International Journal of Computer Applications (0975 8887)
Volume 86 - No. 8, January 2014

An Algorithm for Browsing the Referentially-compressed Genomes

Mohammad Nassef
Department of Computer Science
Faculty of Computers and Information
Cairo University, Egypt

Amr Badr
Department of Computer Science
Faculty of Computers and Information
Cairo University, Egypt

Ibrahim Farag
Department of Computer Science
Faculty of Computers and Information
Cairo University, Egypt

ABSTRACT

Genome resequencing produces enormous amount of data daily. Biologists need to frequently mine this data with the provided processing and storage resources. Therefore, it becomes very critical to professionally store this data in order to efficiently browse it in a frequent manner. Reference-based Compression algorithms (RbCs) showed significant genome compression results compared to the traditional text compression algorithms. By avoiding the complete decompression of the compressed genomes, they can be browsed by performing partial decompressions at specific regions, taking lower runtime and storage resources. This paper introduces the *inCompressi* algorithm that is designed and implemented to efficiently pick sequences from genomes, that are compressed by an existing Reference-based Compression algorithm (RbC), through partial decompressions. Moreover, inCompressi performs a more efficient complete genome decompression compared to the original decompression algorithm. The experimental results showed a significant reduction in both runtime and memory consumption compared to the original algorithm.

General Terms:

Bioinformatics, Compression, Data Management.

Keywords:

Reference-based Genome Compression; Partial Genome Decompression; Browsing Genome Sequences; inCompressi.

1. INTRODUCTION

Many genomic research projects have been established during the last decade to study the genomic variations between individuals from the same species. The HapMap Project [1], the 1000-Genomes Project [2], and the Personal Genome Project [3] are examples of these projects that are interested in the human genome. These projects work on huge feeds of hundreds or even thousands of genome sequences resulted from massive genome sequencing operations [4, 5]. These sequenced genomes need huge storage with high cost. The traditional text compression algorithms compress each genome individually, and so, they cannot exploit the clear similarities between similar genomes. Consequently, the overall storage space of similar genomes is still huge.

Challenged by these limitations, Reference-based Compression algorithms (RbCs) have emerged as a promising alternative. A class of these RbCs depends on Single Nucleotide Polymorphism (SNP) maps [6] to compress a target genome with respect to a specific reference genome. However, they are limited to the discovery and availability of SNP maps and other sequence variations. Another class of RbCs avoids the need to SNP maps by manually extracting the differences of a given target genome with respect to a specific reference genome. These differences may include substitutions, insertions, and deletions that should be applied to the reference genome (during decompression) to reproduce the target genome. Moreover, these RbCs vary in how exactly these differences are extracted, clustered, and encoded.

From one hand, some RbCs referentially compress multiple target genomes together into one dataset by exploiting their common similarities with respect to one or more reference genomes [7–9]. Each of these RbCs supports browsing of specific parts of the compressed genomes in a different way, and with noticeable overhead. For example, the RbC in [7] suffers from the overhead of aligning all the target genomes with respect to the reference genome. It then encodes and stores the differences of every target genome independent of each other, making it easy to directly decode the differences related to the queried sequences. Also, the Rbc in [8] spends high cost on artificially composing the reference genome with sequences that occur frequently between the target genomes. It then allows random access to these common sequences using different indexing techniques. Alternatively, the RbC in [9] divides every genome into blocks, and then compresses every block relative to matches with a selected reference genome. Specific regions of the compressed genomes can be browsed through partial decompressions of specific blocks.

On the other hand, other RbCs avoid the overhead resulting from compressing multiple genomes. They focus on the referential compression of just one target genome with respect to one reference genome. GRS [10] is a simple RbC that is able to extract differences below some threshold, thus, it cannot compress a target genome with excessive differences to a given reference genome. GReEn [11] is another RbC that is able to handle excessive differences efficiently by appling a complex probabilistic approach to determine the frequencies of these differences before encoding them.

Chern et al. at Stanford University [12] developed an RbC that is inspired by the sliding window of LZ77 [13]. Because the work done in this paper is particularly applied to their RbC, this paper will refer to their algorithm as the *"Stanford"* algorithm. Firstly, Stanford generates all matches between a target genome and a given reference genome. It then encodes them into matching instructions that can reproduce the target genome from the reference genome. After that, it starts merging these instructions by recording the differences occurring between every two subsequent instructions. By applying the Stanford algorithm to James Watson's (JW) genome with hg18 genome as a reference, Stanford compresses JW from 2,991 megabytes (MB) down to 6.99 MB, whereas GReEn compresses it into 18.23 MB. Without a reference genome, Gzip merely compresses JW to 834.8 MB. So, if one thousand human genomes are to be compressed, then they can

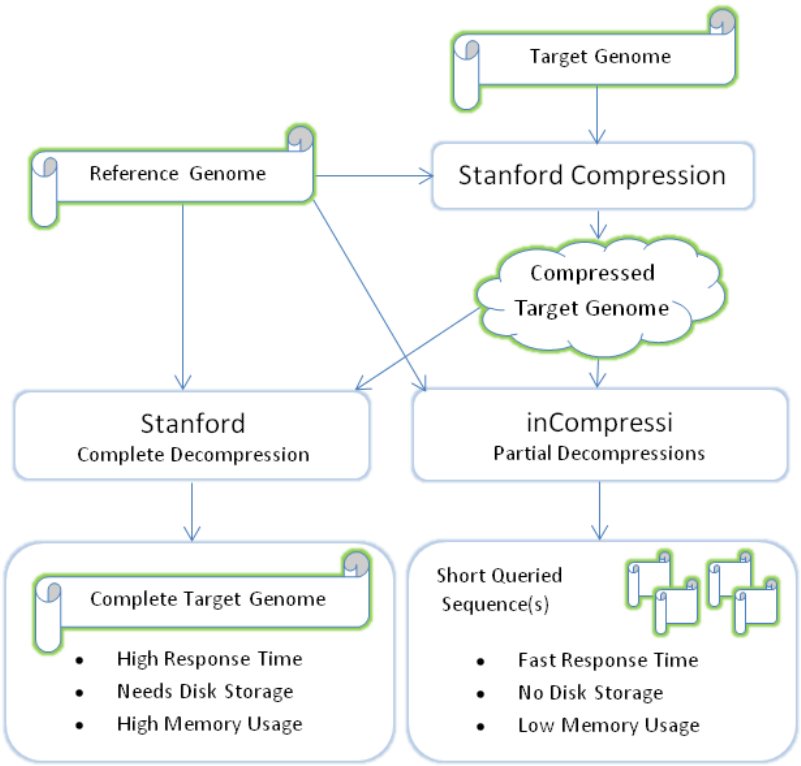

Fig. 1: *Briefing of how the Stanford's compression and decompression work, and how inCompressi can outperform the Stanford's decompression through performing partial decompressions.*

be compressed into 834,800 MB using Gzip. Conversely, Stanford can utilize the same space to referentially compress around 119,000 genomes in addition to one complete genome as a reference. Compared to GReEn, Stanford gave very promising compression ratios in many datasets.

