IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 21 September 2022, accepted 29 September 2022, date of publication 5 October 2022, date of current version 12 October 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3212523*

RESEARCH ARTICLE

# HADC: A Hybrid Compression Approach for DNA Sequences

**SARAH ELNADY, SABAH SAYED, AND AKRAM SALAH**
Computer Science Department, Faculty of Computers and Artificial Intelligence, Cairo University, Cairo 12613, Egypt
Corresponding author: Sarah Elnady (s.elnady@fci-cu.edu.eg)

**ABSTRACT** In the blossoming age of Next Generation Sequencing (NGS) technologies, genome sequencing has become much easier and more affordable. The large number of enormous genomic sequences obtained demand the availability of huge storage space in order to be kept for analysis. Since the storage cost has become an impediment facing biologists, there is a constant need of software that provides efficient compression of genomic sequences. Most general-purpose compression algorithms do not exploit the inherent redundancies that exist in genomic sequences which is the reason for the success and popularity of reference-based compression algorithms. In this research, a new reference-based lossless compression technique is proposed for deoxyribonucleic acid (DNA) sequences stored in FASTA format which can act as a layer above gzip compression. Several experiments were performed to evaluate this technique and the experimental results show that it is able to obtain promising compression ratios saving up to 99.9% space and reaching a gain of 80% for some plant genomes. The proposed technique also succeeds in performing the compression at acceptable time; even saving more than 50% of the time taken by ERGC in most experiments.

**INDEX TERMS** Bioinformatics, DNA sequences, reference-based compression, greedy alignment.

## I. INTRODUCTION

Genome sequencing, defined as the process of determining the complete deoxyribonucleic acid (DNA) sequence (i.e. determining the order of nucleotides in DNA) of an organism, has widely been performed over the past years. It was mainly used for research but with the new and advanced technologies, it is paving its way to be used in clinics for personalized medicine. Genome sequencing is usually done ''in pieces''; i.e., by cutting the genome into small reads since the whole genome can't be sequenced all at once and the available sequencing methods can only handle short reads of DNA at a time. One of the earlier methods of genome sequencing was Sanger sequencing [1] which utilized a high-fidelity DNA-dependent polymerase and dideoxynucleotides to produce a complementary copy to a single strand of the DNA. Despite being an old-fashioned technique, Sanger sequencing is still used for obtaining long contiguous DNA sequence reads. However, this method is considered to be slow and expensive, and this prompted ''Next Generation Sequencing'' (NGS) or ''High-throughput Sequencing'' (HTS) to be developed.

NGS technologies [2] revolutionized the sequencing world as they parallelize the sequencing process, producing thousands or millions of sequences concurrently. Therefore, NGS is typically characterized by being highly scalable, allowing the entire genome to be sequenced at once. Illumina stated that their sequencing systems can deliver data output ranging from 300 kilobases up to multiple terabases in a single run, depending on instrument type and configuration [3]. Since then, sequencing cost has drastically decreased [4] which has led to a remarkable increase in the number of available genomic sequences. Moreover, the gigantic data files produced by modern sequencing technologies demand the availability of huge storage space in order to keep them for future analysis. However, the rate of decrease in hardware cost has been outpaced by the rate of increase of NGS data and so the need for algorithms that provide efficient compression of biological data has emerged. This biological data is usually stored in certain types of files [5] which include FASTQ and SAM/BAM files. One of the key formats for storing genome sequences is the FASTA format in which each base in the

The associate editor coordinating the review of this manuscript and approving it for publication was Vincenzo Conti.

sequence is encoded by a single letter [5]. The reason behind the significance of this text-based format is its convenience for researchers to perform gene studies and sequence analysis [6] such as sequence alignment and comparison, collection of k-mer statistics, and more.

The compression algorithms [7] proposed previously in the literature can be categorized according to several factors such as the data, in other words file type, they compress (assembled data in FASTA files or raw data in FASTQ/SAM/BAM files), the compression scheme (lossless/lossy compression), and their dependence on a reference (reference-free/reference-based compression). Reference-free compression algorithms compress a target sequence without the use of an external reference genome. Some of the popular reference-free compression algorithms are gzip, bzip2 and 7zip. Unfortunately, these general-purpose algorithms do not exploit the redundancies that inherently exist in genomes and so they cannot produce efficient compression with genomic sequences [8]. There lies the gap that provoked the design of new algorithms, namely reference-based algorithms, specifically for compressing genomic sequences. Reference-based compression algorithms require the use of a reference (a sequence relatively similar to the target sequence) and align the target or reads using that reference, then, they store the match locations and any mismatches [9]. Despite resulting in a great compression ratio for large sequences, reference-based compression still faces some difficulties (for example, the reference and the target sequence must have high similarity) [7]. Therefore, there is always room for improvement due to the importance of reducing the size of genomic data to allow rapid transmission and efficient storage.

This is why, in this work, we focused on the category that has quickly become a more prevalent point of research and that is the lossless compression of assembled genomes (in FASTA files), with a reference. As explained before, genomic sequences stored in the FASTA format are convenient for and required by researchers, so developing algorithms that efficiently store these sequences without error will contribute to reducing storage costs and easier, faster, and cheaper transfer over the internet [7]. The evolution of reference-based genome compression algorithms was discussed as well as some of the state-of-the-art algorithms in this field. Then, the materials and methods used to develop a hybrid approach for DNA compression (HADC) are explained. Finally, the experimental results are presented in addition to the conclusions and plans for future work.

