VGRAM [25] employs a q-gram dictionary where the words are of variable length and more representative of the dataset. Again, the limitations to small queries persist (the experimental evaluation reports queries of average size ranging from 8 to 62 characters) and the performance seriously deteriorates as k (the number of edit operations applied to the queries) increases (> 4). An improved vgram-based method is described in [43], but is again limited to small query sizes (varying between 4 and 249 characters). Several methods [18, 7] employ a two level q-gram index to speed up the database search. A q-gram based approximate string matching method is described in [31], where disjoint text substrings of length q are collected by the index at fixed intervals. Finally, [24] introduces several strategies for improving the join cost of the gram lists found during a query search in an inverted q-gram index and shows how to incorporate these strategies into existing filtering methods to improve string matching.

A key property of q-gram based methods, such as the ones mentioned above, is the following: if the query size is $|Q|$ and we are searching for matches with edit distance within k, q can be at most $\lceil |Q|/(k+1) \rceil$ to guarantee no false dismissals. It can be seen that as k increases, q decreases, and thus, the index size becomes larger. Consequently, and also as shown in the experiments, q-gram based approaches can only handle short queries of relatively high similarity to the database. However, the biologically interesting types of queries (e.g. mutated genes) can be significantly long (up to $10,000$ nucleotides or more [22]) and thus, q-gram based methods are not able to handle them efficiently.

Another group of methods has been proposed for *exact string matching*, targeting exact occurrences of the query sequence in a database [4, 9, 10, 16, 20, 27, 28, 37, 41]. However, exact string matching is quite different from the main focus of this paper and thus, these methods are not discussed any further.

Several methods have been developed for aligning biological sequences. FASTA [26, 35] detects locally similar regions between two sequences using only identities and no gaps, and then based on some measure of similarity it re-scores them accordingly. Additional heuristics are proposed in BLAST [1]. Given a query (DNA or protein), BLAST performs a linear scan on the sequence database searching for a set of seeds belonging to the neighborhood of some substrings of the query. Having identified a set of candidate hits, it then extends them both ways, until the accumulated similarity score begins to decrease. Finally, BLAST reports as matches those regions with high statistical significance.

A new version of BLAST, known as BLAST2 [2], improves accuracy by allowing a limited number of insertions and deletions

during the alignment formation and improves search speed by imposing more stringent criteria when performing a local alignment. Further improvements of BLAST include MegaBLAST [44], MPBLAST [21] and miBLAST [19]. MegaBLAST is a greedy algorithm for detecting sequences that differ slightly as a result of sequencing. MPBLAST and miBLAST are different versions of BLAST used for parallel queries.

BLAT [1] builds an index of the database and then given a query, it linearly scans the query searching for matches in the index. Apart from using an inverse index, BLAT differs from BLAST and BLAST2 in that it triggers extensions on any number of perfect hits whereas in BLAST extensions are triggered when one or two hits occur in proximity to each other. Several hash-based approaches [15, 34] have been developed for further speed up. A key limitation of all the above-mentioned variants of BLAST is that their accuracy and retrieval cost deteriorates as the query size increases. As the volume of biological sequence databases increases, all the aforementioned exhaustive systems become prohibitively expensive.

Another key limitation of BLAST-like approaches is that there is no guarantee that the optimal local alignment will be reported. Several methods have been developed to handle this weakness. OASIS [29] employs a best first search technique over a suffix tree for string alignment. The algorithm outperforms BLAST by an order of magnitude, but only for small query sizes (5 to 60); this is one of its major limitations. Another indexing method that uses suffix trees is discussed in [32], whereas [36] discusses an efficient method for suffix tree construction in external memory. Finally, BWT-SW[22] employs a suffix array to speedup local alignment search in biological sequences. It outperforms BLAST for queries of size up to 1000; for larger queries its performance deteriorates. Both OASIS and BWT-SW always find the best local alignment according to Smith-Waterman.

Two reference-based indexing methods for full sequence matching are proposed in [42] that use reference sequences to represent the database. At query time, the edit distance of the query against each reference sequence is computed. Lower and upper bounds are applied to efficiently filter candidate matches. DSIM[8] uses a set of selected reference words formed from high-frequency data sub- strings. SST [11] is used for subsequence matching in biological sequences and maps the biological sequence database to a d-dimensional vector space; this mapping is used to filter a significant portion of the database from consideration during the query process. This method outperforms BLAST by an order of magnitude but only for applications where there exists an extremely high similarity (95% and over) between the query sequence and its match in the database.

The RBSA method proposed in our paper is also related to EBSM [3], which uses precomputed alignments between database sequences and reference sequences for efficient subsequence matching in time series databases. The key differences between RBSA and EBSM stem from the fact that RBSA addresses near-exact string matching under the edit distance or Smith-Waterman, whereas EBSM addresses general time series matching under DTW. RBSA exploits the metric properties of the edit distance, and the additional near-exact matching constraint, to provide either guaranteed correct results (for exact RBSA) or guaranteed high probability of correct results (for approximate RBSA). No equivalent guarantees are present in EBSM. Furthermore, RBSA can handle queries of arbitrary size (query lengths range from 40 to 10,000 in our experiments) by breaking up queries into fixed-size segments, whereas EBSM requires that query lengths be within a relatively narrow range (query lengths range from 152 to 426 in the experiments of [3]), and provides no mechanism for handling queries of arbitrary size.

3. BACKGROUND

In this section we define the edit distance and Smith-Waterman measures used to evaluate similarity between strings (e.g. DNA). We use the terms "string" and "sequence" interchangeably. Throughout the paper, the following notation will be used:

- Q, X are sequences of length $|Q|$ and $|X|$ respectively. Q denotes a query sequence and X denotes a database sequence. Typically $|X| \gg |Q|$. Without loss of generality we assume that the database contains a single very long sequence, since we can always concatenate all the strings stored in the database into a single string.

- Subscripts denote elements of sequences. For example, $Q = (Q_1, \ldots, Q_{|Q|})$.

- For any sequence $X = (X_1, \ldots, X_{|X|})$, given start and end positions s and t respectively, we can define *subsequence* $X^{s:t}$ to be the sequence $(X_s, \ldots, X_t)$, i.e., the part of X that starts at position s and ends at position t. Then, $X_i^{s:t}$ is the i-th element of $X^{s:t}$, and is equal to X_{s+i-1}.

3.1 The Edit Distance

The edit distance $\Delta(A, B)$ is a function measuring how *dissimilar* two strings A and B are. For a more general definition of the edit distance we need to specify a cost for each editing operation, i.e., for each insertion, deletion, and substitution. In this paper we denote these costs as follows:

- C_{ins} denotes the cost of the edit operation that inserts a letter to string A.

- C_{del} denotes the cost of the edit operation that deletes a letter from string A.

- $C_{\text{sub}}(A_j, B_t)$ denotes the cost of the edit operation that replaces letter A_j with some letter $B_t \neq A_j$.

In the general case, $\Delta(A, B)$ is the smallest possible cost of converting A to B using insertions, deletions, and substitutions. In the most common version of ED, $C_{\text{ins}} = C_{\text{del}} = C_{\text{sub}} = 1$, and in that case $\Delta(A, B)$ is the smallest total number of insertions, deletions, and substitutions that can convert A to B. For simplicity, in the remainder of this paper we assume that $C_{\text{ins}} = C_{\text{del}} = C_{\text{sub}} = 1$.