This paper is organized as follows: Section 2 illustrates the functionality of Stanford, including detailed insights into its compression and decompression. Section 3 explores the design of the inCompressi algorithm that efficiently picks sequences from genomes that are referentially compressed using Stanford. Moreover, Section 3 illustrate how inCompressi performs a complete genome decompression compared to the original Stanford's decompression. Section 4 discusses the experimental results of the inCompressi's implementation. Section 5 concludes the paper.

2. STANFORD OVERVIEW

This section explains in details the compression and decompression of the Stanford algorithm that, as depicted by (Figure 1), the contribution of this paper is based on.

Because genome size is relatively big in most cases, it is usually available as individual FASTA-formatted chromosomes. So, Stanford compresses each individual chromosome of a given genome X with its corresponding chromosome of a given reference genome Y. After that, it gathers the compression results (differences) of all chromosomes to form the entire compressed genome.

Let X be the target chromosome to be referentially compressed by a reference chromosome Y. Chromosome X contains N characters: $X^N = X_1^N = \{X_1, ..., X_n\}$, and chromosome Y contains M characters: $Y^M = Y_1^M = \{Y_1, ..., Y_m\}$. Similarly, sequence X_i^j denotes characters $\{X_i, ..., X_j\}$, where $1 \leq i \leq j \leq n$, whereas sequence Y_a^b denotes characters $\{Y_a, ..., Y_b\}$, where $1 \leq a \leq b \leq m$. RC(X,Y) denotes the Stanford's Referential Compression of X with respect to Y:

$$Encoded(H_s, S_s, I_s, D_s) = RC(X, Y) \quad (1)$$

Stanford performs compression by encoding the longest possible matches (H_s) between X and Y (Figure 2), and similarly encoding the differences resulted from merging H_s whenever possible. Two sequences X_i^j and Y_a^b form the longest possible match if they have the same characters, and $X_{j+1} \neq Y_{b+1}$. Differences include substitutions (S_s), insertions (I_s), and deletions (D_s) to be applied to H_s while rebuilding X from Y through a complete Referential Decompression process:

$$X = RD(Y, Encoded(H_s, S_s, I_s, D_s)) \quad (2)$$

Stanford performs decompression by firstly decoding the matching instructions (H_s) and all the differences (S_s, I_s, D_s). It then starts building X by copying the matches H_s from Y, and applying all differences in their reversed order: (D_s, then I_s, and then S_s).

2.1 The Stanford's Compression

For some chromosome X to be referentially compressed by a specific reference chromosome Y, Stanford generates a collection of primitive match instructions (H_s). (Figure 2) shows an example of chromosomes X and Y. Those matches (H_s) will be used during decompression to initially build X from Y. Each instruction is formatted as H_i(Y_offset, length, X_char, Y_char). X_char and Y_char are respectively the different target and reference characters falling immediately after the match H_i, preventing it from being longer. (Figure 3.a) illustrates the initial instructions.

Most of the subsequent H_s are then possibly merged to form longer H_s while substitutions (S_s) being deduced. As in (Figure

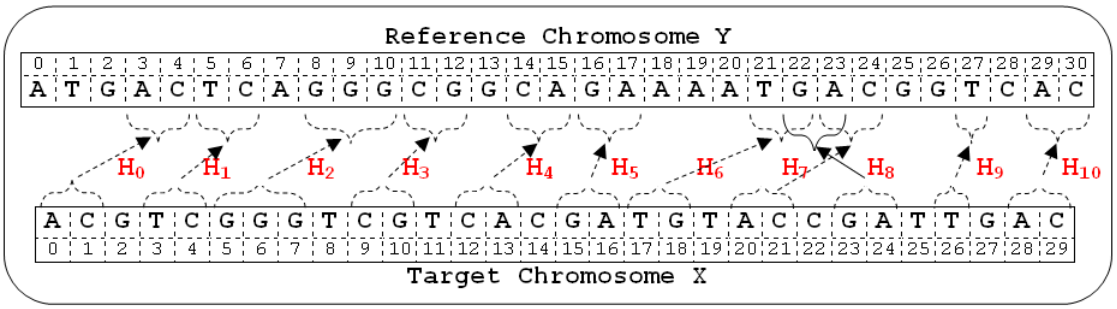

Fig. 2: *The matching instructions (H_s) of chromosome X with respect to chromosome Y.*

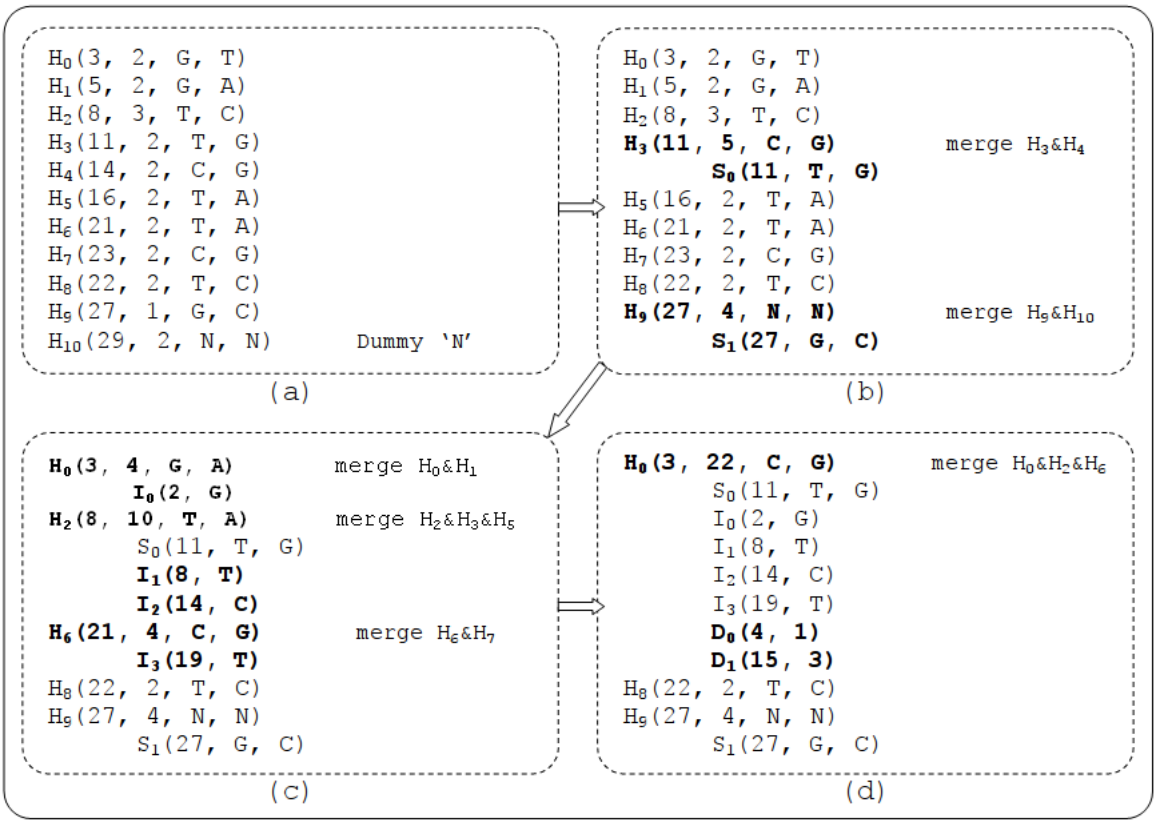

Fig. 3: *Reference-based Instructions and Target-based Differences generated during the Stanford's compression of sample chromosome X using sample chromosome Y as a reference. Both chromosomes are shown in (Figure 2). Bold items refer to the new or altered items.*