## II. RELATED WORKS

As mentioned in the previous section, genomes are stored for analysis and processing and transmitted frequently which requires they be compressed. The most commonly used compression algorithms are general-purpose compression algorithms such as PPMD [10], DEFLATE [11], and BWT [12] which are supported by general-purpose compression tools such as gzip (http://www.gnu.org/software/gzip/). These algorithms do not require the use of an external reference and rely mainly on processing the target and using an encoding scheme on it like run-length encoding for instance. Although these algorithms depend only on the target and perform well over small-sized data, they are not practical for larger sequences and are not able to achieve acceptable compression ratios for large data [8], [13]. This is due to the fact that they do not take the advantage of the redundancies and similarities that inherently exist in the structure of genomes.

Recently, many novel special-purpose algorithms have emerged to effectively compress one or more of the data types embedded in genomic files [14], [15], [16]. The special-purpose reference-free genome compression algorithms use bit-encoding [17] or statistics [18]. An example of the algorithms that use encoding is the lossless compression algorithm for genomes proposed by Al-Okaily et al. [19] which is based on Huffman encoding to replace high frequency bases with shorter codes and low frequency bases with longer codes. On the other hand, statistical-based compression algorithms, such as XM [20] and DNAC-SBE [18], generate a probabilistic statistical model according to the input data, predict the probability of the next nucleotide, and then use encoding schemes [21]. Although these algorithms maintain a low compression time, they still fail to achieve high compression ratios. Therefore, the chase for higher compression ratios has guided researchers to special-purpose reference-based genome compression algorithms [22] which instantly grasped attention for the extremely high compression ratios they can achieve. In the next paragraphs, some of the most efficient and widely used reference-based genome compression algorithms will be discussed.

In 2011, Deorowicz and Grabowski [23] proposed an LZ77-style compression scheme called GDC for the reference-based compression of multiple genomes of the same species. GDC was essentially similar to RLZ-opt [24], which was the state-of-the-art algorithm for compressing collections of genomes, yet it included some key differences. The authors stated that the improved compression ratios of GDC are attributed to both the LZ-parsing scheme they chose and to Huffman encoding, which is more compact than Golomb. GDC proved to outperform GRS [25] in relative compression, become better than RLZ-opt close to three times on human genomes and provide fast random access. Later, GDC 2 was proposed by Deorowicz et al. [26] in 2015 achieving results about 4 times better than what is offered by the other existing compressors in compressing a collection of human diploid genomes. GDC 2 is a two-level algorithm that uses a hash table and performs Ziv–Lempel factoring of sequences at each level.

Moreover, Hsi-Yang Fritz et al. [27] proposed a reference-based compression algorithm that can be used for the efficient storage of DNA sequencing data. However, it is worth noting that their proposed algorithm was developed to be used on FASTQ files that contain both the short sequences and their quality information. The algorithm firstly aligns the short sequences (reads) to a reference and the unaligned reads are

pooled and their base pair information is then stored using specific offsets with substitutions, insertions, or deletions encoded in separate data structures. As for the quality scores, before being compressed using a Huffman-based code, they are stored and a user-defined percentage of quality positions that are identical to the reference are stored [27].

In 2014, Stanford's Ochoa et al. [28] proposed iDoComp which is a state-of-the-art reference-based compression algorithm. The algorithm is composed of three steps: the mapping generation, the post-processing step and the entropy encoding step. The authors stated that the improved compression ratios achieved by the algorithm were largely due to the post-processing step, which modifies the set of instructions produced from the mapping generation in a way that is beneficial to the entropy encoder. iDoComp outperforms previously proposed algorithms in most of the studied cases reaching compression gains of up to 60% in several cases, including *H. sapiens* data with comparable or even better running time.

Both ERGC and NRGC were proposed by Saha and Rajasekaran in 2015 and 2016, respectively [29], [30]. These algorithms adopt the segmentation matching strategy by dividing the reference sequence and the to-be-compressed target sequence into segments of equal length and then matching each segment. ERGC algorithm also uses greedy alignment based on hashing. Although ERGC and NRGC are both effective genome compression algorithms that perform better than the best-known algorithms in most of the cases, they require high similarity between the reference and the target genomes. If the characteristics of the reference sequence and target do not conform to their matching strategy, the compression result will be poor [6].

A recent study performed in 2019 by Yao et al. [6] proposed HRCM, a new compression method capable of compressing a single sequence as well as large collections of sequences. This algorithm is also composed of three stages: information extraction, matching and encoding. HRCM performs two-level matching, and the compression performance is improved through the second-level matching. Experimental verification showed that HRCM performance, including compression ratio, speed, and memory usage, is competitive and sometimes superior to many of the best-known algorithms.

All the reference-based compressors discussed are capable of producing substantially higher compression ratios when compared with reference-free compressors [31]. In 2021, a novel method [32] for compressing large datasets of short reads using feature vectors and clustering algorithms was proposed. It exploits the redundancies between the datasets to improve compression performance [32]. Moreover, Silva et al. [31] stated that an entire human genome of size3 GB can be compressed to approximately 4 MB using a reference, and this is 100 times less than the compressed size produced using reference-free compression [31]. Still, reference-based compression is far from done and definitely has a lot of room to advance. These facts encouraged us to focus on lossless reference-based compression approaches in this work.