Given a query sequence Q and a database sequence X, the best (optimal) *subsequence match* of Q in X is the subsequence $X^{s:t}$ that minimizes $\Delta(Q, X^{s:t})$. We define the subsequence matching cost $D(Q, X)$ as:

$$D(Q, X) = \min\{\Delta(Q, X^{s:t}) | s \in \{1, \ldots, t\}, t \in \{1, \ldots, |X|\}\}.$$
(1)

In describing how to compute $D(Q, X)$ and the corresponding subsequence match $X^{s:t}$, it is useful to define an auxiliary distance $D^{j,t}$, as the smallest possible distance between $Q^{1:j}$ and a *suffix* $X^{s:t}$ of $X^{1:t}$:

$$D^{j,t}(Q, X) = \min\{\Delta(Q^{1:j}, X^{s:t}) | s \in \{1, \ldots, t\}\}.$$
(2)

We also define an auxiliary function $C(Q_j, X_t)$ that denotes the cost of matching letter Q_j with letter X_t:

$$C(Q_j, X_t) = \left\{ \begin{array}{ll} C_{\text{sub}} & \text{if } Q_j \neq X_t \\ 0 & \text{if } Q_j = X_t \end{array} \right.$$
(3)

Computing $D(Q, X)$ and the corresponding best subsequence match of Q in X can be performed using dynamic programming,

by computing $D^{j,t}(Q, X)$ for $j = 1, \ldots, |Q|$ and $t = 1, \ldots, |Q|$, as follows:

initialization:
$$D^{0,0} = 0, D^{j,0} = \infty, D^{0,t} = 0 . \qquad (4)$$

loop:
$$D^{j,t}(Q, X) = \min \begin{cases} D^{j,t-1}(Q, X) + C_{\text{ins}} \\ D^{j-1,t}(Q, X) + C_{\text{del}} \\ D^{j-1,t-1}(Q, X) + C(Q_j, X_t) \end{cases} \quad (5)$$
$$(j = 1, \ldots, |Q|; t = 1, \ldots, |X|) .$$

termination:
$$t^* = \operatorname{argmin}_{t=1,\ldots,|X|} \{ D^{|Q|,t}(Q, X) \} . \qquad (6)$$
$$D(Q, X) = D^{|Q|,t^*}(Q, X) . \qquad (7)$$

It should be clear that evaluating $D(Q, X)$ takes time $O(|Q||X|)$. We should also note that the optimal matching sequence can be found by keeping track, in each application of Equation 5, of the predecessor selected for each (j, t), and by backtracking, at termination, starting at position $(|Q|, t^*)$.

3.2 The Smith-Waterman Measure

A similarity measure $\Lambda(A, B)$, in contrast to a distance measure, measures how *similar* two strings A and B are. If $\Lambda(A, B) = 0$ then A and B are maximally different from each other. The Smith-Waterman measure [39] is a frequently used similarity measure for strings. In order to specify the Smith-Waterman measure, we need to choose values $P_{\text{match}}, P_{\text{sub}}$ and P_{gap}, that stand for the following terms:

- P_{match} is a positive number that denotes the reward for a letter of A being equal to the corresponding letter in B.

- P_{sub} is a negative number that denotes the penalty for a letter of A being substituted by another letter.

- P_{gap} is a negative number that denotes the penalty for deleting a letter of A, or inserting a letter to A.

In the remainder of the paper, and in our experiments, we use $P_{\text{match}} = 2, P_{\text{sub}} = -1$, and $P_{\text{gap}} = -1$, which are commonly used choices for these parameters.

Given a query string Q and a database string X, finding the best (optimal) *local alignment* between Q and X is the task of finding subsequences $Q^{i:j}$ and $X^{s:t}$ that maximize $\Lambda(Q^{i:j}, X^{s:t})$. We define the Smith-Waterman similarity score $L(Q, X)$ as:

$$L(Q, X) = \max\{\Lambda(Q^{i:j}, X^{s:t}) | i \in \{1, \ldots, j\}, j \in \{1, \ldots, |Q|\},$$
$$s \in \{1, \ldots, t\}, t \in \{1, \ldots, |X|\}\} . \qquad (8)$$

In describing how to compute $L(Q, X)$ and the corresponding optimally matching subsequences $Q^{i:j}$ and $X^{s:t}$, it is useful to define an auxiliary score $S^{j,t}$, as the highest matching score between a suffix $Q^{i:j}$ of $Q^{1:j}$ and a *suffix* $X^{s:t}$ of $X^{1:t}$:

$$L^{j,t}(Q, X) = \max\{\Lambda(Q^{i:j}, X^{s:t}) | i \in \{1, \ldots, j\}, s \in \{1, \ldots, t\}\} . \qquad (9)$$

We also define an auxiliary function $P(Q_j, X_t)$ that denotes the reward or penalty of matching letter Q_j with letter X_t:

$$P(Q_j, X_t) = \begin{cases} P_{\text{sub}} & \text{if } Q_j \neq X_t \\ P_{\text{match}} & \text{if } Q_j = X_t \end{cases} \qquad (10)$$

Given Q and X, the Smith-Waterman algorithm identifies optimal subsequences $Q^{i:j}$ and $X^{s:j}$ and the corresponding similarity score $L(Q, X) = \Lambda(Q^{i:j}, X^{s:t})$. The Smith-Waterman algorithm is very similar to the algorithm computing the edit distance, and also proceeds using dynamic programming, by computing $L^{j,t}$ for $j = 1, \ldots, |Q|$ and $t = 1, \ldots, |Q|$, as follows:

initialization:
$$L^{j,0} = 0, L^{0,t} = 0 . \qquad (11)$$

$$L^{j,t}(Q, X) = \max \begin{cases} L^{j,t-1}(Q, X) + P_{\text{gap}} \\ L^{j-1,t}(Q, X) + P_{\text{gap}} \\ L^{j-1,t-1}(Q, X) + P(Q_j, X_t) \\ 0 \end{cases} (12)$$
$$(j = 1, \ldots, |Q|; t = 1, \ldots, |X|) .$$

termination:
$$L(Q, X) = \max_{j=1,\ldots,|Q|, t=1,\ldots,|X|} \{ L^{j,t}(Q, X) \} . \qquad (13)$$

Similar to the edit distance, Smith-Waterman takes time $O(|Q||X|)$, and finding the subsequences of Q and X that give the maximum similarity score can be easily done using backtracking.

4. RBSA FOR FIXED QUERY LENGTH

In this section, we describe the proposed RBSA (Reference-Based String Alignment) method for queries of fixed length. We denote that fixed length as q. In Section 5, we will generalize RBSA to queries of arbitrary length.

RBSA follows a filter-and-refine approach. A set of random reference sequences is generated. For each database position, an alignment score with each reference sequence is computed, and an embedding-based index is constructed using those scores. The embedding is used for fast filtering of database positions that can lead to a potential match. Those positions are then passed to the refine step where the computationally expensive distance measure (edit distance or Smith-Waterman) is applied.

4.1 Embedding Queries and Database Positions

Let Q be a query sequence of fixed length $|Q| = q$, and X be the database sequence. At the core of our method is an embedding definition, that we use to produce one-dimensional ($1D$) mappings, that map every query sequence Q to a number, and that map every database position (X, t) also to a number. We will use these $1D$ mappings to obtain bounds for the optimal subsequence matching or local alignment score *ending* at each database position (X, t), and then we will use those bounds to efficiently prune significant portions of the database.

Let R be a sequence of the same fixed length q as the queries. Using R we can define a 1D embedding F^R, mapping each query sequence into a real number $F^R(Q)$, and also mapping each database position (X, t) into a real number $F^R(X, t)$:

$$F^R(Q) = D^{|R|,|Q|}(R, Q) . \qquad (14)$$
$$F^R(X, t) = D^{|R|,t}(R, X) . \qquad (15)$$

The above equations can be interpreted intuitively as follows: the embedding $F^R(Q)$ of the query is the smallest edit distance matching R to a suffix of Q. The embedding $F^R(X, t)$ of database position (X, t) is the smallest edit distance matching R to a suffix of $X^{1:t}$. If a very close match to Q appears as $X^{s:t}$ in X, then we expect $F^R(Q)$ to be very similar to $F^R(X, t)$. Any sequence R used to define an embedding F^R is called a *reference sequence*.

4.2 Reference-based Bounds for the Edit Distance and Smith-Waterman

Let Q be a query string, X be the database sequence, and t be a position on X. As a reminder, $\Delta(A, B)$ is the edit distance between strings A and B, and $D^{|Q|,t}(Q, X)$ is the smallest edit distance between Q and any subsequence of X ending at position (X, t). To establish an exact reference-based filtering method for the subsequence matching problem, our first step is to establish a lower bound for $D^{|Q|,t}(Q, X)$ based on $F^{R_i}(Q)$ and $F^{R_i}(X, t)$, where R_i is any reference sequence.

