RRCA: Ultra-fast Multiple In-Species Genome Alignments

Sebastian Wandelt and Ulf Leser

Knowledge Management in Bioinformatics,
Humboldt-University of Berlin, Berlin, Germany
E-mail: {wandelt,leser}@informatik.hu-berlin.de

Abstract. Multiple sequence alignment is an important method in Bioinformatics, for instance, to reconstruct phylogenetic trees or for identifying functional domains within genes. Finding an optimal MSA is computationally intractable, and therefore many alignment heuristics were proposed. However, computing MSA for sequences at chromosome/genome scale in a reasonable time with good alignment results remains an open challenge.
In this paper we propose RRCA, a very fast method to compute high-quality in-species MSAs at genome scale. RRCA uses referential compression to efficiently find long common subsequences in to-be-aligned sequences. A colinear sub collection of these subsequences is used for an initial alignment and the not yet covered subsequences are aligned following the same approach recursively. Our evaluation shows that RRCA achieves MSAs at similar quality as current state-of-the-art methods, while often being orders of magnitude faster for all our datasets. For instance, RRCA aligns eight human Chromosome 22 (around 50 MB each) within one minute on a consumer computer; a task that takes hours to days with competitors.

Keywords: multiple sequence alignment, referential compression

1 Introduction

A multiple sequence alignment (MSA) arranges a set of sequences in a rectangular array such that one obtains the greatest number of similar characters in every column of the alignment with the minimal number of columns. An optimal MSA is usually computed following Carillo-Lipman [8] or generalizations based on dynamic programming. Since computing an optimal MSA is NP-complete under the most common cost models [33], the development of scalable approximate alignment methods, necessary in a context where information is growing by the day, is an open challenge [20]. The main application for MSA is in bioinformatics, where it is used, for instance, to reconstruct a phylogenetic tree [34] or for function/gene prediction [17]. Recently, several research results on multi-genome read mapping were published [16, 19, 31], most of which are based on the alignment of many long sequences (genomes or chromosomes) [16, 19]. Also recently, it was shown that very high compression ratios are possible when compressing aligned genome data [11]. Therefore, we believe that scalable sequence alignment (in terms of length and number of sequences) will become even more important in the future.

Practical implementations make use of heuristics to guide the assembly of a MSA (see [9, 28] for reviews and assessment of existing methods). Progressive alignment [23, 27] builds a MSA by combining pairwise sequence alignments (PSA), usually by aligning most similar pairs of sequences first. Hence, progressive alignment methods need a guide tree as an input or compute it on-the-fly. In contrast to progressive alignment, iterative alignment methods [15, 7] repeatedly re-align previously aligned sequences and thus often obtain a better score, because alignment errors obtained in the beginning of the alignment process can be repaired at later stages. A third class of heuristics, often used for PSA only, is based on chaining [36, 29]. The idea underlying

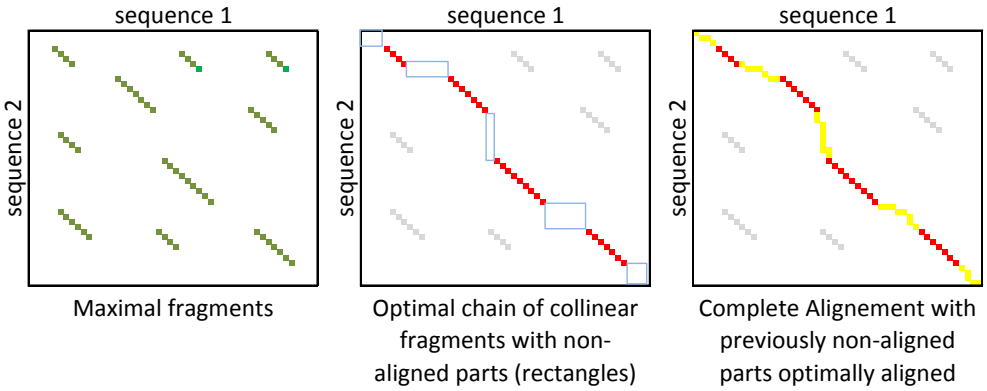

Fig. 1. Dot plot matrices for principal steps in chaining algorithms.

chaining algorithms is visualized in Figure 1: first a set of maximal identical fragments of both sequences is computed (Figure 1, left), then a colinear non-overlapping chain of fragments is identified (Figure 1, middle), and finally all the subsequences in between fragments are aligned following an (usually optimal) MSA algorithm (Figure 1, right); one example in the MSA field is Mugsy [6]. However, many chaining-based methods fail when it comes to long sequences, e.g. whole human chromosomes or MSA problems with more than a few sequences (see our evaluation). There exists further work on mixtures of the three approaches and also on slightly different problems, e.g. alignment-free sequence comparison [35] and alignment of short protein sequences [12].

Our solution: We propose RRCA, Recursive Referential Compression Alignment. Our technique is based on the idea of chaining and uses the referential compression framework FRESCO [32] to identify identical fragments within the collection of sequences. A non-overlapping chain of fragments (Step 2 of chaining in Figure 1) is identified by a greedy strategy, which always selects the longest non-overlapping fragment next. For Step 3 of Figure 1, i.e. aligning subsequences between chain-aligned fragments, we apply the same approach recursively, therefore the name *Recursive Referential Compression Alignment*. The recursion stops if either 1) all to-be-aligned sequences are equal or 2) all sequences are shorter than a given threshold. In the latter case an optimal MSA algorithm is applied.

