UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

OPTIMIZACIÓN EN ALGORITMOS DE FILTRADO DE FASCÍCULOS DE MATERIA BLANCA SUPERFICIAL EN DATOS DE TRACTOGRAFÍAS PROBABILÍSTICAS

POR

Felipe Andrés Pineda Sepúlveda

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Electrónico

Profesores Guía

Dra. Pamela Guevara A.
Dra. Cecilia Hernández R.

Abril 2022
Concepción (Chile)

Agradecimientos

Mis agradecimientos al Proyecto FONDECYT 1190701, por el apoyo otorgado.

A mis profesoras guías, la Dra. Pamela Guevara A. y Dra. Cecilia Hernandez R. Agradezco la oportunidad de haber trabajado con ustedes, quiénes me han fomentado el interés y el encanto por este trabajo y por esta área de investigación. Muchas gracias por el tiempo y la paciencia que me han entregado en el desarrollo de este trabajo.

A mis padres. Muchas gracias por el gran apoyo, que me ha permitido vivir esta etapa de la vida, muchas gracias por toda la confianza que han tenido en mis buenas y malas decisiones, pero sobre todo, gracias por el amor, la paciencia y la entrega que siempre he recibido de ustedes. Si en la universidad viví momentos difíciles, sé que ustedes estuvieron ahí para aliviar mi carga y hoy no estaría redactando esta Memoria si no fuera por todo el amor que ustedes me han entregado.

A mi pareja y mi hijo. Este año en particular fue intenso, pero a pesar de las dificultades siempre han hecho un esfuerzo extra para que yo pueda dedicarme a mis deberes académicos. Muchas gracias por la paciencia que me han entregado, por apoyarme en mis decisiones, por sus ánimos en los momentos dolorosos y por cada momento de felicidad que me entregan todos los días. De ahora en adelante, comenzaremos otra etapa en la vida y espero que nos sigamos apoyado mutuamente como lo hemos hecho hasta ahora.

A mis compañeros y amigos. Les doy las gracias por hacer que cada momento de la universidad sea un momento de alegría, nunca olvidaré las noches de estudio, tardes juego y los cumpleaños improvisados. En especial quiero agradecer a quien fue mi compañero de departamento, contigo formé una gran amistad, siempre fuiste un apoyo y aprendí que podría contar contigo siempre. Espero que en un futuro seamos colegas y amigos.

Gracias a todos.

Sumario

Durante los últimos años se han desarrollado diversas técnicas de imagenología para el estudio del cerbero humano. Una de estas es la imagen por tensor de difusión, la cual permite visualizar fibras cerebrales.

Los algoritmos de segmentación automática de fibras, tienen como objetivo clasificar las fibras en sus respectivos fascículos. Este proceso automatizado, según las especificaciones del algoritmo, pueden generar errores o fascículos ruidosos, donde el ruido corresponde a ciertas fibras del fascículo que son diferentes del resto desde un punto de vista topológico y morfológico.

Los algoritmos de filtrado, fueron diseñados para eliminar las fibras ruidosas de los fascículos. A partir del análisis de un algoritmo de filtrado diseñado en Python, se desarrolla una versión optimizada en C. Este algoritmo utiliza como criterio de filtrado, la distancia SSPD, una métrica que permite establecer una unidad de medida entre curvas de diferentes longitudes, formas y localización.

Además del filtrado, se realiza un algoritmo de reducción dimensional, el cual transforma las fibras a un espacio bidimensional, minimizando los tiempos de cómputo en el proceso de filtrado. A través de un análisis teórico, en C se crean librerías destinadas a realizar estos procedimientos.

En el desarrollo de la implementación en C, se hacen diversas optimizaciones respecto al algoritmo original, siendo la principal, el cómputo de la mitad de la matriz de distancias SSPD, el cual es el procedimiento de mayor costo computacional del algoritmo. Estas optimizaciones, en conjunto con el uso de computación paralela generan un algoritmo de filtrado que es sobre 4000 veces más rápido que su implementación en Python.

Abstract

In recent years, various imaging techniques have been developed to study the human brain. One of these techniques is the difussion tensor imaging,which allows to visualize brain fibers.

The objective of automatic fiber segmentation algorithms is to classify the fibers into their corresponding fascicles. This automated process, according to the algorithm specifications, can generate errors or noisy fascicles, where the noise corresponds to certain fibers of the fascicle that are different from the rest from a topological and morphological point of view.

Filtering algorithms was designed to remove the noisy fibers of the fascicle. Based on the analysis of a filtering algorithm designed in Python, a C optimized version is developed. This algorithm uses the SSP distance as a filtering criterion, a metric that allows establishing a unit of measurement between curves of different lengths, shapes and location.

In addition to filtering, a dimensional reduction algorithm is performed, which transforms the fibers into a two-dimensional space, minimizing the computation times in the filtering process. Through a theoretical analysis, C libraries are created to perform these procedures.

In the development of the C implementation, various optimizations are made with respect to the original algorithm, the main one being the computation of half of the SSP distance matrix, which is the procedure with the highest computational cost of the algorithm. These optimizations, together with the use of parallel computing, generate a filtering algorithm that is over 4000 times faster than the Python implementation.

Tabla de Contenidos

Lista de Tablas

Lista de Figuras

1. Introducción

La tractografía probabilística es una de las técnicas de imagenología que permite reconstruir los tractos neuronales. Estos se obtienen a través de un método derivado de imagen de resonancia magnética (del inglés *Magnetic Resonance Imaging*, MRI), llamada Imágenes con Tensor de Difusión (DTI). Este permite obtener los tractos cuantificando el grado de anisotropía del movimiento agua en los tejidos, lo que permite estimar la orientación de un conjunto de fibras dentro del cerebro.

Una de las aplicaciones que ha permitido la tractografía es la segmentación de las fibras en los diferentes fascículos presentes en el cerebro. Existen una serie de algoritmos que se han desarrollado para este fin, pero debido a la gran cantidad de datos presentes y a la cantidad de operaciones que se deben hacer, el desarrollo de algoritmos de procesamiento de fibras resulta con un alto coste computacional, generando programas con grandes tiempos de ejecución.

En la segmentación de fibras de asociación cortas, debido a su tamaño, localidad y morfología, aplicarles un algoritmo de segmentación utilizados en otro tipo de fibras, generan fascículos con alto nivel de ruido, es decir con una mayor presencia de fibras que no obedecen a la norma del fascículo. Por esto, es necesario realizar un filtrado de fibras ruidosas.