PROPOSITION 1. *For any query Q, database position (X, t), and reference sequence R_i, define $lb_{ED}^{i,t}(Q)$ as follows.*

$$lb_{ED}^{i,t}(Q) = F^{R_i}(X, t) - F^{R_i}(Q). \tag{16}$$

Then, it holds that:

$$lb_{ED}^{i,t}(Q) \leq D^{|Q|,t}(Q, X), \tag{17}$$

and thus $lb_{ED}^{i,t}(Q)$ is a lower bound for the smallest possible edit distance between Q and a subsequence of X ending at (X, t).

Proof: First, we need to make the following auxiliary definitions:

$$M(A, B, t) = \operatorname{argmin}_{B^{s:t}|s=1,\ldots,t}\{\Delta(A, B^{s:t})\}, \tag{18}$$
$$Q' = M(R_i, Q, |Q|). \tag{19}$$

In words, $M(A, B, t)$ is the subsequence of B ending at position (B, t) that has the smallest edit distance from A, and Q' is the suffix of Q that has the smallest edit distance from R_i. Then, we can prove Proposition 1 as follows:

$$
\begin{aligned}
lb_{ED}^{i,t}(Q) &= F^{R_i}(X, t) - F^{R_i}(Q) & (20)\\
&= \Delta(R_i, M(R_i, X, t)) - \Delta(R_i, Q') & (21)\\
&\leq \Delta(R_i, M(Q', X, t)) - \Delta(R_i, Q')) & (22)\\
&\leq \Delta(M(Q', X, t), Q') & (23)\\
&\leq \Delta(M(Q, X, t), Q). & (24)
\end{aligned}
$$

To justify the above derivation, we note the following:

- $\Delta(R_i, M(R_i, X, t)) \leq \Delta(R_i, M(Q', X, t))$ since both $M(R_i, X, t)$ and $M(Q', X, t)$ are subsequences of X ending at (X, t), and $M(R_i, X, t)$ is defined as the subsequence of X ending at (X, t) that has the smallest distance with R_i.

- The edit distance is metric, so the triangle inequality holds, and $\Delta(R_i, M(Q', X, t)) - \Delta(R_i, Q') \leq \Delta(M(Q', X, t), Q')$.

- We can prove $\Delta(M(Q', X, t), Q') \leq \Delta(M(Q, X, t), Q)$ by considering that when we perform the minimal set of edit operations that convert Q to $M(Q, X, t)$, those same operations suffice to convert Q' (which is a suffix of Q) to a suffix of $M(Q, X, t)$. Therefore, the smallest possible edit distance between Q' and a subsequence of X ending at (X, t) cannot be greater than $\Delta(M(Q, X, t), Q)$.

□

If we are actually interested in retrieving optimal matches under the Smith-Waterman similarity measure, as opposed to the edit distance, we can easily convert the lower bound of the edit distance to an upper bound for Smith-Waterman. We can prove the following:

PROPOSITION 2. *For any query Q and database position (X, t), define $ub_{SW}^{i,t}(Q)$ as follows:*

$$ub_{SW}^{i,t}(Q) = 2|Q| - lb_{ED}^{i,t}(Q). \tag{25}$$

Suppose that we define a Smith-Waterman similarity measure using $P_{\text{match}} = 2$, $P_{\text{gap}} = -1$, and $P_{\text{sub}} = -1$. Then, it holds that:

$$ub_{SW}^{i,t}(Q) \geq L^{|Q|,t}(Q, X), \tag{26}$$

where $L^{|Q|,t}(Q, X)$ is the highest Smith-Waterman score between Q and a subsequence of X ending at (X, t). Thus $ub_{SW}^{i,t}(Q)$ is an upper bound for the Smith-Waterman score between Q and any subsequence of X ending at (X, t).

Proof: First, we need to make the following auxiliary definition:

$$M_{SW}(Q, X, t) = \operatorname{argmax}_{X^{s:t}|s=1,\ldots,t}\{\Lambda(Q, X^{s:t})\}. \tag{27}$$

In words, $M_{SW}(Q, X, t)$ is the subsequence of X ending at position (X, t) that has the highest Smith-Waterman score with Q. Then, we can prove Proposition 2 as follows:

$$
\begin{aligned}
ub_{SW}^{i,t}(Q) &= 2|Q| - lb_{ED}^{i,t}(Q) & (28)\\
&\geq 2|Q| - \Delta(Q, M(Q, X, t)) & (29)\\
&\geq 2|Q| - \Delta(Q, M_{SW}(Q, X, t)) & (30)\\
&\geq L^{|Q|,t}(Q, X) & (31)
\end{aligned}
$$

In justifying the above derivation, the most important step is showing that $2|Q| - \Delta(Q, M_{SW}(Q, X, t)) \geq L^{|Q|,t}(Q, X)$. The argument for that is as follows: Consider the optimal alignment (according to Smith-Waterman) between Q and $M_{SW}(Q, X, t)$. If Q perfectly matches $M_{SW}(Q, X, t)$, then the alignment score is $2|Q|$, since we get a reward of $P_{\text{match}} = 2$ for every letter of Q. Any mismatch and gap in the optimal alignment causes the alignment score to decrement by at least 1. Therefore, we know that the number of mismatches and gaps in the optimal alignment cannot be greater than $2|Q| - L^{|Q|,t}(Q, X)$. At the same time, the optimal alignment between Q and $M_{SW}(Q, X, t)$ defines a sequence of edit operations (substitutions for mismatches and insertions or deletions for gaps) that converts Q to $M_{SW}(Q, X, t)$. Consequently, the edit distance between Q and $M_{SW}(Q, X, t)$) cannot exceed $2|Q| - L^{|Q|,t}(Q, X)$.

□

Notice that since Smith-Waterman is a similarity score (and not a distance measure) upper bounds established efficiently during a filtering step can be used to prune away candidate database matches, while guaranteeing that the correct answer will not be pruned. This is quite analogous to the use of lower bounds for efficient filtering when looking for the best matches under a distance measure.

4.3 Offline Selection of Reference Sequences

We have shown how to use reference-based alignment scores computed for database positions and for the query in order to obtain lower bounds of the edit distance or upper bounds for the Smith-Waterman similarity score between the query and subsequences ending at each database position. We say that, for a query Q, database position (X, j) is pruned using R_i, if $lb_{ED}^{i,j}(Q) > \delta q$, where $lb_{ED}^{i,j}(Q)$ is as defined in Eq. 16, and δ is the maximum amount (expressed as fraction of the query length) of difference between the query and its subsequence match that we are willing to tolerate. We note that, if the best match has an edit distance of more than δq from Q, we are not interested in retrieving that match.

The filter step of RBSA, which is described in Section 4.4, prunes database positions using information from reference sequences. However, given a query Q, it would take too much time to check for each database position if it can be pruned using every single reference sequence. Therefore, we perform an off-line preprocessing step, at which we identify, for every database position, the best reference

sequences (out of thousands of available sequences) to use for that position. Intuitively, reference sequences R for which $F^R(X, j)$ is high (meaning that R is far from any subsequence of X ending at position j) tend to provide tighter lower bounds according to Eq. 16. Our reference selection method is inspired by that of [42], although that approach was proposed in the context of full sequence matching.

For the reference selection process, we use two sets: 1) a set $\mathcal{Q}_{\text{sample}} = \{Q_1, \ldots, Q_{|\mathcal{Q}_{\text{sample}}|}\}$ of randomly generated queries with $|Q_i| = q$, and 2) a set of randomly generated reference objects $\mathcal{R} = \{R_1, \ldots, R_{|\mathcal{R}|}\}$ with $|R_i| = q$. For each database position (X, j), the set of reference objects to use for that position are selected using a greedy approach. More specifically, for each position (X, j), we first choose reference object R_j^1 to be the reference sequence R that prunes position (X, j) for the largest number of queries in $\mathcal{Q}_{\text{sample}}$. Then, the queries for which (X, j) is pruned by R_j^1 are removed from $\mathcal{Q}_{\text{sample}}$. Similarly, we choose reference object R_j^i to be the reference sequence R that prunes position (X, j) for the largest number of queries in $\mathcal{Q}_{\text{sample}}$, where $\mathcal{Q}_{\text{sample}}$ has been modified to exclude queries for which (X, j) is pruned using the previously chosen reference objects $R_j^1, \ldots, R_j^{i-1}$.

