Journal of Discrete Algorithms 10 (2012) 15–22

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

An algorithm for mapping short reads to a dynamically changing genomic sequence ☆

Costas S. Iliopoulos [a,b,*], Derrick Kourie [c], Laurent Mouchard [a,b,d], Themba K. Musombuka [c,e], Solon P. Pissis [a], Corne de Ridder [c,e]

[a] King's College London, Dept. of Informatics, Strand, London WC2R 2LS, UK
[b] Curtin University, Digital Ecosystems & Business Intelligence Institute, Center for Stringology & Applications, GPO Box U1987 Perth WA 6845, Australia
[c] University of Pretoria, Dept. of Computer Science, Pretoria 0002, South Africa
[d] University of Rouen, LITIS (EA 4108), System and Information Processing, 76821 Mont Saint Aignan Cedex, France
[e] University of South Africa, School of Computing, P.O. Box 392 UNISA 0003, South Africa

ARTICLE INFO

Article history:
Available online 11 August 2011

Keywords:
String algorithms
Next-generation sequencing
Mapping

ABSTRACT

Next-generation sequencing technologies have redefined the way genome sequencing is performed. They are able to produce tens of millions of short sequences (reads), during a single experiment, and with a much lower cost than previously possible. Due to the dramatic increase in the amount of data generated, a challenging task is to map (align) a set of reads to a reference genome. In this paper, we study a different version of this problem: mapping these reads to a dynamically changing genomic sequence. We propose a new practical algorithm, which employs a suitable data structure that takes into account potential dynamic effects (replacements, insertions, deletions) on the genomic sequence. The presented experimental results demonstrate that the proposed approach can be extended and applied to address the problem of mapping short reads to multiple related genomes.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

Sequencing technology has come a long way since the time when traditional sequencing techniques required many laboratories around the world to cooperate for years in order to sequence the human genome for the first time. The traditional Sanger sequencing methods [8,10,9], developed in the mid 70's, had been the workhorse technology for DNA sequencing for almost 30 years.

Nowadays, sequencing has been reduced to a matter of days or hours and the cost has decreased by many orders of magnitude, making it an accessible experimental procedure to many laboratories. The data resulting from a single sequencing experiment can be quite large, and it is not uncommon to have data from multiple experiments.

Many algorithms and programmes have been published recently to deal with the task of efficiently mapping the millions of short reads onto a single reference sequence [2,5–7]. However none of these algorithms addresses the inherent genomic variability between individuals, opting instead to simply treat it as mismatches, and punish the presence of differences accordingly. But most importantly, since the reads are quite short, even few changes in the sequence, as part of the natural diversity, can cause a read to seemingly best match with a different location of the reference than the one it actually

☆ A preliminary version of this paper appeared in the Proceedings of the International Conference on Bioinformatics & Biomedicine (BIBM 2010), 2010, pp. 133–136.

* Corresponding author at: King's College London, Dept. of Informatics, Strand, London WC2R 2LS, UK.
E-mail address: csi@dcs.kcl.ac.uk (C.S. Iliopoulos).

1570-8667/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.jda.2011.08.006

corresponds to, while others will fail to be mapped entirely. Misaligned reads in turn, lead to false identification of novel variation.

Very few programmes have been published to also take into account this natural variability. GenomeMapper [11] and GSNAP [12] address the issue by accepting a list of known variations and including them in their indexes. GenomeMapper uses an index with a graph structure, which consists of the reference sequence of one of the genomes, and a list of differences in the other genomes compared to the first one, to do the mapping. GSNAP implements the ability to align reads not just to a single reference sequence, but to a reference "space" of all possible combinations of major and minor alleles from databases like dbSNP.[1] However, none of the above programmes actually takes as input multiple reference sequences. Although the combination of a consensus reference and a list of known variants indeed covers the needs for polymorphism-aware mapping in humans, use of multiple full references could prove relevant in cases of other organisms, for which such ample data does not exist.

In this paper, we introduce a new approach to this problem. Accepting that the mutation rate between two random individuals is limited (0.1% on average for humans [4]), as well as the fact that two different assembled versions of a genomic sequence may differ in even fewer positions, we propose a new practical algorithm to address the problem of efficiently mapping short reads to a genomic sequence, which changes dynamically.

In particular, the proposed algorithm makes provision to accommodate dynamical changes that may occur in the reference sequence. With the increasing knowledge of variants, one could simply align against all known genomes for a species separately. This would come with the overhead of redundant alignments in conserved regions. Therefore, if a small number of differences (insertions, deletions, replacements) occur within the reference sequence, it is more appropriate to alter the already mapped reads onto the reference, dynamically. In order to represent the new changes, instead of starting to map the reads to a new related sequence again from scratch, we propose a faster approach, which encompasses a suitable data structure that will allow this flexibility and dynamic effects. Thus, the proposed algorithm can take as input either multiple reference sequences, or a single reference sequence and lists of differences in other sequences compared to the first one.

