Universidad de la República

Facultad de Ingeniería

Instituto de Computación

Compresión de datos generados por secuenciación de ADN por nanoporos

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación

Supervisor:
Dr. Álvaro Martín
Co-Supervisor:
Ing. Guillermo Dufort

Autores:
Nicolás Izquierdo
Gonzalo Larghero
Ingeniería en Computación

Montevideo, 2020

Resumen

La secuenciación de ADN es una de las tecnologías con mayor crecimiento y potencial dentro del siglo XXI. Dentro de este contexto, recientemente la empresa Oxford Nanopore Technologies desarrolló el dispositivo MinION, capaz de secuenciar cadenas de ADN a través de una tecnología novedosa que ofrece ventajas destacables sobre otras anteriores. Como es usual en los datos de secuenciación, los archivos generados (de formato fast5) tienen un tamaño promedio grande, lo cual dificulta su almacenamiento y transmisión. Para esto se apunta a la creación de un compresor de datos, desarrollado específicamente con el objetivo de minimizar el tamaño de este tipo de archivos.

La metodología utilizada para evaluar los resultados es el desarrollo de un software usando el lenguaje C++, de nombre F5Comp, el cual es comparado con otros compresores específicos tanto para el formato fast5 como generales. Se comparan tanto tiempos de ejecución como tasas de compresión.

Los resultados obtenidos de la comparación indican que F5Comp desarrollado específicamente para el formato fast5 tiene mejores resultados en cuanto a reducción de espacio de almacenamiento, al coste de tener mayores tiempos de ejecución.

Índice general

Capítulo 1

Introducción

En estos últimos años la secuenciación de ADN ha tenido un crecimiento acelerado debido a sus grandes aportes a la medicina, así como también a la disminución de sus costos. *Oxford Nanopore Technologies*[1] (ONT), una de las empresas mas importantes dedicadas a este rubro, tiene entre sus productos un dispositivo de secuenciación de ADN portátil llamado MinION. Este permite realizar análisis en pequeñas escalas produciendo archivos en un formato específico denominado fast5.

Actualmente los archivos generados por cada secuenciación de MinION pueden llegar a ocupar cientos de GBs, distribuidos en múltiples archivos mas pequeños. Para dar una idea, los ejemplos de secuenciaciones de ADN provistos por *Genomic DNA project*[2] ocupan desde 4 a 105 GB, por lo que al trabajar con múltiples secuenciaciones la cantidad de datos crece rápidamente. El hecho de que los archivos sean tan masivos genera grandes costos debido a que se necesita una significativa inversión en hardware de almacenamiento. Además aumenta los tiempos de envío y ralentiza cualquier tipo de procesamiento que se vaya a realizar sobre ellos. En el mercado existen distintos compresores especializados en el formato fast5, como por ejemplo Picopore[3] y F5Pack[4], los cuales serán analizados en el capítulo 3. Además se destaca que varios compresores especializados han quedado obsoletos al actualizarse la versión del formato fast5. El objetivo de este proyecto es el desarrollo de un compresor de datos sin pérdida para archivos con formato fast5 utilizando el lenguaje

[1] https://nanoporetech.com/

[2] https://github.com/nanopore-wgs-consortium/NA12878

[3] https://github.com/scottgigante/picopore

[4] https://github.com/mateidavid/fast5/blob/master/python/bin/f5pack

de programación C++; a este compresor lo llamamos F5Comp.

Agregamos además la posibilidad de extraer datos de los archivos como: posibles errores en las lecturas, la cantidad de lecturas total y sus valores. Complementariamente presentamos un utilitario para graficar los datos extraídos, lo cual agrega valor al proyecto.

Algunas de las hipótesis que se ponen a prueba en este proyecto son:

- La baja efectividad de un compresor genérico en comparación con un compresor a medida.
- La obtención de tiempos de compresión menores al usar C++ en comparación con herramientas desarrolladas en otros lenguajes.

A continuación se presenta un análisis de los formatos de archivos, seguido por una revisión de los antecedes y la solución propuesta, tanto el diseño como la implementación. Desarrollamos F5Comp con un enfoque extensivo en el cual agregar nuevos métodos de compresión implica un bajo costo, y modular, ya que se puede intercambiar distintas secciones sin necesidad de ajustes adicionales. Luego se expone el trabajo experimental y las conclusiones, con anexos que podrían resultar de interés a quien quiera continuar con esta investigación. Utilizamos una metodología cuantitativa, basada en el análisis y medición del espacio ocupado por los archivos pre y post compresión. Medimos además los tiempos de compresión, los cuales son incluidos en las comparaciones de los datos obtenidos. La información se representa en forma gráfica para simplificar su comprensión. Los resultados obtenidos de la comparación indican que F5Comp tiene el mejor nivel de compresión entre los métodos testeados, los cuales incluyen dos de compresión genéricos (rar y zip) y un compresor especifico (Picopore). Sin embargo por ser un compresor hecho específicamente para este tipo de datos requiere la lectura del contenido del archivo, no alcanza con leer el archivo como una unidad, por lo que los tiempos de compresión resultaron ser mayores que los de los compresores genéricos.

Capítulo 2

Tecnología de secuenciación por Nanoporos

1. Secuenciación de ADN

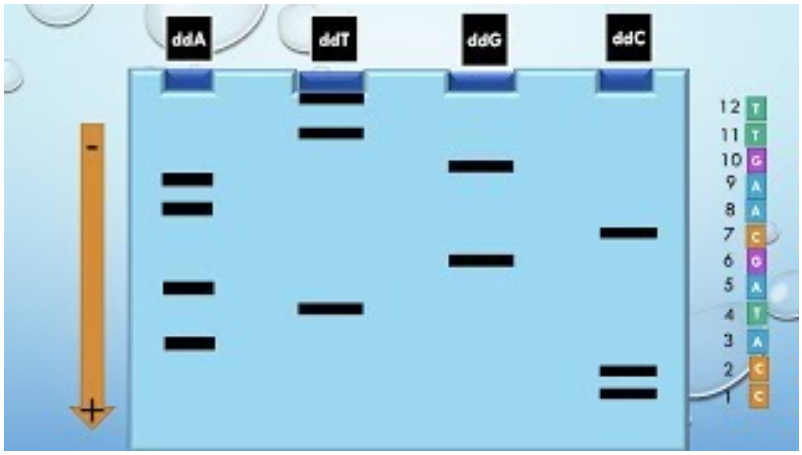
El *ácido desoxirribonucleico* o ADN es un polímero que contiene las instrucciones genéticas usadas en el desarrollo y funcionamiento de todos los seres vivos [1] y tiene entre otras funciones la trasmisión de características de los progenitores a sus descendientes. La molécula del ADN está formada por dos cadenas enrolladas entre sí formando una estructura de doble hélice unidas por pares de bases. Para poder determinar el orden de los nucleótidos A (*adenina*), C (*citosina*), G (*guanina*), T (*timina*) en una secuencia corta de ADN o ARN, el cual a diferencia del ADN consiste en una única cadena, se utiliza un conjunto de técnicas llamado secuenciación de ADN[2]. Esta información se utiliza dentro de los campos de la biología y en especial para aplicaciones dentro de la biomedicina. Por ejemplo, en un futuro podría utilizarse la secuenciación de ADN para ofrecer un tratamiento médico personalizado, a la medida de las necesidades de un individuo, con base en las variantes genéticas de su genoma. Normalmente la secuenciación de ADN requiere al menos uno de dos procesos: PCR (*polymerase chain reaction*- reacción en cadena de la polimerasa) o un etiquetado químico. El primero es un método desarrollado en 1986 que permite obtener rápidamente una gran cantidad de copias de un fragmento de ADN y el segundo es el proceso en el cual se adjunta una molécula a ciertas bio-moléculas (anticuerpos, aminoácidos y proteínas) con el objetivo de identificarlas.

2. Generaciones de tecnologías de secuenciación

2.1. Primera generación

El método de Sanger que da origen a la primera generación de tecnologías de secuenciación consiste en repartir hebras de ADN y enzimas polimerasa [3] (encargadas de replicar el ADN) en cuatro tubos de ensayo distintos, cada uno correspondiente a una base modificada. Al calentar cada tubo, las dos hebras de ADN se separan y las enzimas polimerasa se encargan de completar las bases en los pares correspondientes (A-T y C-G). Esto continúa hasta que aleatoriamente la enzima complete el par con una base modificada, luego de esto se detiene y continua con la siguiente hebra. Finalmente se utiliza un gel para organizar las hebras por largo y definir en qué posición se encuentra cada base. El proceso de lectura se puede visualizar en la figura 2.1, en la cual cada columna corresponde a un tubo distinto y cada marca negra corresponde al final de una hebra.

Figura 2.1: Secuenciación de primera generación[4]



2.2. Segunda generación

En comparación con el método de Sanger, la tecnología de secuenciación de segunda generación utiliza un enfoque de secuenciación masivo y paralelo, en el cual el ADN es extraído y fragmentado. A cada uno de los fragmentos se lo prepara y se le añaden adaptadores, que son secuencias cortas de ADN sintetizadas químicamente que pueden añadirse en los extremos de otras moléculas de ADN. Una vez que todos los fragmentos cuentan con adaptadores, cada uno de ellos es secuenciado

en paralelo. El resultado de secuenciación es por lo tanto un conjunto numeroso de lecturas de fragmentos de ADN. En general, estas lecturas individuales se ensamblan posteriormente para obtener el genoma.

2.3. Tercera generación

La secuenciación de tercera generación o secuenciación de lectura larga es la familia de métodos de secuenciación de ADN actualmente en desarrollo más activo. Las tecnologías de esta familia se diferencian por tener la capacidad de producir lecturas sustancialmente más largas que las de secuenciación de segunda generación. Aunque esto es una gran ventaja, los datos de secuenciación de tercera generación tienen tasas de error mucho más altas, complicando el ensamblaje del genoma y el análisis de los datos. Las principales empresas en desarrollo de tecnologías de tercera generación son Pacific Biosciences , Oxford Nanopore Technologies , Quantapore (CA-EE. UU.) y Stratos (WA-EE. UU.). Cada una de estas empresas utiliza diferentes enfoques al momento de secuenciar. Pacific Biosciences desarrolló una plataforma de secuenciación en tiempo real de una sola molécula (SMRT). Por otro lado Oxford Nanopore Technologies, Stratos Genomics y Quantapore utilizan el método de nanoporos con algunas variaciones entre sí. Por ejemplo Stratos Genomics espacia las bases de ADN en el cual, con inserciones poliméricas con un método llamado secuenciación por expansión, se insertan polímeros entre las bases para facilitar la lectura del ADN.

3. Nanoporos

La *secuenciación por nanoporos* es un método de tercera generación usado para secuenciar biopolímeros, específicamente polímeros en forma de ADN o ARN que a diferencia de los métodos anteriores de secuenciación no requiere de la aplicación de PCR o de un etiquetado químico para su funcionamiento. Es una tecnología portable y escalable que permite análisis en tiempo real de largos fragmentos de ADN o ARN y que funciona monitoreando los cambios en una corriente eléctrica mientras los ácidos nucleicos pasan a través de un nanoporo proteico. Los nanoporos se pueden obtener de muchas formas distintas. En el caso de ONT, los nanoporos se obtienen

de forma biológica utilizando una proteína patentada para generar poros en una membrana. Este tipo de proteínas son comunes en la naturaleza. Por ejemplo, la alfa-hemolisina [5], que se encuentra en las membranas celulares, actúa como canal para la transferencia de moléculas e iones a través de la pared celular. Estos poros tienen una alta estabilidad y diámetro de aproximadamente 1 nm. Los nanoporos de proteínas son fáciles de producir y modificar a bajo costo. Sin embargo, futuras generaciones de dispositivos de secuenciación por nanoporos probablemente usen nanoporos fabricados de forma sintética. Estos nanoporos llamados nanoporos de estados sólido tienen el potencial de mejorar tanto los costos como la escala en comparación con los de origen biológico. Los nanoporos sintéticos se fabrican usualmente enfocando iones o rayos de electrones en una membrana sintética usualmente $SiNx$ o SiO_2 aunque se necesita un mayor desarrollo de estas tecnologías para obtener la precisión natural obtenida por las proteínas.

4. MinION

MinION es un dispositivo portátil de ONT diseñado para la secuenciación y el análisis de muestras de ADN. El dispositivo, que puede verse en la figura 2.2, cuenta con un peso menor a los 500g y una capacidad de secuenciación de hasta 30 gigabases (30.2^{30} bases) por uso, siendo una buena opción para la secuenciación de ADN en pequeñas escalas. MinION produce archivos en formato fast5, los cuales contienen los datos obtenidos a través de las lecturas eléctricas. El formato fast5 es una especificación del formato general de archivos para datos jerárquicos hdf5, definido en la sección 3.1. Luego de aplicarse un proceso de basecalling se asignan las bases nitrogenadas a las lecturas generando archivos de tipo fastq, el cual es el formato de facto para datos de secuenciación descrito en la sección 3.2.

Figura 2.2: Dispositivo MinION

Los dispositivos MinION operan con un software llamado MinKNOW, este lleva a cabo varias tareas incluida la adquisición de datos, análisis en tiempo real y basecalling local. MinKNOW produce archivos fast5 y archivos fastq, según la configuración que le sea provista.