The final outcome is the set $\mathcal{R}^K = \{\mathcal{R}_1^K, \ldots, \mathcal{R}_{|X|}^K\}$, where $\mathcal{R}_j^K$ contains the top K reference objects for position (X, j). For each position (X, j) we also store all values $F^{R_j^i}(X, j)$, for $i = 1, \ldots, K$. The pseudocode for selecting reference objects for each database position is given in Algorithm 1. We should note that the selection of reference objects is an offline process and is executed only once.

4.4 Filter Step

Next we describe the online behavior of RBSA at query time, for queries of fixed size q. The retrieval process, given Q, consists of a filter step and a refine step. Given a query Q, its embeddings $F^{R_i}(Q)$ under all reference objects in $\mathcal{R}$ are computed. Then, for each database position (X, j), each $R_j^i \in \mathcal{R}_j^K$ is considered, until either an R_j^i is found that prunes (X, j), or all $R_j^i \in \mathcal{R}_j^K$ have been considered. In the latter case, position (X, j) is a candidate endpoint of a subsequence match, that will be considered by the refine step. The filter step is described in Algorithm 2.

4.5 Refine Step

The filter step produces set candidates that contains endpoints of possible database matches for the query. At the refine step, each of those candidates is evaluated. Naturally, depending on whether we want to retrieve the best matches according to the edit distance or Smith-Waterman, we use respectively the edit distance or Smith-Waterman to evaluate each candidate endpoint.

For the case of the edit distance, the refine step is shown in Algorithm 3. It is fairly straightforward to adapt that algorithm to work for the Smith-Waterman similarity measure.

4.6 Alphabet Collapsing

The filtering power of RBSA is improved by employing an alphabet collapsing technique. In particular, for the case of DNA sequences the alphabet is $\Sigma = \{A, C, G, T\}$. We can reduce the alphabet size to 2 by applying four possible collapsing schemes:

- Scheme 0: No collapsing (letters remain unchanged).

- Scheme 1: A and C map to X, G and T map to Y.

- Scheme 2: A and G map to X, C and T map to Y.

- Scheme 3: A and T map to X, C and G map to Y.

input : $\mathcal{Q}_{\text{sample}}$: a set of randomly generated queries.
$\mathcal{R}$: a set of reference objects.
$\{F^{R_i}(X, j)\}$: the embeddings of all positions (X, j) under all $R_i \in \mathcal{R}_j$.
X: database sequence.
δ: target dissimilarity percentage.
K: number of reference objects to be returned for each database position.

output : $\{\mathcal{R}_j^K\}$: for each database position (X, j), the set $\mathcal{R}_j^K$ of K reference objects to use for that position.

for $j = 1$ **to** $|X|$ **do**
    // initialize $\mathcal{R}_j^K$ to the empty set.
    $\mathcal{R}_j^K = \{\}$;
    // insert all queries into a list $\mathcal{Q}$.
    $\mathcal{Q} = list(\mathcal{Q}_{\text{sample}})$;
    for $r = 1$ **to** K **do**
        // initialize pruned to zero.
        pruned = $uchar[|\mathcal{R}|] = 0$;
        for each $R_i \in \mathcal{R}$ **do**
            for $k = 1$ **to** $|\mathcal{Q}|$ **do**
                // compute lower bound for the k_{th} query.
                if $(lb_{ED}^{i,j}(Q) > q\delta)$ **then** pruned[i]++;
            end
        end
        BestRef = null; BestPrune = -1;
        for $i = 1$ **to** $|\mathcal{R}|$ **do**
            if pruned[i] > BestPrune **then**
                BestPrune = pruned[i];
                BestRef = R_i;
            end
        end
        $\mathcal{R}_j^K = \mathcal{R}_j^K \cup \{\text{BestRef}\}$;
        // remove pruned queries from Q using QPrune
        $\mathcal{Q} = \text{EliminatePruned}(\mathcal{Q}, j, \text{BestRef})$;
    end
end

Algorithm 1. Selecting reference sequences per database position.

A combination of the four schemes is used to improve the filtering power of RBSA. Let T_i be a transformation function that converts an input string defined in alphabet Σ to its corresponding string defined in scheme i. In the offline selection of reference sequences for each database position (Section 4.3), each reference sequence $R \in \mathcal{R}$ eventually generates four different reference sequences: $T_0(R)$, $T_1(\mathcal{R})$, $T_2(\mathcal{R})$ and $T_3(\mathcal{R})$. The same transformations are also applied to the database thus producing $T_0(X)$, $T_1(X)$, $T_2(X)$ and $T_3(X)$.

Reference object $T_i(R)$ can be used to obtain bounds and prune database positions (X, j) by comparing $F^{T_i(R)}(T_i(Q))$ with $F^{T_i(R)}(T_i(X), j))$. Bounds obtained using any of the transformations T_i are still true for the untransformed sequences, since we can easily show that, for any of the four T_i's, the edit distance $\Delta(A, B) \geq \Delta(T_i(A), T_i(B))$. The offline process for reference selection considers each of the $T_i(R)$'s as a separate candidate reference sequence and typically chooses, for each database position, reference sequences obtained from all letter collapsing schemes.

At query time, the query Q is also converted into each of the four representations, $T_0(Q)$, $T_1(Q)$, $T_2(Q)$ and $T_3(Q)$. Filtering is modified to include these transformations. For each database position (X, j), lower bounds are computed for each T_i.

input   : Q: query.

            X: database sequence.

            δ: target dissimilarity percentage.

            $F^{\mathcal{R}}(Q) = \{F^{R_i}(Q)\}$: embeddings of query Q.

            $\{\mathcal{R}_j^K\}$: the set of reference sequences selected for position (X, j).

            $\{F^{R_j^i}(X, j)\}$: embedding of each database position (X, j) under each reference object $R_j^i \in \mathcal{R}_j^K$.

output   : candidates: database positions to be passed to the refine step.

```
// insert all database positions into list candidates.
candidates = {1, ..., |X|};
// define lower bound cut-off threshold.
threshold = qδ;
for i = 1 to K do
    for j = 1 to |X| do
        x = F^{R_j^i}(X,j) − F^{R_j^i}(Q);
        if x > threshold then
            candidates = candidates − {j};
        end
    end
end
```

Algorithm 2. Filtering with maximum pruning.

We have found empirically that we get more pruning power by combining bounds from the untransformed sequences and bounds from the transformed sequences obtained using letter collapsing. Reference objects obtained via letter collapsing have a larger variance in their distances to database subsequences, thus leading to better pruning. We should underline that in [42] it is also noted (in the context of full sequence matching) that pruning power improves when using reference objects whose distances to database sequences have higher variance, but that approach did not use letter collapsing.

5. RBSA FOR VARIABLE QUERY LENGTH

The discussion in Section 4 addressed the problem of efficient retrieval of subsequence matches for query sequences of fixed length q. In this section we describe how to build upon the solutions proposed for the fixed query length problem to obtain solutions for the variable query length problem. We assume that we have already prepared an index, as described in Section 4.3, for processing queries of fixed size q. In our experiments, $q = 40$.

Let Q be a query. In principle, Q can have arbitrary size, but for simplicity we assume that $|Q| = \alpha q$, for some $\alpha \in \mathbb{N}$. No constraints are placed on α, and α can be different for each query. At query time, the query is broken into non-overlapping segments $Q^1, \ldots, Q^\alpha$ of size q. We now proceed to describe two different methods, one exact, and one approximate, for using results obtained for the different segments Q^i in order to identify the subsequence match for the entire query.

5.1 Exact RBSA

The exact version of RBSA is based on a simple observation: if Q has a subsequence match with edit distance $\leq \delta|Q|$, then at least one of the query segments Q^i has a subsequence match with edit distance $\leq \delta q$. This can be seen by observing that each of the edit operations that transforms Q into its subsequence match is applied to one of the individual query segments. After all edit

input   : Q: query.

            X: database sequence.