Although our method is very intuitive and simple, we show that RRCA finds MSA with comparable quality scores as more sophisticated competitors. At the same time RRCA is often orders of magnitude faster alignment for biological and non-biological datasets. For instance, RRCA can align eight human Chromosome 22 (around 50 MB each) within one minute on a consumer computer; a task that takes hours to days with competitors. Besides, since our technique is based on a compressed representation of all sequences against a reference, RRCA does not have to store the complete collection of raw sequences in main memory during any step of the algorithm. In fact, the more similar the sequences, the more compressed is our representation during the execution of RRCA.

Structure: The structure of the paper is as follows. In Section 2, we define the problem of multiple sequence alignments. We show in Section 3 how RRCA computes a multiple sequence alignment by using referential compression. In Section 4, we evaluate our recursive referential compression algorithm. The paper is concluded in Section 5.

```
a_1: -----JANE*HAD*LOST*HER*JOB*AND*SHE--*-W-AS------*UNHAPPY.
a_2: WHEN*JANE*HAD*LOST*HER*JOB*SHE*-FELT*RE-ALLY----*UNHAPPY.
a_3: -----JANE*HAD*LOST*HER*JOB*AND*SHE--*RE-ALLY*WAS*UNHAPPY.
a_4: WHEN*JANE*HAD*LOST*HER*JOB*S----HE--*BECAME-----*UNHAPPY.
```

Fig. 2. Optimal alignment MSA_{EX} of C_{EX}.

2 Preliminaries

In the following, we present our recursive referential alignment algorithm RRCA. First, the multiple sequence alignment problem is defined.

Definition 1. *[Multiple Sequence Alignment]*
Given a collection $C = \{s_1, ..., s_n\}$ of sequences over an alphabet Σ, let $-$ be a symbol not in Σ. A multiple sequence alignment (MSA) *of C is a collection of sequences $\{a_1, ..., a_n\}$, such that $|a_1| = ... = |a_n|$ and each a_i is obtained from s_i by inserting any number of occurrences of symbol $-$. The term* column i of an alignment $\{a_1, ..., a_n\}$*, refers to the symbols $\{a_1(i), ..., a_n(i)\}$. The* length of an alignment *is the number of columns, i.e. the length of any sequence in the MSA. The special case of $n = 2$ is called* pairwise sequence alignment (PSA).

Example 1. A collection $C_{EX} = \{s_1, s_2, s_3, s_4\}$ contains four highly-similar sequences[1]:

s_1 : `JANE*HAD*LOST*HER*JOB*AND*SHE*WAS*UNHAPPY.`
s_2 : `WHEN*JANE*HAD*LOST*HER*JOB*SHE*FELT*REALLY*UNHAPPY.`
s_3 : `JANE*HAD*LOST*HER*JOB*AND*SHE*REALLY*WAS*UNHAPPY.`
s_4 : `WHEN*JANE*HAD*LOST*HER*JOB*SHE*BECAME*UNHAPPY.`

One multiple sequence alignment of C_{EX} is $MSA_{EX} = \{a_1, a_2, a_3, a_4\}$, shown in Figure 2: Column 2 of our example alignment is $\{-, H, -, H\}$.

Usually, one is interested in alignments that maximizing a given scoring function. An often used scoring function is sum-of-pairs [18]. A *scoring function* takes a pair of symbols from $\Sigma \cup \{-\}$ and returns a real number. Given a scoring function *score*, the *sum-of-pairs score for a column* $\{c_1, ..., c_n\}$ is defined as $\sum_{i<j\leq n} score(c_i, c_j)$. Given an MSA $\{a_1, ..., a_n\}$ of length m, the *sum-of-pairs score* is defined as the sum of the sum-of-pairs score for each column. A MSA is optimal for a collection of sequences $C = \{s_1, ..., s_n\}$, if there exists no other MSA for C with a higher score.

Example 2. Let *score* be defined as follows:

$$score(c_1, c_2) = \begin{cases} 1 & \text{if } c_1 \equiv c_2 \\ 0 & \text{if } c_1 \equiv - \wedge c_2 \equiv - \\ -1 & \text{if } else \end{cases}$$

The score of column $\{A, S, A, S\}$ is $1 * score(A, A) + 1 * score(S, S) + 2 * score(A, S) + 1 * score(S, A) = 1 + 1 + 2 * (-1) + 3 * (-1) = -1$.

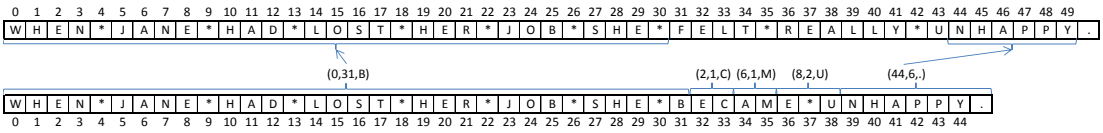

Fig. 3. Example for a referential compression of s_4 (below) against reference s_2 (up)

3 Computing an Alignment with Referential Compression

We present a method for the computation of an initial alignment, usually Step 1 of chaining-based MSA-approaches, i.e. computation of colinear fragments. We use a technique recently emerged in compression of biological sequences: Referential compression. Similar to dictionary-based techniques [37, 24], referential compression algorithms replace long subsequences of the to-be-compressed input with references to a distinct sequence, called reference. The reference is not part of the to-be-compressed input data. Furthermore, the reference is usually static, while dictionaries are being extended during compression phase. During the last years several referential compression algorithms emerged [22, 21, 13, 32]. These algorithms work best if the to-be-compressed sequences are similar to the reference sequence. Impressive results are reported when compressing large collections of sequences: referential compression algorithms achieve compression rates of up to 1000:1 for human genomes, i.e. more than 3 TB of raw data for 1092 genomes is compressed down to few GB, at compressions speeds close to maximum read speeds for state-of-the-art hard disks. We proceed with a formal definition of the referential compression algorithm from [32].