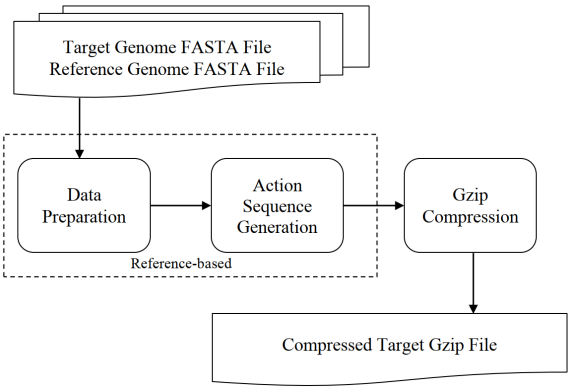


**FIGURE 1.** HADC overview.

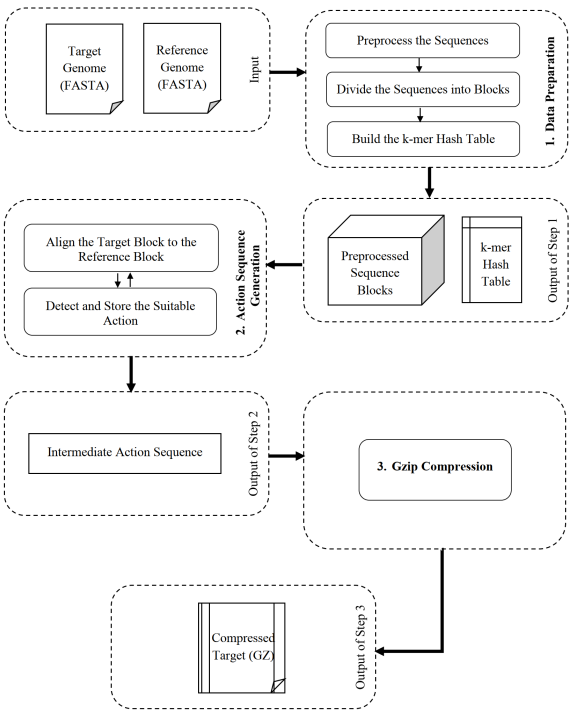


**FIGURE 2.** HADC detailed pipeline.

## III. MATERIALS AND METHODS

In this study, a hybrid DNA compression approach (HADC) is proposed in which a reference-based compression algorithm is used as a layer before the final gzip compression layer. The proposed reference-based compression layer is inspired by the Turing Machine and its mechanism [33], [34]. HADC works by transforming the target sequence (the sequence to be compressed) into a series of actions applied on the reference and saving these actions. Some of these actions are similar to the actions taken by a Turing Machine when it processes an input string to produce an output. So, in other words, the algorithm stores the transition function used by a Turing Machine that transforms the reference into the target. An overview of the proposed approach is shown in Figure 1. The main steps are: 1) sequence preparation and preprocessing, 2) action sequence generation and 3) compression using gzip. Figure 2 illustrates a more detailed pipeline of each step.