            δ: target dissimilarity percentage.

            sorted: an array of candidate endpoints j, sorted in decreasing order of j.

output   : $(X, j_{\text{start}}), (X, j_{\text{end}})$: start and end point of estimated best alignment.

            distance: distance between Q and estimated best alignment.

            columns: number of database positions evaluated by the edit distance dynamic programming.

```
for i = 1 to |X| do
    unchecked[i] = 0;
end
for i = 1 to |sorted| do
    unchecked[sorted[i]] = 1;
end
distance = δ × |Q| + 1;
columns = 0;
n = |sorted|;
// main loop, check all candidates sorted[1], ..., sorted[n].
for k = 1 to n do
    candidate = sorted[k];
    if (unchecked[candidate] == 0) then continue;
    j = candidate + 1;
    for i = |Q| + 1 to 1 do
        cost[i][j] = ∞;
    end
    while (true) do
        j = j − 1;
        if (candidate − j ≥ |Q|δ + 1) then break;
        if (unchecked[j] == 1) then
            unchecked[j] = 0;
            candidate = j; // found another candidate endpoint.
            cost[|Q| + 1][j] = 0;
            endpoint[j + 1] = j;
        else
            cost[|Q| + 1][j] = ∞; // j is not a candidate endpoint.
        end
        for i = |Q| to 1 do
            previous = {(i + 1, j), (i, j + 1), (i + 1, j + 1)};
            (p_i, p_j) = argmin_{(a,b)∈previous} cost[a][b];
            cost[i][j] = D(Q_i, X_j) + cost[p_i][p_j];
            endpoint[i][j] = endpoint[p_i][p_j];
        end
        columns = columns + 1;
    end
end
```

Algorithm 3. The refine step for the edit distance.

operations have been applied, each query segment Q^i has been transformed to a database subsequence. If each query segment Q^i needed more than δq edit operations to be converted to its optimal database match, then the entire query would need more than $\alpha \delta q = \delta|Q|$ operations to be converted to its optimal database match.

Let $X^{s:t}$ be a subsequence match for the entire query Q, with distance $\leq \delta|Q|$. Then, we can show that there exists at least one Q_i that has, within $X^{s:t}$, a subsequence match $X^{s':t'}$ with distance $\leq \delta q$, and such that $t' \in \{t - q(\alpha - i) - \delta|Q|, \ldots, t - q(\alpha - i) + \delta|Q|\}$. Conversely, if for some segment Q^i we have found a match $X^{s':t'}$ with distance $\leq \delta q$, this generates a set of candidate

endpoints for a subsequence match of the entire query. This set of candidate endpoints is equal to $\{t' + q(\alpha - i) - \delta|Q|, \ldots, t + q(\alpha - i) + \delta|Q|\}$.

Let $sorted$ be the union of the sets of candidate endpoints generated from all matches of all segments Q^i, and let's assume that $sorted$ is sorted in descending order. Then, evaluating those candidate endpoints can be done by invoking Algorithm 3, i.e., the exact same algorithm that was used for the refine step of the fixed-query-length version. It should be clear from the preceding paragraphs that this algorithm is guaranteed to identify the correct subsequence match, as long as that match is within edit distance $\delta|Q|$ from Q. As in the fixed-length case, Algorithm 3 can easily be adapted to use Smith-Waterman instead of the edit distance, so as to identify the optimal Smith-Waterman match for the query (but still assuming an edit distance $\leq \delta|Q|$ from Q).

5.2 Approximate RBSA

In the exact version of RBSA we try to find subsequence matches within δq edit distance of each of the α query segments Q^i. An important question, whose answer forms the foundation of the approximate version of RBSA, is the following: what if, instead of using all segments Q^i, we used a single random Q^i? What would be the probability of the endpoint of the subsequence match for the entire query being included in the set of candidate endpoints generated by that single Q^i? It turns out, as we prove next, that under some fairly reasonable assumptions, this probability is at least 50%.

In order to prove the above claim, we need to make some assumptions about the distribution of edit operations needed to convert Q into its optimal subsequence match. We denote the best subsequence match of Q in X as $M(Q, X)$. Since we assume that $\Delta(Q, M(Q, X)) \leq \delta|Q|$, at most $\delta|Q|$ edit operations are needed to convert Q to $M(Q, X)$. Each of these edit operations is applied to one and only one of the α segments Q^i that the query has been partitioned to. We denote by Q^{c_m} the query segment where the m-th edit operation is applied, and by $P(c_m = i)$ the probability that the m-th edit operation is applied to segment Q^i.

PROPOSITION 3. *Let Q be a query, and $M(Q, X)$ be the optimal subsequence match of Q in X. We assume that $\Delta(Q, M(Q, X)) = n \leq \delta|Q|$, $\alpha \geq 4$, $P(c_m = i)$ is uniform over all i, and the distributions $P(c_m = i)$ corresponding to all m are mutually independent. In other words, we assume that the distribution of c_m does not depend on any c_n, for $n \neq m$. Consider the optimal sequence of edit operations that convert Q to $M(Q, X)$. Given any Q^i, there is a probability of at least 50% that, out of those edit operations, at most δq edit operations are applied to Q^i.*

Proof: The probability that exactly k out of the n edit operations are applied to Q^i follows a binomial distribution, where we have n trials, "success" is the case where an edit operation is applied to Q^i, and the probability of success for an individual trial (i.e., a specific edit operation) is $\frac{1}{\alpha}$. The expected number of successes over n trials is $\frac{n}{\alpha}$ (as a reminder, α is defined as $|Q|/q$). If $\alpha \geq 4$, as we assume, the probability of success is ≤ 0.25, and for that case it has been shown [12] that there is at least a 50% probability that the number of successes will not exceed the expected value n/α. Since $n \leq \delta|Q|$, it follows that $\frac{n}{\alpha} \leq \delta\frac{|Q|}{\alpha} = \delta q$, and the probability that at most δq edit operations are applied to Q^i is at least 50%.
□

Based on Proposition 3, by choosing a single Q^i, and generating candidate endpoints for the subsequence match of the entire

query based on subsequence matches retrieved for Q^i, we have a probability of at least 50% to include the correct endpoint (i.e., the endpoint of the optimal subsequence match for the entire query) in those candidates. If the correct point is not included in those candidates, it follows that more than δq edit operations were applied to Q^i. In that case, for any $j \neq i$, the probability that at most δq edit operations are applied to Q^j is still at least 50%, and it is actually higher now that we know that more than δq edit operations were applied to Q^i.

By extending that reasoning, if we generate candidate endpoints for the match of the entire Q using p segments $Q^{i_1}, \ldots, Q^{i_p}$, the probability of not including the correct endpoint in those candidates is at most $\frac{1}{2^p}$, and thus drops exponentially with respect to p. If the correct endpoint is indeed included in those candidates, then the optimal subsequence match is guaranteed to be identified using the same refine step as in exact RBSA, and as in Algorithm 3. In our experiments, we use $p = 10$, so that the probability of retrieving the correct result is at least 99.9%.

6. EXPERIMENTS

The performance of RBSA is evaluated on biological data obtained from the NCBI repository. RBSA is compared with state-of-the-art methods for string matching under the edit distance and the Smith-Waterman similarity measure. With respect to the edit distance, we have compared with Q-grams. With respect to Smith-Waterman, we have compared with:

- BLAST2[2]: the expect value E has been adjusted to achieve retrieval accuracies of 95%, 98% and 100%. In the tables and figures that follow, this adjustment is denoted as $BLASTX$, which means that the E values have been adjusted to guarantee $X\%$ retrieval accuracy compared to Smith-Waterman.

- BWT-SW[22]: a local alignment method that guarantees 100% retrieval accuracy.

For the purposes of the experimental evaluation, we denote the exact version of RBSA as E-RBSA, and the approximate version as A-RBSA. For notation purposes, the distance/similarity measure (edit distance (ED) or Smith-Waterman (SW)) used in the refine step of RBSA is added as a suffix at the end of each notation. For example, E-RBSA-ED is the exact version of RBSA using the edit distance at the refine step, whereas A-RBSA-SW denotes the approximate version of RBSA using Smith-Waterman at the refine step. In the following sections we use the term $RBSA$ to refer to our method in general. The other notation is only used to distinguish within different versions of RBSA when needed.

6.1 Datasets

RBSA has been tested on Human Chromosome 22. The size of this chromosome is 35,059,634 bases. For the experiments described in section 6.2.1, the database sequence consisted of the first 184,309 bases of the chromosome. For the rest of the experiments, the database sequence consisted of the whole chromosome, and thus had a length of 35,059,634 letters. Queries have been extracted from random chromosomes of the mouse genome. Their size varied from 40 to $10K$ nucleotides (i.e., 40 to $10K$ letters) and their similarity to the database varied within 5%, 10% and 15% edit operations, which as also discussed earlier is a reasonable range of δ values needed for the applications targeted by this paper. Several sets of queries have been created, one for each combination of the above parameters. Each set contains 200 queries.