1.1. Antecedentes generales

En el área de la investigación, Python es uno de los lenguajes más utilizados a nivel mundial. Las ventajas de este lenguaje son su sintaxis intuitiva, una amplia cantidad de bibliotecas para diferentes propósitos y un amplio soporte en diferentes comunidades de programadores. Sin embargo, existen lenguajes de programación que son más eficientes en tiempos de ejecución.

En el presente trabajo se realiza un algoritmo en C, el cual es un lenguaje de programación del tipo compilador, cuyos programas resultantes realizan una menor cantidad de procesos internos por instrucción. Gracias a esto y a otras características, genera programas más eficientes en términos de costo computacional que un programa equivalente en Python.

Se realizará la optimización de 2 algoritmos de filtrado de fibras, cuyos tiempos de ejecución

son considerablemente altos. El primero, el algoritmo de Distancia simétrica entre Trayectorias (del inglés *Symmetrized Segment-Path distance*, SSPD), compara la similitud entre fibras en un espacio tridimensional, mientras que el segundo, el algoritmo de Mapeo Isométrico (del inglés *Isometric Mapping*, ISOMAP) transforma las fibras a un plano bidimensional y realiza la comparación usando el algoritmo de SSPD en esta menor dimensión.

1.2. Revisión bibliográfica

1.2.1. Mejora de segmentación Automática de fascículos de materia blanca Superficial en Datos de Tractografías Probabilistica *(Mendoza 2021, [1])*

El objetivo de este trabajo es desarrollar un algoritmo de segmentación de mejor resolución para fibras de la Materia Blanca Superficial (del inglés *Superficial White Matter*, SWM). Dicho trabajo fue realizado con 10 sujetos de la base de datos Proyecto Conectoma Humano (del inglés *Humman Conectome Proyect*, HCP) [2].

El algoritmo descrito en este estudio consta de los siguientes procesos:

- Aplicación de algoritmo de clustering (FFClust), obteniendo los centroides y los clusters correspondiente a cada centroide.
- Se realiza un filtrado de centroides por patrones de conectividad, es decir se eliminan los centroides cuyos extremos estén más alejados del atlas.
- Se reemplaza cada centroide por su respectivo clúster.
- Se hace un filtrado de clusters mediante el cálculo de la distancia SSPD entre las fibras del clúster, descartando aquellas que tengan una mayor distancia respecto a todas las fibras. Este procedimiento se aplica en 2 casos a modo de contrastar resultados y tiempo de ejecución:
    - Se calcula esta distancia en fibras que fueron transformadas a un espacio bidimensional mediante ISOMAP, siendo este algoritmo más rápido.

- Se adapta el algoritmo de distancias SSPD para determinar esta distancia en el espacio tridimensional original, siendo este algoritmo el que genera fascículos más homogéneos.

- Mediante un algoritmo de envolventes conexas se descartan las últimas fibras más alejadas.

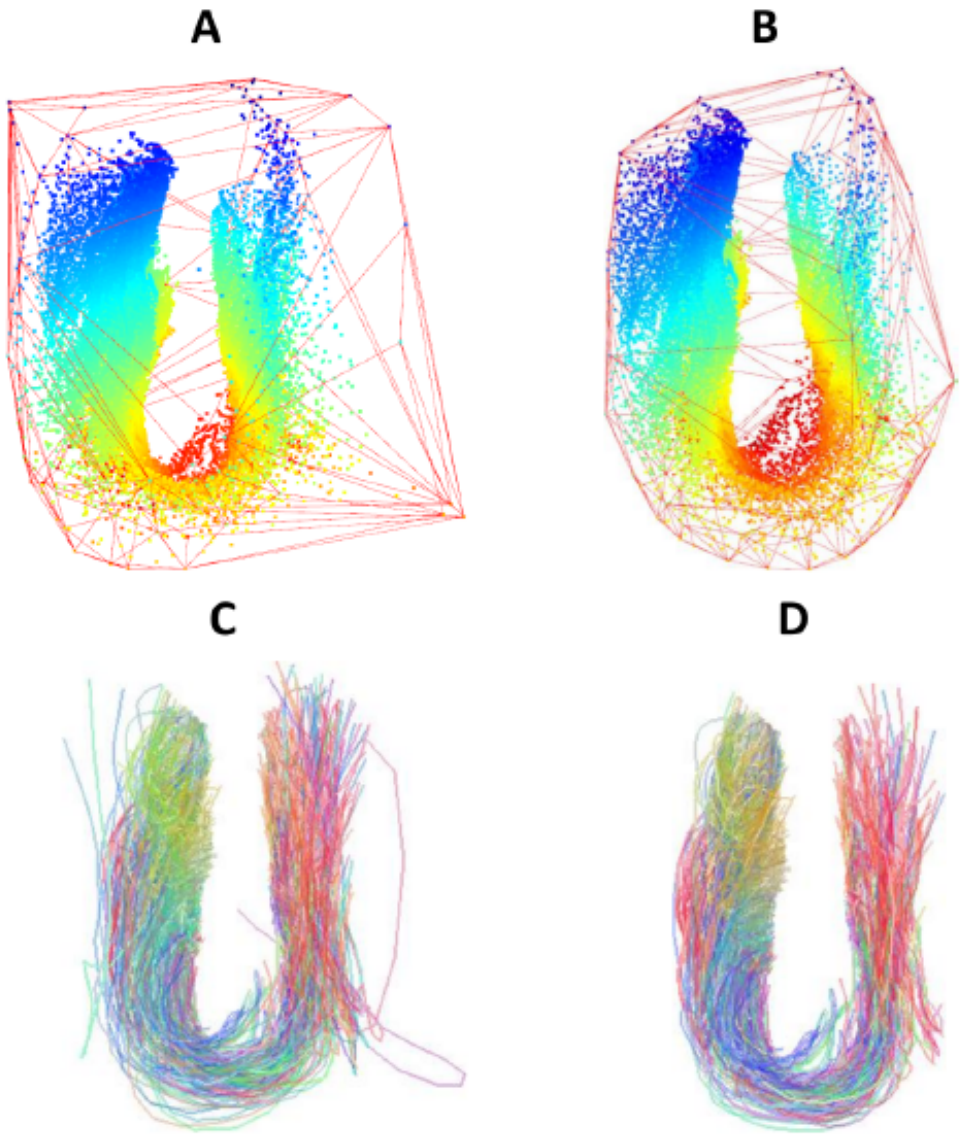


Fig. 1.1: Filtrado por de fibras utilizando envolventes convexas. Fuente: Mendoza, C. (2021) [1]

En la figura 1.1 Se observan los pasos del filtrado por envolventes conexas. En (A) se tienen las envolventes convexas de toda la figura, en (B) se reduce el volumen de la envolvente hasta

tener cierta densidad de puntos dentro de envolvente. En (C) se observa el fascículo completo mientras que en (D) se muestra el fascículo con las fibras ruidosas descartadas.