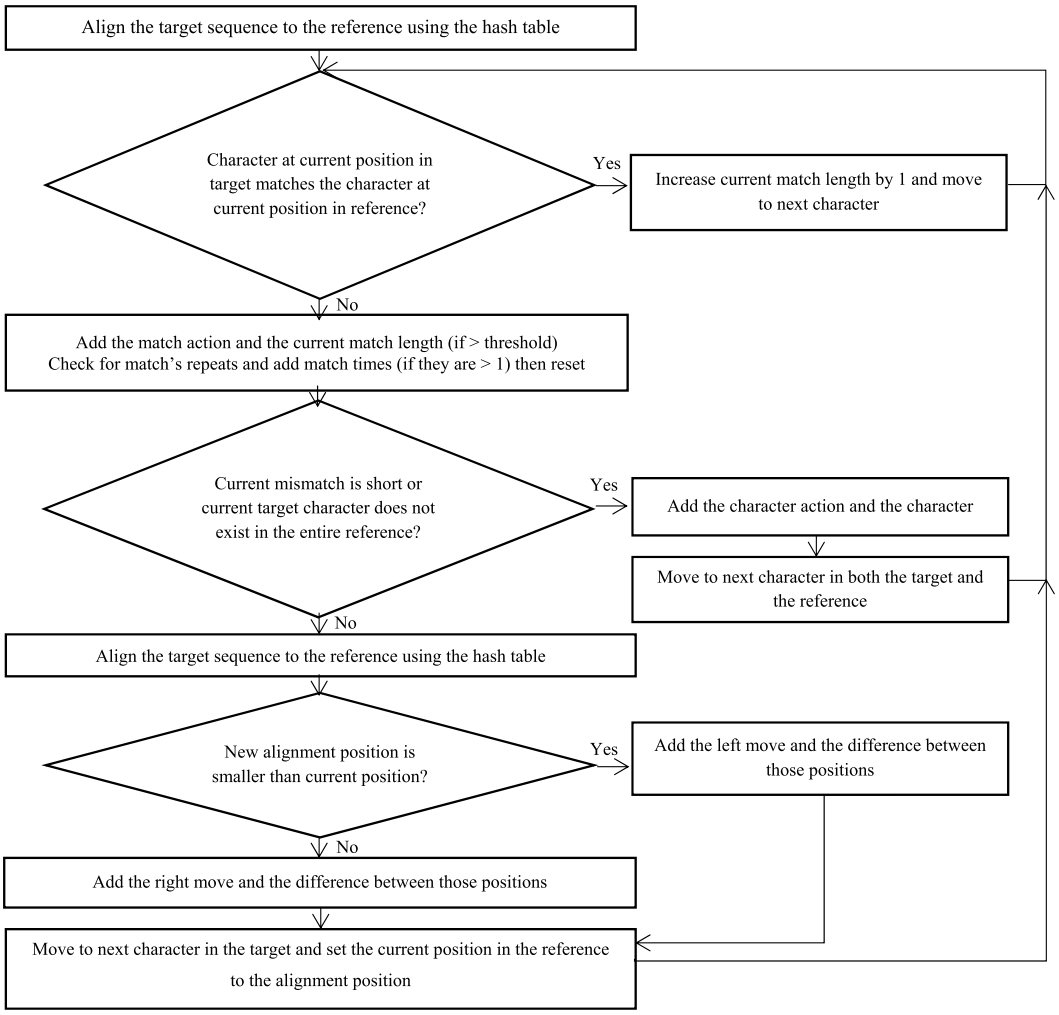


**FIGURE 3.** Flowchart of the action sequence generation process.

### A. DATA PREPARATION STEP

In the data preparation step, the algorithm's hyperparameters are set, the target and the reference sequence are preprocessed by making sure that they are both uppercase and consist of the valid DNA alphabet only (A, C, G, T and N for undetermined bases). Then, HADC adopts a divide and conquer strategy as proposed by Saha and Rajasekaran [29] and the target and the reference are divided into equal-sized blocks. In addition, a k-mer hash table is constructed, initialized and filled. The k-mer hash table is similar to a dictionary or map in which the k-mer is the key and the positions/indexes of its occurrence in the reference are the values. This hash table will be used to quickly yet greedily align the target and the reference sequences instead of using dynamic programming for alignment which is in the order of $O(nm)$ for a target of length $n$ and a reference of length $m$ and this would be infeasible time- and memory-wise.

### B. ACTION SEQUENCE GENERATION STEP

In this step, each pair of corresponding target and reference blocks along with their current hash table are fed to the "Action Sequence Generator" (ASG). The ASG has 5 possible types of actions: i) A match action "M" which is followed by the match length, ii) A character action "C" which is followed by the character, iii) A left move action "L" which is followed by the number of moves made in that direction, iv) A right move action "R" which is similar to the left move action but in the opposite direction, v) A repeat action "MT" which is followed by the number of times a match exists. Figure 3 shows a flowchart of the process of the ASG.

The first step the ASG does is to use the hash table to find the smallest position of the first k-mer in the target sequence and align the target to the reference from that position, saving that action and its information. It is worth mentioning that ASG will choose the alignment that gives the smallest amount of moves as clarified in Algorithm 1. Then, it checks whether the characters at the current positions in the target and reference match. If the characters do match, the current match length, which was initially set to zero, is incremented by 1. However, when the ASG encounters a mismatch, it first checks whether the next part of the target is a repetition of the last detected match and when this is the case, the match times are incremented. Otherwise, the

characters do not match, and the current match length is greater than a certain threshold, a match is recorded, and the match length is saved then reset. At this point, in case of a non-zero repeats number, an "MT" action (match times) is recorded along with the repeats amount which is also reset to zero after that. This means that, a repeat sequence is only considered a repeated match if it does not contain any mismatches (i.e. if it is identical to the previous match). When this is the case, the match times are incremented. Otherwise, if the characters do not match, a match is recorded and the mismatched character(s) are recorded or realignment occurs depending on the mismatch type (short/long).

**Algorithm 1** The Alignment Performed by ASG

**Input:** *k*, *hash_table*, *target_block*, *ref_block*, *target_index*, *previous_ref_index*

1. *target_kmer* = *target_block* [*target_index*:*target_index*+*k*]
2. **if** *hash_table*[*target_kmer*] == **None:**
   a. The target k-mer is not found in the current reference block, so the alignment fails, and the mismatch is treated as short mismatch.
3. *kmer_positions* = *hash_table*[*target_kmer*]
4. Set the *alignment_position* to the first element in the *kmer_positions* list.
5. *difference* = | *previous_ref_index* - *kmer_positions*[0] |
6. **for** *position* = 0 to length of *kmer_positions***:**
   a. *current_difference* = | *previous_ref_index* – *kmer_positions* [*position*]|
   b. **if** *current_difference* < *difference***:**
      i. *difference* = *current_difference*
      ii. *alignment_position* = *kmer_positions* [*position*]

**Output:** *alignment_position*, *difference*

The next step the ASG performs is to check whether the mismatch it just encountered is a short one or a long one. When either the mismatch is short or the current character in the target is a character that does not exist in the reference, the current character is saved with as it is. This is specifically useful in case of single nucleotide polymorphisms (SNPs) which are very common between similar genomes. Otherwise, the target and reference are re-aligned to try to reach a better match and the direction of the movement (left or right) from the previous position to the new one is saved along with the number of moves. After that, the algorithm moves to the next characters and repeats these steps until the entire target is transformed. Finally, the ASG sends the generated action sequence to the gzip compressor. Figure 4(a) illustrates a simple example of the input and output of the ASG.

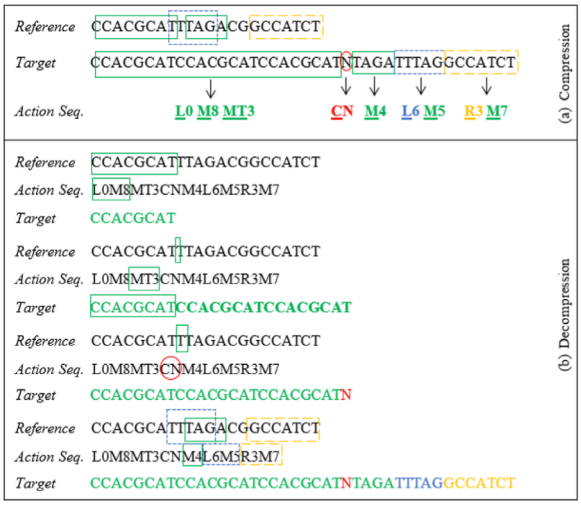


**FIGURE 4.** An example of (a) compression and (b) decompression.

### C. GZIP COMPRESSION STEP

Instead of using simple bit encoding, the gzip compressor is used as the final layer which produces the actual compressed file. Gzip was selected as the final compression layer since it provides efficient compression for general-purpose text [18] which is similar to nature of the produced action sequence and this method proved to produce better results than bit encoding. That's why we feed the action sequence as bytes to gzip and save the output file.