The remainder of the paper is structured as follows. Section 2 presents the basic definitions that are used throughout the paper. In Section 3, we formally define the problem solved in this paper. Section 4 presents a new practical algorithm for addressing the problem of efficiently mapping short reads to a dynamically changing genomic sequence. Finally, in Section 5 we present extensive experimental results, which demonstrate the importance of the proposed approach compared to more traditional approaches, and we briefly conclude in Section 6 with some future proposals.

2. Basic definitions

A *string* or *sequence* is a succession of zero or more symbols from an alphabet Σ of cardinality s; the string with zero symbols is denoted by ϵ. The set of all strings over the alphabet Σ including ϵ, is denoted by Σ^*. The set Σ^+ is defined as $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. A string x of length m is represented by $x[0..m-1]$, where $x[i] \in \Sigma$ for $0 \leqslant i < m$. The length of a string x is denoted by $|x|$. We say that Σ is *bounded* when s is a constant, *unbounded* otherwise. A string w is a factor of x if $x = uwv$ for $u, v \in \Sigma^*$. It is a *prefix* of x if u is empty and a *suffix* of x if v is empty.

Consider the sequences x and y with $x[i], y[i] \in \Sigma \cup \{\epsilon\}$. If $x[i] \neq y[i]$, then we say that $x[i]$ *differs* from $y[i]$. We distinguish among the following three types of differences:

1. A symbol of the first sequence corresponds to a different symbol of the second one, then we say that we have a *mismatch* between the two characters, i.e., $x[i] \neq y[i]$.
2. A symbol of the first sequence corresponds to "no symbol" of the second sequence, that is $x[i] \neq \epsilon$ and $y[i] = \epsilon$. This type of difference is called a *deletion*.
3. A symbol of the second sequence corresponds to "no symbol" of the first sequence, that is $x[i] = \epsilon$ and $y[i] \neq \epsilon$. This type of difference is called an *insertion*.

Another way of seeing this difference is that one can transform string x to y by performing a set of operations. The edit distance, $\delta_E(x, y)$, between strings x and y, is the minimum number of operations required to transform x into y. These operations are *Replacement* of a mismatched symbol, a *Deletion* or an *Insertion* of a symbol. The edit distance is symmetrical, and it holds $0 \leqslant \delta_E(x, y) \leqslant \max(|x|, |y|)$.

Let $t = t[0..n-1]$ and $x = x[0..m-1]$ with $m \leqslant n$. We say that x occurs at position q of t with at most k-differences (or equivalently, a *local alignment of x and t at position q with at most k-differences*), if $t[q..r]$, for some $r \geqslant q$, can be transformed into x by performing at most k of the following operations: inserting, deleting or replacing a symbol.

The Hamming distance δ_H is defined only for strings of the same length. For two strings x and y, $\delta_H(x, y)$ is the number of places in which the two strings differ, i.e. have different characters. The Hamming distance is symmetrical, and it holds $0 \leqslant \delta_H(x, y) \leqslant |x|$. For sake of completeness, we define $\delta_H(x, y) = \infty$ for strings x, y such that $|x| \neq |y|$.

[1] http://www.ncbi.nlm.nih.gov/projects/SNP/.

	0	1	2	3	4	5	6	7	8	9	10	11	$H.o$	$H.p$	$H.c$
t	G	C	A	G	T	A	C	A	G	T	A		-1	4	
	G	C	A	G	A	C	A	G	T	A			1	7	T
	G	C	A	G	A	C	T	A	G	T	A		1	7	C
	G	C	A	G	A	C	C	T	A	G	T	A	-1	8	
$\hat{t}$	G	C	A	G	A	C	C	T	A	T	A				

Fig. 1. Array of differences H.

3. Problem definition

We denote the generated short reads by the set $\{\rho_1, \rho_2, \ldots, \rho_r\}$, and we call them *patterns*. Notice that r is a very large integer number ($r > 10^6$). The length ℓ of each pattern, generated by the next-generation Illumina/Solexa Genome Analyzer, is currently typically between 25 and 100bp long. We denote the genomic sequence by $t = t[0..n-1]$, where $n > 10^6$, and we call it *text*.

We are now in a position to formally define the problem of mapping short reads to a dynamically changing genomic sequence as follows.

Problem 1. Given a set of patterns $\{\rho_1, \rho_2, \ldots, \rho_r\}$ of length ℓ, with $\rho_i \in \Sigma^*$, $\Sigma = \{A, C, G, T\}$ a bounded alphabet, and an integer threshold $h > 0$, find whether ρ_i, for all $1 \leqslant i \leqslant r$, occurs in text t of length n and/or in text $\hat{t}$, where $t, \hat{t} \in \Sigma^*$ and $\delta_E(t, \hat{t}) \leqslant h$.