1.2.2. A Global Geometric Framework for Nonlinear Dimensionality Reduction (*Tenenbaum y de Silva 2000, [3]*)

En esta publicación se da a conocer el algoritmo de ISOMAP, el cual será descrito más detalladamente en la sección 2.2.4 y sus diferencias respecto a otras técnicas de reducción dimensional.

En este documento se detalla el concepto de distancia geodésica, la cual corresponde a la medida del recorrido de menor longitud posible sobre una superficie entre 2 puntos de interés. Si la superficie es un plano, estas distancias serán equivalentes a la distancia euclídea, mientras que si la superficie tiene una forma irregular, la distancia geodésica es mayor a la euclídea.

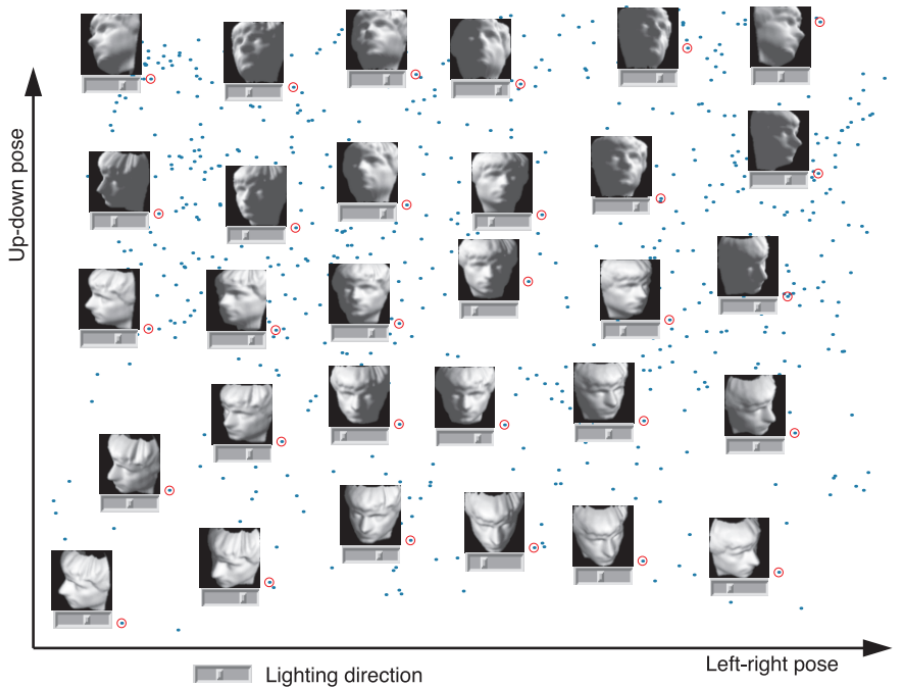


Fig. 1.2: Aplicación de un algoritmo de ISOMAP sobre un conjunto de imágenes de 64 x 64 píxeles. Fuente [3]

.

Además de lo mencionado, en el documento de detallan aplicaciones de ISOMAP en elementos no relacionados con distancias. Una de las aplicaciones es el análisis de un conjunto de

imágenes del renderizado de un rostro en diferentes posiciones y dirección de luz. Se aplicó el algoritmo a una muestra de 698 imágenes de 64 x 64 píxeles, lo que en términos dimensionales son 698 entradas de 4096 dimensiones. Al extraer las primeras 3 dimensiones resultantes, se puede notar que el algoritmo clasifica los datos según la rotación horizontal del rostro, la rotación vertical y la dirección de la luz incidente. Esta aplicación se puede ver en la figura 1.2.

1.2.3. Modern Multidimensional Scaling, Theory and Applications (*Borg y de Groenen 1997, [4]*)

Esta literatura es una enciclopedia completa sobre el algoritmo de Escalamiento Multidimensional (del inglés *Multidimensional Scaling*, MDS). Se detallan sus principios teóricos, principales aplicaciones, métricas de desempeño y optimizaciones mediante aproximaciones numéricas.

Se detallan los diversos tipos de MDS incluyendo el métrico (detallado en la sección 2.2.3) y el no métrico. Este último permite la aplicación del algoritmo a elementos no medibles o más abstractos, como valoraciones, tasas de contagio e incluso aspectos psicológicos.

En la figura 1.3 se observa un ejemplo de aplicación de MDS no métrico, sobre una matriz de correlación de la frecuencia de distintos tipos de crimen en 50 estados de EEUU. Se observa una representación en un plano bidimensional de estos tipos de crímenes para luego ser procesados por otro algoritmo de machine learning.

crime	no	1	2	3	4	5	6	7
murder	1	1.00	0.52	0.34	0.81	0.28	0.06	0.11
rape	2	0.52	1.00	0.55	0.70	0.68	0.60	0.44
robbery	3	0.34	0.55	1.00	0.56	0.62	0.44	0.62
assault	4	0.81	0.70	0.56	1.00	0.52	0.32	0.33
burglary	5	0.28	0.68	0.62	0.52	1.00	0.80	0.70
larceny	6	0.06	0.60	0.44	0.32	0.80	1.00	0.55
auto theft	7	0.11	0.44	0.62	0.33	0.70	0.55	1.00

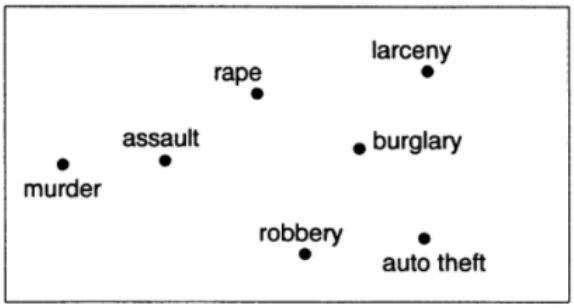


Fig. 1.3: Multidimensional Scaling en matriz de correlación de crímenes. Fuente: Adaptado de [4].

1.2.4. Otras Literaturas de interés

Además de este trabajo se revisaron otros trabajos relacionados como Bese *et al.*, 2016 [5] en el que se profundiza el desarrollo de la distancia SSPD; Roman, 2021 [6], en el cual se estudia la intersección de fascículos para la identificación de grupos similares; Vázquez *et al.*, 2020 [7], en donde se desattla el algoritmo de clústering FFClust y Goicovish 2019, [8] en el que se detalla el uso de CUDA para aceleración de algoritmos de clustering.