5. Basecalling

Basecalling es el proceso por el cual los datos crudos, que contienen las mediciones eléctricas, se procesan para generar una secuencia de bases.
Se pueden encontrar en el mercado varios proveedores de algoritmos de basecalling, como por ejemplo MinKNOW, Albacore o Scrappie. En su mayoría usan redes neuronales profundas para transformar los datos crudos en archivos fastq, los cuales contienen entre otras cosas la secuencia de ADN.

Capítulo 3

Formatos de archivos

1. Hierarchical Data Format (HDF5)

Hierarchical Data Format (HDF) es un formato de archivos diseñado para el almacenamiento y la organización de grandes cantidades de datos desarrollado por el Nacional Center for Supercomputing Applications[5] (NCSA). El NCSA es un organismo de Estados Unidos vinculado con la investigación en informática y telecomunicaciones. Este formato es mantenido por el HDF Group, una organización cuya misión es continuar el desarrollo de tecnologías con base en HDF. Actualmente la ultima versión es hdf5[6].

Entre las principales ventajas de hdf5 se encuentran: el tamaño ilimitado de sus archivos, ser multi-plataforma con bibliotecas de acceso con API en varios lenguajes como Python, C, C++, Fortran y Java, el tener un modelo de datos robusto que permite almacenar tipos de datos heterogéneos y la flexibilidad en los métodos de almacenamiento. Entre sus críticas más grandes se encuentran la falta de soporte para el estándar UTF-8 y el hecho de que los datos almacenados son difíciles de modificar sin usar herramientas externas que realizan copias del archivo.

[5] http://www.ncsa.illinois.edu/

[6] https://portal.hdfgroup.org/display/HDF5/HDF5

1.1. Modelo de datos

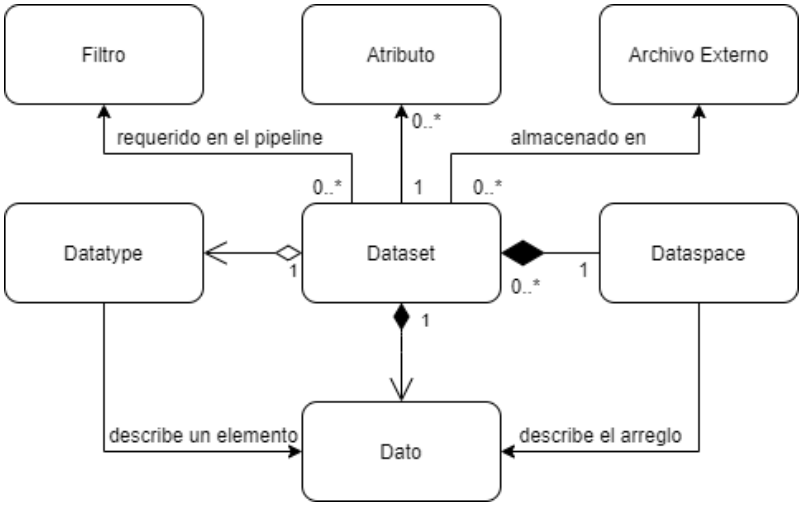
El modelo de datos abstracto (ADM, por sus siglas en inglés) que usa hdf5 se puede ver en la figura 3.1 y define los siguientes elementos:

- **File (archivo)**: es un archivo, que contiene una colección de objetos. Aunque no son los únicos, hay distintos tipos fundamentales de objetos: *groups* y *datasets* que se definen a continuación. Todos los archivos tienen al menos un objeto, el grupo raíz representado con el símbolo "\".

- **Group (grupo)**: es el análogo a una carpeta en un sistema de archivos computacional como por ejemplo el de Unix o Windows. La principal diferencia entre los sistemas de archivos convencionales y los grupos de hdf5, es que estos están conectados como un grafo dirigido, permitiendo referencias circulares. Un grupo puede o no contener objetos y cada objeto debe ser miembro de al menos un grupo (con la excepción del grupo raiz). La pertenencia a un grupo esta implementada con el uso de objetos *link* que vinculan tanto grupos como datasets con el grupo al que pertenecen.

- **Dataset (conjunto de datos)**: es un arreglo multidimensional de datos. Cada dataset tiene un *dataspace*, un *datatype* y una lista de *filters*. El dataspace define las características del arreglo como sus dimensiones, tamaño de cada dimensión, etc. Mientras que un datatype determina las características de los datos individuales. Los filters son transformaciones sobre los de datos aplicadas antes de almacenarlos. Los tres están explicados a continuación. Cabe destacar que todos los elementos del dataset tienen las mismas características en cuanto a su tipo y cantidad de memoria asignada, se puede ver al dataset como una especie de casilleros o lockers en los cuales se almacena información del mismo tipo.

Figura 3.1: Modelo de un *dataset*



- **Dataspace (espacio de datos)**: describe la distribución de los elementos en el arreglo multidimensional. Conceptualmente es un hiper-rectángulo con un máximo de 32 dimensiones, cada una con un tamaño actual y un tamaño máximo, que puede ser ilimitado si así se prefiere. Visto de forma simple, el dataspace es quien define cuál es el tamaño máximo del dataset y en cuantas dimensiones se va a dividir. Existen tres tipos de dataspaces, definidos a continuación:

  - Escalar: contienen un único elemento. Es visto también como un arreglo de cero dimensiones.
  - Simple: define un arreglo multidimensional de elementos. Este es el tipo de dataset más comúnmente usado.
  - Nulo: no contiene ningún tipo de elemento.

- **Datatype (tipo de datos)**: describe el tipo de información en los elementos de un dataset. Puede ser desde un tipo de datos atómico, como un entero de 32 bits, hasta tipos de datos compuestos, definidos por múltiples tipos atómicos. Un tipo de datos se puede almacenar dentro de un archivo hdf5 con un nombre. A estos tipos de datos almacenados con un nombre se les llama *named datatype* y se pueden utilizar como cualquier otro datatype predefinido.

- **Filter (filtro)**: permiten la aplicación de transformaciones de forma secuencial sobre los datos de un dataset y se almacenan en la *property list* del mismo. El usuario puede definir sus propios filters y se organizan de forma que la salida de uno sea la entrada del siguiente. Los filters más usados son los que se encargan de la compresión de datos, esto es particularmente cierto para los filters aportados por la comunidad.

- **Property List (lista de propiedades)**: es una colección de pares nombre-valor. Cada propiedad tiene un nombre, un datatype y un valor. Las listas de propiedades se adjuntan a los objetos hdf5, algunas son permanentes y otras modificables a través de la API de acceso estándar.

- **Attribute (atributo)**: Todos los objetos de hdf5 (groups, datasets y named datatypes) pueden tener atributos definidos por el usuario. Estos atributos se usan para documentar metadatos del objeto y se almacenan junto con él. Cada atributo tiene un nombre y el dato que almacena, definido, al igual que en un dataset, con un dataspace y un datatype. De hecho un atributo es similar a un dataset con las siguiente limitaciones:

  - No se puede compartir entre múltiples objetos.
  - Tiene un límite de mil bytes.
  - A diferencia de los de un dataset, la API estándar los lee y escribe en solo un acceso, y no se puede acceder en parte, sino como una unidad.
  - Los atributos no tienen otros atributos asociados.

1.2. Chunking

Hdf5 cuenta con una función de fragmentación de datos o *chunking*. Esta consiste en dividir el contenido de un dataset en pequeños fragmentos denominados *chunks*, que son comprimidos y almacenados individualmente. Una de las ventajas de realizar este procedimiento en lugar de comprimir el contenido completo del dataset, es que al querer acceder a algunos valores específicos de este, solo hace falta descomprimir el chunk correspondiente. Este método de chunking permite acelerar muchas funciones que se realizan sobre subsecciones de un dataset.

En la figura 3.2 se puede ver representado un dataset de dos dimensiones, visto como una matriz de 9x9, y en la figura 3.3 el mismo dataset dividido en 9 chunks de 3x3. Como cada uno de los chunks es comprimido individualmente, los accesos pueden ser realizados sin descomprimir el dataset completo.

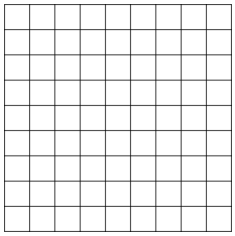


Figura 3.2: Dataset contiguo

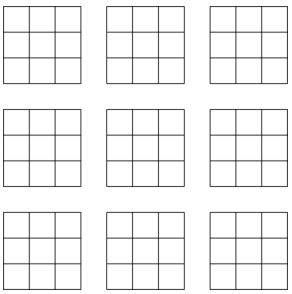


Figura 3.3: Dataset con chunks de 3x3

1.3. Herramientas de hdf5

HDF Group cuenta con una gran variedad de herramientas que facilitan el manejo de archivos hdf5. Todas ellas están incluidas en la biblioteca básica de hdf5. Entre las más utilizadas están:

- **h5repack:** realiza una copia de un archivo de entrada, le aplica filtros o configuraciones y genera un archivo de salida. Esta copia no se genera copiando espacios de memoria sino usando los métodos provistos por la biblioteca de hdf5, es decir, se construye el archivo parte por parte. Se puede elegir si se quiere copiar el archivo completo o solo secciones y al elegir el filtro a utilizar se puede seleccionar un filtro de compresión. Para poder utilizar un filtro implementado por terceros es necesario que se coloquen los binarios del filtro dentro de la estructura de archivos de la biblioteca hdf5.

- **h5ls:** imprime información sobre un archivo hdf5. Entre los datos impresos se encuentra la estructura jerárquica y los links correspondientes, las dimensiones y el tamaño de los datasets. En la figura 3.4 se puede observar un ejemplo de la salida de h5ls.

- **h5dump:** permite visualizar todos los datos de un archivo HDF5 en forma legible. Por la gran cantidad de contenido que se puede encontrar en los archivos, h5dump cuenta con varios filtros para facilitar la legibilidad.

- **h5debug:** muestra datos de un archivo hdf5 a bajo nivel, incluyendo filters, codificaciones y direcciones de memoria.
- **h5diff:** compara dos archivos hdf5 e imprime las diferencias encontradas entre estos. Es posible que dos archivos no sean exactamente iguales bit a bit pero los datos reales que se encuentran contenidos sí lo sean.

2. Fast5

El formato *fast5* de ONT es una especialización del formato hdf5. Para que un archivo hdf5 pueda considerarse un fast5 debe cumplir con una estructura particular; todo archivo fast5 es un archivo hdf5 pero no todo archivo hdf5 es un archivo fast5. Si bien no hay una especificación formal del formato de archivos fast5, en el proyecto validator[7] de ONT se puede encontrar la estructura que deben tener los archivos para considerarse válidos. El desarrollo de nuestro proyecto se basó en datos provistos por validator, más los conseguidos a través de distintas herramientas como h5dump y h5ls.

Figura 3.4: Visualización de un fragmento de la estructura de un archivo fast5 utilizando la herramienta h5ls

```
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44 Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Analyses Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Analyses/Basecall_1D_000 Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Analyses/Basecall_1D_000/BaseCalled_template Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Analyses/Basecall_1D_000/BaseCalled_template/Fastq Dataset {SCALAR}
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Analyses/Basecall_1D_000/Summary Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Analyses/Basecall_1D_000/Summary/basecall_1d_template Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Analyses/Segmentation_000 Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Analyses/Segmentation_000/Summary Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Analyses/Segmentation_000/Summary/segmentation Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Raw Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/Raw/Signal Dataset {5294/Inf}
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/channel_id Group
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/context_tags Group, same as /read_002ff12e-2b6a-4eb1-a7a3-63790c000171/context_tags
/read_f4ce9376-7c7b-4742-a29f-b5eb192ace44/tracking_id Group, same as /read_002ff12e-2b6a-4eb1-a7a3-63790c000171/tracking_id
```

Los datos principales contenidos en los archivos fast5 son las medidas de corriente, en pico-amperios, que se generan al secuenciar ADN con alguno de los dispositivos provistos por ONT. Además, opcionalmente se pueden encontrar datos *post-basecalling* como los eventos y el resultado del proceso de basecalling en formato fastq descrito en la sección 2.1.

[7] https://github.com/nanoporetech/ont_h5_validator

Tradicionalmente la estructura de un archivo fast5 estaba dividida en tres grupos:

- **Analyses:** este grupo contiene los datos correspondientes al basecalling, entre estos, se encuentra la secuenciación en formato fastq, datos estadísticos correspondientes a las lecturas y los eventos. Los eventos son una agregación de lecturas contiguas que idealmente corresponde cada uno a una base y cada base tiene su propio evento. Sin embargo, como el proceso puede generar ruido, puede darse que algunas bases no tengan eventos asociados o por el contrario tengan más de uno.

- **Raw:** en este grupo, dentro de un dataset llamado *Signal*, se guardan los datos crudos de las lecturas en pico-amperios. También se pueden encontrar datos relacionados a las lecturas tales como la duración de la lectura, la media de las mediciones y un número que la identifica, entre otros. En la figura 3.5 se observa un ejemplo generado con la herramienta h5dump. Las mediciones de corriente en pico-amperios son almacenados como enteros de 16bits.

- **UniqueGlobalKey:** almacena meta-datos de las lecturas, tales como la frecuencia de toma de lecturas del aparato y parámetros utilizados para la codificación de las lecturas.