We also denote the list (array) of differences as $H[0..h-1]$, where $0 < h \ll n$, similarly as in [11]. Array H stores triplets such that for each triplet $(o, p, c) \in H[i]$, where $0 \leqslant i < h$, $H[i].o$ represents the edit operation (0 for replacement, 1 for insertion, -1 for deletion) applied in position $H[i].p$ of t. In the case of replacement or insertion, $H[i].c$ represents the new symbol (base). The array is constructed in such a way that it is already sorted by $H[i].p$, i.e. $H[i].p \leqslant H[i+1].p$, for all $0 \leqslant i < h-1$. As an example, see Fig. 1. Notice that H describes how text t can evolve into text $\hat{t}$. In the case that $t = t[0..n-1]$, $\hat{t}$, and threshold h are given such that $\delta_E(t, \hat{t}) \leqslant h$, array H can be computed in $\mathcal{O}(hn)$ time and space [3].

4. The Dynamic-Mapping algorithm

The focus of this section is to describe a suitable data structure that will allow us to dynamically alter the already mapped patterns of a text. Thus, if we have a text t and a non-exact copy of t, say $\hat{t}$, then we want to change the already mapped patterns of t, to reflect the ones that are present in $\hat{t}$. Therefore, if the patterns have already been mapped to t, we want to avoid the mapping to $\hat{t}$ from scratch, but rather alter the already mapped patterns of t.

To contribute to the efficiency of the proposed procedure we will use word-level parallelism by storing factors of t and $\hat{t}$ into computer words. These words will be referred to as *signatures*. The signature $\sigma(x)$ of a string x is obtained by transforming the string to its binary equivalent. This is done by using 2-bits-per-base encoding of the DNA alphabet, and storing its decimal value into a computer word. The reason for using the signatures is that we can easily preprocess the text by indexing its factors in a data structure.

An outline of the Dynamic-Mapping algorithm is as follows.

(i) Preprocessing Phase

Without loss of generality, we split each factor of length ℓ of t, $t_i = t[i..i+\ell-1]$, into ν equal fragments $t_i^j = t[i + j\ell/\nu..i + (j+1)\ell/\nu - 1]$, for all $0 \leqslant i < n - \ell + 1$, $0 \leqslant j < \nu$. We build an array of linked lists $\mathcal{L}[s]$, for all $0 \leqslant s < 2^{2\ell/\nu}$. We compute $\sigma(t_i^j)$, the signature of t_i^j, and insert the couple (u, v) in $\mathcal{L}[\sigma(t_i^j)]$, where for each $(u, v) \in \mathcal{L}[\sigma(t_i^j)]$, u represents the starting position of t_i^j in t, and v indicates whether there exists a mapped pattern at $t[u - (\nu-1)\ell/\nu..u - (\nu-1)\ell/\nu + \ell - 1]$ ($v = 1$), or not ($v = 0$). Thus, the couples $(u, v) \in \mathcal{L}[s]$, for all $0 \leqslant s < 2^{2\ell/\nu}$, are sorted by u.

(ii) Pattern Mapping

The algorithm for mapping a pattern ρ to the text t is outlined in Algorithm 1. It matches all occurrences of ρ in t by updating $\mathcal{L}$ (setting $v = 1$ to the corresponding elements). In the case that Algorithm 1 returns *true*, then ρ is added to a new list $\mathcal{M}$ of mapped patterns. In the case that Algorithm 1 returns *false*, then ρ is added to a new list $\mathcal{U}$ of unmapped patterns. Note that function $f(j, d)$, $1 \leqslant j < \nu$, determines the positional distance Δ between two observations of signatures $\sigma(\rho^j)$ and $\sigma(\rho^{j-1})$ at slots d_j and d_{j-1} of the lists $\mathcal{L}[\sigma(\rho^j)]$ and $\mathcal{L}[\sigma(\rho^{j-1})]$, respectively, i.e. $\Delta = \mathcal{L}[\sigma(\rho^j)][d_j].u - \mathcal{L}[\sigma(\rho^{j-1})][d_{j-1}].u$.

In practice, for long patterns, e.g. $\ell = 72$, and small number of fragments, e.g. $\nu = 3$, it would be impractical to keep $\mathcal{L}$ in memory. Hence, the supported lengths of fragment range from 5 to 13, similarly as in [11] and [12]. As a result, the proposed approach for pattern mapping can only be applied for a prefix of the pattern. If the length of fragment is 12 and $\nu = 3$, we consider the prefix of length 36 of the pattern, as a seed, to do the mapping, and then align the suffix of the pattern to the corresponding factor of the text.