2. Marco Teórico

2.1. Fibras Cerebrales

2.1.1. Materia blanca y fibras de asociación

El sistema nervioso está formado principalmente por células denominadas neuronas. Estas son las encargadas de realizar labores sensitivas, motoras y cognitivas del cuerpo.

La estructura general de una neurona se compone del cuerpo celular o soma, las dendritas y el axón, como se puede apreciar en la figura 2.1. El axón cumple la función de unirse a otras neuronas por medio de la sinapsis y el conjunto de axones es lo que conforma los nervios o fibras neuronales.

En el cerebro, los somas y las dendritas se ubican en mayor proporción en la zona exterior (corteza cerebral) lo que se conoce como materia gris. Mientras que los axones atraviesan la zona más profunda del cerebro, permitiendo la sinapsis entre las distintas neuronas y regiones del cerebro. Debido a esto, la parte interior del cerebro tiene una estructura más fibrosa y se le conoce como sustancia blanca o Materia Blanca (del inglés *White Matter*, WM).

Las fibras de asociación o fibras neuronales corresponden al conjunto de axones que conectan determinadas regiones del cerebro. Existen las fibras de asociación largas, las cuales conectan regiones más distanciadas y fibras de asociación cortas, las cuales conectan regiones más adyacentes entre sí.

Debido a la localidad de las regiones, las fibras de asociación cortas tienen menor longitud y su forma curva es más pronunciada y definida, en comparación con las fibras de asociación largas. Por ello a estas fibras también se les conoce como fibras en "U". Además, como las regiones a comunicar están más próximas entre sí, las fibras que conectan estas regiones se ubican más cerca de la corteza cerebral, por lo que esta zona se denomina materia blanca superficial (SWM).

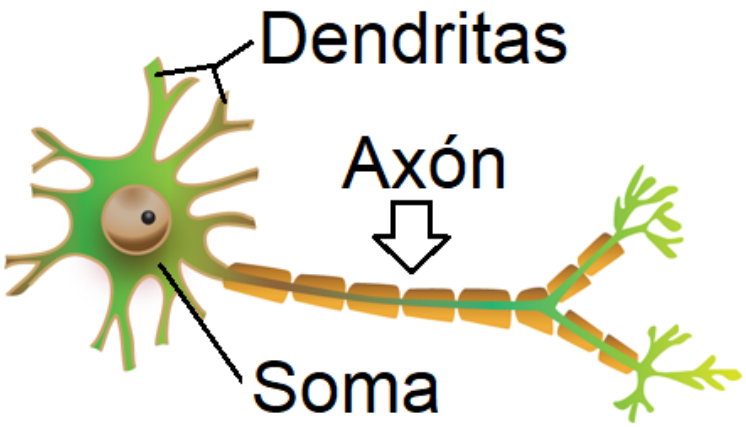


Fig. 2.1: Diagrama de una neurona y sus principales componentes. Adaptado de [9]

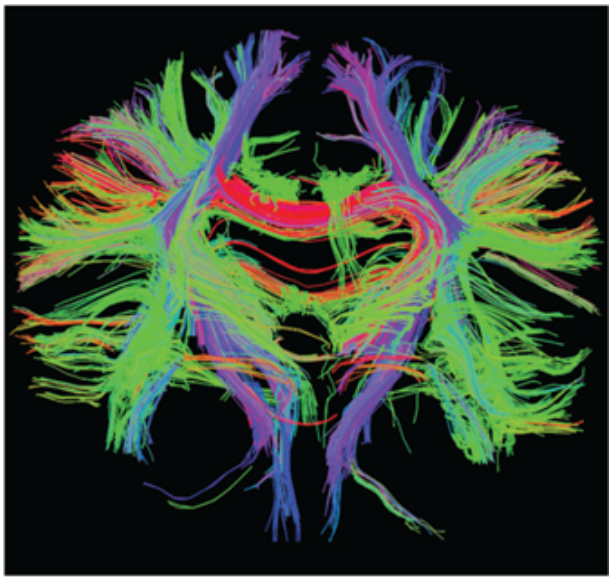
2.1.2. Tractografía

La tractografía es una técnica que permite obtener una reconstrucción tridimensional, que representa trayectorias estimadas de un conjunto de tractos o fibras neuronales. Estas se adquieren principalmente a través de imágenes de Resonancia Magnética por Difusión (del inglés *diffusion Magnetic Resonance Imaging*, dMRI), un método de imagenología no invasiva que ha permitido realizar estudios *in vivo* del sistema nervioso [10].

En dMRI hay diversas aplicaciones y técnicas que pueden utilizarse con diferentes objetivos. Para la obtención de las tractografías, se usa una técnica denominada Imagen por Tensor de Difusión (del inglés *Diffusion Tensor Imaging*, DTI), la que puede estimar las posiciones de los tractos neuronales en base a la medición de la difusión de las moléculas de agua en las distintas zonas del cerebro.

En la figura 2.2 se observa un conjunto de fibras obtenidas mediante DTI donde se aprecian conexiones entre diferentes áreas de la corteza.

Esta técnica permite obtener un numeroso conjunto de curvas en un espacio tridimensional, las cuales representan, una estimación de los principales fascículos de fibras nerviosas [11].



Fuente: John H. Martin: *Neuroanatomía texto y atlas,* 4e: www.accessmedicina.com

Fig. 2.2: Imagen por tensor de difusión. Fuente [10]

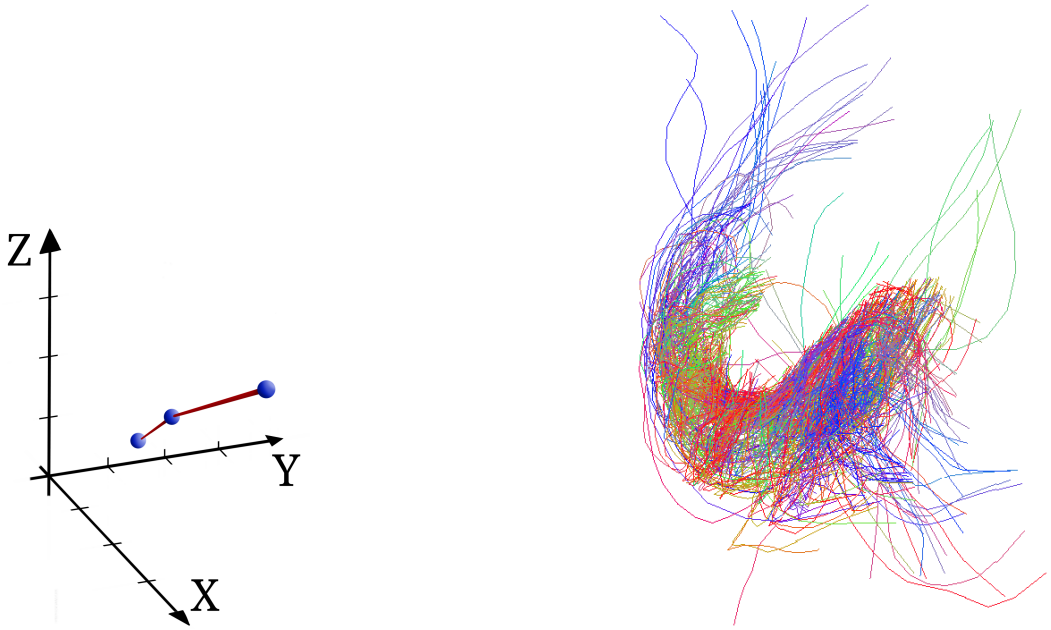
Es necesario resaltar que las fibras obtenidas mediante tractografía no representan las fibras neuronales individualmente, sino que representan una estimación de un conjunto de fibras (axones) neuronales.