Figura 3.5: Algunos atributos contenidos en el grupo Raw obtenidos utilizando la herramienta h5dump

```
GROUP "Raw" {
   ATTRIBUTE "duration" {
      DATATYPE  H5T_STD_U32LE
      DATASPACE  SCALAR
      DATA {
      (0): 11645
      }
   }
   ATTRIBUTE "median_before" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SCALAR
      DATA {
      (0): 221.711
      }
   }
   ATTRIBUTE "read_id" {
      DATATYPE  H5T_STRING {
         STRSIZE 37;
         STRPAD H5T_STR_NULLTERM;
         CSET H5T_CSET_ASCII;
         CTYPE H5T_C_S1;
      }
      DATASPACE  SCALAR
      DATA {
      (0): "034755c4-faa6-4adf-bf62-d9d9bebf1e03"
      }
   }
   ATTRIBUTE "read_number" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SCALAR
      DATA {
      (0): 94
      }
   }
```

2.1. Fastq

Un archivo fastq es un archivo de texto que contiene los datos de secuencia de ADN que se desprenden a partir de las lecturas y el algoritmo de basecalling.
En general un archivo fastq contiene muchos reads, cada uno de los cuales se representa con 4 líneas descritas a continuación y ejemplificadas en la figura 3.6.

- Un identificador de secuencia con información sobre la lectura y el aparato utilizado.
- Una secuencia de bases representadas como las letras: A, C, T y G. Ocasionalmente se utilizan otras letras para representar situaciones especiales, por ejemplo N para una base no identificada correctamente.
- Un separador, generalmente un símbolo de +, eventualmente seguido de una copia del identificador de secuencia.
- Una secuencia de calificaciones de calidad sobre las lecturas. Se usan caracteres ASCII para definir con cuánta seguridad se puede afirmar que la lectura corresponde a esa base. La secuencia de bases y la secuencia de calificaciones de calidad cuentan con el mismo largo ya que a la i-ésima base le corresponde la i-ésima calificación.

Figura 3.6: Ejemplo de un archivo fastq

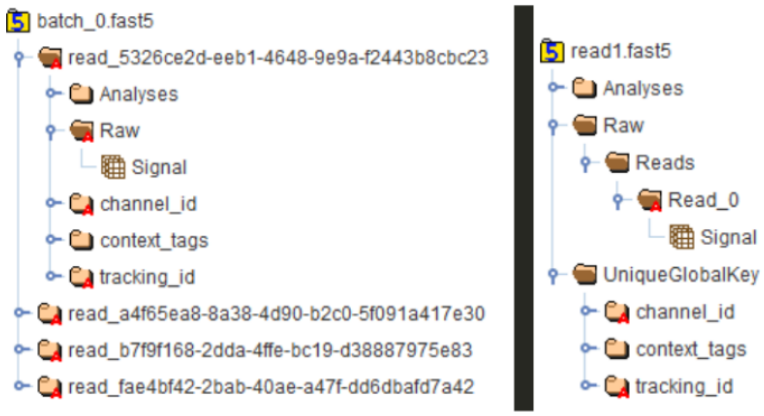
```
@ML-P2-14:9:000H003HG:1:11102:17290:1073 1:N:0:TCCTGAGC+GCGATCTA
TTTGGTAACAGCATGAATTATTCTAGCCACTAAAACTCTATGAACATCTTGTGAAGGTTTCAGATAGAGCCTGAAGTACACAGAGAACAATTCTTAAAAAA
+
AAAAAEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE<AEEEEEEE
```

En los archivos fast5 que incluyen información de basecalling, se puede encontrar el fastq correspondiente a cada lectura.

2.2. Multifast5

Con el avance de la tecnología, los dispositivos y la velocidad de procesamiento se han realizado mejoras sobre los datos guardados en los archivos fast5.
Entre los cambios realizados, desde diciembre del 2018 se pueden encontrar los archivos multilecturas o *multifast5*. Este nuevo formato posee más de una sesión de secuenciación y descarta los eventos intermedios, generando así estructuras más compactas. En este tipo de archivos se puede encontrar, en la raíz, un número de grupos correspondientes a las múltiples lecturas, y dentro de cada uno de estos grupos, se encuentra la estructura de un archivo fast5 tradicional. Es decir, un archivo multifast5 es una suerte de paquete de lecturas que incluye ciertas optimizaciones que permiten contener la misma información que múltiples archivos fast5 en uno solo, ocupando menos espacio de almacenamiento.

Figura 3.7: Comparación de la estructura de multifast5 (izquierda) y fast5 (derecha)



Los cambios en la estructura de los archivos (ejemplificada en la figura 3.7) provoca que los compresores especializados en fast5, a menos que sean mantenidos activamente, se vuelvan obsoletos. Ante esto, la mejor defensa es la continua actualización y el desarrollo de un código mantenible.

Capítulo 4

Antecedentes

En este capitulo se describen tres distintos tipos de compresión que son utilizadas como base para F5Comp, así como también distintos sistemas que trabajan con archivos de tipo fast5. Finalmente se presenta una investigación de la universidad de Stanford que plantea una comparación entre compresores de este tipo de archivo.

1. Compresión

1.1. Huffman

La codificación de Huffman [6] es un algoritmo usado para compresión de datos que utiliza un código de longitud variable para codificar cada uno de los símbolos de una determinada cadena. El código de Huffman se define en función de una distribución de probabilidad sobre el alfabeto de símbolos que componen la cadena. Los símbolos más probables se codifican con palabras de código más cortas y viceversa, de modo que la esperanza del largo de código es la menor posible para dicha distribución.

1.2. GZIP

Gzip [7] es una abreviatura de GNU ZIP, un software libre utilizado para compresión en UNIX. Gzip se basa en el algoritmo *Deflate* [8] que es una combinación del *LZ77* [9] y la codificación Huffman.

LZ77 tiene un funcionamiento muy intuitivo; básicamente recorre la cadena a comprimir hasta encontrar una secuencia que se repite (de al menos un cierto largo mínimo) y una vez encontrada se codifican tres datos: la distancia entre la aparición de la secuencia y su repetición, el largo de la secuencia repetida y el símbolo siguiente a la secuencia. Este proceso se repite secuencialmente hasta el final de la cadena a comprimir.

Para hacer más fácil el desarrollo del software que usa este método de compresión, se creó la biblioteca zlib [10], la cual soporta el formato de ficheros gzip y la compresión deflate. Esta biblioteca es de uso común debido a que es pequeña, eficiente y muy versátil. Debido a su popularidad, se la integró como filtro por defecto en hdf5.

Gzip ofrece un rango de nueve niveles de compresión donde, a medida que aumentamos el nivel de compresión, aumenta el tiempo de compresión a la par que mejora el nivel de compresión.

1.3. SZIP

SZIP [11] es un método de compresión de datos sin pérdida incluido entre los filtros de compresión de los productos de HDF desde la versión 4. Es una implementación del algoritmo extended-Rice [12], que utiliza un subconjunto de la familia de codificaciones del método de Golomb [13], y fue desarrollado en la Universidad de Nuevo México e integrado con hdf4 por investigadores y desarrolladores de dicha universidad. SZIP puede usarse como un filtro opcional para los archivos hdf5.

2. Productos

2.1. Picopore

Picopore es un software desarrollado en Python por Scott Gigante [8]. Su propósito es reducir el espacio de almacenamiento de los archivos fast5 generados por las lecturas del dispositivo MinION. Picopore utiliza varias técnicas de reducción de espacio, entre ellas, las provistas por el formato de archivo hdf5, eliminación de

[8] https://github.com/scottgigante/picopore

datos duplicados y asignación de memoria más eficiente dentro del archivo, entre otras.

Técnicas de compresión

- **Compresión gzip:** Se aplica una compresión gzip sobre los datos almacenados en el archivo hdf5 con opciones de nivel de compresión entre 1 y 9. La compresión predeterminada que usa ONT es gzip en el nivel 3; Picopore aumenta esto al nivel 9.

- **Asignación dinámica de memoria:** Los datos almacenados en los datasets de hdf5 son generalmente números enteros representados con 16, 32 o 64 bits, o números reales representados como punto flotante de precisión simple o doble. Picopore reduce espacio de almacenamiento al analizar el rango de valores en cada conjunto de datos, para determinar el número mínimo de bytes requeridos para su representación, cambiando el tipo de dato al más ajustado posible.

- **Colapso de estructura de archivo:** Picopore reduce el espacio en disco utilizado por los metadatos de un archivo al colapsar la estructura del directorio. Es decir, cada atributo, dataset y demás tipos de objetos son trasladados al grupo raíz, cambiando su nombre por la ruta que tenía previamente. Esto disminuye el espacio de almacenamiento utilizado ya que se libera la memoria perteneciente a los grupos.

- **Indexado de datos duplicados:** Picopore reduce el uso de espacio en disco al eliminar datos de detección de eventos duplicados generados en el basecalling y reemplazándolos por referencias a sus copias.

- **Descarte de datos intermedios:** Picopore permite a los usuarios eliminar los datos intermedios generados durante el proceso de basecalling al generar el fastq a partir de los datos crudos. Estos datos no pueden ser recuperados, salvo que se repita el proceso de basecalling.

Configuraciones

Picopore cuenta con tres tipos de configuraciones posibles para comprimir un archivo fast5:

- **Lossless:** consiste en una compresión gzip nativa del formato hdf5, la cual es aplicada a todos los datasets. Además, asigna dinámicamente el tipo de datos a cada dataset dependiendo de los valores que estos contengan. Es decir que, si los datos están representados como enteros sin signo de 16 bits, pero todos son menores que 2^8, se les cambia el tipo de dato a entero sin signo de 8 bits. Este modo es rápido y permite el análisis de datos por cualquier software existente ya que no cambia la estructura del archivo.

- **Deep-Lossless:** realiza la compresión lossless descrita en el punto anterior, luego aplica un colapso de la estructura de archivos y finalmente un indexado de datos duplicados. Este modo obtiene mejores resultados de compresión sin eliminar ningún dato, pero tiene la desventaja de que requiere descomprimir antes de que se pueda analizar los datos con otro software.

- **Raw:** realiza la compresión lossless, así como la eliminación de datos intermedios, eventos, atributos y logs, dejando únicamente los datos correspondientes a las lecturas crudas. Este modo es rápido y obtiene la mayor reducción de tamaño de archivo. Sin embargo, el archivo comprimido solo puede ser analizado por herramientas que puedan trabajar solo con lecturas crudas, como por ejemplo Nanopolish[9]. Los datos perdidos en este tipo de compresión no pueden ser recuperados, salvo que se repita el proceso de basecalling.

2.2. Poretools

Poretools es una herramienta desarrollada en la universidad de Birmingham [14], cuyo principal objetivo es la conversión de archivos fast5 a otros formatos útiles como fasta y fastq. Poretools cuenta también con una funcionalidad de visualización de datos. Esta herramienta es un software libre desarrollado en el lenguaje Python, y su código fuente puede ser encontrado en su repositorio público[10].

[9] https://github.com/jts/nanopolish

[10] https://github.com/arq5x/poretools

Conversión de archivos fast5

La funcionalidad fundamental proporcionada por Poretools es la capacidad de convertir los datos de salida de una lectura de MinION en formato fast5 a fasta o fastq para facilitar su análisis. Esto se logra con los comandos fasta y fastq de la herramienta, los cuales, para poder ser utilizados necesitan que los archivos cuenten con datos de basecalling.

$$poretools \quad fasta \quad /path/to/fast5/example.fast5$$

$$poretools \quad fastq \quad /path/to/fast5/example.fast5$$

Poretools también provee una funcionalidad para combinar múltiples archivos fast5 en un único archivo tar para facilitar el procesamiento y transporte de los datos.

$$poretools \quad combine \quad -o \quad run.tar \quad /path/to/fast5/directory/fastq$$

Obtención de datos estadísticos

Entre las funcionalidades estadísticas más usadas de Poretools se encuentran *stats* y *events*. La primera imprime datos estadísticos sobre un *dataset* de crudos particular, tales como el total de lecturas, el total de pares de bases, mínimo, máximo y media. La funcionalidad de events imprime los datos de todos los eventos de una lectura particular. Estas funcionalidades son ejecutadas de la siguiente manera:

$$poretools \quad stats \quad /path/to/fast5/example.fast5$$

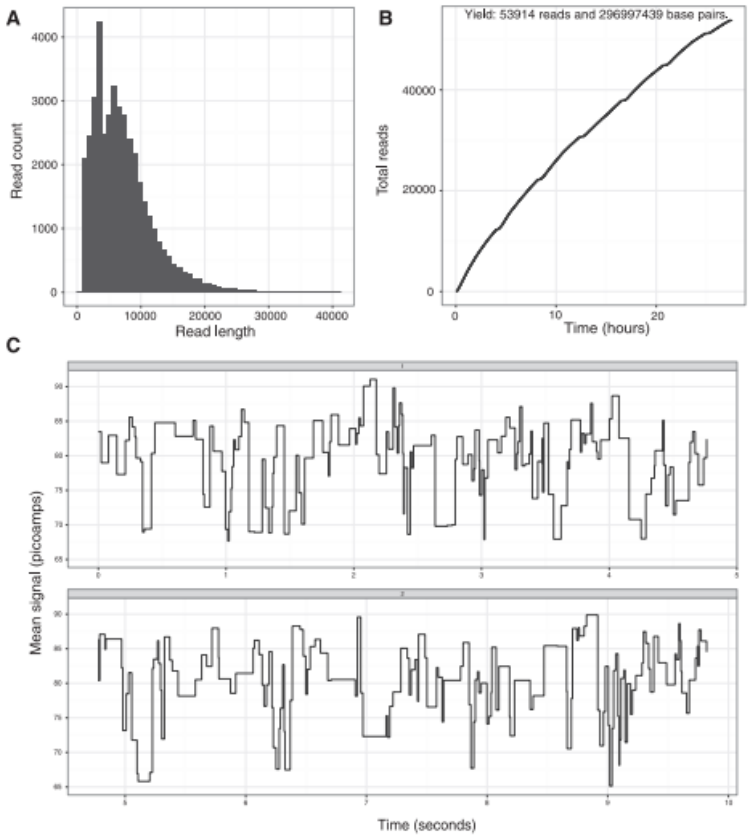
$$poretools \quad events \quad /path/to/fast5/example.fast5$$

Representación gráfica de datos

Poretools cuenta con una funcionalidad de visualización de datos tanto crudos como de basecalling. También puede generar reportes sobre la cantidad de lecturas por unidad de tiempo y el largo de estas, como se puede ver en la figura 4.1, tomada directamente de su manual[11].