### D. DECOMPRESSION PROCESS

As for the decompression process of the compressed file, it is straightforward by applying the reverse of the compression process. The decompression process does not require the construction of a hash table. During decompression, the compressed file is decompressed by gzip to produce the intermediate action sequence file. The intermediate action sequence file is then processed such that the actions recorded are applied to the reference in order to generate the sequence again. Figure 4(b) shows the decompression process on the original example in Figure 4(a). The main contribution of this work lies in the ability of the ASG to support repeated matches in addition to multiple alignments using the left and right actions. Moreover, having the intermediate action sequence format may allow for faster search in the target for genes of known location in the reference.

## IV. RESULTS

In this section, the performance of the proposed method HADC was evaluated in terms of both the compression ratio, which is the ratio of the size of original sequence to the size of compressed sequence as defined by Hosseini et al. [5], and compression time. In addition, the results are compared with some of the previously proposed compression algorithms

and the gain as well as the space saving were calculated as described by Yao et al. [6].

### A. DATASET

More than 30 experiments using several different target and reference sequences and different hyperparameter values were conducted to test the performance of HADC. These targets and references are introduced in Table 1. The dataset used can be easily retrieved with open access and mostly contains FASTA files of genomes from various organisms. This includes *H. sapiens* and *E. coli* sequencing data which were obtained from GenBank on National Center for Biotechnology Information (NCBI), two versions of *S. cerevisiae* sequencing data obtained from University of California Santa Cruz (UCSC), and more as shown in Table 1. A reference genome is selected such that it is used to compress another target genome of the same species as proposed by several studies [7], [8], [28]. However, it is worth mentioning that HADC works on targets and references that do not necessarily belong to the same species.

**TABLE 1. Dataset overview.**

| Organism | Genome (FASTA file) | | Size (MB) | Retrieved from |
|---|---|---|---|---|
| *H. sapiens* | Ref. | HSCHM8 | 141 | http://biomirror.aarnet.edu.au/biomirror/ncbigenomes/H_sapiens/Assembled_chromosomes/seq/ |
| | Target | HS8 | 140 | https://www.ncbi.nlm.nih.gov/nuccore/568815590 |
| *A. thaliana* | Ref. | TAIR9 | 115 | https://www.arabidopsis.org/download_files/Genes/TAIR9_genome_release/TAIR9_chr_all.fas |
| | Target | TAIR10 | 115 | https://www.arabidopsis.org/download_files/Genes/TAIR10_genome_release/TAIR10_chromosome_files/TAIR10_chr_all.fas |
| *H. sapiens* | Ref. | HSCHM16 | 88.7 | http://biomirror.aarnet.edu.au/biomirror/ncbigenomes/H_sapiens/Assembled_chromosomes/seq/ |
| | Target | HS16 | 87.3 | https://www.ncbi.nlm.nih.gov/nuccore/NC_000016.10 |
| *S. cerevisiae* | Ref. | sacCer2 | 11.8 | https://hgdownload.soe.ucsc.edu/goldenPath/sacCer2/bigZips/ |
| | Target | sacCer3 | 11.8 | https://hgdownload.soe.ucsc.edu/goldenPath/sacCer3/bigZips/ |
| *E. coli* | Ref. | NC_017651.1 | 4.87 | https://www.ncbi.nlm.nih.gov/nuccore/NC_017651.1 |
| | Target | NC_017652.1 | 4.87 | https://www.ncbi.nlm.nih.gov/nuccore/NC_017652.1 |

Of course, higher compression ratios are achieved when the target and reference are from the same species since they will be highly similar.

### B. EXPERIMENTAL SETUP

HADC was implemented and tested in Python. The experiments were performed on a machine running 64-bit Ubuntu with 3.3 GHz Intel® Core™ i9-9940X CPU and 32 GB RAM.

The hyperparameters of HADC, such as the number of blocks, the value of $k$, play an important role in the compression results. Different values of these hyperparameters were tested for each dataset. Regarding the value of $k$, experiments showed that the best results were achieved using a $k$ value of 9; since using larger values of $k$ (10, 12, 21) usually caused a dramatic increase in the compression time with slight improvement to the compression ratio, while using smaller values of $k$ (6, 7, 8) decreased the compression ratio a lot, thus the value 9 proved to be the sweet spot for $k$ as shown in Table 2.

**TABLE 2. HADC performance using different values of k.**

| Value of k | Compression Ratio | | Compression Time (min) | |
|---|---|---|---|---|
| | HS8 | NC_017652.1 | HS8 | NC_017652.1 |
| 7 | 48.3 | 59,980 | 5 | 0.06 |
| **9** | **50.1** | **61,556** | **8** | **0.11** |
| 11 | 51.5 | 60,123 | 13 | 0.5 |

**TABLE 3. Compression performance results.**

| Dataset | Compression Method | Performance Metric | |
|---|---|---|---|
| | | Comp. Ratio | Comp. Time (min) |
| HS8 | iDoComp | 241.7 | 78 |
| | GDC 2 | 239.7 | 54 |
| | ERGC | 4.4 | 150 |
| | HRCM | 252 | 50 |
| | HADC | 50.1 | **8** |
| TAIR10 | iDoComp | 59,180 | **0.12** |
| | GDC 2 | 39,453 | 0.33 |
| | ERGC | 14,795 | 0.3 |
| | HRCM | 23,672 | 0.27 |
| | HADC | **83,790** | 0.3 |
| HS16 | iDoComp | 173.9 | 69 |
| | GDC 2 | 193.3 | 50 |
| | ERGC | 9.16 | 131 |
| | HRCM | 190.8 | 48 |
| | HADC | 86.4 | **8.5** |
| sacCer3 | iDoComp | 6,054 | 0.19 |
| | GDC 2 | 3,027 | 0.5 |
| | ERGC | 2,018 | 0.9 |
| | HRCM | 2,018 | 0.21 |
| | HADC | 2,471 | **0.1** |
| NC_017652.1 | iDoComp | 57,976 | 0.2 |
| | GDC 2 | 20,603 | 0.6 |
| | ERGC | 15,735 | 0.86 |
| | HRCM | 26,829 | 0.18 |
| | HADC | **61,556** | **0.11** |