2.1.3. Fibras

En el área de programación nos referimos al término fibra como una curva obtenida mediante una técnica de tractografía. Esta curva está descrita como una secuencia ordenada de puntos unidos a través de segmentos en un espacio tridimensional.

Cada punto se compone de 3 valores correspondientes a sus coordenadas en los 3 ejes en el espacio tridimensional. En la figura 2.3a se puede observar una fibra conformada por 3 puntos y en 2.3b un fascículo obtenido mediante un algoritmo de segmentación.

De ahora en adelante, el término trayectoria o fibra se utilizará para referirse únicamente a la curva descrita anteriormente y el término fibra neuronal para referirse a la estructura presente en la sustancia blanca.



(a) Una fibra con 3 puntos **(b) Conjunto de fibras**

Fig. 2.3: Visualización de fibras

2.1.4. Fascículos

En el área de la neurología, un fascículo es un conjunto de fibras nerviosas cuyos extremos conectan las mismas regiones y por ello tienen localización y morfología similares. En el cerebro, los fascículos conectan distintas zonas que pueden ser lejanas o adyacentes y estas conexiones cumplen funciones específicas según las zonas a conectar.

A nivel de programación, un fascículo corresponderá a un conjunto determinado de fibras. Este conjunto se almacena en un par de archivos ".bundles" y ".bundlesdata". El primero es un archivo de texto, el cual almacena información general del fascículo como su nombre, la cantidad de fibras, el orden de escritura de los bytes, etc. Mientras que el archivo ".bundlesdata" corresponde a un archivo binario almacena las coordenadas de las fibras.

La estructura de los archivos ".bundlesdata" comienza con un número entero que indica la cantidad "N" de fibras hay en el fascículo, seguido de "$3N$" números flotantes, correspondientes a las coordenadas de las fibras en el espacio tridimensional. Debido a esta estructura, durante la lectura del archivo, se deben agrupar los datos en los "N" grupos de 3 correspondientes a cada punto. Esta estructura se repite en todo el archivo y permite almacenar en un fascículo, fibras con diferentes cantidades de puntos.

También se permite almacenar múltiples fascículos en un solo par de archivos. En los archivos ".bundles" se tiene una par etiqueta-valor que indica desde cual fibra comienza el fascículo indicado por la etiqueta. Cuando se tiene solo un fascículo, solo existe un par etiqueta-valor cuyo valor es 0, indicando que el fascículo comienza desde la fibra 0, pero si se almacena más de un fascículo, en el archivo ".bundles" se tendrá más de un par etiqueta-valor, donde cada fascículo comienza con la fibra en la posición indicada por valor y termina en la posición anterior al valor de la posición del fascículo siguiente.

2.2. Algoritmos usados en el filtro

2.2.1. Segmentación Automática de Fibras

La segmentación de fibras es la acción de etiquetar un conjunto de fibras, en los fascículos correspondientes.

Los algoritmos de segmentación pueden clasificarse en 2 grupos: Segmentación basada en Regiones de Interés (del inglés *Regions of Interest*, ROI) y clustering basado en similitudes de fibra [6]. Adicionalmente, se puede hacer un sistema de segmentación basado en una combinación de ambos métodos.

En el caso del análisis por ROI, se tiene un conjunto de datos debidamente etiquetados de regiones de la corteza cerebral. El criterio de clasificación se basa en seleccionar 2 regiones y etiquetar en un fascículo, las fibras cuyos extremos estén ubicados al interior de las regiones seleccionadas. Esta técnica tiene mayor precisión respecto a la localización de las fibras, pero no considera el análisis de la morfología de estas.

En la segmentación basada en similitud entre fibras, se tiene un conjunto de fascículos de fibras debidamente etiquetados en sus respectivos fascículos. Se comparan las medidas y la morfología de las fibras sin etiquetar con cada una de las fibras etiquetadas. De esta forma, a las fibras sin etiquetar se les asignan las etiquetas de las fibras que más se le asemeje del atlas, formando así los fascículos. El desempeño de este algoritmo es variable según el número de puntos que componen las fibras.

En el algoritmo de segmentación, además del proceso de etiquetado de fibras, se puede hacer un proceso de filtrado, en el cual se eliminan las fibras más ruidosas, es decir aquellas que

tienen una morfología y/o localidad más diferenciada en comparación con el resto de fibras del fascículo.

2.2.2. Distancia SSPD

La distancia simétrica entre trayectorias o SSPD, introducida originalmente por Beese et al. (2016) [5], es una métrica diseñada para calcular las distancias entre 2 curvas o trayectorias en un plano bidimensional, con el fin de obtener un parámetro para medir similitud entre trayectorias.

Una trayectoria corresponde a una sucesión finita de puntos, donde cada par consecutivo de puntos está unido por un segmento recto. Por lo tanto, un algoritmo para calcular la distancia entre 2 trayectorias está en función de sus puntos y segmentos. Se observa un ejemplo de trayectoria tridimensional es la fibra de la imagen 2.3a, la cual está compuesta por 3 puntos y 2 segmentos en un espacio tridimensional.