[11] https://poretools.readthedocs.io/_/downloads/en/latest/pdf/

Figura 4.1: Gráficas generadas por poretools



2.3. F5Pack

F5Pack[12] es una herramienta hecha para empaquetar archivos fast5. Esta herramienta está incluida en la biblioteca de fast5, más concretamente en el wrapper Python de esta biblioteca.

F5Pack considera que hay tres tipos de contenidos en un archivo fast5: datos, eventos y configuraciones correspondientes a los datos en el dataset *Raw*, los eventos en el grupo de *Analyses* y a todo el resto del archivo, respectivamente. Para cada uno de ellos aplica las siguientes compresiones:

- **Datasets**: utilizando la observación de que las muestras varían lentamente durante una lectura, se codifican las diferencias entre estas con un código Huffman.

- **Eventos**: cada evento, el cual es una agregación de lecturas contiguas correspondiente a una base, contiene cuatro datos, a los cuales se les aplica una

[12]https://github.com/mateidavid/fast5/blob/master/python/bin/f5pack

determinada compresión:

- *start*: es el identificador de la primera muestra contenida en el evento. Este dato se guarda como *skip*, es decir la diferencia entre el identificador de la última muestra de un evento y la primera muestra del evento siguiente menos uno. Ya que en la mayoría de los casos el final de un evento coincide con el comienzo del siguiente, este valor es normalmente cero.
- *length*: es la cantidad de muestras que tiene el evento. En este caso para comprimirlas se utiliza al igual que en los datos crudos una codificación Huffman
- *mean y stdv*: es la media y la desviación estándar de las muestras del evento. Estas no se almacenan, se re-calculan a partir de los datos crudos al momento de descomprimir.

- **Configuración**: actualmente F5Pack descarta todos los datos de configuración, es decir, se realiza una compresión de datos con pérdida

Limitaciones

Existen varios basecallers diferentes para los datos de ONT: Metrichor, MinKNOW, Albacore y Nanonet, entre otros. A pesar de trabajar con el mismo tipo de datos, la estructura de los eventos generados post-basecalling no está estandarizada y se pueden encontrar pequeñas diferencias entre ellos. En principio, f5pack está diseñado para funcionar con datos *pre-basecalling* y *post-basecalling*, sin embargo hay una lista de errores conocidos referentes a discrepancias de formato de los eventos entre las herramientas disponibles en el mercado actual.

3. Publicaciones

3.1. Compresión con pérdida de lecturas crudas

En una reciente investigación desarrollada en la universidad de Standford [15] se plantea una comparación entre compresores sin pérdida diseñados para datos crudos de nanoporos, como por ejemplo Picopore y el filtro desarrollado por terceros VBZ[13], y compresores con pérdida como SZ[14] y LFZip[15]. En este contexto, la investigación plantea que se puede tener una mejora importante en la compresión de un archivo fast5 al costo de una cierta distorsión en los datos crudos. Cada uno de los compresores con pérdida tiene un parámetro que indica el error máximo tolerable en la reconstrucción del archivo comprimido. Este parámetro está entre 1 y 10, siendo 1 el más parecido al archivo original y 10 el más alejado. Para poner los rangos de error en contexto, hay que tener en cuenta que una lectura típica es de 60 pA y el valor típico de ruido en la lectura es de 1-2 pA. Los ajustes de error máximo de 1 y 10 corresponden a valores de corriente de 0,17 pA y 1,7 pA, respectivamente. Las pruebas realizadas revelan que mediante el uso de compresores de datos con pérdida, y al costo de tener una exactitud de un 95 %-97 % con respecto a los archivos originales, se pueden obtener reducciones en algunos casos de hasta un 65 % del espacio ocupado por las lecturas crudas. Como conclusión, en esta investigación, se afirma que en la mayoría de los archivos se puede lograr una compresión de 40-50 % con respecto a los métodos de compresión sin perdida con un impacto despreciable en la precisión. Más aún, si estas lecturas crudas se utilizaran para realizar un basecall, la coincidencia entre el fastq generado y el original sería igual o superior al 97 %.

[13] https://github.com/nanoporetech/vbz_compression
[14] https://github.com/szcompressor/SZ
[15] https://github.com/shubhamchandak94/LFZip

Capítulo 5

Diseño e Implementación

En este capítulo describimos tanto el diseño como la implementación de nuestro compresor. En la sección 1 se definen sus requerimientos y en la sección 2 se los analiza de forma tal que el diseño sea más fácil de comprender. En las secciones 3 y 4 se da una explicación a grandes rasgos de las herramientas utilizadas para el desarrollo del compresor, así como también los componentes en que se lo divide.

1. Requerimientos

El objetivo de este proyecto tal como se planteó es desarrollar un utilitario similar a Picopore pero con algunas característica diferentes que son de interés para la investigación que se desarrolla en este momento en el Núcleo de Teoría de la Información. Específicamente, estas características son:

- Desarrollo en lenguaje C++.
- Facilidad para extraer información estadística de los datos almacenados en los archivos en formato fast5.
- Sustitución de gzip por otro tipo de compresores y evaluación de su desempeño.

Durante el desarrollo del proyecto aunque el objetivo se mantuvo bastante estable las técnicas de compresión aplicadas por Picopore fueron en su mayoría abandonadas y reemplazadas por técnicas más avanzadas. Nos enfocamos en la variante multi-fast5 de los archivos fast5, ya que es la variante utilizada actualmente. De todos

modos no solo se mantiene la compatibilidad con los archivos fast5 tradicionales sino que se mantienen algunas herramientas de compresión especificas para estos.

2. Análisis

2.1. Compresión

Para analizar las técnicas que se aplicaron para comprimir los datos, es una buena idea recordar cuáles usan las herramientas ya existentes como Picopore y F5Pack.

- **Picopore:**

    1. Aumentar el parámetro de nivel de compresión de *gzip* dentro de la configuración de hdf5.
    2. Reducir el tamaño de representación de tipos de datos al mínimo.
    3. Colapsar la estructura de archivos.
    4. Indexar datos duplicados de los eventos pre y post *basecalling*.
    5. Eliminar datos intermedios, eventos, atributos y logs.

- **F5Pack:**

    6. Usar codificación Huffman para los datos crudos.
    7. Reducir la estructura de eventos a una más compacta que permita recalcular datos eliminados, como por ejemplo media y desviación estándar.
    8. Eliminar datos correspondientes a configuración.

Dentro de estas ocho técnicas de compresión, hay algunas que no aplican al formato de archivos en el cual nos enfocamos, debido a que tanto Picopore como F5Pack son herramientas creadas para la compresión de archivos con el formato original de fast5 y no han sido adaptadas al nuevo formato multi-lectura. Estos cambios al nuevo formato comenzaron a finales de septiembre del 2018 [16], varios meses después de la ultima actualización de F5pack y años después de la última actualización de Picopore. De las técnicas que no aplican al nuevo formato, descartamos las técnicas 3 y 4 por su complejidad de implementación.

[16] https://community.nanoporetech.com/posts/fast5-file-format-change

2.2. Extracción de datos

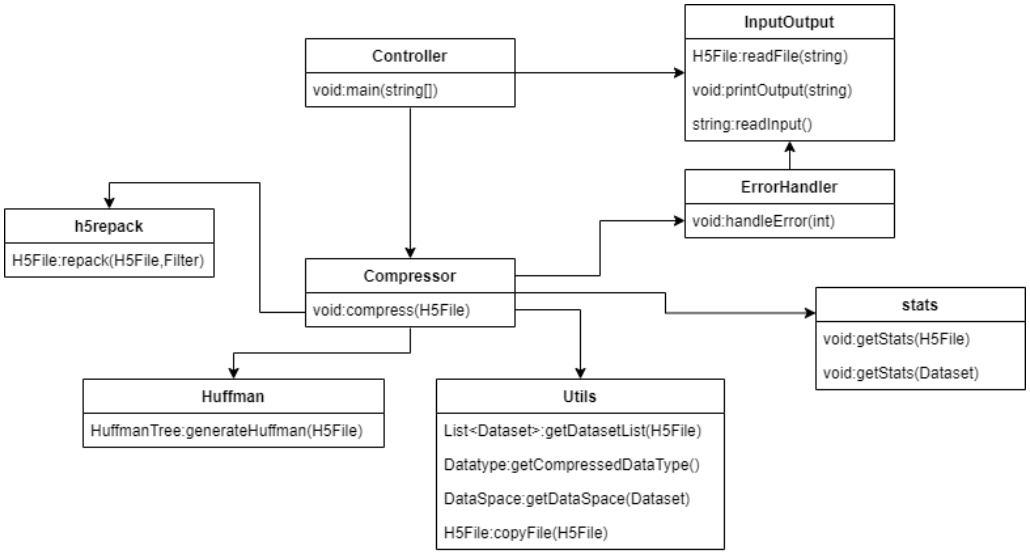
La extracción de datos consiste en la lectura de un archivo de entrada y a continuación la impresión de datos útiles para el usuario. Estos datos pertenecen a ciertos datasets que el usuario selecciona. Se desarrollan dos tipos de extracciones de datos. El primero consiste en la extracción de todos los datasets correspondientes a las lecturas del archivo fast5, junto los metadatos incluidos en estos (atributos). El segundo tipo consiste en la extracción de datos y metadatos de un único dataset de entre los contenidos en el archivo. Los datos de los datasets son impresos en un archivo .csv que puede ser leído por cualquier software que trabaje con este formato. También se implementa un utilitario que muestre los datos en forma gráfica utilizando Octave.

3. Diseño

Los sistemas de compresión funcionan básicamente aplicando transformaciones de forma secuencial en un conjunto de datos. Solo por el hecho de ser un sistema de compresión sabemos que es necesaria una etapa de lectura de información, una etapa de compresión y una etapa de escritura. Se tiene que tener en cuenta también que, según el manual de usuario de hdf5[17], hdf5 no posee en este momento ningún mecanismo para eliminar un dataset de un archivo y reclamar el espacio de almacenamiento ocupado. Solamente cuenta con una función para quitar las referencias a los espacios de memoria ocupados por este pero sin liberarlos realmente.
El diseño utilizado responde a estas necesidades utilizando la estructura mostrada en la figura 5.1 y la función de sus componentes se puede ver a continuación.

[17] https://support.hdfgroup.org/HDF5/doc/UG/HDF5_Users_GuideResponsive%20HTML5/

Figura 5.1: Arquitectura de la solución



- InputOutput: componente encargado de la entrada y salida tanto de los archivos como de mensajes en consola.
- Controller: este componente fue implementado con la idea de manejar la paralelización del componente compresser.
- Compressor: componente encargado de los procesos de compresión y descompresion.
- H5repack: lidia con los problemas de hdf5 para la modificación de datasets.
- Huffman: componente encargado de la generación de los arboles de Huffman.
- Utils: provee las funciones auxiliares para el resto de los componentes.
- Stats: se utiliza para la generación de datos estadísticos.
- ErrorHandler: maneja las excepciones dentro del sistema.

El flujo de datos dentro del sistema en un proceso de compresión es

$$InputOutput \rightarrow Controller \rightarrow Compressor \rightarrow Repack \rightarrow InputOutput,$$

mientras que para la generación de datos estadísticos el flujo es

$$InputOutput \rightarrow Controller \rightarrow Stats \rightarrow InputOutput.$$

3.1. Tipos de compresión

Gzip

La compresión gzip consiste en generar una copia del archivo original en el cual el nivel de compresión gzip nativo del formato fast5 es aumentado de 3 a 9 (máximo).

Huffman

La compresión Huffman consiste en utilizar un árbol de Huffman para codificar los datasets correspondientes a lecturas encontrados en el archivo. Para esto no codificamos las lecturas crudas directamente, sino las diferencias entre lecturas consecutivas. Esta decisión está fundamentada en la observación empírica de que la mayoría de las lecturas tienen valores cercanos a la lectura previa, por lo que las diferencias tienden a ser pequeñas. El tamaño del árbol de Huffman está limitado, por lo que las diferencias menos ocurrentes puede que no tengan una codificación asociada. Para estos casos se utilizó una codificación particular que llamaremos *Token*, la cual antecede a una diferencia que no tiene codificación y simplemente se guarda como entero de 16 bits. El uso de diferencias y la acotación del árbol de Huffman (junto con la utilización del *token*) serán justificadas en la sección de decisiones de diseño.

Eliminación de logs

La eliminación de logs consiste en generar un nuevo archivo removiendo cualquier información relacionada con logs y metadatos emitidos por el aparato de lectura. Esto deja únicamente datos y metadatos relacionados con las lecturas y los basecalling asociados.