Input : $\mathcal{L}$, ρ, ℓ, ν
Output: *true* if ρ occurs in t

{Compute the signature of each fragment ρ^j of ρ}
for $(j \in [0, \nu-1]) \rightarrow (d_j, s^j) := (0, \sigma(\rho^j))$; **rof**
$(matched, j) := (false, 1)$;
do $(d_0 \leq |\mathcal{L}[s^0]|) \rightarrow$
  $\Delta := f(j, d)$;
  do $(d_j \leq |\mathcal{L}[s^j]|) \wedge (\Delta < \ell/\nu) \rightarrow$
    {Calculate the next positional distance Δ}
    $(d_j, \Delta) := (d_j + 1, f(j, d))$;
  od
  if $((\Delta = \ell/\nu) \wedge (j < \nu - 1)) \rightarrow$
    {$\mathcal{L}[s^m]$ found for $m = 0 \ldots j$}
    $j := j + 1$;
  ▯ $((\Delta = \ell/\nu) \wedge (j = \nu - 1)) \rightarrow$
    {$\mathcal{L}[s^m]$ found for $m = 0 \ldots \nu - 1$. Record match & continue}
    $\mathcal{L}[s^j][d_j].v := 1$;
    $(matched, j, d_0) := (true, 1, d_0 + 1)$;
  ▯ $(\Delta > \ell/\nu) \rightarrow$
    {$\mathcal{L}[s^j]$ not found. Match not possible from d_0. Restart from $d_0 + 1$}
    $(j, d_0) := (1, d_0 + 1)$;
  ▯ $(d_j \geq |\mathcal{L}[s^j]|)) \rightarrow$
    {No more matches possible. Terminate search}
    $(j, d_0) := (1, d_0 + 1)$;
  fi
od
return *matched*

Algorithm 1. The algorithm for mapping pattern ρ to the text t.

	0	1	2	3	4	5	6	7	8	9	10	11	*H.o*	*H.p*	*P*
String t	G	C	A	G	T	A	C	A	G	T	A		-1	4	-1
	G	C	A	G	A	C	A	G	T	A			1	7	0
	G	C	A	G	A	C	T	A	G	T	A		1	7	1
	G	C	A	G	A	C	C	T	A	G	T	A	-1	8	0
String $\hat{t}$	G	C	A	G	A	C	C	T	A	T	A				

Fig. 2. Array of prefix sums P.

	0	1	2	3	4	5	6	7	8	9	10
String t	G	C	**A**	**G**	**T**	**A**	**C**	**A**	**G**	**T**	**A**
String $\hat{t}$	G	C	A	G	A	C	C	T	A	T	A

Fig. 3. Affected fragments: $t[4-3+1+i..4+i]$, for all $0 \leqslant i < 3$, which are $t[2..4] = \text{AGT}$, $t[3..5] = \text{GTA}$ and $t[4..6] = \text{TAC}$, and $t[7-3+1+i..7+i]$, for all $0 \leqslant i < 2$, which are $t[5..7] = \text{ACA}$ and $t[6..8] = \text{CAG}$, and $t[7-3+1+i..7+i]$, for all $0 \leqslant i < 2$, which are $t[5..7] = \text{ACA}$ and $t[6..8] = \text{CAG}$, and $t[8-3+1+i..8+i]$, for all $0 \leqslant i < 3$, which are $t[6..8] = \text{CAG}$, $t[7..9] = \text{AGT}$ and $t[8..10] = \text{GTA}$.

(III) Dynamic Update

Assume that we have a new text $\hat{t}$, where $\delta_E(t, \hat{t}) \leqslant h$. We compute array H and a new array P, where $P[i] = \sum_{j=0}^{i-1} H[j].o$ represents the prefix sum of $H[i].o$, for all $0 \leqslant i < h$. As an example, see Fig. 2.

Assume that we have an edit operation $H[\lambda].o$, for some $0 \leqslant \lambda < h$, in position $H[\lambda].p = p$ of t. We compute the signatures of all the ℓ/ν fragments of t, affected by operation $H[\lambda].o$. Let s_j be the signature of the jth affected fragments of t, and $\mathcal{L}[s_j][q]$ the qth element of the linked list $\mathcal{L}[s_j]$. For each edit operation $H[\lambda].o$, for all $0 \leqslant \lambda < h$, the affected fragments are defined as follows.

- *Replacement*: $t[p - \ell/\nu + 1 + i..p + i]$, for all $0 \leqslant i < \ell/\nu$
- *Insertion*: $t[p - \ell/\nu + 1 + i..p + i]$, for all $0 \leqslant i < \ell/\nu - 1$
- *Deletion*: $t[p - \ell/\nu + 1 + i..p + i]$, for all $0 \leqslant i < \ell/\nu$

Similarly as in Algorithm 1, we find and delete the affected fragment from $\mathcal{L}[s_j]$, such that $\mathcal{L}[s_j][q].u = p - \ell/\nu + 1 + i$, to denote that it is no longer a fragment of $\hat{t}$. As an example, see Fig. 3 for $\ell = 6$ and $\nu = 2$, which uses the array H from Fig. 1.