La distancia SSPD, corresponde a un promedio de las distancias entre los puntos que conforman una trayectoria y los segmentos más cercanos de otra. Para las trayectorias A y B compuestas por m y n puntos respectivamente, el algoritmo de cómputo de la distancia SSPD procede de la siguiente manera:

1. **Cálculo de la distancia de punto segmento D_{ps} de cada punto de la fibra A a los segmentos de la fibra B ($D_{ps}(A, B)$):**

   Para un punto 'p' y un segmento 'S' definido por 2 puntos s_1 y s_2, la distancia punto segmento, será la distancia euclídea entre 'p' y su proyección 'p^{proy}' en el segmento 'S'. Si la proyección 'p^{proy}' no está contenida en el segmento, entonces será la distancia euclídea entre 'p' y el extremo más cercano de 'S'.

   $$D_{ps}(p, S) = \begin{cases} d(p, p^{proy}) & si\ p^{proy} \in S \\ min(d(p, s_1)\ ,\ d(p, s_2)) & de\ otra\ manera \end{cases} \tag{2.1}$$

   Calcular la distancia D_{ps} entre 2 trayectorias corresponde a conocer todas las distancias D_{ps} desde los puntos de una trayectoria, respecto a los segmentos de la otra. En este caso el resultado de $D_{ps}(A, B)$ corresponde a una matriz de '$m \times (n-1)$' distancias, donde cada fila se corresponde con un punto de 'A' y cada columna con un segmento de 'B'. Es necesario notar que esta métrica se realiza sobre los puntos de una curva y los segmentos de la otra, por lo que esta operación no es conmutativa ($D_{ps}(A, B) \neq D_{ps}(B, A)$).

2. **Calculo de la distancia punto-trayectoria D_{pt} entre los puntos de la curva A y los segmentos de B ($D_{pt}(A, B)$):**

   Para un punto 'p' y una trayectoria 'T' definida por una sucesión finita de k puntos t_i y $k-1$ segmentos ts_i, la distancia punto trayectoria, será la distancia entre 'p' y el segmento más cercano de la trayectoria 'T', es decir, la menor distancia D_{ps}.

$$D_{pt}(p, T) = min\ _{i\in[1,\ldots,k-1]}(D_{ps}(p, ts_i)) \tag{2.2}$$

   En el caso de $D_{pt}(A, B)$ corresponde al conjunto de las distancias por cada punto de la curva 'A' respecto a la sección más cercana de la curva 'B'. A nivel analítico, esta será la menor distancia D_{ps} por cada punto de 'A'. De forma análoga al caso anterior, esta operación no es conmutativa.

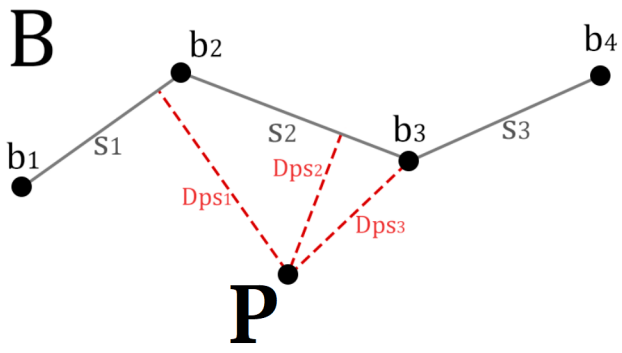


Fig. 2.4: Distancias punto segmento entre un punto P y la trayectoria B.

   En la figura 2.4 se observa la distancia punto segmento de un punto P a cada uno de los 3 segmentos de la trayectoria B. En este caso la distancia D_{ps2}, al ser la menor de las 3 distancias, corresponde a la distancia punto trayectoria.

3. **Cálculo de la distancia entre trayectorias D_{SPD} entre los puntos de la curva A y los segmentos de la curva B, ($D_{SPD}(A, B)$):**

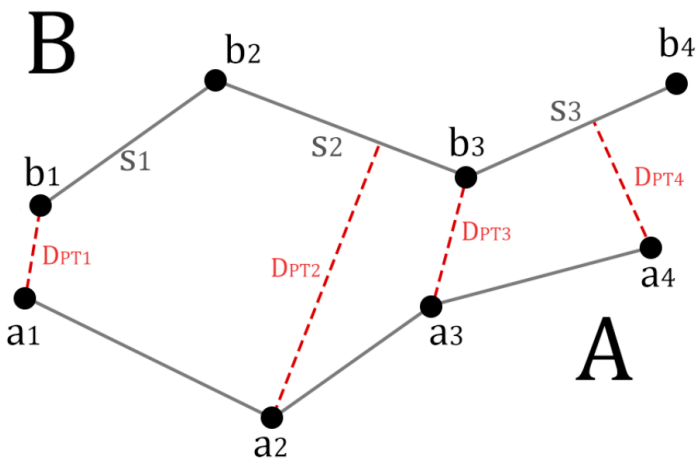
   La Distancia entre Trayectorias (del inglés *Segment-Path distance*, SPD) corresponde al promedio del conjunto de distancias punto trayectoria entre 'A' y 'B' ($D_{pt}(A, B)$). Esta operación no es conmutativa.

$$D_{SPD}(A, B) = \frac{1}{m}\sum_{i=1}^{m} D_{pt}(p_i, B) \tag{2.3}$$

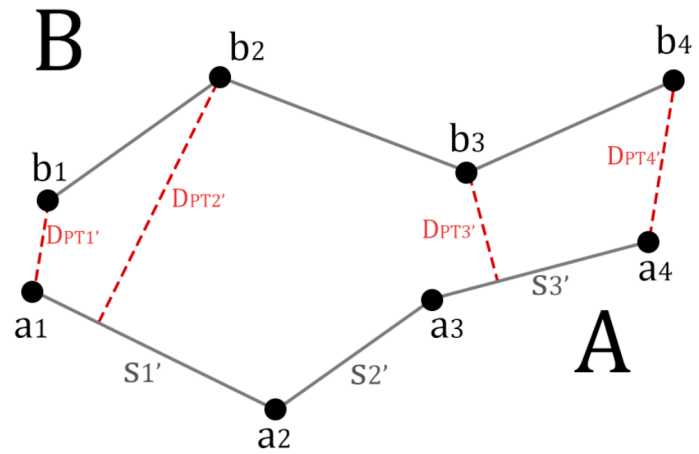
4. **Cálculo de la distancia D_{SSPD} entre las curvas A y B ($D_{SSPD}(A, B)$):**

Corresponde al promedio de las distancias D_{SPD} de los puntos de la curva 'A' respecto a los segmentos de 'B' ($D_{SPD}(A, B)$) y de los puntos de la curva 'B' respecto a los segmentos de 'A' ($D_{SPD}(B, A)$), por lo que se repiten los pasos 1-3 conmutando las curvas 'A' y 'B'. Esta operación es conmutativa.

$$D_{SSPD}(A, B) = \frac{D_{SPD}(A, B) + D_{SPD}(B, A)}{2} \tag{2.4}$$



(a) Distancias punto trayectoria desde la curva A a la curva B. $D_{pt}(A, B)$



(b) Distancias punto trayectoria desde la curva B a la curva A. $D_{pt}(B, A)$

Fig. 2.5: Distancias SPD

En las figuras 2.5a y 2.5b se observa la diferencia entre las distancias D_{pt} de los puntos de A respecto a los segmentos de B y vice-versa. La distancia SPD corresponde al promedio de todas D_{pt} por fibra y la distancia SSPD será el promedio resultante entre las 2 distancias SPD.