Extracción de lecturas

La extracción de lecturas consiste en crear un nuevo archivo fast5 que contenga únicamente los datasets correspondientes a las lecturas. En esta funcionalidad se utiliza también una compresión gzip de nivel 9. Esta compresión no es reversible, por lo que si se quisieran acceder a datos distintos a las lecturas, se tendrá que ir directo al archivo original.

3.2. Decisiones de diseño

Integración de utilitario H5repack

Como se menciona en secciones anteriores, el utilitario h5repack se utiliza para solucionar el problema de la modificación de datasets. Se toma la decisión de incluir su código fuente al compresor para que pueda ser utilizado sin necesidad de instalar software adicional.

Uso de chunking

El chunking es una función de fragmentación de datos que usa hdf5 para mejorar la eficiencia de los accesos, lo cual permite descomprimir solo los fragmentos que se quieren leer y no el dataset completo. Esta funcionalidad si bien mejora la eficiencia en la lectura, tiene una penalización en cuanto a espacio en disco, lo cual se decidió priorizar. Como medida para minimizar el espacio en disco, se asignó a cada dataset un único chunk del tamaño total del dataset.

Codificación de las diferencias entre lecturas

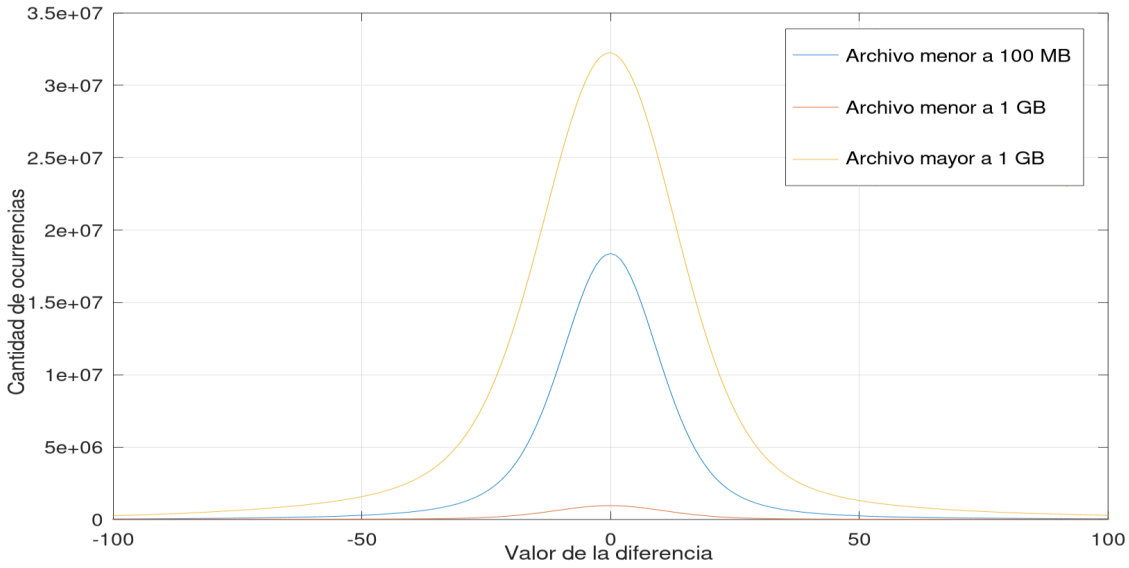
Para mejorar la performance de Huffman es importante que el conjunto a codificar tenga la mayor cantidad de valores repetidos posible. La primer idea fue mapear el universo completo de valores posibles. Para esto se tomaron como conjunto de muestra 10 archivos de tamaños representativos (entre 100 MB y 1.5 GB) y se graficaron todas las lecturas. En este proceso se vio que las lecturas se encuentran distribuidas en un rango de valores amplio, pero que la diferencia entre lecturas consecutivas es pequeña en la mayoría de los casos. Por lo tanto se decidió codificar la diferencia de las lecturas ya que hay una mayor cantidad de repeticiones, además de encontrarse en un conjunto menor a las lecturas originales.

Tamaño del árbol Huffman

El tamaño del árbol de Huffman fue uno de los primeros problemas que se presentó al implementar esta codificación. En los archivos de prueba se dan casos de lecturas muy por encima del promedio, que generan diferencias altas. Se cree que estas anomalías se deben a errores de medición. La ocurrencia de anomalías obligaría

a mapear el conjunto de enteros de 32 bits en un código de Huffman por lo que se optó por una opción más optimista. Sólo se mapean los valores de diferencias más frecuentes, al resto de los valores se los codifica como enteros de 32 bits precedidos por un código de Huffman especial, llamado Token, que funciona como un código de escape. Este Token es el mismo para todos los elementos fuera del rango codificado con Huffman, y servirá para que a la hora de decodificar se sepa cuándo se trata de un valor fuera de rango codificado como entero de 32 bits. La siguiente decisión tomada fue evaluar qué umbral usar para determinar si un valor es codificado con Huffman o con 32 bits y Token. En la figura 5.2 se pueden ver tres casos de ejemplo tomados de los archivos de muestra, en los cuales se puede apreciar que la diferencia entre los valores de muestras contiguas es menor a 100 unidades en la mayoría de las lecturas (un número menor al 1 % son las lecturas que se encuentran a mayor distancia de sus lecturas contiguas).

Figura 5.2: Cantidad de ocurrencias de diferencias entre lecturas consecutivas



Como consecuencia de estas observaciones, se evaluó que los valores que se encuentran entre -200 y 200 serán codificados con Huffman, mientras que los que están por fuera se codificarán como un Token más un entero de 32 bits.

4. Implementación

Herramientas utilizadas

Clion

CLion[18] es un entorno de desarrollo de C++ que fue seleccionado como herramienta para la implementación debido a su buena integración con CMake. CMake es una herramienta de código abierto, multi plataforma, diseñada para la automatización de la compilación, integración y testeo de software para un ambiente específico (en este caso Unix). CMake ya viene integrado con CLion, de forma que al crear el proyecto se generan los archivos de configuración necesarios automáticamente. CMake es sugerido por HDF Group para la generación de herramientas que utilicen las bibliotecas de hdf5 de forma directa en código. Aunque podría haberse utilizado cualquier otro entorno de desarrollo, es recomendable mantener el vínculo con CMake como herramienta de compilación e integración.

Hdf5

Las bibliotecas de hdf5 utilizadas son las del release 1.10.5 de febrero de 2019[19]. La versión más reciente es la 1.12.0, que fue liberada en marzo del 2020. Esta versión ofrece varias herramientas nuevas, pero lamentablemente fue liberada varios meses luego de comenzado el desarrollo de este proyecto, por lo que preferimos mantener la versión en funcionamiento. Por otro lado la versión 1.10.6, de diciembre de 2019, no ofrece ventajas significativas con respecto a la 1.10.5, por lo que no consideraramos conveniente un posible cambio de versión. Estas bibliotecas proveen las funcionalidades mínimas necesarias para el manejo de archivos hdf5 y, por extensión, de fast5.

Boost

Boost[20] es una serie de bibliotecas para el lenguaje C++ desarrolladas inicialmente en 1999 por Beman Dawes y David Abrahams, y mantenidas y actualizadas

[18] https://www.jetbrains.com/es-es/clion/
[19] https://www.hdfgroup.org/packages/hdf5-1105-source/
[20] https://www.boost.org/

hasta el día de hoy por su comunidad. Boost provee soporte para tareas y estructuras como álgebra lineal, generación de números pseudoaleatorios, *multi-threading*, procesamiento de imágenes, expresiones regulares y tests unitarios. Particularmente se utilizó para manejo de archivos y de cadena de caracteres. Boost cuenta con una licencia gratuita de código abierto, tanto para uso comercial como no comercial.

Huffman

Huffman es un utilitario[21] que se encarga de la creación de un árbol de Huffman a partir de una serie de valores junto con su cantidad de ocurrencias. Internamente es una clase que va asignando códigos de longitud variable a cada uno de los valores ingresados (hojas), los cuales tendrán menor longitud cuánto mayor sea su cantidad de ocurrencias. Para definir el código a asignar se utiliza el algoritmo de Huffman[6]. Una vez resueltos los códigos de Huffman para cada valor ingresado, esos se devuelven en formato de diccionario para ser utilizado posteriormente. Esta clase es utilizada únicamente para el proceso de generación de árbol Huffman a partir de archivos de entrenamiento, es decir que no interviene en la compresión.

Componentes

Para una mejor organización y entendimiento de F5Comp, se decide dividirlo en seis componentes o clases que se definen a continuación. Para más detalles o mayor información sobre las funcionalidades disponibles y cómo usarlas, dirigirse al manual del desarrollador adjunto a este informe.

InputOutput

Este componente es el encargado de manejar la comunicación entre el compresor y la consola, así como también de la creación de los archivos. En un principio parsea la entrada del usuario distinguiendo así el nivel de compresión que se desea realizar y examina el archivo indicado verificando si se trata de una compresión o descompresión. Finalmente cuando termina la compresión/descompresión imprime en consola el tiempo transcurrido en el proceso así como también el porcentaje de compresión, si corresponde.

[21] https://github.com/JellyBella/Static-Huffman-Encoding-Decoding-and-Search

h5repack

Este componente es una modificación de la herramienta con el mismo nombre provista por hdf5. Lamentablemente hdf5 no provee ninguna funcionalidad en su biblioteca para eliminar elementos desvinculados de sus archivos y reclamar ese espacio de almacenamiento, por lo que se debe recurrir a la herramienta externa h5repack. Sin embargo, esto implicaría que el compresor no fuera auto contenido, por lo tanto se adaptó el código fuente, incorporando solamente la funcionalidad de copia de elementos vinculados al código de F5Comp.

Compressor

Compressor es el módulo más importante. A partir de la entrada del usuario, decide cómo manejar los datos de los archivos para comprimirlos o descomprimirlos. Se comunica con el resto de los componentes utilizando las funciones que estos proveen. Sobre este componente se encuentra la responsabilidad de orquestar todo el proceso de compresión y descompresión. Se implementan 5 compresiones distintas: gzip, Huffman, eliminación de logs, extracción de lecturas y szip. Las compresiones gzip y szip funcionan de forma análoga, ambas extraen cada uno de los datasets, se copia su contenido en un buffer, y luego se lo desvincula del archivo. A continuación se le aplica h5repack generando un archivo limpio al que se le insertan los datasets previamente desvinculados, solo que esta vez se editan sus dataspaces para indicar qué filtro de compresión usar. La compresión Huffman en cambio, luego de copiar los datasets a un buffer, estos son procesados y a cada lectura se la reemplaza por su codificación Huffman. Luego, al igual que las compresiones gzip y szip, se aplica h5repack y se insertan los datasets codificados. La eliminación de logs recorre recursivamente la estructura de directorio del archivo fast5 y desvincula cualquier objeto relacionado con los logs. Finalmente se cambia el nivel de compresión gzip a 9 y utiliza la herramienta h5repack para crear un nuevo archivo. Finalmente la extracción de lecturas simplemente copia todos los datasets correspondientes a lecturas a un buffer y lo utiliza para crear un nuevo archivo.

Stats

Este componente es utilizado para imprimir en consola o en archivos (mediante el componente InputOutput) datos extraídos de los datasets que el usuario desee. Se encarga de dar un formato .csv a los datos que lo necesiten y hacer conversiones entre distintas unidades.

Utils

Este componente transversal posee todo tipo de funcionalidades útiles y reutilizables para el resto de los componentes. Entre ellas se encuentran búsquedas, creación de configuraciones, conversiones de tipos de datos y operaciones sobre cadenas de caracteres.

ErrorHandler

Este componente es el encargado de procesar los errores. En caso de ocurrir, borra archivos temporales o creados de forma incompleta. También se comunica con el componente InputOutput para notificar al usuario por consola información detallada del error o algún código de error que explique la razón de este y cómo solucionarlo.

Capítulo 6

Resultados experimentales

Para evaluar el desempeño del compresor se realizan pruebas comparativas de velocidad y porcentaje de compresión. Se compraran los resultados con un compresor específico del formato fast5, Picopore, y dos compresores generales como lo son rar y zip.

1. Ambiente de pruebas

Las pruebas fueron realizadas en una máquina virtual Ubuntu, usando un procesador Ryzen 5 3600 y 8 GB de RAM (de los 16 pertenecientes a la maquina host) con una frecuencia de 3200 MHz. La versión de Ubuntu utilizada fue la 19.10 sobre un kernel Linux de versión 5.3.0-64.

2. Casos de prueba

Para la ejecución de las pruebas se tomaron 15 archivos representativos del repositorio de github de *Oxford Nanopore Human Reference Datasets*[22] . Se eligió este repositorio debido a su publicación de datos reales sobre secuenciación del genoma humano, así como el uso de software y hardware actualizado. Además tiene una gran variedad de archivos fast5 de distintos tamaños, lo que permite evaluar el compresor en varios escenarios. Fue también de especial importancia que los datos recopilados en este repositorio hayan sido proporcionados por distintos centros, ya que esto ase-

[22]https://github.com/nanopore-wgs-consortium/NA12878

gura variedad de condiciones de generación de archivos.