The other hyperparameter is the number of blocks to divide the reference and target into. This hyperparameter was mostly data-dependent as a highly similar reference and target produced higher compression ratios when they were divided into a small number of blocks or even compressed as one block. On the other hand, an unsimilar reference and target produced better compression ratios when divided into several blocks (10 or more).

## V. EXPERIMENTAL RESULTS & DISCUSSION

The results of these of experiments were recorded and compared to the results of other state-of-the-art reference-based compression algorithms such as HRCM [6], GDC 2 [26], iDoComp [28] and so on as shown in Table 3. The performance of gzip is recorded in Table 4 in order to show the improvement achieved by adding the reference-based compression layer added in HADC above gzip.

The results in Table 3 show that HADC performs efficient compression with extremely high compression ratios in some cases. Specifically, for small datasets where the reference and the target are very similar and only include some single nucleotide polymorphisms (SNPs), which is the case for most genomes of the same species, the compression ratio is higher than competitive reference-based algorithms as in the cases of the *A. thaliana* and *E. coli* genomes used reaching a gain of up to 82% and 74% (compared to ERGC) in *A. thaliana* and *E. coli* respectively. However, with different genomes,

**TABLE 4. Gzip performance results.**

| Dataset | Compression Method | Performance Metric | |
|---|---|---|---|
| | | Comp. Ratio | Comp. Time (min) |
| HS8 | gzip | 3.4 | 1 |
| | HADC | **50.1** | 8 |
| TAIR10 | gzip | 3.4 | 0.4 |
| | HADC | **83,790** | **0.3** |
| HS16 | gzip | 3.8 | 0.5 |
| | HADC | **86.4** | 8.5 |
| sacCer3 | gzip | 3.3 | 0.16 |
| | HADC | **2,471** | **0.1** |
| NC_017652.1 | gzip | 3.3 | 0.03 |
| | HADC | **61,556** | 0.11 |

HADC is inferior to iDoComp, GDC 2 and HRCM. Those three algorithms outperform HADC in the *H. sapiens* dataset but HADC outperforms ERGC for all the mentioned datasets.

To further clarify the effect of the compression ratio, the compression result expressed as bits per base was also calculated for each algorithm on each dataset. This was done by dividing the compressed file size over the number of bases in the original file. Figure 5(a) illustrates the bits per base metric in a line plot and indeed confirms that HADC gives the best results for *A. thaliana* and *E. coli* datasets and closely competitive results for *S. cerevisiae*.

As for the compression time, results in Figure 5(b) showed that HADC was the best. Out of all the tested reference-based algorithms, HADC provides the least compression time for all datasets, except the TAIR (*A. thaliana*) dataset in which the algorithm's compression time was 18 seconds while iDoComp's compression time was 7.2 seconds for this dataset. It can also be inferred from Table 3 that HADC saves more than 50% of the time taken by ERGC in most experiments. It is worth noting that the time to build the hash table is included in the recorded compression times of HADC.

Compared to the performance of gzip, HADC produces remarkably higher compression ratios in every test case reaching gains of up to 99.9%. This is obviously due to the use of the reference-based compression layer which generates an action sequence suitable for being compressed using general-purpose compression tools like gzip. Despite the fact that HADC transforms gzip from being a reference-free general-purpose compression tool to a being a reference-based special-purpose tool for FASTA files, using gzip as a final layer in HADC provides an extremely higher compression ratio than using simple bit encoding methods while consuming little time; from 0.5 to 2 seconds.

Based on these results, we conclude that, with a good choice of the reference, in order to reach the best compression ratios, HADC is more appropriate for bacteria and plants (small genome size) while iDoComp, GDC 2 and HRCM are the algorithms that produce better compression ratios on the *H. sapiens* genome.

The memory used by HADC is mainly consumed in the hash table and the reference and target block data. The hash table size is directly proportional to the reference block size because during the compression of a target block, a hash table is built for only the corresponding reference block. That being said, the memory consumption directly depends on the number of blocks chosen by the user.

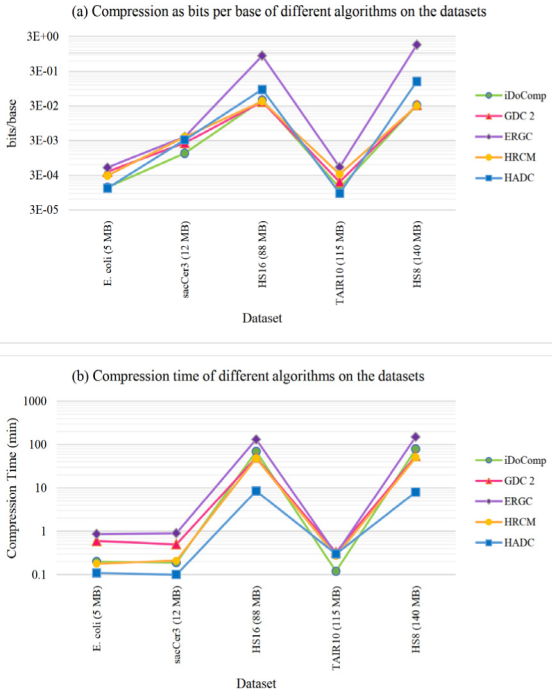


**FIGURE 5. Compression results expressed as (a) bits per base and (b) time in minutes of different algorithms on the datasets.**

## VI. CONCLUSION

In this work, a new hybrid lossless compression method for DNA sequences (HADC) is proposed. HADC performs a layer of reference-based compression to produce an intermediate action sequence followed by a layer of reference-free compression. The intermediate action sequence file may allow for faster search in the target for genes of known location in the reference and this is an open area that requires more exploration and future research. The experimental results show that HADC has acceptable and sometimes competitive compression ratio and compression speed. However, this method still has a lot of room for enhancement, and also more comprehensive testing is needed to further verify the results.