Este algoritmo original fue diseñado para el cálculo de la distancia entre curvas en un plano bidimensional, pero en Mendoza (2021) [1] se realizó una modificación del algoritmo para ser utilizado en curvas en un espacio tridimensional.

2.2.3. Multidimensional Scaling (MDS)

En el área del Machine Learning los algoritmos de reducción de dimensionalidad corresponden a procesos destinados a reducir la cantidad variables necesarias para representar cierta información.

Asumiendo que se tiene una muestra con una cantidad "e" de elementos y cada elemento es definido por una cantidad "c_D" de características, el total de datos se representaría por una matriz de tamaño "$e \times c_D$". En términos dimensionales, se dice que cada elemento tiene "c_D" dimensiones.

El algoritmo de MDS reduce la cantidad de características necesarias para representar cada elemento, pero considerando el peso de todas las características. De esta forma, al reducir a "c_d" características (siendo $c_d < c_D$), el total de datos necesarios para representar la muestra se reduce a "$e \times c_d$".

Existen diferentes variantes de MDS (clásico, métrico, no métrico y generalizado), cada una con sus ventajas y desventajas en términos de precisión, manejo de datos y costo computacional. En el caso particular de trabajar con distancias (el caso de interés de este estudio), se puede aplicar el algoritmo de MDS clásico, también conocido como Análisis de Coordenadas Principales (del inglés *Principal Coordinate Analisys*, PCoA), cuya aplicación se detalla a continuación.

El objetivo de aplicar el algoritmo de PCoA a una matriz "D" que contiene las distancias entre "e" puntos, es obtener una matriz de coordenadas "X" cuyas filas representan las coordenadas de los elementos en un espacio de "c_d" dimensiones, manteniendo las distancias establecidas por "D". El algoritmo permite elegir con cuantas dimensiones se desean representar esos puntos, pero el caso general y el utilizado en este estudio es hacer una reducción a un plano bidimensional.

El algoritmo se puede deducir a partir de la generalización de la expresión de distancia Euclídea, la cual permite conocer la longitud del segmento d_{ij} que une los puntos x_i y x_j en un

espacio bidimensional [4].

$$d_{ij}(x) = \sqrt{(x_{j1} - x_{i1})^2 + (x_{j2} - x_{i2})^2}$$

Por motivos de factorización, se trabaja con el cuadrado de d_{ij}.

$$d_{ij}^2(x) = (x_{j1} - x_{i1})^2 + (x_{j2} - x_{i2})^2$$

En el plano bidimensional, la distancia al cuadrado corresponde a la suma de 2 componentes al cuadrado, en un espacio tridimensional se sumarán 3 componentes y así sucesivamente. A partir de lo anterior, se puede generalizar la expresión para calcular la distancia euclídea al cuadrado entre los puntos x_i y x_j en un espacio de "c_d" dimensiones.

$$d_{ij}^2(x) = \sum_{k=1}^{c_d} (x_{jk} - x_{ik})^2 \tag{2.5}$$

Desarrollando la expresión anterior, se obtiene una ecuación que se generalizará matricialmente, para una mayor cantidad de puntos.

$$d_{ij}^2(x) = \sum_{k=1}^{c_d} (x_{ik}^2 + x_{jk}^2 - 2x_{ik}x_{jk})$$

$$d_{ij}^2(x) = \sum_{k=1}^{c_d} x_{ik}^2 + \sum_{k=1}^{c_d} x_{jk}^2 - 2\sum_{k=1}^{c_d} x_{ik}x_{jk}$$

$$d_{ij}^2(x) = x_{iT} + x_{jT} - 2x_{ijT} \tag{2.6}$$

Donde:

$$x_{iT} = \sum_{k=1}^{c_d} x_{ik}^2 \qquad x_{jT} = \sum_{k=1}^{c_d} x_{jk}^2 \qquad x_{ijT} = \sum_{k=1}^{c_d} x_{ik}x_{jk}$$

En lugar de calcular la distancia entre 2 puntos, se tienen todas las distancias entre todos los pares posibles de un grupo de "p" puntos. Esto corresponde a una matriz de distancias "D" de tamaño "$p \times p$", cuyos elementos representan las distancias entre los "n" puntos y los índices de las filas y columnas se corresponden con los índices de dichos puntos.

Como cada elemento es una distancia, la matriz será simétrica, ya que la distancia de un punto "x_i" a un punto "x_j" será la misma que la del punto "x_j" al punto "x_i". La matriz será no negativa, ya que las distancias son iguales o mayores a 0 y de diagonal nula, ya que la diagonal representa la distancia de cada punto consigo mismo.

$$D = \begin{bmatrix} 0 & d_{12} & ... & d_{1n} \\ d_{21} & 0 & ... & d_{2n} \\ ... & ... & ... & ... \\ d_{n1} & d_{n2} & ... & 0 \end{bmatrix}$$

De la expresión de distancia al cuadrado "d_{ij}^2" para 2 puntos (ecuación 2.5), se puede generar una expresión matricial para equivalente "$D^{(2)}$" para p puntos.

$$D^{(2)} = d_{ij}^2 \quad ; \quad d_{ij} \in D$$

Para esta matriz "$D^{(2)}$" existe una matriz "X", que corresponde a las coordenadas de los "n" puntos en un espacio de "m" dimensiones.

$$D^{(2)} = \begin{bmatrix} 0 & d_{12}^2 & ... & d_{1n}^2 \\ d_{21}^2 & 0 & ... & d_{2n}^2 \\ ... & ... & ... & ... \\ d_{n1}^2 & d_{n2}^2 & ... & 0 \end{bmatrix} \qquad X = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1m} \\ x_{21} & x_{22} & ... & x_{2m} \\ ... & ... & ... & ... \\ x_{n1} & x_{n2} & ... & x_{nm} \end{bmatrix}$$

La matriz de coordenadas "X" tiene por propiedad que el promedio de sus columnas es igual a 0. De esta forma se asegura que su distancias no son alteradas debido a traslaciones de los puntos.