Para las pruebas de compresión se tomaron 5 archivos de tamaño inferior a los 250 MB, 5 archivos con tamaño superior a los 250 MB e inferior a 1 GB, y 5 archivos con un tamaño superior a 1.5 GB, como se puede ver en el cuadro 6.1. Se considera que en estas tres franjas se encuentran los archivos de pequeño, mediano y gran porte. Para la comparación de resultados, se tomaron 5 compresiones distintas: F5Comp-gzip, F5Comp-Huffman, rar, zip y Picopore. Siendo F5Comp-gzip y F5Comp-Huffman dos modos de compresión correspondientes al compresor F5Comp. En cuanto a Picopore, existen dos tipos de compresión disponibles, lossless y deep-lossless. Para estas pruebas no se ha podido utilizar deep-lossless ya que no es compatible con el formato de archivo multifast5.

Cuadro 6.1: Tamaño de archivos utilizados

Nombre	Tamaño (MB)
Test1	36.5
Test2	98.2
Test3	98.2
Test4	152.8
Test5	224.0

Nombre	Tamaño (MB)
Test6	555.6
Test7	566.7
Test8	589.1
Test9	591.5
Test10	605.4

Nombre	Tamaño (MB)
Test11	1591.2
Test12	1602.6
Test13	1625.0
Test14	1640.6
Test15	1653.5

3. Composición de archivos de prueba

Los archivos multifast5 se pueden agrupar en dos tipos: los que contienen datos correspondientes al basecalling (fastq) y los que solo contienen lecturas. El compresor desarrollado se enfoca en las lecturas crudas, por lo que los archivos usados son del segundo tipo. Como ejemplo se muestran a continuación las composiciones de dos archivos de tamaño similar, el primero perteneciente a los datos de prueba utilizados y el segundo a los ejemplos de Picopore[23]. La proporción de espacio ocupado por los distintos componentes de archivos que solo contienen lecturas crudas se muestra en la figura 6.1. En la figura 6.2 se muestra esta proporción para archivos que incluyen datos de basecalling.

[23]https://github.com/scottgigante/picopore/tree/master/sample_data

Figura 6.1: Proporción de espacio ocupado sin basecalling

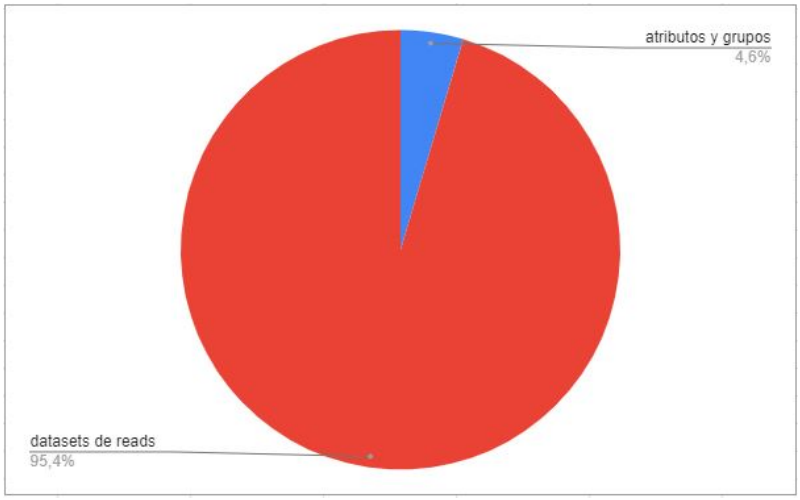
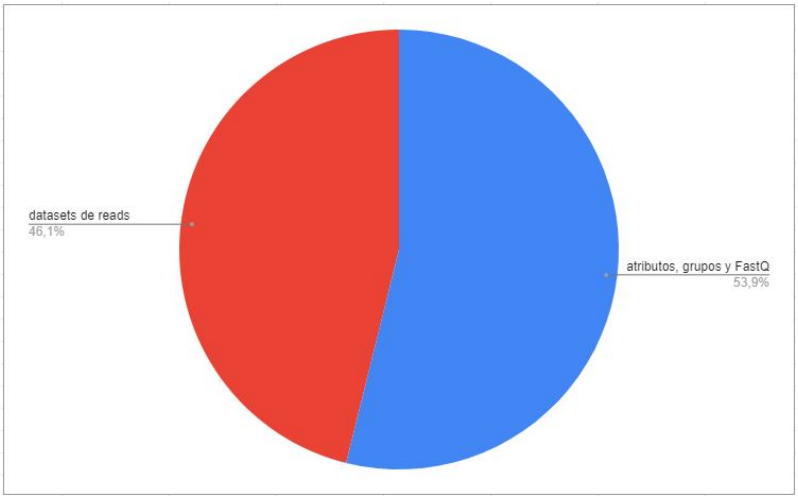


Figura 6.2: Proporción de espacio ocupado con basecalling



Se puede observar que en los archivos que no cuentan con basecalling, la proporción de espacio de almacenamiento ocupado por las lecturas es cercana al 95 %, mientras que en los archivos con basecalling, la proporción es cercana al 45 %. F5Comp es un software orientado a la compresión de lecturas crudas, es decir, que a mayor porcentaje de espacio de almacenamiento ocupado por lecturas crudas, mayor será la efectividad de la compresión. Por lo tanto F5Comp comprimirá en mayor medida archivos sin datos de basecalling.

4. Test de Compresión

El test de compresión consiste en ejecutar los 5 tipos distintos de compresiones y evaluar las tasas de compresión obtenidas por cada algoritmo para cada archivo de prueba. Esta tasa se calcula de la siguiente forma:

$$\frac{tamañoComprimido}{tamañoOriginal},$$

donde las variables *tamañoOriginal* y *tamañoComprimido* representan el espacio ocupado por el archivo antes y después de aplicarse la compresión, respectivamente. Cuanto menor sea la tasa de compresión, más eficiente el algoritmo. Cabe aclarar que F5Comp está hecho a medida para los archivos fast5, lo cual provee una clara ventaja con respecto a los compresores genéricos como zip y rar. También, como se mencionó previamente, no se ha podido utilizar la compresión *deep-lossless* de Picopore ya que no es compatible con el tipo de archivo multi-read que se desea probar. En el cuadro 6.2 se muestran las tasas de compresión de los distintos algoritmos para los archivos de prueba. Luego, en las figuras 6.3, 6.4 y 6.5 se pueden ver las comparaciones de tasas según el tamaño de los archivos.

Cuadro 6.2: Tasas de compresión

Archivos	F5Comp-gzip	F5Comp-huffman	rar	zip	picopore
1	0,999	**0,712**	0,910	0,918	0,995
2	0,999	**0,775**	0,928	0,935	0,999
3	0,996	**0,775**	0,927	0,928	0,999
4	0,996	**0,688**	0,950	0,954	0,996
5	0,996	**0,698**	0,919	0,927	0,996
6	0,999	**0,765**	0,925	0,933	0,999
7	0,999	**0,767**	0,927	0,926	0,999
8	0,996	**0,769**	0,927	0,933	0,996
9	0,996	**0,766**	0,927	0,933	0,996
10	0,996	**0,771**	0,928	0,934	0,996
11	0,988	**0,707**	0,977	0,979	0,988
12	0,988	**0,706**	0,978	0,979	0,988
13	0,988	**0,705**	0,978	0,979	0,988
14	0,988	**0,708**	0,978	0,980	0,988
15	0,988	**0,707**	0,978	0,980	0,988

Figura 6.3: Tasa de compresión de archivos menores a 250MB

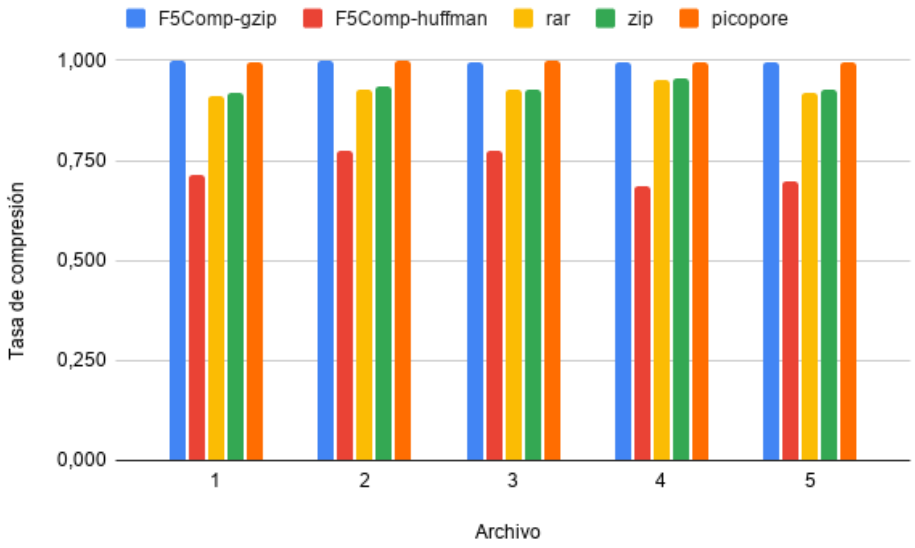


Figura 6.4: Tasa de compresión de archivos menores a 1 GB

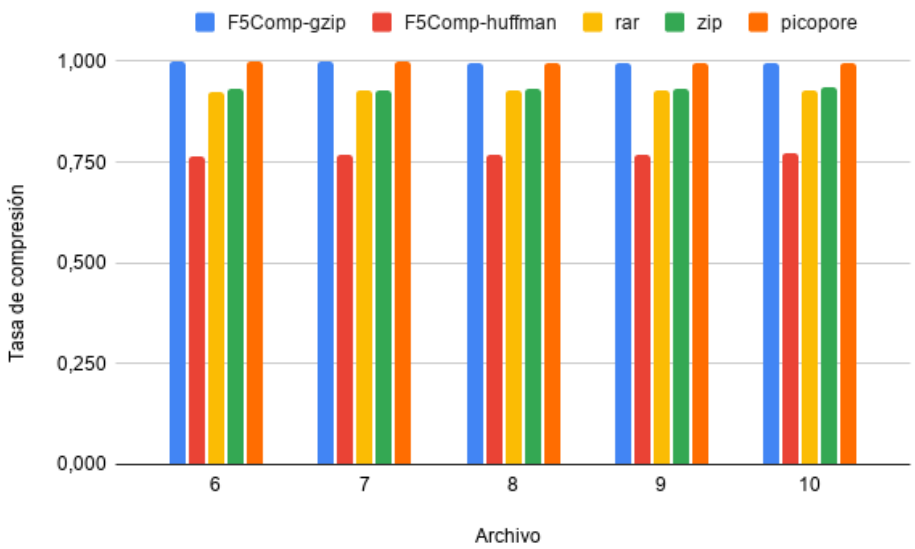
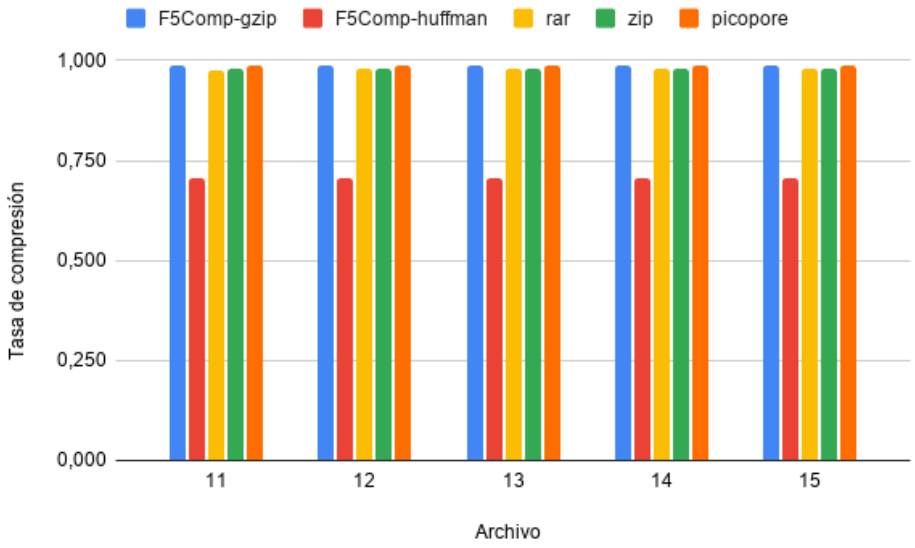


Figura 6.5: Tasa de compresión de archivos mayores a 1.5 GB



A partir de los gráficos se pueden desprender varias observaciones. La primera es que de forma consistente, la compresión utilizando F5Comp-Huffman es notablemente superior al resto. Una forma de apreciar esta diferencia es a través de la *diferencia relativa porcentual* entre la tasa de compresión de F5Comp-Huffman y la de los demás compresores. La *diferencia relativa porcentual* se calcula como:

$$\frac{tasa_x - tasa_{F5Comp}}{tasa_{F5Comp}} \times 100,$$

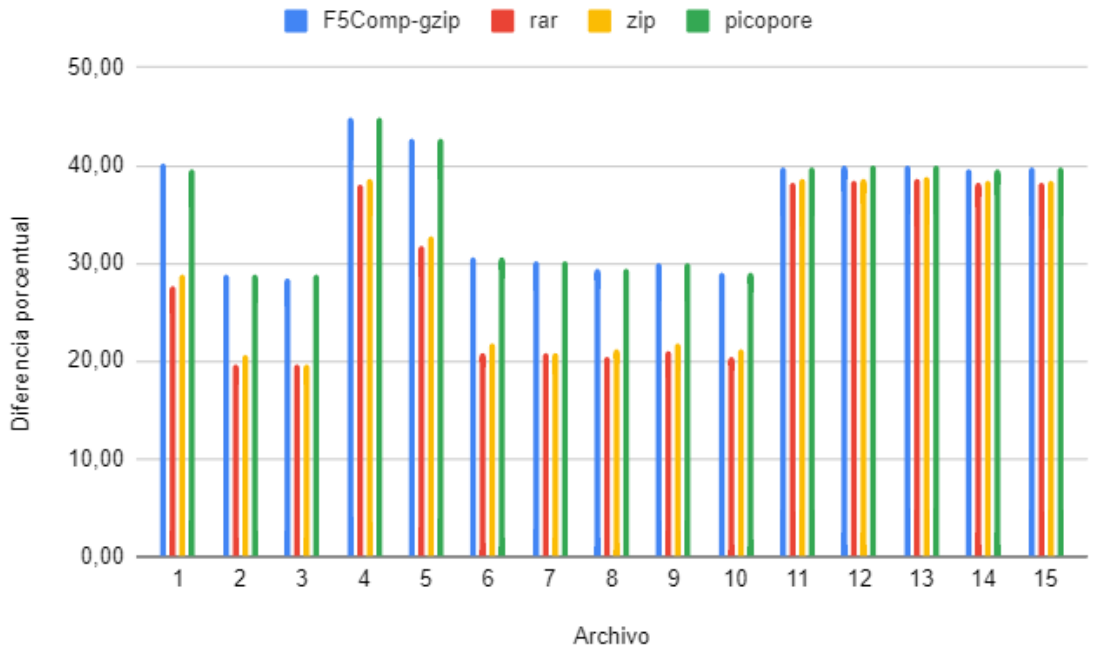
donde las variables $tasa_x$ y $tasa_{F5Comp}$ son las tasas de compresión del método a comparar y de F5Comp-Huffman, respectivamente.