In the next stages, the compression ratio will continue to improve by analyzing more features of the sequence files and the intermediate action file and creating an additional post-processing layer after the ASG. The possibility of parallelizing the algorithm performed by ASG will also be studied and tested and this will greatly improve the compression speed enabling HADC to perform better on multiple FASTA files. In addition, more components will be added to the current technique to incorporate compressing FASTQ files as well.

## REFERENCES

[1] F. Sanger, S. Nicklen, and A. R. Coulson, "DNA sequencing with chain-terminating inhibitors," *Proc. Nat. Acad. Sci. USA*, vol. 74, no. 12, pp. 5463–5467, Dec. 1977, doi: 10.1073/pnas.74.12.5463.

[2] D. S. Horner, G. Pavesi, T. Castrignano, P. D. De Meo, S. Liuni, M. Sammeth, E. Picardi, and G. Pesole, "Bioinformatics approaches for genomics and post genomics applications of next-generation sequencing," *Briefings Bioinf.*, vol. 11, no. 2, pp. 181–197, Mar. 2010, doi: 10.1093/bib/bbp046.
[3] B. T. Wilhelm and J.-R. Landry, "RNA-seq—Quantitative measurement of expression through massively parallel RNA-sequencing," *Methods*, vol. 48, no. 3, pp. 249–257, Jul. 2009, doi: 10.1016/j.ymeth.2009.03.016.
[4] E. E. Schadt, S. Turner, and A. Kasarskis, "A window into third-generation sequencing," *Hum. Mol. Genet.*, vol. 19, no. R2, pp. R227–R240, Oct. 2010, doi: 10.1093/hmg/ddq416.
[5] M. Hosseini, D. Pratas, and A. Pinho, "A survey on data compression methods for biological sequences," *Information*, vol. 7, no. 4, pp. 1–23, 2016, doi: 10.3390/info7040056.
[6] H. Yao, Y. Ji, K. Li, S. Liu, J. He, and R. Wang, "HRCM: An efficient hybrid referential compression method for genomic big data," *BioMed Res. Int.*, vol. 2019, pp. 1–13, Nov. 2019, doi: 10.1155/2019/3108950.
[7] S. Deorowicz and S. Grabowski, "Data compression for sequencing data," *Algorithms Mol. Biol.*, vol. 8, no. 1, pp. 1–13, Jan. 2013, doi: 10.1186/1748-7188-8-25.
[8] Z. Zhu, Y. Zhang, Z. Ji, S. He, and X. Yang, "High-throughput DNA sequence data compression," *Briefings Bioinf.*, vol. 16, no. 1, pp. 1–15, Jan. 2015, doi: 10.1093/bib/bbt087.
[9] M. Nicolae, S. Pathak, and S. Rajasekaran, "LFQC: A lossless compression algorithm for FASTQ files," *Bioinformatics*, vol. 31, no. 20, pp. 3276–3281, Oct. 2015.
[10] J. G. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Commun.*, vol. COM-32, no. 4, pp. 396–402, Apr. 1984, doi: 10.1109/tcom.1984.1096090.
[11] P. Deutsch, "DEFLATE compressed data format specification version 1.3," RFC Editor, USA, Tech. Rep. RFC1951, 1996, doi: 10.17487/rfc1951.
[12] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, pp. 1754–1760, Jul. 2009, doi: 10.1093/bioinformatics/btp324.
[13] Y. Zhang, L. Li, Y. Yang, X. Yang, S. He, and Z. Zhu, "Light-weight reference-based compression of FASTQ data," *BMC Bioinf.*, vol. 16, no. 1, pp. 1–8, Dec. 2015, doi: 10.1186/s12859-015-0628-7.
[14] S. Chandak, K. Tatwawadi, I. Ochoa, M. Hernaez, and T. Weissman, "SPRING: A next-generation compressor for FASTQ data," *Bioinformatics*, vol. 35, no. 15, pp. 2674–2676, Aug. 2019, doi: 10.1093/bioinformatics/bty1015.
[15] S. Deorowicz and A. Danek, "GTShark: Genotype compression in large projects," *Bioinformatics*, vol. 35, no. 22, pp. 4791–4793, 2019, doi: 10.1093/bioinformatics/btz508.
[16] D. Lan, R. Tobler, Y. Souilmi, and B. Llamas, "Genozip: A universal extensible genomic data compressor," *Bioinformatics*, vol. 37, no. 16, pp. 2225–2230, Aug. 2021, doi: 10.1093/bioinformatics/btab102.
[17] L. Chen, S. Lu, and J. Ram, "Compressed pattern matching in DNA sequences," in *Proc. IEEE Comput. Syst. Bioinf. Conf.*, Aug. 2004, pp. 62–68, doi: 10.1109/CSB.2004.1332418.
[18] D. Mansouri, X. Yuan, and A. Saidani, "A new lossless DNA compression algorithm based on a single-block encoding scheme," *Algorithms*, vol. 13, no. 4, p. 99, Apr. 2020, doi: 10.3390/a13040099.
[19] A. Al-Okaily, B. Almarri, S. A. Yami, and C.-H. Huang, "Toward a better compression for DNA sequences using Huffman encoding," *J. Comput. Biol.*, vol. 24, no. 4, pp. 280–288, Apr. 2017, doi: 10.1089/cmb.2016.0151.
[20] M. D. Cao, T. I. Dix, L. Allison, and C. Mears, "A simple statistical algorithm for biological sequence compression," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2007, pp. 43–52, doi: 10.1109/dcc.2007.7.
[21] S. Wandelt, M. Bux, and U. Leser, "Trends in genome compression," *Current Bioinf.*, vol. 9, no. 3, pp. 315–326, May 2014, doi: 10.2174/15748936096661405160101 43.