6.1.1 Performance Measures

The two key measures of performance in this context are accuracy and efficiency. E-RBSA is **exact** meaning that it is always guaranteed to find the optimal match for each query. Hence its accuracy is always 100%. On the other hand, A-RBSA is **approximate**, therefore we use the term **Retrieval Accuracy (RA)** to express the percentage of the correct nearest neighbors found over the total number of queries. Efficiency is measured based on the **Retrieval Runtime Percentage (RRP)** for each query. RRP is defined as follows:

$$RRP \quad = \quad \frac{RBSA \ in \ sec}{brute \ force \ in \ sec} \ 100\%. \qquad (32)$$

For our experiments the brute-force case is the full dynamic programming algorithm. Efficiency is also measured based on the **cell cost** for each query, which is the percentage of database positions visited during the refine step.

Specifically, two sets of experiments have been performed: for the first set, the edit distance has been used at the refine step, whereas for the second we used the Smith-Waterman similarity measure. The system was implemented in C++, and run on an AMD Opteron 8220 SE processor running at 2.8GHz. For all the experiments, parameter K of Algorithm 1 was set to 50.

6.2 Experimental Results

First we show the experimental performance of RBSA when the edit distance is used at the refine step. In this case, the main competitors are the q-gram based methods. Then, we compare the performance of RBSA on Smith-Waterman against BLAST and BWT-SW. To provide a thorough experimental analysis we show the performance of RBSA considering the following factors: 1) the effect of letter collapsing, 2) the effect of query size and δ, and 3) the effect of the number of reference objects used for the filter step.

6.2.1 Edit Distance: Comparison with Q-grams

The major competitors in the case of edit distance are the q-gram based approaches. Their inefficiency for long queries with a relatively large deviation from the database has already been discussed earlier in this paper. In Table 1 we show that their pruning power deteriorates for queries of size larger than 100 and for values of δ that exceed 5%. For this experiment only, we used a small dataset that included the first 184,309 bases of Human Chromosome 22. The queries had a match within $\delta = 5\%$, 10% and 15%. The experiment was organized as follows: for each query size $|Q|$ we used a set of sliding windows $\mathcal{W}$ with size varying in $[|Q|(1 - \delta), \ |Q|(1 + \delta)]$. The database was scanned using $\mathcal{W}$ and all possible sequences were enumerated. For each query size and δ value we show the cell cost for the optimal q value. Clearly, for query sizes larger than 100 or δ values greater than 10%, the pruning power of q-grams deteriorates significantly, rendering them inappropriate for such string searches in large string databases. Due to this observation, we did not perform any further experiments with q-gram based state-of-the-art methods for subsequence matching. Also, an application of a full sequence matching q-gram based algorithm like [24, 25] would not work either as these algorithms are designed for full sequence matching (as pointed out by one of the authors of [24, 25]).

For the experiments described in the remainder of this section the database sequence is the whole Human Chromosome 22. Next, we show the performance of E-RBSA-ED in terms of retrieval runtime percentage and cell cost for various query sizes and various δ values on Human Chromosome 22. E-RBSA-ED is not significantly affected by the query size as regards its retrieval runtime

Cell cost of E-RBSA-ED vs. Q-grams				
Method	$|Q|$	δ=5%	δ=10%	δ=15%
Q-grams	20	2.1% (q=9)	8.2% (q=6)	28.4% (q=4)
RBSA	40	0.55%	1.02%	1.47%
Q-grams	40	3.2% (q=10)	9.3% (q=7)	31.9% (q=5)
Q-grams	100	15.3% (q=15)	27.4% (q=8)	58.8% (q=6)
RBSA	200	0.32%	0.89%	1.22%
Q-grams	200	32.9% (q=17)	45.5% (q=9)	73.7% (q=6)

Table 1: Cell cost of Q-grams vs. E-RBSA-ED (exact RBSA using edit distance at the refine step) for different query sizes and different values of δ. For E-RBSA-ED, alphabet collapsing has not been applied. For q-grams, the best q value for each case is shown. Notice that the database used in this experiment contains the first 184,309 nucleotides of Human Chromosome 22, i.e. $|X| = 184{,}309$.

RRP of E-RBSA-ED				
δ	$|Q|$=40	$|Q|$=200	$|Q|$=2,000	$|Q|$=10,000
15%	3.49%	3.50%	3.52%	3.56%
10%	0.89%	0.91%	0.91%	0.94%
5%	0.27%	0.28%	0.28%	0.29%

Cell cost of E-RBSA-ED				
δ	$|Q|$=40	$|Q|$=200	$|Q|$=2,000	$|Q|$=10,000
15%	1.12%	1.01%	0.87%	0.76%
10%	0.11%	0.10%	0.088%	0.077%
5%	0.01%	0.011%	0.009%	0.008%

Table 2: RRP and cell cost of E-RBSA-ED (exact RBSA using edit distance at the refine step) for various query sizes and various δ values without applying letter collapsing. The number of reference objects used at the filter step is 50.

percentage. We also note that larger query sizes lead to smaller cell cost. This behavior is expected since the longer the query size the more segments will be used for pruning; thus the pruning power increases. With respect to δ, it is clear that as the similarity of the query to the database increases, E-RBSA-ED improves both in terms of retrieval runtime percentage and cell cost. Table 2 summarizes the results.

6.2.2 Effect of Alphabet Collapsing

Table 3 shows how alphabest collapsing affects the performance of E-RBSA-ED and E-RBSA-SW. From the experiments we can see that applying alphabet collapsing can improve the performance of E-RBSA in most cases by factors of 1.3 and 1.55 or more, in terms of retrieval runtime percentage and cell cost respectively.

6.2.3 RBSA-SW: Comparison with BLAST and BWT-SW

For the remaining part of our experimental analysis we focus on the performance of RBSA on local alignment, i.e. when the Smith-Waterman similarity measure is used at the refine step. The performance of RBSA-SW is compared against two state-of-the-art local alignment methods, BLAST and BWT-SW, for various query sizes and δ values. We also show the significant improvement on both retrieval runtime percentage and cell cost for the approximate version of RBSA (i.e. A-RBSA-SW). For the following experiments, alphabet collapsing has been applied. Notice that A-RBSA-SW has not been studied for query sizes 40 and 200 since the number of possible chunks in both cases is extremely small to guarantee a high retrieval accuracy. Our findings are summarized in Table 4. For clarity purposes, the same results are also shown in Figure 1. It can be seen that A-RBSA outperforms BLAST by more than an order of magnitude for large queries (2,000 and 10,000). The

RRP of E-RBSA-ED with Alphabet Collapsing				
RBSA	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
Coll.	2.342%	2.386%	2.400%	2.473%
Uncoll.	3.49%	3.50%	3.52%	3.56%

Cell cost of E-RBSA-ED with Alphabet Collapsing				
RBSA	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
Coll.	0.735%	0.663%	0.571%	0.499%
Uncoll.	1.12%	1.01%	0.87%	0.76%

RRP of E-RBSA-SW with Alphabet Collapsing				
RBSA	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
Coll.	2.630%	2.679%	2.695%	2.777%
Uncoll.	3.579%	3.638%	3.660%	3.754%

Cell cost of E-RBSA-SW with Alphabet Collapsing				
RBSA	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
Coll.	0.826%	0.745%	0.641%	0.560%
Uncoll.	1.358%	1.224%	1.055%	0.921%

Table 3: RRP and cell cost of E-RBSA-ED (exact RBSA using edit distance at the refine step) and E-RBSA-SW (exact RBSA using Smith-Waterman at the refine step) for various query sizes and $\delta = 15\%$. The first column describes whether alphabet collapsing has been used (*Coll.*) or not (*Uncoll.*). The number of reference objects used at the filter step is 50.

retrieval accuracy of A-RBSA is $\geq 99.5\%$ for all the experiments described in this section. For $\delta = 15\%$ and 10%, A-RBSA has a retrieval accuracy of 99.5% when $|Q| = 2,000$, and 100% when $|Q| = 10,000$. For $\delta = 5\%$, A-RBSA achieves 100% accuracy for both query sizes. As regards BWT-SW, in terms of retrieval runtime percentage it outperforms BLAST and E-RBSA by over an order of magnitude for $|Q| = 40$ and is up to almost 3 times faster than BLAST for $|Q| = 200$. Its performance deteriorates, however, as $|Q|$ becomes larger.