En el cuadro 6.3 y en la figura 6.6 se puede observar esta diferencia.

Cuadro 6.3: Tabla de diferencia relativa porcentual entre la tasa de compresión de F5Comp-Huffman y la del resto de los compresores

Archivo	F5Comp-gzip	rar	zip	picopore
1	40,24	27,69	28,85	39,62
2	28,91	19,71	20,63	28,91
3	28,53	19,58	19,71	28,91
4	44,81	38,06	38,73	44,81
5	42,67	31,73	32,82	42,67
6	30,52	20,88	21,84	30,52
7	30,27	20,86	20,75	30,27
8	29,44	20,48	21,28	29,44
9	29,98	20,94	21,73	29,98
10	29,11	20,39	21,16	29,11
11	39,87	38,33	38,55	39,87
12	39,97	38,47	38,69	39,98
13	40,09	38,63	38,84	40,09
14	39,62	38,19	38,39	39,62
15	39,74	38,31	38,51	39,74

Figura 6.6: Gráfica de diferencia porcentual de la tasa de compresión de F5Comp-Huffman y la del resto de los compresores



Como se puede ver, la diferencia es siempre de al menos casi veinte puntos porcentuales, alcanzando en ciertos archivos los cuarenta puntos porcentuales. Lo más importante desde el punto de vista práctico es que para los archivos de mayor tamaño la diferencia se estabiliza con F5Comp-Huffman comprimiendo un 39 % más que el resto de los métodos.

También se puede ver esta diferencia en una tasa global de compresión, la cual se calcula para cada compresor como:

$$\frac{\sum_{i=1}^{15} tamañoComprimido_i}{\sum_{i=1}^{15} tamañoOriginal_i},$$

donde $tamañoComprimido_i$ y $tamañoOriginal_i$ son los epacios de almacenamiento ocupados por el archivo i antes y despues de la compresión, respectivamente. En este caso se separaron los resultados en función de las categorías de los archivos ya antes mencionadas.

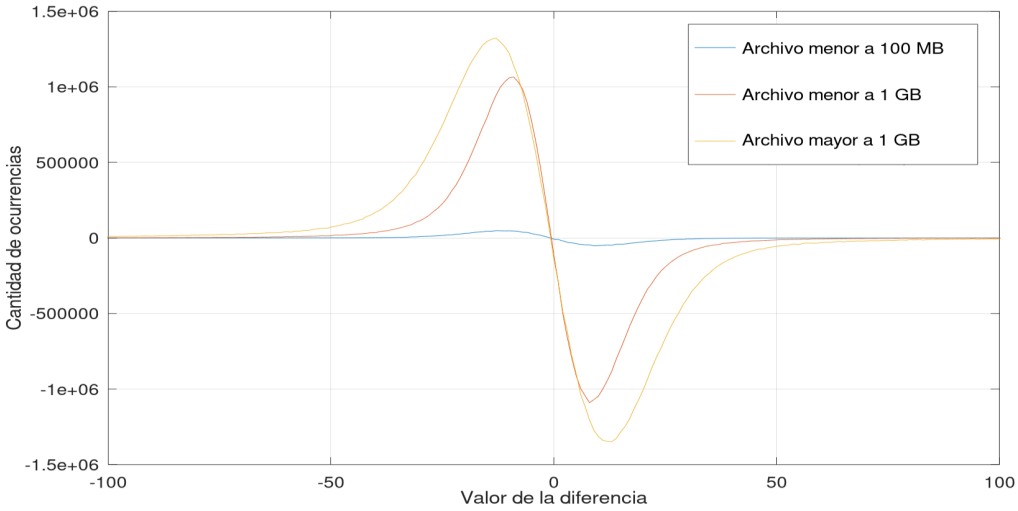
Cuadro 6.4: Tasas de compresión en función de tamaños de archivos

Compresión	Archivos pequeños	Archivos medianos	Archivos grandes
F5Comp-gzip	0,9969	0,9971	0,9883
F5Comp-huffman	0,7210	0,7679	0,7066
rar	0,9288	0,9269	0,9779
zip	0,9346	0,9319	0,9794
picopore	0,9967	0,9971	0,9883

Del cuadro 6.4 se desprende que la de tasa global de compresión de F5Comp es significativamente mejor que la de los demás. Una segunda observación es que la compresión F5Comp-gzip y la compresión lossless de picopore son muy similares. El parecido en estas compresiones se da porque la compresión lossless de Picopore solamente cambia la compresión gzip nativa de hdf5 de un nivel 3 (nivel por defecto) a un nivel 9, y ésta es justamente la compresión realizada por el modo gzip de F5Comp. La tasa de compresión para los primeros dos niveles de gzip es de 0.831 y de 0.828, manteniéndose prácticamente sin modificaciones hasta el nivel 9 el cual tiene una tasa de 0.825. Dicho de otra forma, los primeros dos niveles de la compresión gzip son los que tienen una mayor tasa de compresión con respecto al nivel anterior, mientras que los niveles 3 y 9 no presenta grandes cambios en cuanto a espacio ahorrado. Esta es la razón por la cual F5Comp-gzip y Picopore obtienen tasas de compresión cercanas a 1, que no representan niveles de compresión significativos.

Una tercera y última observación a partir de los datos obtenidos es que F5Comp-Huffman mejora su porcentaje de compresión con respecto a compresores genéricos cuando se trata de archivos más grandes. Esto se debe a que la concentración de las diferencias alrededor del cero es mayor en los archivos de mayor tamaño por lo que la compresión es más eficiente. Si se toman los valores de la figura 5.2 como una función y se la deriva, se observa una mayor pendiente para los archivos más grandes (ver figura 6.7), indicando que cuanto más grande el archivo, más se concentran las diferencias en los valores cercanos al cero. Como Huffman asigna palabras de menor tamaño a los valores mas cercanos al cero, esto permite que F5Comp-Huffman sea más eficiente en archivos más grandes.

Figura 6.7: Variación de ocurrencias de diferencias entre valores contiguos



5. Test de Velocidad

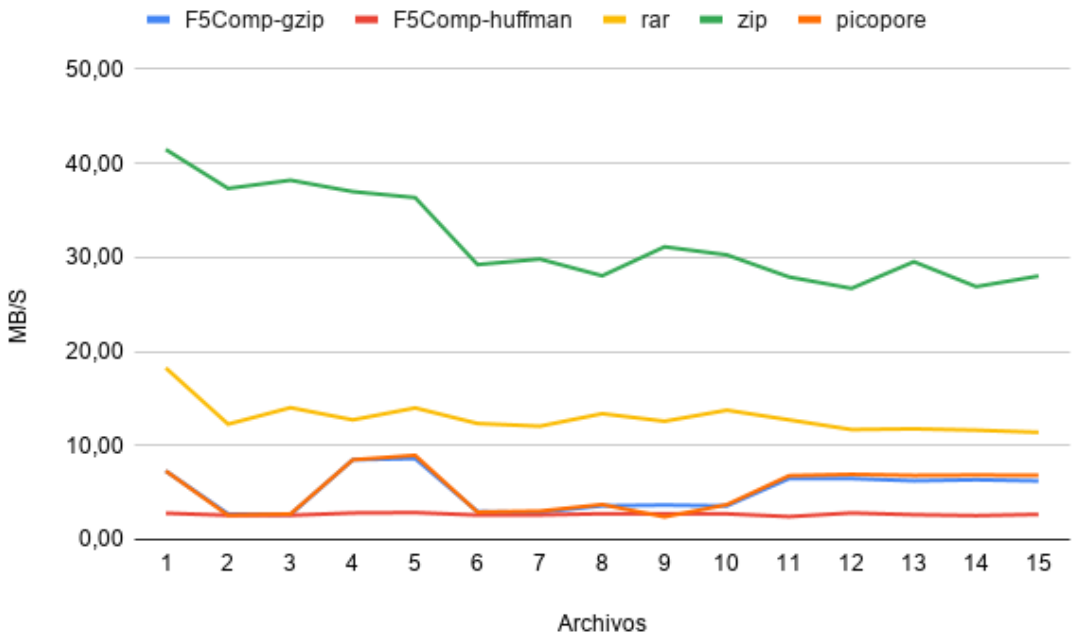
El test de velocidad consiste en ejecutar los 5 tipos distintos de compresiones y evaluar el tiempo de ejecución en función del tamaño de los archivos, obteniendo así una medida de velocidad en MBs por segundo. Para esto, se miden los tiempos de ejecución de los tests realizados previamente, cinco veces cada uno, y se promedian las cinco mediciones calculando la desviación estándar en cada caso. Finalmente se dividen los tamaños de cada archivo entre el promedio del tiempo de compresión y se obtiene una medición de velocidad de compresión. En la tabla 6.5 presentamos los tiempos medios de ejecución obtenidos con cada compresor para cada uno de los archivos de prueba, especificando la desviación estándar como un intervalo en torno a la media.

Cuadro 6.5: Mediciones de tiempo medio de ejecución y desviación estándar para cada compresor en cada uno de los archivos de prueba (en segundos)

Archivo	F5Comp-gzip	F5Comp-huffman	rar	zip	picopore
1	4,9 ± 0,1	13,4 ± 0,3	2,5 ± 0,1	0,9 ± 0	4,8 ± 0
2	38,3 ± 0,1	42,2 ± 0,3	7,9 ± 0,3	2,4 ± 0	36,8 ± 0,1
3	38 ± 0,2	42,2 ± 0,5	7,7 ± 0,2	2,3 ± 0	36,7 ± 0,2
4	18,7 ± 0,1	56,4 ± 0,3	12,3 ± 0,2	3,7 ± 0	18,5 ± 0,3
5	27,2 ± 0,2	80,1 ± 1,5	18 ± 0,4	5,4 ± 0,2	26,6 ± 0,1
6	198,3 ± 5,7	236 ± 1,6	47,4 ± 2,2	15,8 ± 0,3	182,1 ± 0,7
7	193,4 ± 1,2	241,2 ± 3,9	48,8 ± 1,5	14,8 ± 0,2	186,1 ± 0,4
8	165,8 ± 0,9	235,5 ± 1,2	52,4 ± 2,4	20 ± 1,8	160 ± 1,3
9	163,5 ± 0,6	235,8 ± 1,7	50,5 ± 1,9	21,4 ± 1,9	155,2 ± 1,7
10	176,9 ± 1	241,9 ± 1,8	54,4 ± 1,7	21,6 ± 0,7	166,8 ± 1,1
11	256 ± 8,4	616,4 ± 7,4	153 ± 9,3	58,2 ± 1	217,6 ± 3,6
12	245,6 ± 9,5	615,2 ± 8,9	150,9 ± 4,9	57,1 ± 2,3	223,1 ± 4,9
13	240,1 ± 4,7	621,9 ± 6	141,5 ± 7,5	57,7 ± 2,4	226,3 ± 2,3
14	248,4 ± 8,7	628 ± 5,3	137,3 ± 7	58,2 ± 1,8	237,4 ± 7,5
15	249,2 ± 5,1	629,9 ± 2,2	145,7 ± 6,4	57,5 ± 4,8	231,4 ± 1,8

Luego, al analizar las velocidades de compresión de la figura 6.8, observamos que los compresores genéricos como rar y zip son sumamente rápidos en comparación con F5Comp y Picopore.

Figura 6.8: Velocidad de compresión según distintos métodos



Otra observación es que la velocidad de compresión en todos los casos, incluyendo F5Comp-Huffman, es relativamente constante independientemente del tamaño del archivo. Esto es bueno, ya que indica que todos los compresores tienen la capacidad de escalar hacia archivos de mayor tamaño teniendo tiempos de ejecución aceptables. Se puede observar también que las velocidades de compresión de picopore y F5Comp-gzip varían de igual forma según el contenido de los archivos, fenómeno que no sucede con F5Comp-Huffman. En el caso de la velocidad de compresión de F5Comp-Huffman, se puede ver que es mas lenta que el resto ya que su técnica de compresión implica codificar muestra a muestra usando Huffman, lo cual es una tarea computacionalmente compleja.

