SCA-NGS: Secure compression algorithm for next generation sequencing data using genetic operators and block sorting

Muhammad Sardaraz and Muhammad Tahir
Department of Computer Science, Faculty of Information Sciences & Technology, COMSATS University Islamabad, Attock Campus, Attock, Punjab, Pakistan

Abstract
Recent advancements in sequencing methods have led to significant increase in sequencing data. Increase in sequencing data leads to research challenges such as storage, transfer, processing, etc. data compression techniques have been opted to cope with the storage of these data. There have been good achievements in compression ratio and execution time. This fast-paced advancement has raised major concerns about the security of data. Confidentiality, integrity, authenticity of data needs to be ensured. This paper presents a novel lossless reference-free algorithm that focuses on data compression along with encryption to achieve security in addition to other parameters. The proposed algorithm uses preprocessing of data before applying general-purpose compression library. Genetic algorithm is used to encrypt the data. The technique is validated with experimental results on benchmark datasets. Comparative analysis with state-of-the-art techniques is presented. The results show that the proposed method achieves better results in comparison to existing methods.

Keywords
NGS data, data compression, encryption, genetic algorithm

Introduction
Recent advancements in sequencing methods have led to tremendous increase in sequencing data due to which many problems arise such as memory, storage, transfer, processing, confidentiality, etc. In recent years, the rate of generating
biological data has increased significantly. In early times sequencing cost was high with low speed of data generation. Modern machines generate data at a high speed at reduced cost.² The 1000 genome project has produced a huge amount of data, that is, the generated data in the first 6 months of the project is more than the total data of NCBI GenBank data in 21 years before the project.³ The Beijing Institute of genomics produced 30 petabytes of raw data per year.⁴ The increase has left behind the computing power of modern computers.² This huge increase has motivated researchers to develop compression tools to handle the huge amount of data. There have been good achievements in compression ratio and execution time.

Data compression is a technique to reduce the size of data and address issues related to storage and transfer. There has been extensive development in compression tools for genomic data. Different techniques have been formulated for the compression of biological sequences. Techniques for genomic data take FASTA files as input and process data with both lossy and lossless techniques. For NGS data FASTQ and other formats, files are taken as input and processed with both lossy and lossless compression. Different generalized methods have been used for data compression. One of these methods is the statistical method. The statistical approach is a technique widely used for compression. The technique is based on encoding one symbol at a time and works on the probability of occurrence of symbols. The general methods of this class are Huffman and arithmetic coding. Another technique known as the Lempel-Ziv method is based on the classic creation of dictionaries.⁵ Dictionary consists of repeated substrings in data. The specialty of Lempel-Ziv is that it records the whole genomic dataset in dictionary rather than only a reference genome. In data compression methods, the permutation of data can be utilized to improve compression, for example, the assembled genomic data. This method yields better results in terms of compression ratio. Examples of this method include table compression and Burrow Wheeler Transform (BWT).

Different approaches have been opted to compress biological data. Due to the unique characteristics of biological data, general-purpose compression techniques are not suitable to compress these data. Taking this motivation into consideration, specialized compressors have been developed. These include reference-free or reverential compression tools. Some approaches use lossy compression whereas others use lossless compression. Due to high transfer costs, the data need to be stored in the cloud. The data in cloud environment need some security measures. Despite the rich literature for the data compression technique, general-purpose compression methods are still employed for compressing genome data. Many of the repositories use general-purpose compression tools. Specialized compression needs to target industrial trends of compression and develop tools that can replace general-purpose compression techniques as well as consider data security.

The fast-paced advancements have raised major concerns for the security of data. Large volumes of data have raised concerns for secure storage, privacy, and accessibility of data.⁶ Encryption along with compression is the solution to address these issues. Cryptographic schemes can be applied for the security and confidential accessibility of data.⁶ The use of cloud computing and blockchain (for
bioinformatics applications) opens new challenges for compression tools. Encryption of compressed datasets is a challenging task that needs to address security along with other parameters. General-purpose compression and encryption tools can directly be applied to genome data, but these tools cannot take advantage of the characteristics of the genomic data. The characteristics include small alphabet size, repetitive nature of the occurrence of the patterns, palindromes, etc. Most of the existing solutions do not consider security as a parameter. Only a few tools have addressed the problem of data security. Adding this parameter to the existing method will affect execution time and other parameters.

This article presents a novel compression and encryption algorithm for NGS data. The framework utilizes general-purpose compression library for compression of quality scores and other components of the data. Bases are pre-processed before applying general-purpose compressor. The resulting data are encrypted using Genetic Algorithm (GA). Crossover and mutation phases of GA are used to encrypt the data with a given key. Comparative experimental results analysis on benchmark datasets shows the effectiveness of the proposed method. The paper is organized as follows. Section 2 presents related work followed by materials and methods in section 3. Section 4 presents the results and discussion and finally, section V concludes the article.