4.a), instructions $H_i(m_i, l_i, t_i, r_i)$ and $H_{i+1}(m_{i+1}, l_{i+1}, t_{i+1}, r_{i+1})$ can be merged if $m_i + l_i + 1 = m_{i+1}$. In other words, H_i and H_{i+1} were not combined in one match because the characters t_i and r_i are different. If so, H_{i+1} is merged with H_i, and the new H_i will be: $H_i(m_i, l_i+l_{i+1}+1, t_{i+1}, r_{i+1})$. The resulted substitution $S(n_i, t_i, r_i)$ is then appended to S_s, where n_i is the target offset of t_i. In (Figure 2), instruction H_4 could be merged with instruction H_3 after forming the substitution S_0 that contains the reference character 'G' which should be replaced with the target character 'T' during decompression. Similarly, instruction H_{10} is merged with instruction H_9 after forming the substitution S_1. (Figure 3.b) shows the updated list of instructions as well as the newly created substitutions.

Next, the remaining H_s are checked for extension with possible embedded insertions (I_s). As in (Figure 4.b), instructions $H_i(m_i, l_i, t_i, r_i)$ and $H_{i+1}(m_{i+1}, l_{i+1}, t_{i+1}, r_{i+1})$ can be merged if: $m_i+l_i = m_{i+1}$. This means that t_i prevented H_i and H_{i+1} from forming a whole match. Hence, Stanford merges H_{i+1} with H_i to be $H_i(m_i, l_i+l_{i+1}, t_{i+1}, r_{i+1})$, and a new insertion $I(n_i, t_i)$ is added to I_s, where n_i is the target offset of t_i. In (Figure 2), instructions H_0 and H_1 satisfy the insertion condition because they are adjacent to each other in the reference chromosome, whereas they have one character ('G') falling in between in the target chromosome. Hence, an insertion I_0 is created. (Figure 3.c) shows the updatd instructions as well as all the created insertions.

At last, a final round of merging H_s is done while exploring possible deletions (D_s). As in (Figure 4.c), two instructions $H_i(m_i, l_i, t_i, r_i)$ and $H_{i+1}(m_{i+1}, l_{i+1}, t_{i+1}, r_{i+1})$ could be merged if: $2 \leq m_{i+1} - (m_i + l_i) \leq max\_del$, where Max_del is the maximum allowed length for deletions. This means that r_i (and its successive characters occurring before H_{i+1}) prevented H_i and H_{i+1} from being combined into one match (Figure 4.c). In this

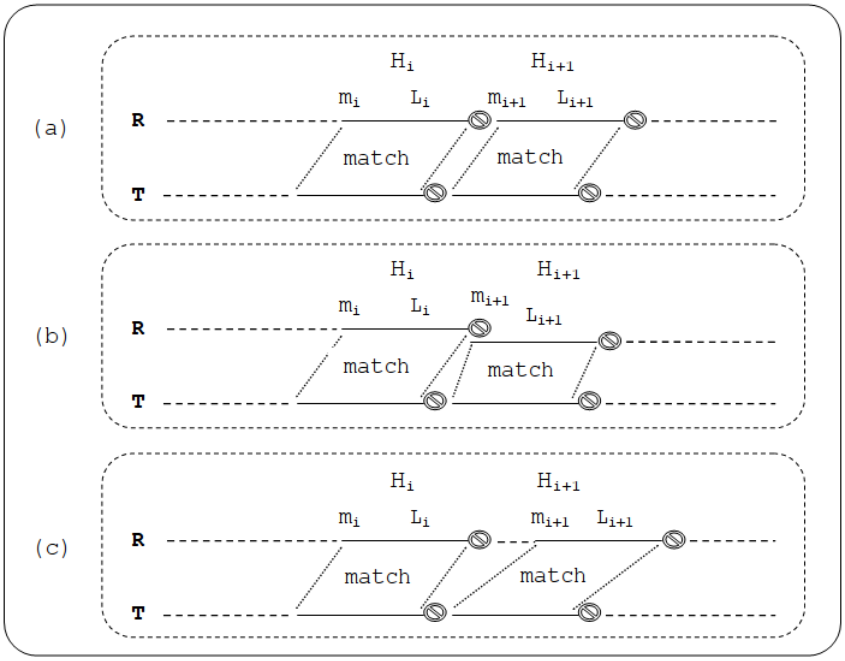

Fig. 4: *An example of (a) a substitution, (b) an insertion, and (c) a deletion that could result from merging two subsequent instructions during the Stanford's compression. R is a reference chromosome, and T is a target chromosome. Instructions H_i and H_{i+1} refer to matches of T with respect to R. The "No" symbol denotes the different ending characters in T and R successive to each match instruction.*

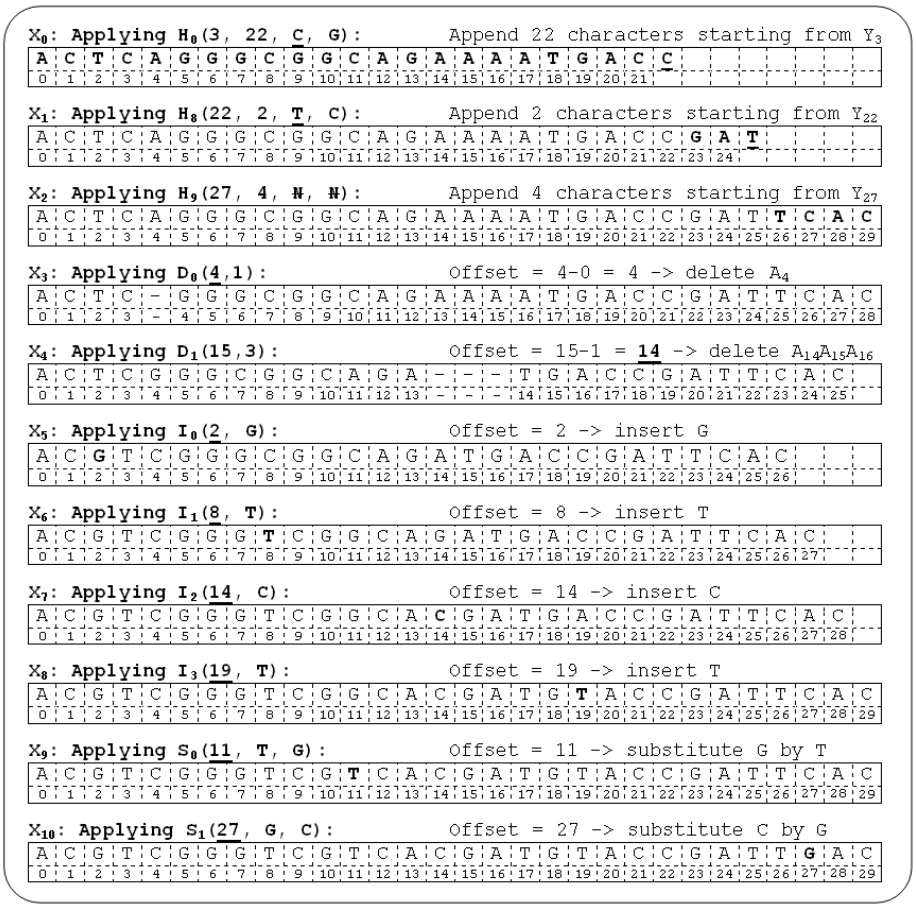

Fig. 5: *Example of Stanford's decompression of chromosome X previously depicted in (Figure 2) and compressed in (Figure 3).*