Capítulo 7

Conclusiones y trabajo a futuro

1. Conclusiones

Consideramos apropiado comenzar las conclusiones de este proyecto con un resumen de los resultados obtenidos, no solo en base a los test experimentales sino también de algunas ventajas cualitativas inherentes al método de desarrollo utilizado. Como resumen de los resultados experimentales se puede decir que F5Comp obtuvo al menos un 20 % de ventaja en comparación con los compresores genéricos e incluso demostró comprimir mejor que Picopore en el formato de fast5 original, para el cual quedaron muchas técnicas de compresión sin implementar. Hubiera sido interesante la comparación con la compresión Deep-Lossless de Picopore en los archivos multi-fast5 pero la falta de mantenimiento de Picopore en versiones mas recientes lo hizo imposible. Los tiempos de compresión además resultaron ser mayores, pero esto es comprensible ya que el procesamiento de datos es más complejo. Se muestra evidente que al trabajar con este tipo de archivos las ventajas de tener una estructura dada y un tipo de datos limitados permiten obtener compresiones fuera del alcance de cualquier otro compresor. Si se aplicaran las mismas estrategias para un archivo hdf5 genérico no se podría haber obtenido la diferencia de almacenamiento que se obtuvo. Esto fue sin dudas gracias a la regularidad de los datos producidos por MinION y el formato ya establecido de fast5. Por ultimo una gran ventaja que no es tan sencilla de medir es el hecho de que se mantiene el formato hdf5 permitiendo el uso de todas las herramientas disponibles incluso luego de la compresión. Lamentablemente no ocurre lo mismo

con el formato fast5, el cual se pierde debido a que mantenerlo hubiera limitado las opciones al momento de comprimir. Creemos que hubiera sido posible obtener una mejor compresión rompiendo el formato hdf5; al coste de perder el acceso a las herramientas provistas para hdf5; queda como trabajo futuro explorar si esto valdría o no la pena.

En el transcurso de este proyecto se encontraron diversas dificultades que complicaron el avance. La primera barrera encontrada fue la falta de conocimiento en compresión por parte del equipo, esto llevó a dedicar tiempo de estudio sobre métodos de compresión y su implementación. Fue una muy buena oportunidad para expandir conocimientos e investigar sobre un área de la computación en la que poco se había trabajado. La misma situación se presentó con hdf5; se trataba de un formato completamente desconocido y no muy fácil de comprender, lo cual llevó a pasar por una etapa de investigación que fue muy provechosa en el transcurso del proyecto.

Una segunda complicación encontrada fue la dificultad de encontrar documentación de hdf5 que fuera útil para implementar un programa en C++ que usara estas bibliotecas. Si bien HDFGroup cuenta con un wrapper en C++, este no cuenta con todas las facilidades que tiene por ejemplo el de Python (el cual es más completo). En más de una ocasión se tuvo que utilizar soluciones alternativas para realizar ciertas funcionalidades que con el wrapper de python se podrían haber hecho de forma inmediata. En adición, cabe mencionar que la comunidad de desarrolladores de hdf5 en C++ fue menor de lo que esperábamos, aumentando así las dificultades para solucionar errores ocurridos en la implementación.

El tercer y último gran problema encontrado fue la dificultad de encontrar información oficial sobre el formato fast5. Si bien encontramos diversos foros con intercambios sobre este tema, los cambios de versión de fast5 hacen a que mucha de esta información esté desactualizada. El más claro ejemplo es el del cambio de formato de fast5 simple a multifast5. Muy avanzado el proyecto se obtuvo acceso a los foros de ONT en los cuales se publica la información oficial de cada actualización, lo que sirvió para corroborar los conocimientos adquiridos en el transcurso del proyecto.

Si bien se logró un compresor que cumple con los objetivos, en el correr del proyecto se cometieron algunos errores que enlentecieron el progreso. Entre ellos el mas grande fue basarse en los archivos de prueba de Picopore para hacer las compresiones, el formato de estos archivos correspondían a una versión de fast5 anterior a lo que se usa actualmente, por lo que el tiempo dedicado a adaptar el compresor a este tipo de archivos pudo haber sido aprovechado en otras cosas como por ejemplo nuevos métodos de compresión.
Otro error cometido fue posponer la descompresión más tiempo del conveniente. Sucedió que al esperar a tener varios métodos de compresión para desarrollar la descompresión, el tiempo de pruebas se extendió más de lo previsto. Si se hubieran desarrollado las dos cosas en paralelo se hubiera optimizado el tiempo utilizado.
Finalmente parados en este lugar en el cual se tiene un compresor terminado, entendemos que podríamos haber explorado con más detenimiento la idea de cambiar el formato hdf5 por otro, lo cual podría ayudar a reducir el espacio en disco utilizado. Si bien usar las bibliotecas hdf5 trae muchas facilidades, cambiar a otro formato pudo haber redundado en mejores niveles de compresión.

Como conclusiones finales, creemos que el proyecto desarrollado puede ser de gran interés para la comunidad científica dedicada a la secuenciación de ADN ya que provee una manera de disminuir el espacio en disco utilizado por los archivos procedentes de los dispositivos de ONT. Incluso el código puede ser tomado como base para compresores de mejor calidad y eficiencia. Finalmente cabe destacar que este proyecto para el equipo fue todo un reto y que el conocimiento que adquirimos fue muy valioso.

2. Trabajo futuro

2.1. Implementación de nuevos filtros de compresión

En este proyecto se utilizaron dos filtros de compresión distintos, gzip y szip. Estos filtros ya están integrados con hdf5 por lo que usarlos no causó mayores dificultades. Una posible mejora o trabajo a futuro sería integrar otros filtros de compresión desarrollados por terceros[24], como por ejemplo ZFP[25] o VBZ[26]. Estos filtros simplemente aplican compresiones sobre los datasets antes de ser guardados en disco.

Para integrar los filtros desarrollados por terceros se debe ir uno a uno a los repositorios correspondientes y seguir los pasos necesarios para compilarlos y agregarlos al proyecto. Por el costo necesario de agregar nuevos filtros y verificar su funcionamiento se decidió que sea una posible mejora o un trabajo futuro.

2.2. Completar técnicas faltantes para archivos fast5 de versiones anteriores

Dentro de las técnicas implementadas para las versiones anteriores el obstáculo más grande encontrado es el manejo de los datos de eventos. Los eventos, a diferencia de las lecturas en Signal, son un dato estructurado que contiene el inicio del evento, la duración, la desviación estándar y la media de los valores en el evento. Para los nuevos formatos de fast5 los eventos al momento de lectura fueron eliminados y solo aparecen luego del basecalling de forma más completa, pero para las versiones anteriores de fast5 los eventos se generaban en la lectura y eran corregidos en el proceso de basecalling almacenándose ambas versiones en el archivo (ambos en la sección de Analyses). Al estar estos formatos anteriores fuera del foco de este proyecto no fue posible completar su compresión, en particular sería posible eliminar los datos de media y desviación estándar de los eventos y volverlos a calcular en el proceso de de-compresión a partir de las lecturas. Tanto la media como la desviación estándar son almacenados como datos de tipo double, de 8 bytes cada uno, por lo que sería

[24] https://support.hdfgroup.org/services/filters.html

[25] https://github.com/LLNL/H5Z-ZFP

[26] https://github.com/nanoporetech/vbz_compression

bastante beneficioso eliminar estos datos de la versión comprimida. Por otro lado esto solo aplicaría para los archivos generados antes del 2018, aunque podría usarse como base para un compresor de archivos post-basecall ya que estos aún poseen una estructura de eventos (un poco diferente a la generada por el lector).

2.3. Paralelismo para múltiples archivos

Una de las ventajas proporcionadas por Picopore es el uso de multithreading para el procesamiento de varios archivos en simultaneo. En este sentido el compresor desarrollado no cuenta con compresión en paralelo, por lo que se encontraría en desventaja si se quiere comprimir varios archivos. El uso de múltiples hilos puede ser una tarea compleja y tanto el desarrollo como las pruebas requieren de un tiempo del que no se disponía. Por esta razón se decidió no implementar multithreading internamente, como forma alternativa para procesar múltiples archivos se deberá llamar al compresor de forma individual una vez por cada archivo.
Como el uso de multithreading mejoraría el rendimiento del compresor para múltiples archivos ya que se tendría un mejor control de la memoria usada, se decidió mantenerlo como una mejora a futuro que le aportaría gran valor al proyecto si se implementara.

2.4. Interfaz gráfica para visualizar datos estadísticos

El compresor desarrollado contiene una funcionalidad que genera en archivos csv información sobre los datasets que el usuario decida. Para poder visualizar esta información se debe usar algún software que pueda leer este tipo de archivos y generar gráficos (por ejemplo Excel, Matlab u Octave). Esto puede ser incómodo para el usuario ya que se agrega un trabajo adicional para poder visualizar los datos. Una mejora a futuro podría ser la integración con alguna biblioteca que permita graficar tablas de datos tales como Mathplotlib[27] o gnuPlot[28], o incluso tener una integración con otros programas a quienes se les delegue la realización de los gráficos.

[27] https://github.com/lava/matplotlib-cpp
[28] http://www.gnuplot.info/

2.5. Evaluar el consumo de memoria

Dentro de este proyecto, por razones de tiempo y alcance, no se evaluó el consumo de memoria de F5Comp, un dato que podría ser de utilidad al momento de compararlo con otros compresores. Si bien la memoria de la máquina virtual utilizada para las pruebas resultó ser suficiente, es de interés realizar pruebas más exhaustivas que indiquen cuánta memoria es recomendada para el correcto funcionamiento del compresor.

Bibliografia

[1] Antonio Alcalá Malavé. "Qué es el ADN". En: *Genética de la emoción: el origen de la enfermedad*. Ediciones B, 2015.

[2] Benjamin Chain James M. Heather. "The sequence of sequencers: The history of sequencing DNA". En: *US National Library of Medicine* (nov. de 2015).

[3] M.M. Hingorani. "Polymerase". En: *Encyclopedia of Genetics*. Ed. por Sydney Brenner y Jefferey H. Miller. New York: Academic Press, 2001, págs. 1497 -1499. ISBN: 978-0-12-227080-2. DOI: https://doi.org/10.1006/rwgn.2001.1010. URL: http://www.sciencedirect.com/science/article/pii/B0122270800010107.

[4] W Gilbert A M Maxam. "A new method for sequencing DNA." En: *Proceedings of the National Academy of Sciences* (feb. de 1977).

[5] Giovanni Di Muccio y col. "Insights into protein sequencing with an α-Hemolysin nanopore by atomistic simulations". En: *Scientific Reports* 9.1 (2019), pág. 6440. ISSN: 2045-2322. DOI: 10.1038/s41598-019-42867-7. URL: https://doi.org/10.1038/s41598-019-42867-7.

[6] D. A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes". En: *Proceedings of the IRE* 40.9 (1952), págs. 1098-1101. DOI: 10.1109/JRPROC.1952.273898.

[7] L. Peter Deutsch y col. *GZIP file format specification version 4.3*. RFC 1952. http://www.rfc-editor.org/rfc/rfc1952.txt. RFC Editor, 1996. URL: http://www.rfc-editor.org/rfc/rfc1952.txt.

[8] L. Peter Deutsch. *DEFLATE Compressed Data Format Specification version 1.3*. RFC 1951. http://www.rfc-editor.org/rfc/rfc1951.txt. RFC Editor, 1996. URL: http://www.rfc-editor.org/rfc/rfc1951.txt.

[9] Jacob Ziv y Abraham Lempel. "A universal algorithm for sequential data compression". En: *IEEE TRANSACTIONS ON INFORMATION THEORY* 23.3 (1977), págs. 337-343.

[10] L. Peter Deutsch y Jean-Loup Gailly. *ZLIB Compressed Data Format Specification version 3.3*. RFC 1950. `http://www.rfc-editor.org/rfc/rfc1950.txt`. RFC Editor, 1996. URL: `http://www.rfc-editor.org/rfc/rfc1950.txt`.

[11] Pen-Shu Yeh y col. "Implementation of CCSDS Lossless Data Compression in HDF". En: *Earth Science Technology Conference* (feb. de 2002).

[12] Robert Rice. "Some Practical Universal Noiseless Coding Techniques". En: (jun. de 1994), pág. 130.

[13] Solomon W. Golomb. "Run-length encodings". En: *IEEE TRANSACTIONS ON INFORMATION THEORY* 12.3 (1966), pág. 399.

[14] Nicholas J. Loman y Aaron R. Quinlan. "Poretools: a toolkit for analyzing nanopore sequence data". En: *bioRxiv* (2014). DOI: `10.1101/007401`. eprint: `https://www.biorxiv.org/content/early/2014/07/23/007401.full.pdf`. URL: `https://www.biorxiv.org/content/early/2014/07/23/007401`.

[15] Shubham Chandak y col. "Impact of lossy compression of nanopore raw signal data on basecalling and consensus accuracy". En: *bioRxiv* (2020). DOI: `10.1101/2020.04.19.049262`. eprint: `https://www.biorxiv.org/content/early/2020/07/17/2020.04.19.049262.full.pdf`. URL: `https://www.biorxiv.org/content/early/2020/07/17/2020.04.19.049262`.

Índice de figuras