Definition 2 (Referential Compression [32]).
A referential match entry *(rme) is a triple* $\langle start, length, mismatch\rangle$*, where start is a number indicating the start of a match within a reference sequence, length denotes the match length*[2]*, and mismatch denotes a symbol. The* length *of a referential match entry rme, denoted* $|rme|$*, is* $length + 1$*. Given sequences s and a reference* ref*, a* referential compression *of s with respect to* ref*, is a list of referential match entries,* $[\langle s_1, l_1, m_1\rangle, ..., \langle s_n, l_n, m_n\rangle]$*, such that* $(ref(s_1, l_1) \circ m_1) \circ (ref(s_2, l_2) \circ m_2) \circ ... \circ (ref(s_n, l_n) \circ m_n) = s$*., where* $\circ$ *denotes the concatenation of two strings.*

The *offset* of a referential match entry rme_i in a referential compression $rc = [rme_1, ..., rme_n]$, denoted $offset(rc, rme_i)$, is defined as $\sum_{j<i} |rme_j|$. Given a rme $\langle start, length, mismatch\rangle$, we write the expression $(start, length, mismatch) \in rc$, if and only if $\langle start, length, mismatch\rangle$ is an element in the referential compression rc.

An algorithm for computing a referential compression is shown in Algorithm 1. To create a referential compression of input sequence s with respect to ref, the algorithm matches prefixes of s with subsequences of ref using a compressed suffix tree on ref. The longest such prefix is removed from s, encoded as a rme and added to rc. The algorithm terminates once s contains no more symbols. Algorithm 1 is a greedy algorithm, i.e. it always takes the longest prefix of the to-be-compressed which can be found in the reference. Any greedy algorithm computes a minimal representation, i.e. the size of the compressed sequence is minimal, if the dictionary for the reference is fixed and the size of a dictionary entry is constant [10].

[1] We use * instead of white-spaces for presentation purposes.
[2] Match length: Number of symbols for which to-be-compressed sequence and reference coincide.

Algorithm 1 Referential Compression Algorithm

Input: to-be-compressed sequence s and reference sequence ref
Output: referential compression rc of s with respect to ref

```
Let rc be an empty list
while |s| ≠ 0 do
    Let pre be the longest prefix of s occurring in ref, and let i be a position of an occurrence of pre in ref
    if s ≠ pre then
        Add ⟨i, |pre|, s(|pre|)⟩ to the end of rc
        Remove the first |pre| + 1 symbols from s
    else
        Add ⟨i, |pre| − 1, s(|pre| − 1)⟩ to the end of rc
        Remove the prefix pre from s
    end if
end while
```

Example 3. One example referential compression for Sequence s_4 with respect to the reference sequence s_2 is shown in Figure 3. The input is compressed into five referential match entries. The first referential match entry is $\langle 0, 31, B\rangle$ and describes a match for the first 31 characters of sequence s_4 at position 0 of the reference. The mismatch character is B (in the reference an F is found instead of a B). The offset of $\langle 6, 1, M\rangle$ is $|\langle 0, 31, B\rangle| + |\langle 2, 1, C\rangle| = 34$.

3.1 Computing an Initial Alignment

In RRCA, an initial alignment is a chain of colinear fragments, where large overlapping parts of all to-be-aligned the sequences are used as fragments (see Figure 1). These fragments can be obtained, for instance, by computing the longest common subsequences. However, often q-gram-based methods are used: all q-grams of to-be-aligned sequences are computed and then the longest chain of colinear q-grams is extracted. The process of computing and chaining these q-grams is highly time-consuming for long sequences, because a sequence of length n contains $n - q + 1$ q-grams. Even increasing q slightly does not make the problem easier to solve for long sequences. Moreover, if q is chosen too large then similarities between to-be-aligned sequences might be missed by the alignment algorithm.

We use referential compression for the computation of an initial alignment instead. Given a collection of to-be-aligned sequences, we pick one sequence as a reference ref and compress all sequences referentially against ref. Given the referential compressions of all sequences, we extract overlapping parts from the referential match entries, as a base for a chain of colinear fragments. The main advantage of our approach, compared to q-gram-based algorithms, is that referential match entries can represent arbitrary long sequences, and therefore arbitrary long fragments. This allows us to identify fragments with different degrees of similarity using a homogeneous approach, independent from a fixed value q. In our implementation we have always chosen the longest sequence as a reference. Given k sequences of maximum length n, finding the longest sequence takes $\mathrm{O}(k)$ and compression of all sequences against the reference takes $\mathrm{O}(k * n)$, since the compression is computed in linear time in the length [32].