case, Stanford merges H_{i+1} with H_i to be $H_i(m_i, m_{i+1}+l_{i+1}-m_i, t_{i+1}, r_{i+1})$, and a new deletion D($n_i$+$l_i$, ($m_{i+1}$–1)–($l_i$+$m_i$)) is added to D_s. For example, after the past merge of instructions H_0 and H_1, the new version of instruction H_0 can be merged with instruction H_2 after deleting character 'A' from offset 7 in the reference chromosome (Figure 2). So, the deletion D_0 is created as in (Figure 3.d). After that, instruction H_6 can be merged with last version of instruction H_0, and a new deletion (D_1) is created. During decompression, Stanford copies the match instruction H_0 from the reference chromosome, and then, the deletion D_1 will help in deleting the three characters "AAA" at offset 18 from the reference chromosome to get a mature part of the target chromosome.

Notice that the offsets of all differences (S_s, I_s, and D_s) refer to offsets in the target chromosome X, because all differences will be applied on an initial copy of X built from Y when the decompression starts. Finally, Huffman and Golomb [14] encodings are applied to all offset numbers held by (H_s, S_s, I_s, and D_s). Moreover, the characters embedded in (H_s, S_s, and I_s) are then binary encoded to reduce their total size.

2.2 The Stanford's Decompression

To decompress the target chromosome X, Stanford loads the reference chromosome Y and decodes all the list of H_s, S_s, I_s, and D_s. After that, it starts building an initial copy of X by copying all H_s, including their encoded characters, from Y (Figure 5).

Critically, to decompress X correctly, all differences must be executed in its reverse order: executing all D_s, then all I_s, then all S_s. Violating this constraint will result in a different X, simply because the offsets of S_s were computed while characters of I_s are already there in X. Similarly, offsets of I_s were generated while characters referred to by D_s are not yet existing in X. So, for offsets of S_s and I_s to be valid during decompression, D_s must be executed first, followed by I_s, and then S_s. In addition, subsequent D_s must adjust their offsets by the characters deleted by the execution of past D_s. That is because when an offset of any D_i was computed, the characters of past D_s were already counted in the past matches. (Figure 5) illustrates the Stanford's decompression of the sample chromosome X depicted in (Figure 2). X_{10} is the mature copy of chromosome X obtained after complete referential decompression. This is the way adopted by the Stanford's decompression in detail. As shown, in order to obtain any partial sequence from chromosome X, it is necessary to execute all the match instructions H_s, followed by all the differences in its reversed order (D_s, then I_s, then S_s), resulting in big time and temporary storage overhead.

2.3 Stanford Character Encoding

The Stanford's compression encodes every ending-character pair of (H_s, S_s, and I_s) into variable number of bits in order to reduce their overall size. For example, if the reference and target ending characters of some H_i are ('A' and 'G') respectively, their binary encoding will be "01", whereas if the ending characters are ('A' and 'N'), their binary encoding will be "101".

During the Stanford's decompression, the reference character must be in hand in order to contribute in decoding the target character. So, for the former ending-character pair ('A' and 'G'), the target character 'G' is decoded by using both its corresponding reference character and its encoded value "A01", whereas for the later character pair ('A' and 'N'), the target character 'N' is decoded using "A101".

3. METHODS

This section describes the inCompressi algorithm in detail. It then provides three simplified examples of how inCompressi operates while performing partial decompressions.

3.1 inCompressi

The title inCompressi is inspired from the Latin expressions "in vivo", "in vitro", and "in silico", which respectively refer to biological operations performed on "living bodies", "lab samples", and "computer simulations". Similarly, inCompressi refers to picking partial sequences from compressed genomes or chromosomes without their complete decompression (Figure 1). Moreover, inCompressi can also decompress an entire chromosome via successive partial decompressions. The key strength behind inCompressi is summarized in how it can efficiently re-adjust the offset of the needed sequence to determine its location inside the reference chromosome without the real execution of differences preceding this sequence inside the target chromosome. In turn, it adjusts the offsets of the differences falling inside the picked sequence before applying them. Moreover, inCompressi can efficiently handle cases where the needed sequence spans multiple match instructions.

Suppose that a biologist want to pick a sequence Q with offset O_{seq} and length L_{seq} from a specific target chromosome T. If T was referentially compressed by Stanford using a reference chromosome R, then this biologist have two ways to obtain the sequence Q. Traditionally, he can decompress the overall genome using the Stanford's decompression, and then pick Q from the decompressed chromosome T using the given (O_{seq}, L_{seq}) pair. As stated in the previous section, the Stanford's decompression builds an initial genome (all chromosomes) from the reference genome (by executing all H_s on all chromosomes), and then obtains the final genome by executing all the differences (D_s, I_s, and S_s) on the initial genome (Figure 5). Obviously, that traditional genome decompression results in huge runtime and storage overhead.

Alternatively, the biologist can use inCompressi to pick the same sequence Q from chromosome T with the same (O_{seq}, L_{seq}) pair, however, without the genome's complete decompression, and even without T's complete decompression. Common to the Stanford's decompression, inCompressi loads and decodes the match instructions (H_s) and differences (S_s, I_s, and D_s) of the overall compressed genome. Unlike the Stanford's decompression, inCompressi (1) only keeps the match instructions (H_s) and differences (D_s, I_s, and S_s) related to chromosome T, and then, (2) correctly locates and picks the sequence Q from the reference chromosome R (by only executing H_s referring to Q in R), and finally, (3) only applies the differences (of D_s, I_s, and S_s) falling inside Q. That way, the biologist would obtain the same sequences that would be traditionally picked after the Stanford's complete genome decompression. The core functionality of inCompressi is shown in (Listing 1).

To pick Q from R, inCompressi has to determine the absolute offset (Q_r) of Q inside R, which is mainly dependent on how far it is distant ($R_{H->Q}$) from H_i's absolute offset in R (H_r) (Figure 6):

$$Q_r = H_r + R_{H->Q} \tag{3}$$

Unfortunately, the biologist only have O_{seq}, which represents the actual offset (Q_a) of Q inside the final chromosome T. So, the distance $R_{H->Q}$ cannot be calculated without knowing where initially H_i is assumed to start during building the initial chromosome T. We said "initially" because when Stanford executes H_s to build the initial T from R, each H_i will have its own fake target offset (H_f), which is calculated by accumulating the lengths of H_s preceding H_i, in addition to their ending characters (Eqn 4). For example, in (Figure 5), H_9 has fake offset H_f equals to 26 inside the sequence X_2.

$$H_f = \sum_{j=0}^{j=i-1} (H_j\_length + 1) \tag{4}$$

Actually, H_f may not necessarily be the actual offset (H_a) of H_i inside the final chromosome T; because, possible deletions