6.2.4 RBSA-SW: Effect of the Number of Reference Objects used for Filtering

Experiments so far, assumed that 50 reference objects are assigned to each database position. In this section, we show the effect of the number of reference objects assigned per database point on the performance of RBSA-SW. We experiment on two query sizes, 200 and 2,000 with $\delta = 10\%$. Also for these experiments alphabet collapsing has been applied. Table 5 summarizes our findings with respect to retrieval runtime percentage and cell cost. Clearly as the number of reference objects decreases, both retrieval runtime percentage and cell cost deteriorate. In particular, if less that 30 reference objects are used, RBSA-SW outperforms the brute-force Smith-Waterman by a factor smaller than 3.5, and for 10 reference objects this factor is less than 2.

6.2.5 RBSA-SW: Experiment on Queries with Various δ Values

Finally we created a set of queries where δ varies from 1% to 15% in increments of 2%. Two query sizes have been studied, 200 and 2,000. We have created one query set per query size using different δ values. The total number of queries in each set is 400. Also, 50 reference objects have been used at the filter step. Results on retrieval runtime percentage and cell cost are summarized in Table 6. For the set of queries with size 2,000 we show the approximate version of RBSA. At the refine step we have used the Smith-Waterman similarity measure. For both query sizes, RBSA is at least one order of magnitude faster than BLAST and BWT-SW. The retrieval accuracy of A-RBSA is 99.75%.

RRP of RBSA-SW vs. BWT-SW and BLAST for $\delta = 15\%$				
Method	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
A-RBSA			0.476%	0.086%
E-RBSA	2.630%	2.679%	2.695%	2.777%
BWT-SW	0.34%	3.30%	8.63%	12.72%
BLAST95	11.17%	7.57%	7.46%	7.84%
BLAST98	16.34%	7.88%	7.60%	8.11%
BLAST100	19.35%	9.29%	8.20%	9.66%

RRP of RBSA-SW vs. BWT-SW and BLAST for $\delta = 10\%$				
Method	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
A-RBSA			0.087%	0.018%
E-RBSA	0.481%	0.490%	0.493%	0.508%
BWT-SW	0.204%	2.600%	6.889%	8.900%
BLAST95	4.623%	3.133%	3.086%	3.243%
BLAST98	6.783%	3.271%	3.155%	3.362%
BLAST100	8.251%	3.965%	3.498%	4.118%

RRP of RBSA-SW vs. BWT-SW and BLAST for $\delta = 5\%$				
Method	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
A-RBSA			0.019%	0.0053%
E-RBSA	0.106%	0.108%	0.109%	0.112%
BWT-SW	0.083%	0.688%	2.170%	5.460%
BLAST95	4.293%	2.910%	2.866%	3.011%
BLAST98	6.231%	3.005%	2.898%	3.089%
BLAST100	7.437%	3.573%	3.153%	3.711%

Cell cost of RBSA-SW vs. BWT-SW and BLAST for $\delta = 15\%$				
Method	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
A-RBSA			0.126%	0.024%
E-RBSA	0.826%	0.745%	0.641%	0.560%
BWT-SW	0.017%	1.298%	6.107%	7.347%
BLAST95	6.032%	3.972%	3.751%	4.641%
BLAST98	8.98%	4.73%	4.55%	5.56%
BLAST100	9.35%	5.87%	5.44%	6.6%

Cell cost of RBSA-SW vs. BWT-SW and BLAST for $\delta = 10\%$				
Method	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
A-RBSA			0.016	0.003%
E-RBSA	0.103%	0.093%	0.080%	0.070%
BWT-SW	0.015%	1.166%	5.483%	6.596%
BLAST95	4.974%	3.175%	2.793%	3.127%
BLAST98	7.917%	4.170%	4.011%	4.902%
BLAST100	9.862%	4.936%	4.574%	5.550%

Cell cost of RBSA-SW vs. BWT-SW and BLAST for $\delta = 5\%$				
Method	|Q|=40	|Q|=200	|Q|=2,000	|Q|=10,000
A-RBSA			0.001%	0.0002%
E-RBSA	0.010%	0.009%	0.008%	0.007%
BWT-SW	0.012%	0.911%	4.285%	5.155%
BLAST95	4.428%	2.397%	1.800%	2.512%
BLAST98	5.998%	3.216%	2.242%	3.123%
BLAST100	6.150%	4.583%	3.278%	3.479%

Table 4: RRP and cell cost of BLAST and BWT-SW vs. A-RBSA-SW (approximate RBSA using Smith-Waterman at the refine step) and E-RBSA-SW (exact RBSA using Smith-Waterman at the refine step). The number of reference objects used at the filter step is 50. Results are shown for $\delta = 15\%, 10\%,$ and 5%.

To summarize our findings, A-RBSA can support relatively large queries without significant loss in retrieval accuracy and outperforms current state-of-the-art local alignment methods (BLAST and BWT-SW) by over an order of magnitude in terms of retrieval runtime percentage. For completeness we should mention that the average retrieval runtime for the brute-force local alignment computation for queries of size 40, 200, 2,000 and 10,000 is 28.5, 132.4, 1317.8 and 6620.1 seconds respectively.
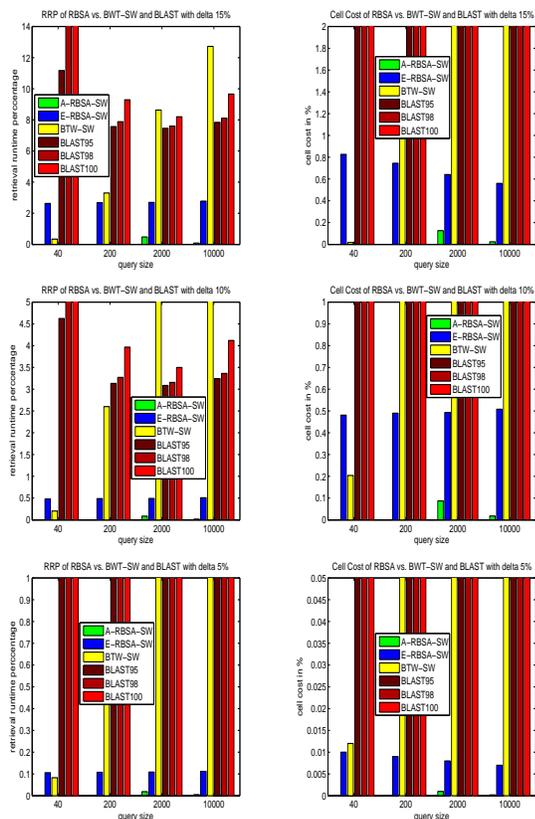
Figure 1: RRP (on left column) and cell cost (on right column) of BLAST and BWT-SW vs. A-RBSA-SW (approximate RBSA using Smith-Waterman at the refine step) and E-RBSA-SW (exact RBSA using Smith-Waterman at the refine step). The number of reference objects used at the filter step is 50. Also, $\delta = 15\%$ on top row, $\delta = 10\%$ on middle row, and $\delta = 5\%$ on bottom row. Notice that A-RBSA has only been applied for query sizes of $2,000$ and $10,000$ and for the latter it can be barely seen due to its low cost.

RRP and cell cost of RBSA-SW varying # of references				
	RRP		Cell Cost	
# of references	$|Q|$=200	$|Q|$=2,000	$|Q|$=200	$|Q|$=2,000
50	0.490%	0.493%	0.093%	0.080%
40	1.143%	1.149%	0.217%	0.187%
30	7.873%	7.920%	1.498%	1.290%
20	28.440%	28.609%	5.411%	4.661%
10	64.743%	65.126%	9.012%	8.931%

Table 5: RRP and cell cost of E-RBSA-SW (exact RBSA using Smith-Waterman at the refine step) varying the number of reference objects assigned to each database point.