Another advantage of using referential compression is as follows: We do not need to keep all uncompressed sequence in main memory at any time. For computing a referential compression of a sequence s, we only need s plus the reference sequence and an index over the reference sequence in main memory. After compression of s we proceed with the compression of remaining sequences, and only keep the compressed representations of previously compressed sequences in main memory. This is an important step towards alignment of many very long sequences on consumer computers. In Example 4, we show the referential compression of s_1 to s_4 (from Example 1) with s_2 as a reference.

Example 4. The longest sequence in C_{EX} is s_2. We obtain the following referential compressions $RC = \{rc_1, ..., rc_4\}$ for each sequence against s_2 as a reference:

$$rc_1 = \{(5, 22, A), (7, 1, D), (26, 5, W), (6, 1, S), (42, 8, .)\}$$
$$rc_2 = \{(0, 50, .)\}$$
$$rc_3 = \{(5, 22, A), (7, 1, D), (26, 5, R), (37, 6, W), (6, 1, S), (42, 8, .)\}$$
$$rc_4 = \{(0, 31, B), (2, 1, C), (6, 1, M), (8, 2, U), (44, 6, .)\}$$

Definition 3 (Alignment Fragments).
Given a collection of sequences $C = \{s_1, ..., s_n\}$, we say that $f = ((astart_1, ..., astart_n), alength)$ is a fragment for C*, if $s_1[astart_1, alength] = ... = s_n[astart_n, alength]$. Two fragments $f_1 = ((astart_{1,1}, ..., astart_{1,n}), alength_1)$ and $f_2 = ((astart_{2,1}, ..., astart_{2,n}), alength_2)$ are* strictly consecutive*, if $astart_{1,i} + alength_1 \leq astart_{2,i}$ for all $i \leq n$. An* initial alignment for C *is a collection of fragments $\{f_1, ..., f_m\}$ for C, such that each pair of fragments f_i and f_{i+1} is strictly consecutive.*

An initial alignment from Definition 3 splits a collection of sequences into different blocks, such that all sequences in C coincide for every second block (with unaligned blocks in between). Below, we describe how to compute an initial alignment by using referential compression. Intuitively, if two referential match entries overlap, i.e. point to the same subsequence of a reference, then the overlapping part is identical in referential match entries, and thus in their uncompressed sequences. We extract all intersections of referential match entries from all sequences in C. In Definition 4, we define an intersection operation on referential match entries in order to identify equal referenced subsequences.

Definition 4. *Given a collection R of referential match entries, $rme_1 = (s_1, l_1, m_1)$,..., $rme_n = (s_n, l_n, m_n)$, let $s = MAX(s_i)$ and $l = MIN(s_i + l_i) - MAX(s_i)$. The* intersection of R*, denoted $\bigcap_{i \leq n} rme_i$, is defined as the pair (s, l), if $l \geq 0$, and undefined otherwise.*

The result of intersecting $rme_1 = (42, 8, .)$ with $rme_2 = (44, 6, .)$ is the pair $(44, 6)$. The intersection between $rme_1 = (5, 22, A)$ with $rme_2 = (44, 6, .)$ is not defined, since they refer to different (non-overlapping) parts of the reference.

Definition 5 (Referential Agreement).
Given a set of referentially compressed sequences $RC = \{cs_1, ..., cs_n\}$, the referential agreement of RC *is defined as $RefAgree(RC) = \{(s, l) \mid \exists rme_1 \in cs_1, ..., rme_n \in cs_n.(s, l) = \bigcap_{i \leq n} rme_i\}$.*

Informally, the referential agreement of RC defines all the areas of the reference which are referenced by at least one referential match entry of *each* referentially compressed sequence. An upper bound for the time-complexity for computation of the referential agreement is quadratic in the number of referential match entries, since all referential match entries have to be intersected. We reduce the time complexity for this step to $O(k^2 * n * \log n)$ as follows. We create an interval tree [25] for each compressed sequence in $O(k * n)$ (with intervals defined by start and length of each referential match entry), and then find for each referential match entry (there are $O(k * n)$ such entries) its overlapping counterparts by probing k interval trees in $O(k * \log n)$.

Given the set of agreements for RC, we compute partial alignments, which build the basis for an initial alignment. To compute the partial alignments, we need to trace back positions in the sequences which contributed to the referential agreement. The function $TRACE$ computes all such positions for an element of a referential agreement and a referentially compressed sequence.

Algorithm 2 Initial Alignment Algorithm

Input: Collection of sequences $C = \{s_1, ..., s_n\}$
Output: Collection of alignment fragments
1: Let $fragments = \emptyset$
2: Select one $s \in C$ as ref
3: Compress all $s_i \in C$ against ref. The result is $RC = \{cs_1, ..., cs_n\}$
4: Compute $RefAgree(RC)$
5: **for all** $(s, l) \in RefAgree(RC)$ **do**
6: Compute $TRACEALL((s, l), RC)$
7: Add fragment $(TRACEALL((s, l), RC), l)$ to $fragments$
8: **end for**
9: Sort $fragments$ by second component (i.e. length of the fragment)
10: Let $consfragments$ be a consistent sub collection of $fragments$
11: Sort $consfragments$ by second component (i.e. start positions of the fragment)
12: return $consfragments$

Definition 6 (TRACE).
Given a referential compression cs and a pair (s, l)*, we define* $TRACE((s, l), cs) = \{s - start_i \mid \exists length_i, mismatch_i.(start_i, length_i, mismatch_i) \in cs \wedge start_i \leq s \wedge s + l \leq start_i + length\}$.