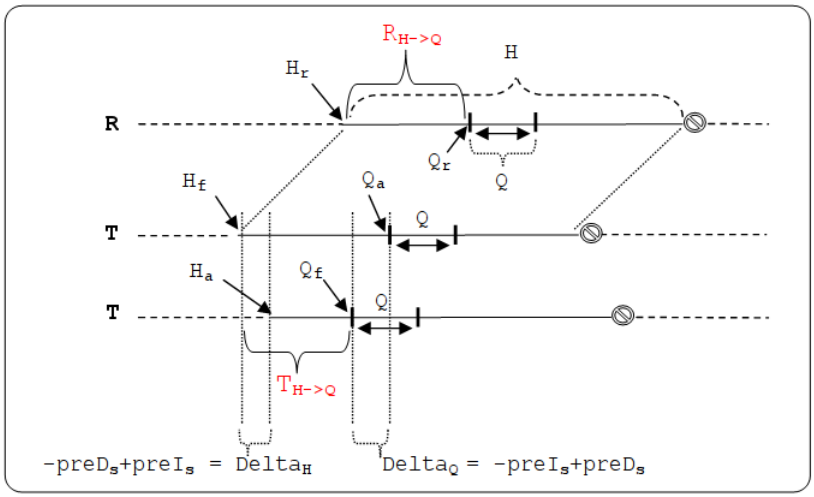

Fig. 6: *Calculating $R_{H->Q}$, the distance of query Q inside a match instruction H in reference Chromosome R, using either fake (H_f, Q_f) or actual (H_a, Q_a) target offsets of H and Q, while assuming that insertions occur more than deletions prior to both H and Q in chromosome T. $Delta_Q$ provides the calculation of Q_f from Q_a by backtracking the insertions and deletions that would be done before Q to have its final offset Q_a. Conversely, $Delta_H$ could be used to calculate H_a from H_f by simulating the execution of deletions and insertions occuring before instruction H.*

($preD_s$) and insertions ($preI_s$) will eventually be applied to T in offsets preceding H_i. Generally speaking, to calculate the instruction's actual offset from its fake offset (Eqn 5), inCompressi just simulates applying the prior deletions ($preD_s$) by subtracting their length from the fake offset, and it similarly simulates the execution of the prior insertions ($preI_s$) by adding their count to the fake offset. This is highlighted by the term $Delta_H$ in (Figure 6).

$$H_a = H_f - preD_s + preI_s \tag{5}$$

Likewise, Q_a, the actual offset of Q inside the final chromosome T, can be used to calculate the fake offset (Q_f) of Q inside the initial version of T (Eqn 6). To calculate Q_f, Q_a is altered by (1) subtracting how many characters were already inserted, and (2) adding how many characters were already deleted before the offset Q_a in the final version of T (represented by $Delta_Q$ in (Figure 6)):

$$Q_f = Q_a - preI_s + preD_s \tag{6}$$

(Eqn 7) shows and (Figure 6) depicts four equivalent ways to calculate the distance $R_{H->Q}$ using target offsets, however, inCompressi makes use of the first alternative. After calculating $R_{H->Q}$, Q_r can be calculated by (Eqn 3), and then inCompressi can seek to offset Q_r in chromosome R to pick the sequence Q with the given length L_{seq}.

$$R_{H->Q} = T_{H->Q} = \begin{cases} Q_f - H_f \\ Q_f - H_a + Delta_H \\ Q_a - H_f - Delta_Q \\ Q_a - H_a - Delta_Q + Delta_H \end{cases} \tag{7}$$

The past scenario is valid in case Q is completely falling inside the same H_i. Potentially, Q may span multiple subsequent H_s in R, and so, inCompressi has to determine the subsequent instructions it should manipulate to build the overall Q. For example, if executing H_i resulted in the first L characters of Q ($SubQ_0$), then inCompressi executes instruction H_{i+1} from its beginning with length (L_{seq}–L). The same process is repeated until inCompressi recovers the complete Q from the subsequent H_s. The pseudo code at (Listing 1) briefly states the overall inCompressi algorithm.

Finally, after picking Q (or $SubQ_i$) from a specific H_i, inCompressi has to apply the internal differences (inD_s, inI_s, and inS_s) that falls inside Q (or $SubQ_i$). Before applying these differences, their offsets have to be corrected and then applied in separation of their preceding differences. At first, if there are multiple inD_s to be applied to $SubQ_i$, then starting from deletion inD_1, each inD_i must have its offset adjusted by: (1) subtracting the number of characters already deleted by inD_0 till inD_{i-1} in $SubQ_i$ (Eqn 8), and (2) subtracting inD_s and adding inI_s that are already applied inside the past $SubQ_s$ ($SubQ_0$ till $SubQ_{i-1}$). Secondly, inI_s and inS_s must have their offsets corrected by the number of characters of the preceding non-deleted D_s ($preD_s$) and non-inserted I_s ($preI_s$). That is because for the Stanford's decompression to work correctly, it has to apply all D_s before it starts applying I_s or S_s, and then it starts applying all I_s before applying S_s. So, for inCompressi to have corrected offsets for inI_s and inS_s, it should shift these offsets forward with the non-deleted $preD_s$, and shift them backward with the non-inserted $preI_s$. (Eqn 9) shows how inCompressi calculates this shift using the fake offset Q_f that is previously calculated by (Eqn 6).

$$inD_i\_Offset = inD_i\_Offset - \sum_{j=0}^{j=i-1} (inD_j\_length) \tag{8}$$

$$PreviousD_s\&I_sShift = Q_f - Q_a \tag{9}$$

Sometimes, the characters to be deleted from Q may be more than the characters to be inserted, or vice versa. In the former case, the final Q will be shorter than required, and so, inCompressi picks the remaining characters the same way Q is picked. In the later case, inCompressi discards the extra characters found at the end of Q.

Listing 1: *A simplified pseudo-code for inCompressi picking a sequence Q with offset O_{seq} and length L_{seq} from a chromosome T that is referentially compressed using chromosome R. Assuming that the match instructions (H_s) and differences (S_s, I_s, and D_s) of T are already decoded.*

PickSequenceFromChromosome(R, H_s, S_s, I_s, D_s, O_{seq}, L_{seq}):
{
1: $NextL_T$ = 0
2: $Q_a = L_{seq}$
3: Q_f = GetQueryFakeOffset(O_{seq}, I_s, D_s)
4: $PreviousD_s\&I_sShift = Q_f - Q_a$
5: $AppliedD_s\&I_sShift$ = 0
6:
7: Foreach H_i in H_s:
8: $H_f = NextL_T$
9: H_r = offset of H_i in R
10: L_H = length of H_i
11: $CharOfH_i$ = Decoded H_i's character
12: $NextL_T = NextL_T + L_H$
13:
14: if $H_f \leq$ Q$_f$ and $Q_f + L_{seq} \leq$ H$_f + L_H$:
15: $VisitH_{i+1}$ = false
16: $NextQ_f$ = -1
17: else if $H_f \leq$ Q$_f$ and Q_f ¡ $H_f + L_H$:
18: $VisitH_{i+1}$ = true
19: $NextQ_f = NextL_T$ + 1
20: $RemainingL_{seq} = L_{seq}$ - ($NextL_T$ - Q_f)
21: $L_{seq} = L_{seq}$ - $RemainingL_{seq}$
22:
23: $R_{H->Q} = Q_f - H_f$
24: $Q_r = H_r + R_{H->Q}$
25: Sub_Q = R.ReadSequence(Q_r, L_{seq})
26: $Sub_Q = ApplyD_sOnQ(D_s, Sub_Q, AppliedD_s\&I_sShift)$
27: $Sub_Q = ApplyI_sOnQ(I_s, Sub_Q, PreviousD_s\&I_sShift)$
28: $Sub_Q = ApplyS_sOnQ(S_s, Sub_Q, PreviousD_s\&I_sShift)$
29: $AppliedD_sI_sShift = L_{seq}$ - Length(Sub_Q)
30: Q = Q + Sub_Q
31:
32: if $VisitH_{i+1}$ == false or $RemainingL_{seq}$ == 0:
33: exit_loop
34:
35: $Q_f = NextQ_f$
36: $L_{seq} = RemainingL_{seq}$
37: End foreach
38: Print Q
}