**Literature review**

Sequence data compression techniques can be either genomic data compressors or NGS data compressors. The main difference between genomic data and NGS data is the presence of quality scores in NGS data. Like genomic sequences, NGS data also consists of headers and bases. The quality score has a broad range of alphabets that makes standard compression tools and specialized tools for genomic data inappropriate for this type of data. Both types of compression tools consist of referential or reference-free methods. In referential mode, a reference sequence is used to compress the target sequence. In reference-free mode, no reference sequence is used. This section presents compression techniques for NGS datasets. The discussion covers reference-free compression tools. Readers are referred to reviews 2,4,7 on sequence data compression for further details on different methods and categories.

Literature consists of techniques based on various methods. NAF is a tool and a format for storing compressed DNA sequences.8 The tool supports multiple formats for compression. The algorithm works in lossless reference-free mode. Like many other techniques for DNA sequence data compression, NAF also splits sequences into different streams. FASTQ files are split into bases, quality scores, and headers. Each component is processed separately. First, sequences are concatenated and then converted into 4-bits encoding. In the end, general-purpose compression library is used to further compressed the data. The main advantage of NAF is the fast execution time in both compression and decompression. The tool also consumes less memory in compression. Another compression tool named minicom is proposed for reads compression.9 Reads are indexed with $k$-minimizers
and sub-groups are created based on the similarity of the minimizers. In the next step, contigs are constructed in each group which are converted to larger contigs on the basis of minimizer-index suffix-prefix overlap similarity. The process is repeated until all contigs are merged. Comparative experimental results are shown to validate the performance of minicom. Another technique considers data compression and encryption as essential parameters for low-cost storage and transmission. The proposed algorithm uses reverse palindromes, genetic palindrome, and sub-string for compression. Substrings of different lengths are replaced with ASCII values. In the end, encryption is applied over the compressed to ensure the security of the data. Comparative experimental results are presented to validate the proposed method. FaStore is another tool for compressing raw sequencing data. The algorithm works in lossless reference-free mode. The proposed method exploits the redundancy in reads to achieve better compression. The procedure includes read clustering, optional reads re-clustering, and compression stages. Each stage has further steps to achieve the desired goals. The tool has additional options for users to discard any portion of data, that is, headers or quality scores to reduce the generated file size. The input sequences are split into different streams and specialized compressors are used to compress each stream. The algorithm has also the ability to preserve the pairing information between reads. Comparative experimental results are presented to validate the performance of the proposed method. Researchers have also worked to exploit inter-similarities between sequences to achieve better compression gains. Another algorithm LFQC applies Huffman coding for quality score compression. The algorithm divides quality score data into several chunks. Each chunk is encoded individually. The algorithm avoids successive # symbols to reduce quality score alphabets. The data are compressed with a context mixing algorithm. A similar procedure is used to compress bases. FQSqueezer is based on partial prediction and dynamic Markov-coder algorithms. FQSqueezer compresses single-end and pair-end reads of variable length. The algorithm uses previously available techniques for compression, that is, prediction by partial matching and Dynamic Markov Coder. These techniques have been improved for organizing large dictionaries, estimation of sequencing errors, reordering reads, and sharing substrings among reads. The algorithm performs better than other compared methods and shows a significant gain in compression ratio. The main drawbacks are huge memory acquisition and long execution time. Accelerated implementation of FQSqueezer is also available. The existing method runs on multi-core CPUs for multi-threading. The accelerated version uses GPUs for this purpose. The objective is to improve the performance. Improvement gain in performance is validated with experimental validations. PgRC technique is based on extracting the shortest
identical superstring among similar reads.\textsuperscript{17} PgRC shows better compression ratio than other compared techniques. The technique also has the advantage of decompression speed over the competitors. The algorithm works on approximating the shortest common superstring. Firstly, reads are partitioned based on quality and number of symbols. Reads that relate without errors are known as high-quality reads and reads that are matched including errors are known as low-quality reads. Low-quality reads that cannot be mapped wholly are partially matched with the relevant pseudo-genome area. The tools can be used to compress reads in FASTQ format. Cryfa\textsuperscript{6} is proposed to secure many formats of genomic data such as FASTA, FASTQ, VCF, SAM, and BAM. The algorithm not only applies encryption to data but also compresses the data. Cryfa uses advanced encryption methods that have a key shuffling mechanism to secure data. The method is faster in compression speed than other algorithms. To encrypt and compress data are split into three streams, that is, headers, bases, and quality scores. The separate data streams are preprocessed and transformed into ASCII characters. Then a key is generated to shuffle the content. Finally, AES encryption is used to secure the data. The main advantage is low memory consumption with faster execution speed. However, the compression ratio is low compared to other methods. MZPAQ\textsuperscript{18} is developed by combining previously developed techniques, that is MFCompress,\textsuperscript{19} ZPAQ.\textsuperscript{14} The technique divides data into three streams and uses strong context-mixing algorithm. The procedure then generates output in the form of single binary file. The compression process utilizes MFCompress for identifiers, bases, and ZPAQ for quality scores compression. SPRING\textsuperscript{20} is another tool proposed that exhibits different modes of compression. The tool has different features and modes that carry out the alignment of reads, lossy quality value compression, lossless large read compression, and frequent access, etc. The lossy mode records information such as alignment, assembly, variant calling. Reads can be effectively retrieved through lossy mode. The tool has support for variable length reads, random access, and support for high coverage datasets. The performance of the proposed method is evaluated with extensive experimental evaluation. The algorithm proposed in Chandak et al.\textsuperscript{21} compresses both unaligned and aligned reads. In both scenarios, the algorithm achieves higher compression ratio over general-purpose compression algorithms and shows improvement gain. Reads are first preprocessed to remove non-ACGT values and are stored in separate files. Reads are ordered based on the location in the genome through hash-based substrings. The algorithm has three phases, that is, reordering, encoding, and compression. Reads are reordered by mapping their loci in the genome. After reordering, reads are encoded to remove repetition and the parameters are saved in separate files. Finally, reads are compressed through Lempel-Ziv and BWT compressors. LFastqC\textsuperscript{22} compression tool splits each read into three data streams and compresses the streams using two compression tools, that is, lpaq8\textsuperscript{23} and MFCompress.\textsuperscript{19} Lpaq8 utilizes context-mixing algorithm. Context mixing uses prediction of weighted combination of estimates from different models. In LFastqC, MFCompress is used for DNA compression. Reads undergo minimal preprocessing before compression. The compression ratio
is better than some general-purpose compressors and specialized algorithms. Ipaq8 is used to compress quality scores. FCompress is another tool that splits FASTQ sequence files into three separate streams. Bases are compressed with the dictionary method by taking segments of four bases. In the next phase, 7z compressor is used to compress the values obtained in the first phase. Headers are also compressed with 7z compressor. The quality score is compressed with Huffman coding. Huffman coding is applied in blocks to achieve better compression results. The method is evaluated with experimental results. GTZ is another approach for FASTQ data compression. The technique is also capable of transmitting compressed data to the cloud server. The approach splits FASTQ files into different streams and uses context modeling for estimating probabilities of characteristics of the input data and arithmetic coding for compression. Results on benchmark datasets are presented to validate the proposed technique. Another compression technique is proposed in Fan et al. The technique integrates FM-Index and complementary context models for compression. FM-Index is used to find exact matches between the two sequences. For mismatches, complementary context models are used. The technique is evaluated with benchmark datasets. A compression and encryption tool named GP2R processes the data in two tiers in lossless reference-free mode. In the first tier, the technique searches for substrings using exact genetic palindrome, palindrome, reverse. A library is created using corresponding ASCII characters. In the second tier, the modified RSA technique is used for encryption.