	0	1	2	3	4	5	6	7	$H.p$	$H.o$	P
String t	C	A	T	G	**G**	**A**	**C**	A	1	0	0
	C	G	T	G	G	A	C	A	2	-1	-1
	C	G	G	G	A	C	A		7	1	0
String $\hat{t}$	C	G	G	**G**	**A**	**C**	G	A			

Fig. 4. New position of $t[4..6] = \text{GAC}$ is $P[\text{NEW}(H, p)] + p = P[1] + 4 = 3$.

	0	1	2	3	4	5	6	7	8	9	10
String t	G	C	A	G	T	A	C	A	G	T	A
String $\hat{t}$	G	C	**A**	**G**	**A**	**C**	**C**	**T**	**A**	**T**	**A**

Fig. 5. New fragments: $\hat{t}[4-1-i..4-1+3-1-i]$, for all $0 \leqslant i < 2$, which is $\hat{t}[3..5] = \text{GAC}$, $\hat{t}[2..4] = \text{AGA}$, and $\hat{t}[7+0-i-1..7+0+3-2-i]$, for all $0 \leqslant i < 3$, which is $\hat{t}[6..8] = \text{CTA}$, $\hat{t}[5..7] = \text{CCT}$, $\hat{t}[4..6] = \text{ACC}$, and $\hat{t}[7+1-i-1..7+1+3-2-i]$, for all $0 \leqslant i < 3$, which is $\hat{t}[7..9] = \text{TAT}$, $\hat{t}[6..8] = \text{CTA}$, $\hat{t}[5..7] = \text{CCT}$, and $\hat{t}[8+0-i..8+0+3-1-i]$, for all $0 \leqslant i < 2$, which is $\hat{t}[8..10] = \text{ATA}$, $\hat{t}[7..9] = \text{TAT}$.

If $\mathcal{L}[s_j][q].v = 1$, i.e. there exist mapped patterns on that fragment of t, then we need to unmap those patterns by adding them to the list of unmapped patterns $\mathcal{U}$. For each edit operation $H[\lambda].o$, for all $0 \leqslant \lambda < h$, the added patterns are defined as follows.

- *Replacement*: $t[p - \ell + 1 + i..p + i]$, for all $0 \leqslant i < \ell$
- *Insertion*: $t[p - \ell + 1 + i..p + i]$, for all $0 \leqslant i < \ell - 1$
- *Deletion*: $t[p - \ell + 1 + i..p + i]$, for all $0 \leqslant i < \ell$

Note that the unmapped patterns were initially mapped at fragment $t[p - \ell/\nu + 1 + i..p + i]$.

Let $NEW(H, p)$ be a binary-search operation that returns the maximum index i such that $H[i].p \leqslant p$. We compute and store in place the new position of each fragment in our structure. The new positions can be computed as $P[NEW(H, p)] + p$. As an example, see Fig. 4.

We compute the signatures of all the ℓ/ν new fragments of $\hat{t}$, affected by $H[\lambda].o$. Let s_j be the signature of the jth new fragment of $\hat{t}$. For each edit operation $H[\lambda].o$, for all $0 \leqslant \lambda < h$, the new fragments are defined as follows.

- *Replacement*: $\hat{t}[p + P[\lambda] - i..p + P[\lambda] + \ell/\nu - 1 - i]$, for all $0 \leqslant i < \ell/\nu$
- *Insertion*: $\hat{t}[p + P[\lambda] - 1 - i..p + P[\lambda] + \ell/\nu - 2 - i]$, for all $0 \leqslant i < \ell/\nu$
- *Deletion*: $\hat{t}[p + P[\lambda] - i..p + P[\lambda] + \ell/\nu - 2 - i]$, for all $0 \leqslant i < \ell/\nu - 1$

Similarly as in Algorithm 1, we insert the new fragment as the qth element $\mathcal{L}[s_j][q]$ of $\mathcal{L}[s_j]$, to denote that it is a fragment of $\hat{t}$, by preserving the ascending order of the positions in the elements. As an example, see Fig. 5 for $\ell = 6$ and $\nu = 2$, which uses the array H from Fig. 1.

(iv) Pattern Re-Mapping

The algorithm for re-mapping the patterns of list $\mathcal{U}$ to the new text $\hat{t}$ is identical to Algorithm 1.

Lemma 4.1. *Given a set of patterns $\{\rho_1, \rho_2, \ldots, \rho_r\}$ of length ℓ, a text $t = t[0..n-1]$, and the number of fragments ν, the* Dynamic-Mapping *algorithm finds whether ρ_i, for all $1 \leqslant i \leqslant r$, occurs in t, in time $\mathcal{O}(n + r\nu|Q|)$, where $|Q|$ is the size of the largest linked list in $\mathcal{L}$.*