The traces from compressed sequences need to be combined carefully, since several traces of a single compressed sequence might cause an overlap in the same region of the reference. For instance, a compressed sequence such as $\{(5, 10, A), (7, 12, .)\}$ references the same reference subsequence $ref(7, 8)$ two times. For the computation of an alignment, we only want to use one of the two referential match entries, either $(5, 10, A)$ or $(7, 12, .)$, once an overlap with another referential match entry at the same subsequence is found. The function TRACEALL in Definition 7 applies the following heuristic: We pick trace positions from each compressed sequence, such that the difference to the average of all traces is minimized. Thus, local subsequence matches are preferred over more distant matches.

Definition 7 (TRACEALL).
Given a set of referentially compressed sequences $RC = \{cs_1, ..., cs_n\}$ *and an element* (s, l) *of the referential agreement of RC, we let* $U = \bigcup_{i \leq n} TRACE((s, l), cs_i)$. $TRACEALL((s, l), RC) = \{p_1, ..., p_n\}$*, such that each* p_i *is the nearest value to* $AVG(U)$ *in* $TRACE((s, l), cs_i)$*. Note that there is at least one value in* $TRACE((s, l), cs_i)$ *for each* i*.*

Given Definition 7, we compute a set of partial alignments (of different quality). For computation of a complete initial alignment, we need to select a consistent (strictly consecutive) subset of these partial alignments. Our algorithm for computing an initial alignment by referential compression is shown in Algorithm 2. The consistent sub collection of $fragments$ (Line 10) is computed by always choosing the longest fragment next, i.e. starting form an empty set of fragments, we repeatedly add the longest not yet used consistent fragment, until no more consistent fragment is left. There exists multiple other heuristics. Finding an optimal chain of colinear non-overlapping fragments is exponential in the number of sequences [5]. Continuing Example 4, we have that $RefAgree(RC) = \{(5, 22), (26, 5), (44, 6)\}$. The initial alignment returned by Algorithm 2, is $\{((0, 5, 0, 5), 22), ((35, 44, 42, 39), 6), ((25, 26, 25, 26), 5)\}$. This initial alignment is shown in Figure 4.

3.2 Completing an Alignment with Recursive Referential Compression

In the previous subsection, we showed how to computed an initial alignment, based on referential compression. Chaining-based approaches usually compute an optimal alignment for subsequences not contained in any fragment, for instance the sequences 'WHEN*' and 'WHEN*' in Figure 4. Three reasons can cause these subsequences not to be part of an initial alignment in RRCA:

Algorithm 3 RRCA Algorithm

Input: Collection of sequences $C = \{s_1, ..., s_n\}$
Output: MSA for C
1: **if** $MAX_{i \leq n}(|s_i|) < \delta$ **then**
2: Return an optimal MSA of C
3: **else**
4: Let $fragments$ be an initial alignment of C
5: **if** $fragments$ is empty **then**
6: Return an optimal MSA of C
7: **else**
8: **for all** (non-empty) sequences S not aligned in $fragments$ **do**
9: Let $res_S = RRCA(S)$
10: **end for**
11: Return the (alternating) concatenation of all initially aligned fragments and recursively aligned sequence collections res_S
12: **end if**
13: **end if**

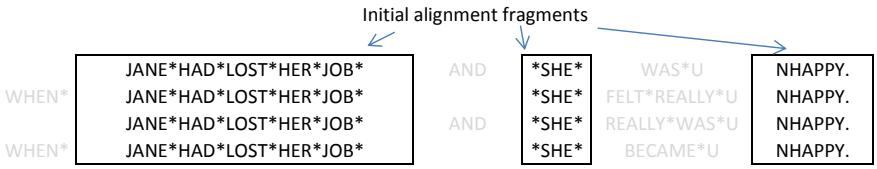

Fig. 4. Initial alignment for C_{EX}.

1. There is a larger insertion/deletion and *not all* sequences contribute a referential match entry.
2. Our greedy referential compression algorithm chose different ways to encode the same subsequences (equality of subsequences cannot be decided by referential agreement).
3. The subsequences are really just not similar.

Therefore, we propose a new strategy as follows: instead computing an optimal MSA of unaligned sequences directly, we repeat the computation of an initial alignment for unaligned (non-empty) subsequences. For instance, the two sequences 'WHEN*' and 'WHEN*' can be perfectly aligned by an initial alignment using referential compression with one of the two sequences as reference. In this case there is no need to compute an (computationally expensive) optimal alignment.

Our algorithm for recursively aligning referential compressions is shown in Algorithm 3. If the maximum length of a sequence in C is shorter than a fixed δ, then the algorithm computes an optimal MSA, following Needleman-Wunsch [26] (Line 1-2), and returns the result. Otherwise, an initial alignment following Algorithm 2 is computed (Line 4-12). If the initial alignment contains no fragment, i.e. referential compression cannot identify a common subsequence of all sequences in the input, then the algorithm computes and returns an optimal MSA as well (Line 6). If the initial alignment of the input contains at least one fragment, then the algorithm recursively computes a MSA for each set of subsequences not covered by fragments (Line 8-11).

Given the initial alignment from Figure 4, we have three blocks not covered by the initial alignment: one block containing two times 'WHEN*', one block containing two times 'AND' and one block before the fragment containing 'NHAPPY'. The first two blocks are aligned immediately by one recursive call each. The last block will be aligned by computing an optimal alignment, since no initial alignment can be found. The result of RRCA is optimal and shown in Figure 2.