Material and methods

This section presents the details of the proposed algorithm. The methodology consists of the steps shown in Figure 1. In the first step, the proposed method splits data into different streams, that is, header, bases, and quality scores. Bases are preprocessed by splitting into combinations of four bases. ASCII values of each combination are processed further. In the next phase, each component is converted to binary, and GA-based encryption is used for encryption. In the final phase, headers, bases, and quality scores are compressed with general-purpose compression library.

Compression and encryption

DNA sequences are taken as the input string. First, all non-ACGT characters are removed from the input string. All lines containing Ns are transferred to separate file and their positions are recorded. This input sequence is further divided into chunks of four. We can make a dictionary of 256 possible combinations of four letters. The position of each chunk in the array is observed. The positions are taken as integer values. To speed up the process, binary search is used to search the desired position in the array. This strategy yields compression ratio equal to 2 bits per base compression. The integer values are then converted to equivalent ASCII
values and converted to binary. These values are transferred to the encryption module and then to the BSC library for further compression. For decompression purpose, each binary value is transformed back to ASCII value where it is replaced by four characters.

To compress other components, each stream is encrypted and processed with BSC compression library. BSC is a reference-free program that uses block sorting method for data compression. The method uses parallel multi-threading approach to compress large partitions of data. Memory usage depends on the number of blocks being processed at a given time. Blocks can be of varied sizes. Algorithm 1 shows the steps in the compression process of the proposed method. Algorithm 1 shows the steps involved in compression.

The proposed algorithm uses symmetric key encryption scheme based on GA, that is, crossover and mutation operators are used to encrypt the data. The procedure is shown in Figure 2. Algorithm 2 shows the steps involved. All components, that is, DNA, header, and quality score data are processed similarly. In the crossover phase, both the key and text are converted to binary. A random point is selected as crossover point. The point is selected between 1 and 8 as the text and key are represented in 8 bits. This point is stored along with the key for decryption. To encrypt the text, the XOR of the selected bit in each byte of key and text is taken and the bit in the text is replaced with the new bit. Changing 1 bit in a character changes the character as well and thus remains unknown. In the mutation