[22] S. Christley, Y. Lu, C. Li, and X. Xie, "Human genomes as email attachments," *Bioinformatics*, vol. 25, no. 2, pp. 274–275, Jan. 2009, doi: 10.1093/bioinformatics/btn582.
[23] S. Deorowicz and S. Grabowski, "Robust relative compression of genomes with random access," *Bioinformatics*, vol. 27, no. 21, pp. 2979–2986, Nov. 2011, doi: 10.1093/bioinformatics/btr505.
[24] S. Kuruppu, S. J. Puglisi, and J. Zobel, "Optimized relative Lempel-Ziv compression of genomes," in *Proc. 34th Australas. Comput. Sci. Conf.*, vol. 113, 2011, pp. 91–98. [Online]. Available: https://dl.acm.org/doi/10.5555/2459296.2459307
[25] C. Wang and D. Zhang, "A novel compression tool for efficient storage of genome resequencing data," *Nucleic Acids Res.*, vol. 39, no. 7, pp. 1–6, 2011, doi: 10.1093/nar/gkr009.
[26] S. Deorowicz, A. Danek, and M. Niemiec, "GDC 2: Compression of large collections of genomes," *Sci. Rep.*, vol. 5, no. 1, pp. 1–12, Sep. 2015, doi: 10.1038/srep11565.
[27] M. H.-Y. Fritz, R. Leinonen, G. Cochrane, and E. Birney, "Efficient storage of high throughput DNA sequencing data using reference-based compression," *Genome Res.*, vol. 21, no. 5, pp. 734–740, May 2011, doi: 10.1101/gr.114819.110.
[28] I. Ochoa, M. Hernaez, and T. Weissman, "iDoComp: A compression scheme for assembled genomes," *Bioinformatics*, vol. 31, no. 5, pp. 626–633, Mar. 2015, doi: 10.1093/bioinformatics/btu698.
[29] S. Saha and S. Rajasekaran, "ERGC: An efficient referential genome compression algorithm," *Bioinformatics*, vol. 31, no. 21, pp. 3468–3475, Nov. 2015, doi: 10.1093/bioinformatics/btv399.
[30] S. Saha and S. Rajasekaran, "NRGC: A novel referential genome compression algorithm," *Bioinformatics*, vol. 32, no. 22, pp. 3405–3412, 2016, doi: 10.1093/bioinformatics/btw505.
[31] M. Silva, D. Pratas, and A. J. Pinho, "Efficient DNA sequence compression with neural networks," *GigaScience*, vol. 9, no. 11, Nov. 2020, Art. no. giaa119, doi: 10.1093/gigascience/giaa119.
[32] T. Tang and J. Li, "Transformation of FASTA files into feature vectors for unsupervised compression of short reads databases," *J. Bioinf. Comput. Biol.*, vol. 19, no. 1, Feb. 2021, Art. no. 2050048, doi: 10.1142/s0219720020500481.
[33] A. M. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proc. London Math. Soc.*, vol. 42, no. 1, pp. 230–265, 1937, doi: 10.1112/plms/s2-42.1.230.
[34] L. De Mol, "Turing machines," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed. Stanford, CA, USA: The Metaphysics Research Lab, 2021. [Online]. Available: https://plato.stanford.edu/archives/win2021/entries/turing-machine/

**SARAH ELNADY** received the bachelor's degree in computer science from the Faculty of Computers and Artificial Intelligence, Cairo University, Egypt, in 2017. She is currently pursuing the master's degree in bioinformatics. She is also working as a Teacher Assistant at the Faculty of Computers and Artificial intelligence, Cairo University. Her research interests include bioinformatics and soft computing, artificial intelligence, machine learning, and image processing.

**SABAH SAYED** received the Ph.D. degree in computer science "a computational framework for colorectal cancer," in 2019. She is currently working as a Teacher at the Faculty of Computers and Artificial intelligence, Cairo University, Egypt. She has many scientific research articles published in international journals in the topics of bioinformatics, artificial intelligence, machine learning. Her research interests include bioinformatics and biomedical, cloud computing, soft computing, image processing, artificial intelligence, data mining, high performance computing, optimization, and meta-heuristics techniques.

**AKRAM SALAH** graduated in mechanical engineering. He received the M.Sc. and Ph.D. degrees in software engineering from North Dakota State University and the Ph.D. degree in computer and information sciences from the University of Alabama at Birmingham, Birmingham, AL, USA, in 1986. He taught at The American University in Cairo, Michigan State University, and Cairo University, before he joined North Dakota State University, where he designed and started a graduate program. He worked in computer programming for seven years, before he got his Ph.D. degree. He is currently a Professor with the Faculty of Computer and Information, Cairo University. He has more than 100 published papers. His current research interests include data, knowledge, software engineering, semantics, and Semantic Web.