3.2 Simple inCompressi Runs

inCompressi works on chromosomes with sizes ranging from tens to hundreds of Megabytes, however, this paper gives illustrative examples of picking short sequences from a simple sequence representing a random chromosome. Assume that chromosome X is referentially compressed using chromosome Y (as in Figure 2), and that the match instructions (H_s) and differences (S_s, I_s, and D_s) of the compressed chromosome X are loaded and decoded to be the same as in (Figure 3.d).

Table 1. : *Instructions used throughout the following examples.*

Instruction H_i	H_r	H_f	$L_H + 1$
H_0(3, 22, C, G)	3	0	22+1=23
H_8(22, 2, T, C)	22	0+23=23	2+1=3
H_9(27, 4, ~~N~~, ~~N~~)	27	23+3=26	4+0=4

(Table 1) respectively shows H_s, their Y_offset (H_r), their fake X_offset (H_f), and their length (L_H+1) including the ending character of each instruction (except the last dummy character). Recall that the Stanford's decompression has to use all H_s to build an initial X chromosome (as X_2 in (Figure 5)). Conversely, inCompressi only uses H_s that are spanned by the sequence to be decompressed.

Example 1: Q = X(O_{seq}, L_{seq}) = X(17, 4)

* $Q_a = L_{seq}$ = 17
* $Q_f = Q_a + (D_0+D_1) - (I_0+I_1+I_2)$ = 17 + (1+3) – (1+1+1) = 18
* Q = X(18, 4) ⇒ Q ∈ H_0
* $R_{H->Q} = Q_f - H_f$ = 18 – 0 = 18
* $Q_r = H_r + R_{H->Q}$ = 3 + 18 = 21
* Q = Y(Q_r, L_{seq}) = Y(21, 4) = TGAC
* Applying D_s, I_s, S_s, if any, falling inside Q (between offsets 18 and 21):
* No D_s to apply.
* Calculating $I_s\&S_s$Shift = $Q_a - Q_f$ = 17 – 18 = –1
* Applying I_3(19, T) on Q: Offset =19 – (–1) = 20 ⇒ Q = TG**T**AC
* No S_s to apply.
* Ignoring the extra 'C' ... ⇒ Q = TGTA
* Q is identical to X(17, 4) of X_{10} in (Figure 5).

Example 2: Q = X(O_{seq}, L_{seq}) = X(4, 10)

* $Q_a = L_{seq}$ = 4
* $Q_f = Q_a - I_0$ = 4 – 1 = 3
* Q = X(3, 10) ⇒ Q ∈ H_0
* $R_{H->Q} = Q_f - H_f$ = 3 – 0 = 3
* $Q_r = H_r + R_{H->Q}$ = 3 + 3 = 6
* Q = Y(Q_r, L_{seq}) = Y(6, 10) = CAGGGCGGCA
* Applying D_s, I_s, S_s, if any, falling inside Q (between offsets 3 and 12):
* Applying D_0(4, 1) on Q: ⇒ Q = C–GGGCGGCA
* Calculating $I_s\&S_s$Shift = $Q_a - Q_f$ = 4 – 3 = 1
* Applying I_1(8, T) on Q: Offset = 8 – 1 = 7 ⇒ Q = CGGG**T**CGGCA
* Applying S_0(11, T, G) on Q: Offset = 11 – 1 = 10 ⇒ Q = CGGGTCG**T**CA
* Q is identical to X(4, 10) of X_{10} in (Figure 5).

Example 3: Q = X(O_{seq}, L_{seq}) = X(21, 7)

* $Q_a = L_{seq}$ = 21
* $Q_f = Q_a + (D_0+D_1) - (I_0+I_1+I_2+I_3)$ = 21 + (1+3) – (1+1+1+1) = 21
* Q = X(21, 7) ⇒ Q spans H_0, H_8, and H_9.
* Recursively fetching $SubQ_s$ as shown in the past examples:
* $SubQ_0$ = X(21, 2): $SubQ_0$ = C + (C)
* $SubQ_1$ = X(23, 3): $SubQ_1$ = GA + (T)
* $SubQ_2$ = X(26, 2): $SubQ_2$ = TG
* Q = $SubQ_0 + SubQ_1 + SubQ_2$ = CC + GAT + TG = CCGATTG
* Q is identical to X(21, 7) of X_{10} in (Figure 5).

4. RESULTS AND DISCUSSION

4.1 The Experimental Environment

The performed experimental results are based on two datasets. The first dataset contains two versions of the Arabidopsis thaliana (TAIR) genome, namely the *TAIR8* (TAIR8 Website) and the *TAIR9* (TAIR9 Website) genomes. Each of these genomes has a total size of 120 MB approximately, and is divided into five chromosomes which are stored separately in FASTA-formatted files. The second dataset consists of two versions of the human genome, namely the *hg18* (HG18 Website) and the *yh* (YH Website) genomes. Each of these genomes has a total size of 3,000 MB approximately, and is divided into twenty five FASTA-formatted chromosome files.

The TAIR experiments are performed on a machine with an Intel Core2 Duo CPU @ 2.10 GHz (a single core is used) with 3.77 GB of RAM, whereas the human genome experiments are executed on a machine with an Intel Core i5 CPU @ 2.50 GHz (a single core is used) with 3.89 GB of RAM. Both machines run on LinuxMint-14 OS.

4.2 The Implementation Details

As Stanford was implemented in Python, inCompressi is also implemented in Python in order to be comparable with the original performance of the Stanford's decompression. In addition, biologists sometimes need to visually analyze specific regions of the genomic sequences in a convenient way. Motivated by this need, the inCompressi's implementation is augmented with a desktop application, namely, inCompressi_Navigator. Aided by this application, biologists can visually browse their genomes, which are already compressed by Stanford, without their complete decompression. The reader can find more details about the complete implementation of inCompressi at this link: `www.fci.cu.edu.eg/~mnassef/implementations/inCompressi.php`.

4.3 Partial Decompressions using inCompressi

This subsection discusses the performance of inCompressi during its partial decompression of variable length sequences from the referentially compressed TAIR9 and yh genomes. The performed experiments picked sequences with lengths ranging from 1,000 characters to entire chromosomes. Most of these sequences were picked from each chromosome at different offsets, however, inCompressi gave similar results independent of the queried offset. To ensure the experimental validity of inCompressi, all sequences picked from the referentially compressed genomes using inCompressi were compared with the same sequences from the original non-compressed chromosomes.