![Figure 1. Architecture of SCA-NGS.](image-url)
Algorithm 1. Compression procedure

begin
read line l from FASTQ file
for each base line b
    if b contains non-AGCT characters
        store b in separate file along with line number
        process b with bsc library
    else
        initialize pattern array A (Array of 256 possible combinations of four DNA bases)
    end if
for counter i set to 0
    split b into segment s of length 4
    search A for s (use binary search to find the index of s in A)
    store index of si
    increment i by 4
end for
call encryption procedure (algorithm 2)
compress the resulting data with bsc library
end if
end for
end

Figure 2. An illustrative example of the encryption process. In the evaluation of the proposed algorithm, the key length was kept equal to the length of the sequence line.
operator, another point is selected as the mutation point. This point is different from the crossover point in the range of 1 and 8. The selected bit in each byte of the data is flipped. The mutation point is also saved with the key to be used for decryption.

Decompression and decryption

All files generated during compression are decompressed first with general-purpose compression library, that is, BSC. After decompression with BSC, decryption is done with the help of key and random points for crossover and mutation generated during the encryption process. In the next step, DNA is decoded as ASCII values, segments of four bases are retrieved from the array and finally, all files are combined to regenerate the original FASTQ file. Figure 3 shows the steps involved in the decompression process. In the decryption phase, the encryption process is reversed. First, the mutation is performed by flipping the random bit in each byte of the encrypted text. In the next phase, that is, crossover the XOR of the random bit (selected in encryption phase) of each byte of the text and key is taken and the bit in data is updated accordingly to recover the plain data.

Results and discussion

This section presents the experimental results of the proposed method in comparison to other state-of-the-art techniques in the literature. Experiments are
performed on NGS datasets to validate the performance of the proposed SCA-NGS. The datasets are used to validate many compression tools and review articles for comparing compression programs.²,⁷,²⁴,³⁰ Datasets used in experiments are publicly available on the NCBI website. The datasets are related to different species (Human, plants, worm, fungus, and bacteria) of varying sizes (1.7–64.18 GB). Datasets are generated with different platforms (Illumina, 454, Solid, and Ion Torrent). The varying size of datasets acquired from multiple platforms helps in the careful evaluation of experimental results. Details related to the dataset are shown in Table 1.

Compression programs for FASTQ format are selected based on the type of compression method used, streams of data targeted for compression, and

**Figure 3.** The process of decompression.

**Table 1.** NGS datasets used for experiments.

| Datasets  | Species            | Number of reads | File size (MBs) |
|-----------|--------------------|----------------|-----------------|
| SRR489793 | C. elegans         | 56,851,258     | 13132.48        |
| SRR801793 | L. pneumophila     | 10,812,922     | 2818.11         |
| ERR022075 | E. coli            | 45,440,200     | 11253.16        |
| SRR003177 | Homo sapiens       | 1,504,571      | 1672.78         |
| SRR125858 | Homo sapiens       | 124,815,011    | 52172.64        |
| SRR935126 | A. Thaliana        | 49,719,116     | 10039.24        |
| SRR611141 | Homo sapiens       | 4,853,655      | 1799.86         |
| SRR400039 | Homo sapiens       | 124,331,027    | 65723.77        |
requirements of resources. Some programs are designed only for read compression whereas some programs use lossy compression. Programs selected for comparison with SCA-NGS are lossless reference-free tools designed for FASTQ file compression. Specialized compression tools used for comparison include SPRING, NAF, and FaStore. General-purpose compression tools include 7z, Bzip2, and gzip. These tools are used to make baseline for comparison. Recent tools that use encryption along with compression are used for comparison of encryption and compression parameters of SCA-NGS.

First compression parameters are compared with compression techniques using different parameters followed by compression and encryption results. Comparative results in terms of compression and decompression time, compression and decompression memory, compression ratio, compression and encryption time, and decompression and decryption time are presented. Table 2 shows the description of various parameters used for comparison.

All programs were executed on a computer having Intel Core i5 2.4 GHz processor, equipped with 16 GB RAM, and running Ubuntu 16.04 operating system. Each tool or program was compiled and run with respective compilers. Programs were executed with default options. SPRING and NAF use lossless mode as the default option whereas FaStore was executed with lossless option in compression mode.

\[
\text{Compression Ratio} = \frac{\text{Uncompressed size}}{\text{Compressed size}} \tag{1}
\]

Table 3 shows the comparative results of the proposed algorithm with other techniques. The results for various datasets show that the proposed algorithm produces better results as compared to other algorithms. SCA-NGS shows moderate results in terms of memory consumption for all datasets. General-purpose compression tools score best in terms of memory usage among other algorithms. The

| Parameters          | Description                                                                 |
|---------------------|-----------------------------------------------------------------------------|
| Compression ratio   | The ratio between the compressed and decompressed file size is calculated with equation (1). |
| Compression time    | The time required to compress a file is known as compression time.           |
| Compression memory  | Peak memory required by a compressor during compression.                     |
| Decompression time  | The time required to decompress a file is known as decompression Time.       |
| Decompression memory| Peak memory required by a compressor during decompression.                   |