Proof. By using the sliding window mechanism, we read $t[0..n-1]$ from left to right, calculating the signature $\sigma(t_i^j)$ of each factor of length ℓ/ν of t, $t_i^j = t[i + j\ell/\nu..i + (j+1)\ell/\nu - 1]$, for all $0 \leqslant i < n - \ell + 1$, $0 \leqslant j < \nu$, and we add the couple (u, v), $u = i + j\ell/\nu$, $v = 0$, as the last element of $\mathcal{L}[\sigma(t_i^j)]$ in time $\mathcal{O}(1)$. As soon as we compute $\sigma(t_0^0)$, then each of the rest signatures can be retrieved in constant time (using "shift"-type operation), resulting in a total of $\mathcal{O}(n)$ for the Preprocessing Phase. Trivially, the resulting lists of signatures are sorted by the position u in ascending order. Since each linked list $\mathcal{L}[s]$, for all $0 \leqslant s < 2^{2\ell/\nu}$, is sorted by u, the Pattern Mapping (see Algorithm 1) can be done in time $\mathcal{O}(\nu|Q|)$, i.e. the worst case is that all ν fragments of a pattern occur in the last elements of linked lists of size $|Q|$, resulting in a total time of $\mathcal{O}(r\nu|Q|)$. Hence, asymptotically, the total amount of time is $\mathcal{O}(n + r\nu|Q|)$. □

Lemma 4.2. *Given a set of patterns $\mathcal{M}$ of length ℓ mapped to $t = t[0..n-1]$, a set $\mathcal{U}$ of unmapped patterns, a text $\hat{t}$, and an integer threshold $h > 0$, such that $\delta_E(t, \hat{t}) \leqslant h$, the* Dynamic-Mapping *algorithm finds whether the patterns in $\mathcal{U} \cup \mathcal{M}$ occur in $\hat{t}$, in time $\mathcal{O}(hn + h\frac{\ell}{\nu}|Q| + |\mathcal{U}|\nu|Q|)$.*

Proof. The array H of the edit operations can be computed in $\mathcal{O}(hn)$ time [3]. The array P of the prefix sums can be computed in time $\mathcal{O}(h)$ from H. For each edit operation $H[i].o$, for all $0 \leqslant i < h$, we need to find and delete $\mathcal{O}(\frac{\ell}{\nu})$ elements from $\mathcal{L}$, add $\mathcal{O}(\ell)$ patterns to $\mathcal{U}$, and then add $\mathcal{O}(\frac{\ell}{\nu})$ new elements in $\mathcal{L}$. In addition, we compute the new position of

Table 1
Mapping 4,639,636 36bp-long simulated reads to four sequences of the E. coli chromosome (4,639,675bp), and 1,000,000 36bp-long simulated reads to four sequences of a mouse chromosome X region (1,000,000bp).

Reference	1st alignment	2nd alignment	3rd alignment	4th alignment
E. coli	97,61%	93,87%	97,61%	93,87%
Mouse	74,04%	71,05%	74,04%	71,05%

Table 2
The total number of reads to be mapped by the Dynamic-Mapping algorithm in each alignment.

Reads	1st alignment	2nd alignment	3rd alignment	4th alignment
E. coli	4,639,636	286,714	286,714	286,714
Mouse	1,000,000	178,004	177,962	178,004

each fragment of $\hat{t}$ in our structure by using a binary search operation on array P in time $\mathcal{O}(n \log h)$. These result in a total time of $\mathcal{O}(hn + h\frac{\ell}{\nu}|Q| + n \log h)$ for the Dynamic Update. The Pattern Re-Mapping (see Algorithm 1) can be done in time $\mathcal{O}(|\mathcal{U}|\nu|Q|)$. Hence, asymptotically, the total amount of time is $\mathcal{O}(hn + h\frac{\ell}{\nu}|Q| + |\mathcal{U}|\nu|Q|)$. □

Theorem 4.3. *The* Dynamic-Mapping *algorithm can solve Problem 1 in time* $\mathcal{O}(r\nu|Q| + hn + h\frac{\ell}{\nu}|Q|)$.

Proof. Immediate from Lemma 4.1 and Lemma 4.2. □

In the case that the array of differences H is given, the total complexity of Dynamic-Mapping algorithm can be reduced to $\mathcal{O}(r\nu|Q| + n \log h + h\frac{\ell}{\nu}|Q|)$.

Theorem 4.4. *The* Dynamic-Mapping *algorithm can solve Problem 1 using* $\mathcal{O}(2^{\ell/\nu} + hn - \ell/\nu + r\ell)$ *space.*

Proof. The array $\mathcal{L}$ consists of exactly $2^{2\ell/\nu}$ linked lists indexing the $2^{2\ell/\nu}$ possible signatures of length $2\ell/\nu$. A text of length n consists of exactly $n - \ell/\nu + 1$ factors of length ℓ/ν, and thus the total number of elements contained in the linked lists of $\mathcal{L}$ is exactly $n - \ell/\nu + 1$. The space required for keeping the r patterns (each of length ℓ) in memory is $r\ell$. Additionally, in the case that array H is not given, an extra hn space is required for computing array H [3]. □

5. Experimental results

The proposed algorithm was implemented in *C* programming language, and was developed under GNU/Linux operating system. The programme takes as input arguments, either multiple files of the reference sequences, or one file with a single reference sequence, all in FASTA format, and lists of differences of other sequences compared to the first one. In addition, it takes a file with the short reads in FASTA format, and then produces a SOAP-like tab-delimited text file with the hits, one for each sequence, as output.