(Table 2) shows the memory usage and the average runtime in milliseconds (ms) of inCompressi performing miscellaneous partial decompressions over the five chromosomes of the TAIR9 genome. (Figure 7.a) shows how inCompressi is efficient in picking sequences with length ranging from 100 to 100,000 characters. It almost has the same memory usage (7 MB) and average runtime (18-20 ms) independent of the offset or the length of the sequence being queried. Picking sequences with lengths greater than 100,000 characters costs relatively longer runtime and higher memory consumption.

Table 2. : *Average runtime and memory usage for sequences picked by (a) inCompressi, and (b) inCompressi-Blocks with block length 100,000, from the referentially compressed TAIR9 genome.*

	(a) *inCompressi*		**(b)** *inCompressi-Blocks*	
Query Length	Memory Usage (MB)	Average Runtime (ms)	Memory Usage (MB)	Average Runtime (ms)
100	7	18		
10,000	7	19		
100,000	7	20		
500,000	8	24	8	24
1,000,000	11	44	9	30
3,000,000	18	125	13	56
5,000,000	26	242	17	79
10,000,000	45	661	26	136
Chromosome	56	1,428	36	593

The impressive performance of inCompressi was motivating to perform partial decompressions in subsequent fixed-length blocks of length 100,000 characters (inCompressi-Blocks), where the shorter the block to be extracted, the lower the expected runtime and memory consumption. For example, to decompress some chromosome X_1^{1000} in blocks of length 100, inCompressi-Blocks partially decompress chromosome X in ten subsequent blocks: X_1^{100}, X_{101}^{200}, ... X_{901}^{1000}. (Table 2) shows the performance of inCompressi-Blocks alongside the normal performance of inCompressi. Using inCompressi-Blocks, the complete chromosome decompression is significantly enhanced to have the maximum memory usage of 36 MB instead of 56 MB, and average runtime of 593 sec instead of 1,428 sec. (Figure 7.b) shows how inCompressi-Blocks overcomes inCompressi's performance during picking longer sequences.

Table 3. : *Average runtime and memory usage for sequences picked by inCompressi-Blocks with block length 1,000,000 characters from the first chromosome of the referentially compressed yh human genome. Around 540 MB of memory and 32 sec of runtime are consumed in loading and decoding the compressed data.*

	inCompressi-Blocks	
Query Length	Memory Usage (MB)	Average Runtime (sec)
1,000	551	33
10,000	551	33
100,000	551	33
1,000,000	552	33
10,000,000	563	34
100,000,000	745	36
Chromosome	790	42

The performance of picking partial sequences from the referentially compressed yh human genome is shown in (Table 3). Compared to Stanford, inCompressi-Blocks significantly consumes less memory and reduces runtime from 19 minutes to less than one minute. (Table 3) also shows that picking sequences with length 1,000 to 10,000,000 characters almost have the same memory and runtime.

4.4 Complete Genome Decompression

This subsection investigates the performance of inCompressi's complete genome decompression compared to the original Stanford's decompression. (Table 4) shows the measured storage in MB and runtime in seconds (Sec) for each implementation during the referential decompression of the TAIR9 genome using TAIR8 as a reference. Compared to Stanford, inCompressi significantly takes lower runtime and memory because it builds the mature target chromosome one match instruction at a time, and so, it applies insertions (or deletions) to relatively shorter memory buffers. (Figure 8.a) illustrates the average runtime and the memory usage obtained by trying inCompressi-Blocks using different block lengths. It appears that running inCompressi-Blocks with block length 100,000 resulted in the best performance for the TAIR9 genome. Compared to Stanford, inCompressi-Blocks shows a drastic reduction in both memory consumption and runtime (around the tenth). Also, (Figure 8.a) shows that using very short block length (such as 100 and 1,000 characters) is not recommended, because when block length becomes very short, inCompressi-Blocks has to access the list of matches and differences more frequently while building more blocks. Thus, using very short and very long block lengths should be avoided.

The experimental results on the human genome are shown in (Table 5) during the referential decompression of the yh genome using the hg18 genome as a reference. (Table 5) shows how inCompressi is perfectly scalable for manipulating the human genomes. Moreover, as shown in (Figure 8.b), running

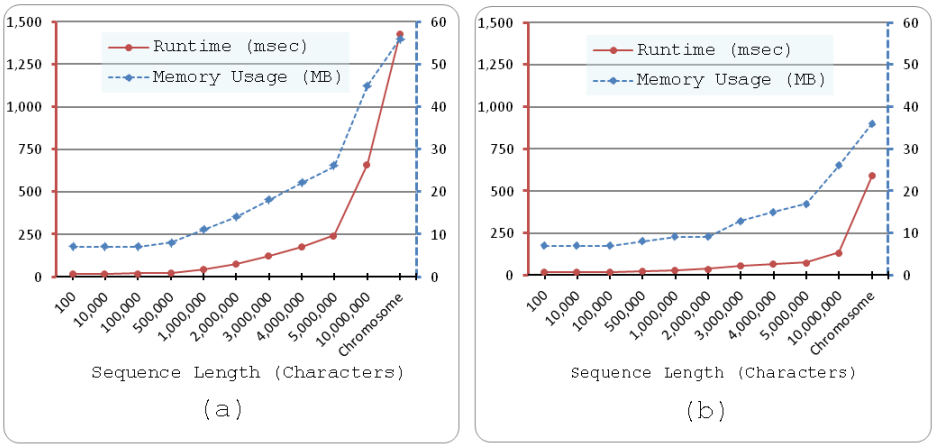

Fig. 7: *The performance of partial decompressions using (a) inCompressi, and (b) inCompressi-Blocks with block length 100,000. inCompressi-Blocks outperforms inCompressi in picking sequences longer than 500,000 characters.*

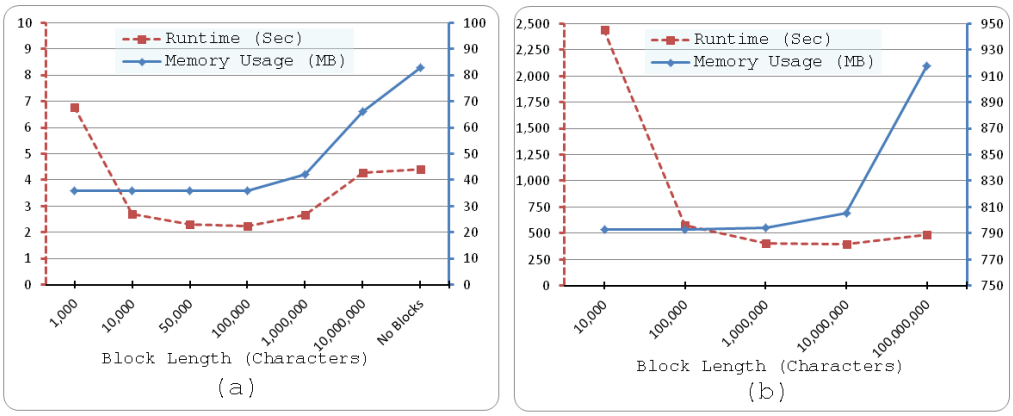

Fig. 8: *Average runtime and memory usage of inCompressi-Blocks performing complete genome decompression for (a) the TAIR9 genome, and (b) the yh human genome, with different block lengths. Using block lengths of 100,000 and 1,000,000 gave the best runtime and lowest memory consumption for the TAIR9 and yh genomes respectively.*