			SeqAn		Mugsy		T-Coffee		Mafft		RRCA	
	Dataset	Length	Score	Time (s)	Score	Time (s)	Score	Time (s)	Score	Time (s)	Score	Time (s)
4 sequences	AT-1	500	2,958	0.1	2,958	0.5	2,958	0.1	2,958	0.1	2,958	0.0
	AT-1	16,000	95,712	174.5	95,712	0.8	95,712	109.8	95,712	0.4	95,712	0.0
	AT-1	2,048,000	*	*	12,216,048	23.6	*	*	12,213,022	431.5	12,216,012	0.6
	AT-1	30,000,000	*	*	*	*	*	*	*	*	178,138,742	24.1
	H-22	500	3,000	0.0	3,000	0.5	3,000	0.1	3,000	0.1	3,000	0.0
	H-22	16,000	95,884	86.6	95,884	0.6	95,884	72.5	95,884	0.3	95,884	0.0
	H-22	1,024,000	*	*	6,138,592	28.6	*	*	5,750,316	203.5	6,138,558	0.2
	H-22	30,000,000	*	*	*	*	*	*	*	*	179,407,238	12.8
	Y-wg	500	-1,040	0.2	-7,362	0.5	-2,330	0.1	-3,848	0.1	-1,370	0.0
	Y-wg	16,000	-7,948	290.8	-117,826	0.9	-100,066	148.2	-104,334	24.5	-119,590	0.0
	Y-wg	512,000	*	*	2,643,044	5.8	*	*	2,666,154	298.7	2,623,888	0.3
	Y-wg	8,192,000	*	*	43,999,160	169.1	*	*	*	*	43,086,892	6.6
8 sequences	AT-1	500	13,782	0.3	13,782	1.0	13,782	0.2	13,758	0.1	13,782	0.0
	AT-1	16,000	*	*	446,636	1.8	446,636	509.7	446,612	0.7	446,636	0.0
	AT-1	2,048,000	*	*	57,035,820	94.8	*	*	*	*	57,028,046	1.4
	AT-1	30,000,000	*	*	*	*	*	*	*	*	831,372,614	70.6
	H-22	500	14,000	0.0	14,000	1.0	14,000	0.2	14,000	0.1	14,000	0.0
	H-22	16,000	447,508	592.7	447,508	1.8	447,508	497.6	447,508	0.7	447,508	0.0
	H-22	2,048,000	*	*	56,965,626	265.0	*	*	*	*	56,912,358	0.6
	H-22	30,000,000	*	*	*	*	*	*	*	*	837,014,656	17.3
	Y-wg	500	-12,756	1.0	-63,924	1.0	-18,070	0.9	-18,048	0.2	-12,756	0.0
	Y-wg	16,000	*	*	-639,344	2.2	-603,404	737.5	-575,412	59.3	-416,700	0.1
	Y-wg	2,048,000	*	*	46,053,822	175.0	*	*	*	*	43,192,182	4.4
	Y-wg	8,192,000	*	*	*	*	*	*	*	*	197059310	15.5

Fig. 5. Comparison of MSA-methods for biological datasets (time in seconds). The score is computed as sum-of-pairs of the computed MSA with (match=1, mismatch=-1, gap=-1); larger scores are better. Computations that did not finish on time are marked with *.

4 Discussion

In the following section, we evaluate our proposed scheme. All experiments were run on a computer with 16 GB RAM and Intel Core i7-2670QM. We evaluate our method on five different datasets with different degrees of similarity. Three biological datasets: a collection of eight human Chromosome 22 (H-22) of the 1000 Genome project [4], a collection of eight Chromosome 1 from Arabidopsis thaliana (AT-1), taken from the 1001 Genomes project [2], release GMINordborg2010[3], and a collection of eight yeast genomes [1] (Y-wg). We have chosen these species since their sequences have different degrees of inner-species similarity. In our experiments RRCA was set to always choose the longest sequence as a reference. If all sequences have the same length, as initially in our experiment, one sequence is chosen randomly.

We compare RRCA against one optimal MSA algorithm (part of SeqAn [14]) and three approximate solutions (Mugsy [6], T-Coffee [27], and Mafft [3]), in Figure 5. We ran tests on the biological datasets with different lengths. If a program took longer than 15 minutes to complete a test, it was stopped (indicated by a $*$ in Figure 5). It can be seen that the optimal algorithm can only compute a MSA for rather short sequences within 15 minutes. The score obtained by all algorithms is quite similar, with the exception of the least self-similar dataset Y-wg. Overall, RRCA is the fastest MSA algorithm for each single test case, usually orders of magnitude faster than all three approximate competitors.