7. DISCUSSION AND CONCLUSIONS

RBSA uses precomputed alignment scores between reference sequences and database positions to efficiently identify, given a query Q, a relatively small number of candidate subsequence matches in the database. RBSA has an exact version that is guaranteed to find the correct subsequence match, as long as that subsequence match has edit distance of at most $\delta|Q|$ to Q. In our experiments, for $|Q| \geq 200$, the exact version of RBSA outperforms state-of-the-art competitors such as BLAST, BWT, and q-grams.

Furthermore, we present an approximate version of RBSA that,

RRP and cell cost of RBSA-SW vs. BWT-SW and BLAST				
	RRP		Cell Cost	
Method	$|Q|$=200	$|Q|$=2,000	$|Q|$=200	$|Q|$=2,000
RBSA	0.530%	0.098%	0.088%	0.018%
BWT-SW	1.370%	2.958%	0.873%	4.233%
BLAST95	2.727%	2.406%	2.651%	2.640%
BLAST98	2.575%	2.483%	3.823%	3.815%
BLAST100	3.927%	3.304%	4.431%	4.454%

Table 6: RRP and cell cost of RBSA vs. competitors for variable δ values. For query size $2,000$ we have used A-RBSA (the approximate version of RBSA using Smith-Waterman at the refine step). The number of reference objects used at the filter step is 50.

for large queries, can efficiently identify candidate matches by considering only a relatively small number of fixed-size segments of Q. We show that, under some realistic assumptions, the probability of failing to retrieve the correct match, for approximate RBSA, drops exponentially with the number of query segments considered. It is important to note that the number of query segments needed to guarantee a certain probability of success is independent of the $|Q|$, making approximate RBSA scale very well with large query lengths. This version also achieves significant speedups over the exact version and produces speedups of one to two orders of magnitude compared to existing competitors, for $|Q| \geq 2,000$.

An open question is whether we can extend RBSA, so that it does not require matches to be within edit distance $\delta|Q|$ from Q. It will also be interesting to study more extensively the effects of letter collapsing, and analyze theoretically the reasons that letter collapsing improves performance. Finally, we believe that it may turn out to lead to more significant improvements in domains with larger alphabet sizes, such as proteins. We aim to explore these issues in future work.

8. REFERENCES

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[2] S. Altschul, T. Madden, R. Schffer, J.Zhang, Z.Zhang, W.Miller, and D. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, 25:3389–3402, 1997.

[3] V. Athitsos, P. Papapetrou, M. Potamias, G. Kollios, and D. Gunopulos. Approximate embedding-based subsequence matching of time series. In *ACM International Conference on Management of Data (SIGMOD)*, pages 365–378, 2008.

[4] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[5] S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron. q-gram based database searching using a suffix array (quasar). In *International Conference on Computational Molecular Biology (RECOMB)*, pages 77–83, 1999.

[6] S. Burkhardt and J. Kärkkäinen. Better filtering with gapped q-grams. *Fundam. Inf.*, 56(1,2):51–70, 2002.

[7] C. Cao, L. C. Shuai, and A. K. H. Tung. Indexing dna sequences using q-grams. *Database Systems for Advanced Applications*, 3453:4–16, 2005.

[8] X. Cao, B. C. Ooi, A. Tung, H. H. Pang, and K.-L. Tan. DSIM: A distance-based indexing method for genomic sequences. In *IEEE Symposium on Bioinformatics and Bioengineering (BIBE)*, pages 97–104, 2005.

[9] W. I. Chang and E. L. Lawler. Sublinear expected time approximate string matching and biological applications. Technical Report CSD-91-654, University of California at Berkeley, 1991.

[10] M. Crochemore and T. Lecroq. Pattern matching and text compression algorithms. In *ACM Computing Surveys*, 1996.

[11] E. Giladi, M. G. Walker, J. Z. Wang, and W. Volkmuth. SST: an algorithm for finding near-exact sequence matches in time proportional to the logarithm of the database size. *Bioinformatics*, 18(6):873–877, 2002.

[12] K. Hamza. The smallest uniform upper bound on the distance between the mean and the median of the binomial and poisson distributions. *Statistics and Probability Letters*, 23(1):21–25, 1995.

[13] X. Huang and K.-M. Chao. A generalized global alignment algorithm. *Bioinformatics*, 19(2):228–233, 2003.

[14] C. V. Jongeneel. Searching the expressed sequence tag (est) databases: Panning for genes. *Bioinformatics*, 1:76–92, 2000.

[15] K. J. Kalafus, A. R. Jackson, and A. Milosavljevic. Pash: Efficient genome-scale sequence anchoring by positional hashing. *Genome Resources*, 14(4):672678, 2004.

[16] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987.

[17] W. J. Kent. Resource BLAT-The BLAST-Like Alignment Tool. *Genome Research*, 2002.

[18] M.-S. Kim, K.-Y. Whang, J.-G. Lee, and M.-J. Lee. n-gram/2l: A space and time efficient two-level n-gram inverted index structure. In *International Conference on Very Large Data Bases (VLDB)*, pages 325–336, 2005.

[19] Y. J. Kim, A. Boyd, B. D. Athey, and J. M. Patel. miblast: scalable evaluation of a batch of nucleotide sequence queries with blast. *Nucleic Acids Res*, 33:4335–4344, 2005.

[20] D. E. Knuth. The art of computer programming. *Sorting and Searching*, 3, 1973.

[21] I. Korf and W. Gish. Mpblast : improved blast performance with multiplexed queries. *Bioinformatics*, 16:1052–1053, 2000.

[22] T. W. Lam, W. K. Sung, S. L. Tam, C. Wong, and S. Yiu. Compressed indexing and local alignment of dna. *Bioinformatics*, 24(6), 2008.

[23] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics*, 10(8):707–710, 1966.

[24] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *IEEE International Conference on Data Engineering (ICDE)*, pages 257–266, 2008.

[25] C. Li, B. Wang, and X. Yang. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *International Conference on Very Large Data Bases (VLDB)*, pages 303–314, 2007.

[26] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, March 1985.

[27] W. Litwin, R. Mokadem, P. Rigaux, and T. Schwarz. Fast ngram-based string search over data encoded using algebraic signatures. In *International Conference on Very Large Data Bases (VLDB)*, pages 207–218, 2007.

[28] U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 319–327, 1990.

[29] C. Meek, J. M. Patel, and S. Kasetty. Oasis: An online and accurate technique for local-alignment searches on biological sequences. In *International Conference on Very Large Data Bases (VLDB)*, pages 910–921, 2003.

[30] G. Myers. Whole-genome dna sequencing. *Computing in Science and Engg.*, pages 33–43, 1999.

[31] C. Navarro, E. Sutinen, J. Tanninen, and J. Tarhio. Indexing text with approximate q-grams. In *COM '00: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pages 350–363, 2000.

[32] G. Navarro and R. Baeza-Yates. A new indexing method for approximate string matching. In *Combinatorial Pattern Matching, 10th Annual Symposium*, pages 163–185, 1999.

[33] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal on Molecular Biology*, 48(3):443–53, 1970.

[34] Z. Ning, A. J. Cox, and J. C. Mullikin. SSAHA: A fast search method for large dna databases. *Genome Resources*, 11(10):1725–1729, 2001.

[35] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85(8):2444–2448, April 1988.

[36] B. Phoophakdee and M. J. Zaki. TRELLIS+: an effective approach for indexing genome-scale sequences using suffix trees. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pages 90–101, 2008.

[37] D. E. K. J. Pratt and R. Vaughan. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[38] G. J. Russell and J. H. Subak-Sharpe. Similarity of the general designs of protochordates and invertebrates. *Nature*, 266:533–536, 1977.

[39] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[40] E. Ukkonen. Algorithms for approximate string matching. *Information Control*, 64(1-3):100–118, 1985.

[41] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[42] J. Venkateswaran, D. Lachwani, T. Kahveci, and C. Jermaine. Reference-based indexing of sequence databases. In *International Conference on Very Large Databases (VLDB)*, pages 906–917, 2006.

[43] X. Yang, B. Wang, and C. Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *ACM International Conference on Management Of Data (SIGMOD)*, pages 353–364, 2008.

[44] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning dna sequences. *Journal of Computational Biology*, 7:203–214, 2000.