Table 4. : *Memory usage and average runtime taken by the Stanford's decompression, inCompressi, and inCompressi-Blocks, while decompressing the TAIR9 genome using the TAIR8 genome as a reference.*

Algorithm	*Memory Usage* (MB)	*Average Runtime* (Sec)
Stanford's Decompression	300	21.17
inCompressi	83	4.40
inCompressi-Blocks (100,000)	36	2.25

Table 5. : *Memory usage and average runtime taken by the Stanford's decompression, inCompressi, and inCompressi-Blocks, while decompressing the yh human genome using the hg18 human genome as a reference.*

Algorithm	*Memory Usage* (MB)	*Average Runtime* (Sec)
Stanford's Decompression	2,928	1,150
inCompressi	925	514
inCompressi-Blocks (1,000,000)	794	400

inCompressi-Blocks with block length 1,000,000, on the second experimental machine, resulted in the best performance for the yh human genome. It significantly outperforms the Stanford's decompression in both runtime and memory consumption.

4.5 The Implementation Challenges

Based on the Stanford's character encoding described on Subsection 2.3, if inCompressi is going to decompress a sequence at a specific region of the compressed genome, it has to decode the target ending characters of (H_s, S_s, and I_s) falling in the targeted sequence. However, because the character encodings of the compressed genome differ in length, inCompressi cannot bypass the encodings preceding the encodings of the targeted sequence until it has all their reference characters in hand in order to know their lengths, which is impractical.

Alternatively, we modified the character encoding of the Stanford's compression to retain metadata that enables inCompressi to bypass the encodings preceding the targeted sequence. This metadata is written before every character encoding storing its length. For example, by considering the encoding

examples in Subsection 2.3, the new encoding of the ('A' and 'G') pair is "**10**01", whereas the new encoding of ('A' and 'N') is "**11**101". Retaining this metadata has a slight effect on the compression ratio of the overall genome. For instance, the altered Stanford's compression compresses the TAIR9 genome into 3.7 Kilobytes (KB) instead of 3.5 KB, and compresses the yh genome into 10.4 MB instead of 8.5 MB.

On the other hand, because the Stanford's compression cumulatively encodes the differences of the entire genome into the same file, inCompressi repeatedly loads and decodes all the compressed differences of the overall yh genome while picking each single sequence. The resulting overhead is shown by (Table 3), where picking sequences from the yh human genome consumes almost the same memory (540 MB) and runtime (32 sec). This overhead can be avoided by altering the Stanford's compression algorithm to compress each chromosome individually, however, this will slightly affect the compression ratio. Alternatively, inCompressi could reduce this overhead by loading and decoding all the differences once, but then picking multiple sequences using the same decoded differences.

5. CONCLUSION AND FUTURE WORK

This paper showed how it is feasible to browse sequences from the referentially compressed genomes through partial decompressions. Partial decompressions saves both runtime and memory usage, and it also needs no disk space. As the referential compression algorithms differ in how their decompression works, each of them have to have its own tailored partial decompression algorithm. Therefore, the inCompressi algorithm is designed and implemented to efficiently pick sequences from genomes referentially compressed using the Stanford's compression algorithm. In addition to testing inCompressi on versions of the relatively short Arabidopsis-Thaliana genome, inCompressi showed a scalable performance over versions of the human genomes that are very long. On the other hand, inCompressi showed more efficient complete decompression compared to the original Stanford's decompression algorithm. Hence, the biological genome analysis algorithms can utilize inCompressi to browse the Stanford's referentially compressed genomes in reasonable response time and computational resources. Additionally, the inCompressi's implementation is augmented with a desktop application, namely inCompressi_Navigator, that can be used by biologists to visually browse the compressed genomes without their complete decompression.

The inCompressi's implementation could have further runtime enhancement by exploiting the multicore capabilities provided by the nowadays machines. Moreover, inCompressi can be enhanced to search for specific sequences inside genomes referentially compressed by Stanford.

6. ACKNOWLEDGEMENTS

We thank the authors of the Stanford algorithm for providing us with their algorithm's prototype implementation. We also thank our colleague Sara Salem for proofreading this article.

7. REFERENCES

[1] R.A. Gibbs, J.W. Belmont, P. Hardenbol, T.D. Willis, F. Yu, et al. The international hapmap project. *Nature*, 426(6968):789–796, 2003.

[2] N. Siva. 1000 genomes project. *Nature biotechnology*, 26(3):256–256, 2008.

[3] G.M. Church. The personal genome project. *Molecular Systems Biology*, 1(1), 2005.

[4] D.R. Bentley. Whole-genome re-sequencing. *Current opinion in genetics & development*, 16(6):545–552, 2006.

[5] J. Shendure and H. Ji. Next-generation dna sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

[6] R. Sachidanandam, D. Weissman, S.C. Schmidt, J.M. Kakol, L.D. Stein, et al. A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms. *Nature*, 409(6822):928–933, 2001.

[7] L.S. Heath, A. Hou, H. Xia, and L. Zhang. A genome compression algorithm supporting manipulation. In *Proc LSS Comput Syst Bioinform Conf*, volume 9, pages 38–49, 2010.

[8] S. Kuruppu, S.J. Puglisi, and J. Zobel. Reference sequence construction for relative compression of genomes. In *String Processing and Information Retrieval*, pages 420–425. Springer, 2011.

[9] S. Deorowicz and S. Grabowski. Robust relative compression of genomes with random access. *Bioinformatics*, 27(21):2979–2986, 2011.

[10] C. Wang and D. Zhang. A novel compression tool for efficient storage of genome resequencing data. *Nucleic Acids Research*, 39(7):e45–e45, 2011.

[11] A.J. Pinho, D. Pratas, and S.P. Garcia. Green: a tool for efficient compression of genome resequencing data. *Nucleic Acids Research*, 40(4):e27–e27, 2012.

[12] B.G. Chern, I. Ochoa, A. Manolakos, A. No, K. Venkat, and T. Weissman. Reference based genome compression. In *Information Theory Workshop (ITW), 2012 IEEE*, pages 427–431. IEEE, 2012.

[13] A.D. Wyner and J. Ziv. The sliding-window lempel-ziv algorithm is asymptotically optimal. *Proceedings of the IEEE*, 82(6):872–877, 1994.

[14] M.C. Brandon, D.C. Wallace, and P. Baldi. Data structures and compression algorithms for genomic sequence data. *Bioinformatics*, 25(14):1731–1738, 2009.

Websites

TAIR8 Website: `ftp://ftp.arabidopsis.org/home/tair/Sequences/whole_chromosomes/OLD/` [Accessed: 5 December 2012]

TAIR9 Website: `ftp://ftp.arabidopsis.org/home/tair/Sequences/whole_chromosomes/OLD/TAIR9_chromosome_file/` [Accessed: 5 December 2012]

HG18 Website: `http://hgdownload.cse.ucsc.edu/goldenPath/hg18/bigZips/chromFa.zip` [Accessed: 6 Jan 2013]

YH Website: `ftp://public.genomics.org.cn/BGI/yanhuang/fa/` [Accessed: 7 Jan 2013]