Table 2. Description of the parameters used for comparison. The compression ratio is calculated with equation (1).
Table 3. Comparative experimental results of the SCA-NGS with other methods on NGS datasets. CTime and DTime refer to compression and decompression times in seconds. CMem and DMem refer to compression and decompression memory in MBs. CRatio refers to compression ratio. We could not decompress the dataset SRR400039 with NAF due to high memory requirements.

| Programs | CTime | CMem | DTime | DMem | CRatio |
|----------|-------|-------|-------|-------|--------|
| **SRR801793 (2818.11)** | | | | | |
| SCA-NGS | 110 | 1148 | 58 | 1331 | **5.08** |
| SPRING | 386 | 3386 | 45 | 3686 | 5.0 |
| NAF | 30 | 486 | **28** | 1728 | 3.66 |
| FaStore | 673 | 872 | 93 | 736 | 4.69 |
| 7z | 436 | 6.6 | 34 | 0.55 | 3.24 |
| Gzip | 282 | **0.37** | 49 | **0.18** | 3.05 |
| Bzip2 | 439 | 6.4 | 139 | 3.6 | 3.73 |
| **ERR022075 (11253.16)** | | | | | |
| SCA-NGS | 401 | 1311 | 305 | 1528 | **5.48** |
| SPRING | 1847 | 3548 | 153 | 5529 | 5.22 |
| NAF | **126** | 16 | **104** | 2356 | 3.78 |
| FaStore | 2174 | 1847 | 428 | 1132 | 4.87 |
| 7z | 2378 | 6.6 | 200 | 0.55 | 3.39 |
| Gzip | 896 | **0.37** | 231 | **0.18** | 3.15 |
| Bzip2 | 1002 | 6.4 | 457 | 3.6 | 3.84 |
| **SRR125858 (52172.64)** | | | | | |
| SCA-NGS | 1741 | 1638 | 1531 | 2132 | **5.82** |
| SPRING | 4671 | 4104 | 519 | 6310 | 5.8 |
| NAF | **759** | 17 | **484** | 13643 | 4.24 |
| FaStore | 3856 | 2922 | 1121 | 1786 | 5.47 |
| 7z | 10456 | 6.6 | 1272 | 0.55 | 3.65 |
| Gzip | 3814 | **0.37** | 1277 | **0.18** | 3.4 |
| Bzip2 | 4738 | 6.4 | 3260 | 3.6 | 4.1 |
| **SRR611141 (1799.86)** | | | | | |
| SCA-NGS | 44 | 948 | 36 | 1142 | **4.03** |
| SPRING | 319 | 2765 | 45 | 2663 | 3.7 |
| NAF | 32 | 17 | **13** | 444 | 2.81 |
| FaStore | 312 | 1245 | 57 | 1102 | 3.51 |
| 7z | 169 | 6.6 | 19 | 0.55 | 2.62 |
| Gzip | 144 | **0.37** | 21 | **0.18** | 2.48 |
| Bzip2 | 138 | 6.4 | 119 | 3.6 | 3.02 |
| **SRR489793 (13132.48)** | | | | | |
| SCA-NGS | 548 | 1536 | 490 | 1562 | **4.65** |
| SPRING | 1573 | 3174 | 1354 | 3438 | 4.27 |
| NAF | **218** | 16 | **130** | 2765 | 3.53 |
| FaStore | 1272 | 1844 | 224 | 1636 | 4.21 |
| 7z | 2421 | 6.6 | 236 | 0.55 | 3.18 |
| Gzip | 1027 | **0.37** | 297 | **0.18** | 2.97 |
| Bzip2 | 1252 | 6.4 | 849 | 3.6 | 3.6 |
| **SRR935126 (10039.24)** | | | | | |
| SCA-NGS | 282 | 1126 | 193 | 1433 | **5.48** |
| SPRING | 1269 | 2532 | 847 | 2728 | **5.22** |

(continued)
The proposed algorithm yields better compression ratio with low execution time on many datasets as compared to specialized compression tools. In specialized compressors, NAF has smaller execution time in comparison to other tools with the cost of low compression ratio. NAF also has the advantage of low memory consumption in compression however in decompression memory consumption is high. SPRING yields compression ratio closer to SCA-NGS at the cost of higher execution time. Memory consumption of SPRING remains high for both compression and decompression.

For dataset SRR801793 SCA-NGS achieved percent improvement gain of 1.5, 28.4, and 10.65 in terms of compression ratio over SPRING, NAF, and FaStore, respectively. In terms of execution time, NAF is faster than the other methods. The proposed algorithm achieves 71.5, and 83.6 percent improvement gain over SPRING and FaStore.