The fact that we are interested in reporting a read only in a case that it occurs exactly once in the reference sequence (best hit), as well as the fact that the algorithm should also do the alignment on the reverse chain of the reference sequence, complicated the implementation of the algorithm. For instance, notice that a best hit on the first sequence is not necessarily a best hit on the second, since a potential difference in the second sequence may result in that read to be aligned more than once against the second sequence.

In order to validate the correctness of Dynamic-Mapping algorithm, we used two reference sequences in four arguments as input, such that the first argument is the same as the third, and the second the same as the fourth. The first reference sequence is the Escherichia coli strain K-12 substrain MG1655, obtained from GenBank database, and the second one is a simulated sequence generated by inserting 5000 random replacements in the first sequence. The short reads were obtained by simulating 4,639,636 36bp-long reads from the first sequence. In addition, to further validate the correctness of Dynamic-Mapping algorithm, we have conducted the same experiment to map 1,000,000 36bp-long simulated reads, generated from a mouse chromosome X 1Mbp region, obtained from the NCBI library, back to the same region they came from, and to a second reference sequence with 1000 mixed replacements, insertions and deletions.

The results in Table 1 demonstrate the correctness of Dynamic-Mapping algorithm in practice: the percentage of the mapped reads in the first and the second alignment is exactly the same as the third and the fourth, respectively. After the first alignment, which is done in Pattern Mapping, the rest of the alignments are done in Pattern Re-Mapping, which is based only on Dynamic Update, thus, avoiding to map all the reads from scratch. The main advantage of the proposed approach becomes evident in Table 2: the number of reads to be mapped decreases significantly after the first alignment.

In order to check the efficiency of Dynamic-Mapping algorithm, we compared its performance with three reference sequences, to the respective performance of three runs (one for each reference sequence) of SOAP2 (v2.20) [7], which

Table 3
Mapping 62,254,884 40bp-long simulated reads to three sequences of a mouse chromosome 16 region (62,254,923bp). Both programmes were run with 40bp-long seed, and reported exact best hits only.

Programme	Total time	1st alignment	2nd alignment	3rd alignment
SOAP2	3775 s	91,69%	88,26%	88,26%
Dynamic-Mapping	2158 s	91,69%	88,26%	88,26%

Table 4
The individual elapsed times of each step of the Dynamic-Mapping algorithm.

Step	Elapsed time
Preprocessing Phase	34 s
1st Alignment	1343 s
Dynamic Update	198 s
2nd Alignment	144 s
Dynamic Update	263 s
3rd Alignment	175 s

Table 5
Mapping 4,639,636 40bp-long simulated reads to two references of the E. coli chromosome (4,639,675bp) using a list of differences of the second sequence compared to the first.

Programme	Total time
GenomeMapper	70 s
Dynamic-Mapping	30 s

is, up-to-date, one of the most popular and efficient known read aligners. We used three reference sequences in three arguments as input; the first reference sequence is a mouse chromosome 16 region (62,254,923bp), obtained from the NCBI library, and the second and the third are simulated sequences generated by inserting 60,000 random replacements in the first sequence. The short reads were obtained by simulating 62,254,884 40bp-long reads from the first sequence. In each case, effort was made to make the two programmes run in as much similar way as possible, so that the speed and sensitivity comparisons are fair. Thus, SOAP2 was always given the modifier `-l <INT>` to adjust the seed length to be equal to the seed length of Dynamic-Mapping. Furthermore, the programmes were set to report only exact best (non-repetitive) matches, otherwise SOAP2 results would be chosen at random between equal hits. In SOAP2 this was achieved with the use of `-M 0 -r 0` modifiers.

As it is demonstrated by the results in Table 3, Dynamic-Mapping is able to complete the assignment much faster. Dynamic-Mapping finished in 2158 s, while SOAP2 in 3775 s. In terms of reported best hits, both programmes report the same percentage of alignment for all three reference sequences, a fact that further demonstrates the correctness of Dynamic-Mapping algorithm, in practice. In Table 4, the individual elapsed times of each step of the Dynamic-Mapping algorithm for the same experiment are reported. It is demonstrated that after the first alignment, the total time required for a dynamic update and a following alignment is much less than the time required for the first alignment.

As a last experiment, we compared the performance of Dynamic-Mapping algorithm with one reference and a list of differences, to the respective performance of GenomeMapper (v0.3) [11]. We used as input the E. coli chromosome, and as a list of differences 5000 mixed replacements, insertions and deletions. The short reads were obtained by simulating 4,639,636 40bp-long reads from the E. coli chromosome. In Table 5, it is demonstrated that Dynamic-Mapping is able to complete the assignment much faster.