We performed experiments regarding the exact alignment time of random sequences with an extension of Needleman-Wunsch to MSA, as implemented in Seqan. We generated 500 collections of k random sequences with a fixed length. The result for the alignment of the sequences with $k = 4$ and $k = 8$ is shown in Figure 6. We have used the symbolic regression solver Eureqa [30] to estimate a formula for the alignment time in ms, given input length and the number of sequences

[3] `http://1001genomes.org/data/GMI/GMINordborg2010/releases/current/`

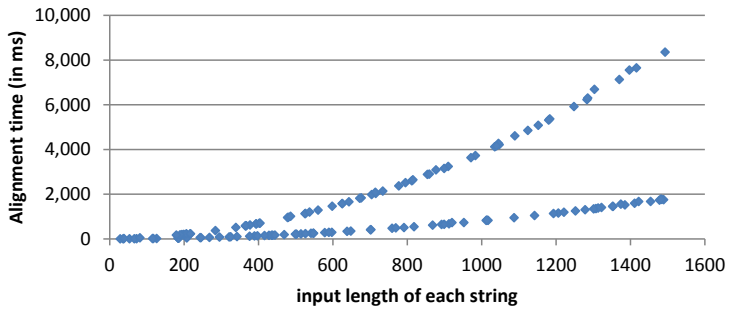

Fig. 6. Alignment of four (lower curve) and eight (upper curve) random sequences of different lengths with SeqAn.

k. The best solution with a size (number of terms) smaller than 10 is $time = 0.0000598 * length^2 * k^2$. This formula helps to estimate the alignment time, and thus, can be used to set the constant δ (the maximum length for exact alignment) from Algorithm 3. In our experiments with RRCA, we have set δ such that computing an optimal alignment in recursive call should not take longer than 100 ms, e.g. for $k = 8$, we obtained $\delta = 161.6$.

We analyzed how much time RRCA spends on different parts of the algorithm for aligning 10 human Chromosome 1 (total runtime was three minutes). Creating the index structure for references, i.e. initial reference and references in recursive calls, dominates the runtime (45.8%). The exact alignment of small fragments has the second highest share of the runtime(16.7%). Decompression of sequences (13.4%), compression of sequences (11.7%), and other parts of RRCA (12.2%) follow, respectively.

We have performed additional experiments for the alignment of protein sequences using benchmark BaliBase 3. Even for the most similar set of sequences (around 40% identity), RRCA cannot find good initial alignments and falls back to computing an optimal alignment. Similar substrings are not long enough in these short protein sequences and often only shared by small subsets of the whole collection. RRCA will not work for the alignment of sequences from different species, e.g. derived from human and mouse: Similar substrings are not long enough to exploit the benefit of referential compression. In addition, computing the referential compression of a mouse chromosome against a human chromosome is very time consuming. To sum up, alignment technique implemented RRCA cannot easily deal with large rearrangements and synteny. RRCA is tailored towards alignment of long sequences from the same species.

5 Conclusion

RRCA recursively computes a MSA using referential compression for fast identification of chaining fragments. We show that RRCA computes nearly optimal alignments for shorter sequences and for long sequences results with a similar score as competitors. RRCA is orders of magnitude faster than competitors and allows to align sequences within few seconds that take hours with other programs.

We see two major directions for future work. First, it should be investigated how to further improve MSA for very long sequences, in terms of alignment time and alignment quality. The key for improvement is 1) to extend our simple greedy strategy for selection of colinear fragments and 2) to find a heuristic for selecting a reference during the recursive compression step. Our results show that the selection of the longest reference already produces good results, but

more sophisticated strategies might yield alignments with higher scores. On the other hand, the run time of sophisticated techniques, which analyze (parts of) each sequence, will undoubtedly increase alignment time. Thus, selecting an efficient strategy for improving alignment times and alignment quality is challenging problem.

In addition, we think that it will be helpful to run an iterative alignment on top of RRCA to improve the quality (scores) of alignments. Second, running times of RRCA can be reduced by investigating different index structures for referential compression. It is important to note that the indexing time of the reference sequences is dominating the runtime (and not the lookup of matches alone). Thus we believe that a lightweight index structure, in terms of indexing time, can further decrease alignment times.

References

1. Overview of the yeast genome. Nature 387(6632 Suppl), 7–65 (May 1997), http://www.nature.com/doifinder/10.1038/42755
2. Whole-genome sequencing of multiple Arabidopsis thaliana populations. Nature Genetics 43(10), 956–963 (Aug 2011), http://dx.doi.org/10.1038/ng.911
3. MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability. Molecular Biology and Evolution 30(4), 772–780 (Apr 2013), http://dx.doi.org/10.1093/molbev/mst010
4. 1000 Genomes Project Consortium: A map of human genome variation from population-scale sequencing. Nature 467(7319), 1061–1073 (Oct 2010), http://dx.doi.org/10.1038/nature09534
5. Abouelhoda, M., Ohlebusch, E.: Multiple genome alignment: Chaining algorithms revisited. In: Baeza-Yates, R., Chvez, E., Crochemore, M. (eds.) Combinatorial Pattern Matching, Lecture Notes in Computer Science, vol. 2676, pp. 1–16. Springer Berlin Heidelberg (2003), http://dx.doi.org/10.1007/3-540-44888-8_1
6. Angiuoli, S.V., Salzberg, S.L.: Mugsy: fast multiple alignment of closely related whole genomes. Bioinformatics 27(3), 334–342 (2011)
7. Brudno, M., Chapman, M., Göttgens, B., Batzoglou, S., Morgenstern, B.: Fast and sensitive multiple alignment of large genomic sequences. BMC Bioinformatics 4, 66 (2003)
8. Carillo, H., Lipman, D.: The multiple sequence alignment problem in biology. SIAM Journal of Applied Math 48, 1073–1082 (1988)
9. Chen, X., Tompa, M.: Comparative assessment of methods for aligning multiple genome sequences. Nat Biotech 28(6), 567–572 (Jun 2010), http://dx.doi.org/10.1038/nbt.1637
10. Cohn, M., Khazan, R.: Parsing with prefix and suffix dictionaries. In: Data Compression Conference. pp. 180–189 (1996)
11. Deorowicz, S., Danek, A., Grabowski, S.: Genome compression: a novel approach for large collections. Bioinformatics 29(20), 2572–2578 (2013)
12. Deorowicz, S., Debudaj-Grabysz, A., Gudy, A.: Kalign-lcs a more accurate and faster variant of kalign2 algorithm for the multiple sequence alignment problem. In: Gruca, D.A., Czachrski, T., Kozielski, S. (eds.) Man-Machine Interactions 3, Advances in Intelligent Systems and Computing, vol. 242, pp. 495–502. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-02309-0_54
13. Deorowicz, S., Grabowski, S.: Robust Relative Compression of Genomes with Random Access. Bioinformatics (Oxford, England) (Sep 2011), http://dx.doi.org/10.1093/bioinformatics/btr505
14. Döring, A., Weese, D., Rausch, T., Reinert, K.: Seqan an efficient, generic C++ library for sequence analysis. BMC Bioinformatics 9 (2008)
15. Edgar, R.C.: Muscle: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics 5(1) (August 2004), http://dx.doi.org/10.1186/1471-2105-5-113
16. Ferrada, H., Gagie, T., Hirvola, T., Puglisi, S.J.: AliBI: An Alignment-Based Index for Genomic Datasets. ArXiv e-prints (Jul 2013)
17. Gross, S.S., Brent, M.R.: Using multiple alignments to improve gene prediction. In: J. Comput. Biol. pp. 379–393. Springer (2005)

18. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York, NY, USA (1997)
19. Huang, L., Popic, V., Batzoglou, S.: Short read alignment with populations of genomes. Bioinformatics (Oxford, England) 29(13), i361–i370 (Jul 2013), http://dx.doi.org/10.1093/bioinformatics/btt215
20. Kemena, C., Notredame, C.: Upcoming challenges for multiple sequence alignment methods in the high-throughput era. Bioinformatics 25(19), 2455–2465 (2009)
21. Kreft, S., Navarro, G.: Lz77-like compression with fast random access. In: Proceedings of the 2010 Data Compression Conference. pp. 239–248. DCC '10, IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/DCC.2010.29
22. Kuruppu, S., Puglisi, S., Zobel, J.: Optimized relative lempel-ziv compression of genomes. In: Australasian Computer Science Conference (2011)
23. Larkin, M., Blackshields, G., Brown: Clustal w and clustal x version 2.0. Bioinformatics 23(21), 2947–2948 (Nov 2007), http://dx.doi.org/10.1093/bioinformatics/btm404
24. Larsson, J., Moffat, A.: Offline dictionary-based compression. In: Proceedings of the IEEE Data Compression Conference. pp. 296–305 (Mar 1999)
25. McCreight, E.: Efficient algorithms for enumerating intersection intervals and rectangles. Tech. rep., Xerox Paolo Alte Research Center (1980)
26. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of molecular biology 48(3), 443–453 (Mar 1970), http://view.ncbi.nlm.nih.gov/pubmed/5420325
27. Notredame, C., Higgins, D.G., Heringa, J.: T-Coffee: A novel method for fast and accurate multiple sequence alignment. Journal of molecular biology 302(1), 205–217 (Sep 2000), http://dx.doi.org/10.1006/jmbi.2000.4042
28. Notredame, C.: Recent Evolutions of Multiple Sequence Alignment Algorithms. PLoS Computational Biology 3(8), e123+ (Aug 2007), http://dx.doi.org/10.1371/journal.pcbi.0030123
29. Roytberg, M., Gambin, A., Noe, L., Lasota, S., Furletova, E., Szczurek, E., Kucherov, G.: On subset seeds for protein alignment. IEEE/ACM Transactions on Computational Biology and Bioinformatics 6(3), 483–494 (Jul 2009), http://dx.doi.org/10.1109/TCBB.2009.4
30. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. Science 324(5923), 81–85 (2009)
31. Schneeberger, K., Hagmann, J., Ossowski, S., Warthmann, N., Gesing, S., Kohlbacher, O., Weigel, D.: Simultaneous alignment of short reads against multiple genomes. Genome biology 10(9), R98+ (Sep 2009), http://dx.doi.org/10.1186/gb-2009-10-9-r98
32. Wandelt, S., Leser, U.: FRESCO: Referential compression of highly-similar sequences. IEEE/ACM Transactions on Computational Biology and Bioinformatics 99(PrePrints), 1 (2013)
33. Wang, L., Jiang, T.: On the complexity of multiple sequence alignment. J. Comput. Biol. 1(4), 337–348 (1994), http://view.ncbi.nlm.nih.gov/pubmed/8790475
34. Wong, K.M., Suchard, M.A., Huelsenbeck, J.P.: Alignment Uncertainty and Genomic Analysis. Science 319(5862), 473–476 (Jan 2008), http://dx.doi.org/10.1126/science.1151532
35. Yu, H.J., Huang, D.S.: Normalized feature vectors: A novel alignment-free sequence comparison method based on the numbers of adjacent amino acids. IEEE/ACM Transactions on Computational Biology and Bioinformatics 10(2), 457–467 (Mar 2013), http://dx.doi.org/10.1109/TCBB.2013.10
36. Zhang, Z., Raghavachari, B., Hardison, R.C., Miller, W.: Chaining multiple-alignment blocks. Journal of Computational Biology 1(3), 217–226 (1994)
37. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23(3), 337–343 (1977)