For the ERR022075 dataset, the proposed algorithm yields better compression ratio as compared to other algorithms. SPRING remains closer to the proposed algorithm. However, the proposed algorithm has the advantage in compression and decompression time. The percent improvement gains of the proposed algorithm over SPRING and FaStore in terms of execution time is 78.2 and 81.5, respectively. NAF leads all the compared methods in execution time. In terms of compression ratio, the proposed algorithm achieves percent improvement gain of 4.74, 23.37, and 11.13 over SPRING, NAF, and FaStore, respectively.

| Programs | CTime | CMem | DTime | DMem | CRatio |
|----------|-------|-------|-------|-------|--------|
| NAF      | 107   | 16    | 73    | 2662  | 4.32   |
| FaStore  | 871   | 1445  | 398   | 1136  | 5.16   |
| 7z       | 2312  | 6.6   | 258   | 0.55  | 3.72   |
| Gzip     | 755   | 0.37  | 175   | 0.18  | 3.42   |
| Bzip2    | 936   | 6.4   | 592   | 3.6   | 4.19   |

SRR003177 (1672.78)

| Programs | CTime | CMem | DTime | DMem | CRatio |
|----------|-------|-------|-------|-------|--------|
| SCA-NGS  | 37    | 1638  | 26    | 1532  | 5.12   |
| SPRING   | 246   | 2472  | 118   | 2532  | 4.97   |
| NAF      | 22    | 16    | 13    | 378   | 3.48   |
| FaStore  | 169   | 1002  | 61    | 856   | 4.79   |
| 7z       | 336   | 6.6   | 19    | 0.55  | 3.3    |
| Gzip     | 129   | 0.37  | 18    | 0.18  | 3.03   |
| Bzip2    | 135   | 6.4   | 86    | 3.6   | 3.73   |

SRR400039 (65723.77)

| Programs | CTime | CMem | DTime | DMem | CRatio |
|----------|-------|-------|-------|-------|--------|
| SCA-NGS  | 2640  | 1556  | 2580  | 1752  | 4.19   |
| SPRING   | 4618  | 3365  | 3112  | 3722  | 4.14   |
| NAF      | 911   | 17    | –     | –     | 3.14   |
| FaStore  | 5416  | 2146  | 1823  | 1540  | 3.94   |
| 7z       | 12255 | 6.6   | 1316  | 0.55  | 3.15   |
| Gzip     | 4498  | 0.37  | 1175  | 0.18  | 2.96   |
| Bzip2    | 6078  | 6.4   | 4137  | 3.6   | 3.64   |

Note: Bold entries show best results for each parameter.
In case of SRR125858 dataset, NAF remains the fastest among all algorithms. SCA-NGS is 62.72 and 54.84 percent faster than SPRING and FaStore, respectively. In terms of compression ratio, SCA-NGS achieves percent improvement gain of 0.34, 27.14, and 6.01 over SPRING, NAF, and FaStore, respectively. In terms of memory consumption, SPRING and FaStore consume more memory in comparison to other methods.

Percent improvement gain of SCA-NGS for dataset SRR61141 in terms of execution time over SPRING and FaStore is 86.2 and 85.84, respectively. Like other datasets, NAF is faster in all compared methods. In terms of compression ratio, percent improvement gain of SCA-NGS over the compared methods is 8.18, 20.34, and 12.9. SPRING and FaStore consume more memory both for compression and decompression.

The proposed algorithm performs better than all other algorithms in compression ratio for dataset SRR489793. NAF is faster in comparison to all compared methods. SCA-NGS achieves significant reduction in file size as compared to other methods. The results show that the proposed algorithm achieved percent improvement gain of 6516 and 56.91 over SPRING and FaStore respectively in terms of execution time. In terms of compression ratio percent improvement gain of SCA-NGS over SPRING, NAF, and FaStore is 8.17, 24.08, and 9.46, respectively.

In case of SRR935135 dataset, SCA-NGS achieved better results in terms of compression ratio and execution time except for NAF. NAF performs both compression and decompression in shorter execution time in comparison to other methods at the cost of compression ratio. Improvement gain attained by SCA-NGS in terms of compression ratio over SPRING, NAF, and FaStore is 4.74, 21.16, and 5.83 respectively. Improvement gain in terms of execution time over SPRING and FaStore is 77.7 and 67.62.

The proposed algorithm yields better compression ratio for the homo-sapiens dataset (SRR003177). Execution time is high as compared to NAF. There is a significant reduction in file size. The proposed algorithm achieved percent improvement gain in execution time of 86.2 and 85.89 over SPRING and FaStore, respectively. In terms of compression ratio, percent improvement gain of the proposed algorithm over the compared methods is 8.18, 20.34, and 12.9.

For the SRR40039 dataset, NAF yields the worst compression ratio with the fastest execution time and low memory consumption for compression. The proposed algorithm has slightly higher gain over SPRING. Percent improvement gain in terms of the compression ratio of SCA-NGS over SPRING, NAF, and FaStore is 1.19, 25, and 5.96 respectively. For the same dataset, the improvement gain in terms of compression time over SPRING and FaStore is 42.83 and 51.25.