The experiments were conducted on a desktop PC, using a single core of a 2.67 GHz Intel Core i7 920 CPU and 8 GB of main memory, running GNU/Linux operating system. The implementation is available at a website,[2] which is set up for maintaining the source code and the documentation. The datasets used in the presented experimental results are also available for further testing on the same website.

6. Conclusion

With the continuous increasing knowledge of variants between individuals, a simple method would be to map a set of reads against all known genomes for a species separately. There exist many different programmes for this task. However, this procedure will come with the overhead of redundant alignments in conserved regions.

[2] http://www.inf.kcl.ac.uk/pg/dynmap.

In this paper, we studied the problem of mapping short reads to a dynamically changing genomic sequence. The Dynamic-Mapping algorithm, firstly proposed in [1], makes provision to accommodate dynamical changes that may occur in the genomic sequence. Therefore, if there occur changes within the genomic sequence, the already mapped reads can be altered dynamically. In order to represent the new changes, instead of starting to map the reads to the new sequence again from scratch, we employ a suitable data structure that allows this flexibility and dynamic effects.

The presented experimental results demonstrate that the proposed approach can gain performance in comparison to more traditional approaches. We have demonstrated that it can match and outperform current popular software, such as SOAP2 and GenomeMapper, in terms of speed, while producing comparable numbers of hits. The presented experimental results are very promising, and they suggest that further research and development in this direction is desirable. The main contribution of this paper is the fact that the proposed algorithm can be extended and applied to address the problem of mapping short reads to multiple related genomes. This advantage should become much more drastic once hundreds of genomes are incorporated into the structure. This should improve the workflow, as the separate handling of separate references by the existing aligners would become increasingly impractical.

It is important to note that Dynamic-Mapping is still under development and several features will be made available soon. Our immediate target is to further optimise and extend our algorithm to accommodate a small number of mismatches within the reads. The data structure of the proposed algorithm is suitable for this extension. The idea of using the pigeonhole principle can be applied similarly as in [2]. The general idea for the k-mismatches problem is that inside any match of a pattern of length m, with at most k-mismatches, there must be at least $m - k$ symbols belonging to the pattern. In our case, by requiring some of the fragments to be exactly matched, the non-candidates can be excluded very quickly. For example, to admit two mismatches, a read can be split into four fragments. The two mismatches can exist in at most two of the fragments (at the same time). Then, if we try all six combinations of the two fragments as the seed, we can catch all hits with two mismatches.

Furthermore, building a complete software tool, which will be based on the presented algorithm, and will be used by biologists for mapping millions of short reads to multiple related genomes, is already on the way.

References

[1] T. Flouri, J. Holub, C.S. Iliopoulos, S.P. Pissis, An algorithm for mapping short reads to a dynamically changing genomic sequence, in: T. Park, S.K.-W. Tsui, L. Chen, M.K. Ng, L. Wong, X. Hu (Eds.), 2010 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2010, Proceedings, Hong Kong, China, 18–21 December 2010, pp. 133–136.
[2] K. Frousios, C.S. Iliopoulos, L. Mouchard, S.P. Pissis, G. Tischler, REAL: an efficient REad ALigner for next generation sequencing reads, in: Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology, BCB '10, ACM, New York, NY, USA, 2010, pp. 154–159.
[3] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, New York, NY, USA, 1997.
[4] L.B. Jorde, S.P. Wooding, Genetic variation, classification and race, Nature Genetics Supplement 36 (2004) S28–S33.
[5] B. Langmead, C. Trapnell, M. Pop, S.L. Salzberg, Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, Genome Biology 10 (2009) R25+.
[6] H. Li, R. Durbin, Fast and accurate short read alignment with Burrows–Wheeler transform, Bioinformatics 25 (2009) 1754–1760.
[7] R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, J. Wang, SOAP2: an improved ultrafast tool for short read alignment, Bioinformatics 25 (2009) 1966–1967.
[8] F. Sanger, A.R. Coulson, A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase, J. Mol. Biol. 94 (1975) 441–448.
[9] F. Sanger, G.M. Air, B.G. Barrell, N.L. Brown, A.R. Coulson, C.A. Fiddes, C.A. Hutchison, P.M. Slocombe, M. Smith, Nucleotide sequence of bacteriophage phi X174 DNA, Nature 265 (1977) 687–695.
[10] F. Sanger, S. Nicklen, A.R. Coulson, DNA sequencing with chain-terminating inhibitors, Proc. Natl. Acad. Sci. USA 74 (1977) 5463–5467.
[11] K. Schneeberger, J. Hagmann, S. Ossowski, N. Warthmann, S. Gesing, O. Kohlbacher, D. Weigel, Simultaneous alignment of short reads against multiple genomes, Genome Biology 10 (2009) R98+.
[12] T.D. Wu, S. Nacu, Fast and SNP-tolerant detection of complex variants and splicing in short reads, Bioinformatics 26 (2010) 873–881.