Table 4 shows the comparative results of the proposed algorithm with Cryfa and GP²R algorithms. The results include encryption time in addition to parameters of the previous results. The results show that the proposed algorithm has much better compression ratio as compared to Cryfa and GP²R on all datasets. In most cases, the size of compressed data with the proposed algorithm is less than half of the size of the compressed data with Cryfa. Cryfa performs better in terms of execution time.
and memory. GP$^2$R is faster and consumes less memory in comparison to SCA-NGS however compression ratio is worst in comparison to SCA-NGS.

### Conclusion

Data compression is extensively studied topic in bioinformatics. Many tools have been developed to compress biological data. Proper measures are required to maintain its confidentiality, integrity, and authenticity. Previously some encryption

| Algorithms      | CETime | CEMem | DDTime | DDMem | CR  |
|-----------------|--------|-------|--------|-------|-----|
| SRR801793 (2818.11) |        |       |        |       |     |
| SCA-NGS         | 180    | 1148  | 58     | 1331  | 5.09|
| Cryfa           | 55.98  | 6     | 51     | 23    | 2.09|
| GP$^2$R         | 75     | 352   | 55     | 302   | 3.56|
| ERR022075 (11253.16) |       |       |        |       |     |
| SCA-NGS         | 552    | 1131  | 305    | 1528  | 5.48|
| Cryfa           | 477    | 6     | 442    | 22    | 1.96|
| GP$^2$R         | 512    | 371   | 431    | 317   | 3.51|
| SRR125858 (52172.64) |       |       |        |       |     |
| SCA-NGS         | 2437   | 1638  | 1531   | 2132  | 4.76|
| Cryfa           | 2047   | 6     | 2362   | 23    | 1.99|
| GP$^2$R         | 2144   | 364   | 2016   | 342   | 3.46|
| SRR611141 (1799.86) |       |       |        |       |     |
| SCA-NGS         | 102    | 948   | 36     | 1142  | 4.03|
| Cryfa           | 41     | 6     | 37     | 21    | 1.94|
| GP$^2$R         | 44     | 342   | 35     | 302   | 3.69|
| SRR489793 (13132.48) |       |       |        |       |     |
| SCA-NGS         | 876    | 1536  | 490    | 1562  | 4.65|
| Cryfa           | 635    | 7     | 716    | 21    | 1.91|
| GP$^2$R         | 691    | 345   | 635    | 331   | 3.51|
| SRR935126 (10039.24) |       |       |        |       |     |
| SCA-NGS         | 412    | 1126  | 193    | 1433  | 5.33|
| Cryfa           | 322    | 6     | 321    | 21    | 1.96|
| GP$^2$R         | 354    | 332   | 321    | 302   | 3.64|
| SRR003177 (1672.78) |       |       |        |       |     |
| SCA-NGS         | 68     | 1638  | 26     | 1532  | 5.12|
| Cryfa           | 50     | 8     | 53     | 28    | 1.85|
| GP$^2$R         | 62     | 352   | 45     | 321   | 3.58|
| SRR400039 (65723.77) |       |       |        |       |     |
| SCA-NGS         | 3231   | 1556  | 2580   | 1752  | 4.14|
| Cryfa           | 2387   | 7     | 2413   | 21    | 2.0 |
| GP$^2$R         | 2561   | 348   | 2525   | 341   | 3.67|

Note: Bold entries show best results for each parameter.
techniques were applied but they proved to be futile enough to gain good compression advantage. The compression ratio was much lower than other state-of-the-art algorithms. Encryption adversely impacts the execution time, memory acquisition. We have proposed a secure method for encryption and compression of NGS data. Our compression ratio with encryption is comparatively better as compared to previous algorithms. However, there could still be improvements over encryption time. Results vary over different architectures and processors. The nature of the dataset also impacts the overall compression gain. The security of data has been applied through crossover and mutation operations of the genetic algorithm. The compression of data has been applied by the grouping of data to different ASCII values and later transforming them into binary values. A general-purpose compression library is used to compress different components.

Declaration of conflicting interests
The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding
The author(s) received no financial support for the research, authorship, and/or publication of this article.

ORCID iD
Muhammad Sardaraz https://orcid.org/0000-0002-7169-8683

References
1. Sardaraz M, Tahir M, Ikram AA, et al. SeqCompress: an algorithm for biological sequence compression. Genomics 2014; 104: 225–228.
2. Sardaraz M, Tahir M and Ikram AA. Advances in high throughput DNA sequence data compression. J Bioinform Comput Biol 2016; 14: 1630002.
3. Kahn SD. On the future of genomic data. Science 2011; 331: 728–729.
4. Deorowicz S and Grabowski S. Data compression for sequencing data. Algorithms Mol Biol 2013; 8: 25.
5. Seroussi G and Lempel A. Lempel-Ziv compression scheme with enhanced adaption. Google Patents, 1993.
6. Hosseini M, Pratas D and Pinho AJ. Cryfa: a secure encryption tool for genomic data. Bioinformatics 2019; 35: 146–148.
7. Zhu Z, Zhang Y, Ji Z, et al. High-throughput DNA sequence data compression. Brief Bioinform 2015; 16: 1–15.
8. Kryukov K, Ueda MT, Nakagawa S, et al. Nucleotide archival format (NAF) enables efficient lossless reference-free compression of DNA sequences. Bioinformatics 2019; 35: 3826–3828.
9. Liu Y, Yu Z, Dinger ME, et al. Index suffix–prefix overlaps by (w, k)-minimizer to generate long contigs for reads compression. Bioinformatics 2019; 35: 2066–2074.
10. Hossein SM, De D and Mohapatra PKD. DNA sequence compression using RP/GP 2 method with information storage and security. *Microsyst Technol* 2020; 26: 2159–2172.

11. Roguski Ł, Ochoa I, Hernaez M, et al. FaStore: a space-saving solution for raw sequencing data. *Bioinformatics* 2018; 34: 2748–2756.

12. Tang T and Li J. Transformation of FASTA files into feature vectors for unsupervised compression of short reads databases. *J Bioinform Comput Biol* 2021; 19: 2050048.

13. Nicolae M, Pathak S and Rajasekaran S. LFQC: a lossless compression algorithm for FASTQ files. *Bioinformatics* 2015; 31: 3276–3281.

14. ZPAQ Home Page, http://mattmahoney.net/dc/zpaq.html (accessed 25 July 2020).

15. Deorowicz S. FQSqueezer: k-mer-based compression of sequencing data. *Sci Rep* 2020; 10: 1–9.

16. Amich M, De Luca P and Fiscale S. Accelerated implementation of FQSqueezer novel genomic compression method. In: *2020 19th international symposium on parallel and distributed computing (ISPDC)*, 2020, Warsaw, Poland: pp.158–163. IEEE.

17. Kowalski TM and Grabowski S. PgRC: pseudogenome-based read compressor. *Bioinformatics* 2020; 36: 2082–2089.

18. El Allali A and Arshad M. MZPAQ: a FASTQ data compression tool. *Source Code Biol Med* 2019; 14: 3.

19. Pinho AJ and Pratas D. MFCompress: a compression tool for FASTA and multi-FASTA data. *Bioinformatics* 2014; 30: 117–118.

20. Chandak S, Tatwawadi K, Ochoa I, et al. SPRING: a next-generation compressor for FASTQ data. *Bioinformatics* 2019; 35: 2674–2676.

21. Chandak S, Tatwawadi K and Weissman T. Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis. *Bioinformatics* 2018; 34: 558–567.

22. Al Yami S and Huang C-H. LFastqC: a lossless non-reference-based FASTQ compressor. *PLoS One* 2019; 14: e0224806.

23. LPAQ Home Page, http://mattmahoney.net/dc/#lpaq (accessed 25 July 2020).

24. Sardaraz M and Tahir M. FCompress: an algorithm for FASTQ sequence data compression. *Curr Bioinform* 2019; 14: 123–129.

25. Xing Y, Li G, Wang Z, et al. GTZ: a fast compression and cloud transmission tool optimized for FASTQ files. *BMC Bioinform* 2017; 18: 233–242.

26. Fan W, Dai W, Li Y, et al. Complementary contextual models with FM-index for DNA compression. In: *2017 data compression conference (DCC)*, 2017, Utah, USA: pp.82–91. IEEE.

27. Hossein SM, De D, Mohapatra PKD, et al. DNA sequences compression by GP² R and selective encryption using modified RSA technique. *IEEE Access* 2020; 8: 76880–76895.

28. BSC Home Page, https://libbsc.com/ (accessed 25 July 2020).

29. Tahir M, Sardaraz M, Mehmood Z, et al. CryptoGA: a cryptosystem based on genetic algorithm for cloud data security. *Clust Comput* 2021; 24: 739–752.

30. Roguski Ł and Deorowicz S. DSRC 2—industry-oriented compression of FASTQ files. *Bioinformatics* 2014; 30: 2213–2215.

**Author biographies**

Muhammad Sardaraz received his master’s degree in computer science from Foundation University Islamabad. He completed his PhD in Computer Science in 2016 from Iqra University Islamabad, Pakistan. He worked as a lecturer in the Department of Computer
Science, University of Wah, Wah Cantt, Pakistan. Presently Dr. Sardaraz is working as an assistant professor in the Department of Computer Science COMSATS University Islamabad, Attock campus, Pakistan. His research interests are cloud computing, cluster and grid computing, and bioinformatics.

Muhammad Tahir completed PhD (Computer Science) from the Department of Computing and Technology, Iqra University, Islamabad, Pakistan in 2016. He worked as a lecturer in the Department of Computer Science, University of Wah, Wah Cantt, Pakistan. He is currently working as an assistant professor in the Department of Computer Science COMSATS University Islamabad, Attock campus, Pakistan. His research interests are in parallel and distributed computing, Hadoop Mapreduce framework, IoT, bioinformatics.