A partir de la expresión $d_{ij}^2(x)$ definida para un par de puntos de coordenadas x_i y x_j (ecuación 2.6), se puede generalizar en una expresión matricial equivalente "$D^{(2)}(X)$" en función de la matriz de coordenadas "X" para el total de "p" puntos presentes en cada fibra.

$$D^{(2)}(X) = X_{iT} + X_{jT} - 2XX^T$$

Donde

$$X_{iT} = \sum_{k=1}^{c_d} \begin{bmatrix} x_{1k}^2 & x_{1k}^2 & ... & x_{1k}^2 \\ x_{2k}^2 & x_{2k}^2 & ... & x_{2k}^2 \\ ... & ... & ... & ... \\ x_{pk}^2 & x_{pk}^2 & ... & x_{pk}^2 \end{bmatrix} \tag{2.7}$$

$$X_{jT} = \sum_{k=1}^{c_d} \begin{bmatrix} x_{1k}^2 & x_{2k}^2 & ... & x_{pk}^2 \\ x_{1k}^2 & x_{2k}^2 & ... & x_{pk}^2 \\ ... & ... & ... & ... \\ x_{1k}^2 & x_{2k}^2 & ... & x_{pk}^2 \end{bmatrix} \tag{2.8}$$

$$XX^T = \sum_{k=1}^{c_d} \begin{bmatrix} x_{1k}^2 & x_{1k}x_{2k} & ... & x_{1k}x_{pk} \\ x_{2k}x_{1k} & x_{2k}^2 & ... & x_{2k}x_{pk} \\ ... & ... & ... & ... \\ x_{pk}x_{1k} & x_{pk}x_{2k} & ... & x_{pk}^2 \end{bmatrix} \tag{2.9}$$

La matriz "$D^2(X)$" se multiplica por la matriz de centrado "C", por la izquierda y por la derecha. La matriz de centrado de tamaño p se construye a plartir de la matriz identidad I y una matriz cuadrada J donde todos sus elementos son 1.

$$C_p = I_p - \frac{1}{p} J_p$$

La multiplicación por la matriz "C" tiene el efecto de restar el promedio por columna (si se multiplica por la izquierda) o por fila (si se multiplica por la derecha) a cada elemento de la matriz a la cual está multiplicando. A este proceso se le denomina doble centrado. De esta forma, se define una nueva matriz "B".

$$B = -\frac{1}{2} C\, D^{(2)}(X)\, C$$

$$B = -\frac{1}{2} C\, [X_{iT} + X_{jT} - 2XX^T]\, C$$

$$B = -\frac{1}{2} C\, X_{iT}\, C - \frac{1}{2} C\, X_{jT}\, C + C\, XX^T\, C$$

Se puede observar que cada fila de X_{iT} (ecuación 2.7) esta compuesta por el mismo elemento "p" veces, por ello, el promedio de la fila será el mismo valor de cada elemento y consecuencia de esto, al restar el promedio por fila a cada elemento de la matriz X_{iT}, se tendrá una matriz nula. Se puede hacer una razonamiento análogo con la matriz X_{jT} (ecuación 2.8), pero usando el promedio por columnas. Por lo tanto, al multiplicar estas matrices por la matriz de centrado, por la derecha y por la izquierda respectivamente, se anulan ambas matrices.

$$B = -\frac{1}{2} C\, [X_{iT}\, C] - \frac{1}{2} [C\, X_{jT}]\, C + C\, XX^T\, C$$

$$B = -\frac{1}{2} C\, [0]_n - \frac{1}{2} [0]_n\, C + C\, XX^T\, C$$

$$B = C\, XX^T\, C$$

Al principio del proceso se estableció como propiedad que el promedio de X por columnas es 0, con el fin de no alterar las distancias debido a las traslaciones. Por lo tanto, al multiplicar $C\, X$

el resultado será la misma matriz X y también sucede de forma análoga con la multiplicación $X^T C$.

$$B = C\,X\,X^T\,C$$

$$B = [C\,X]\,[X^T\,C]$$

$$B = X\,X^T$$

Finalmente se tiene que $X\,X^T$ es una matriz simétrica, por lo que se puede realizar la descomposición a través de sus vectores (Q) y valores (Λ) propios para así obtener la matriz X.

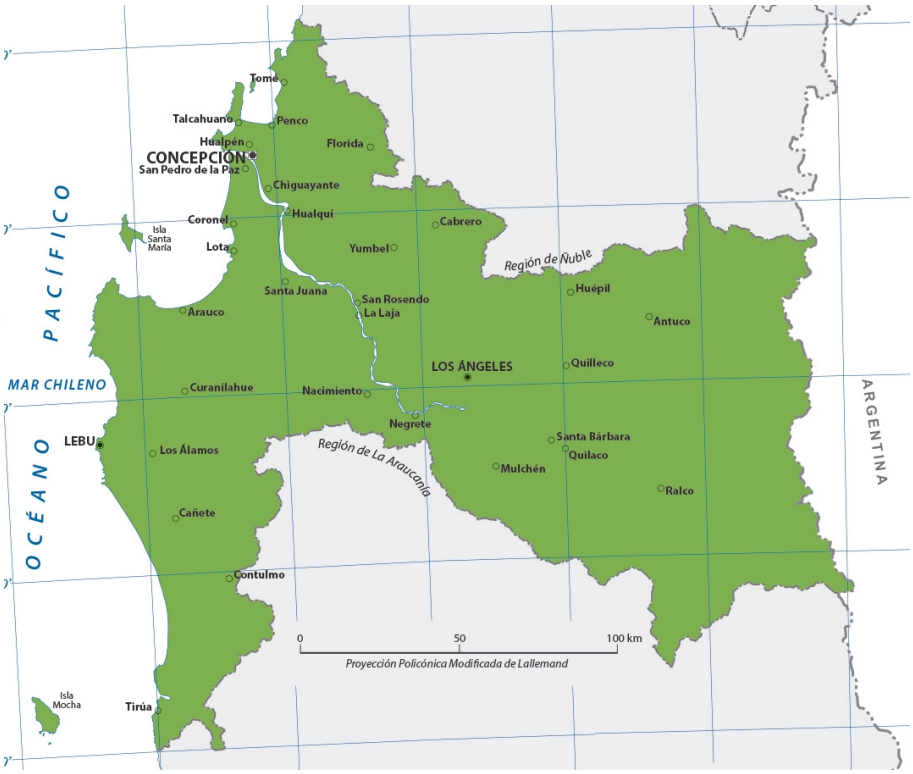
$$X\,X^T = Q\,\Lambda\,Q^T$$

$$X\,X^T = (Q\,\Lambda^{\frac{1}{2}})\,(Q\,\Lambda^{\frac{1}{2}})^T$$

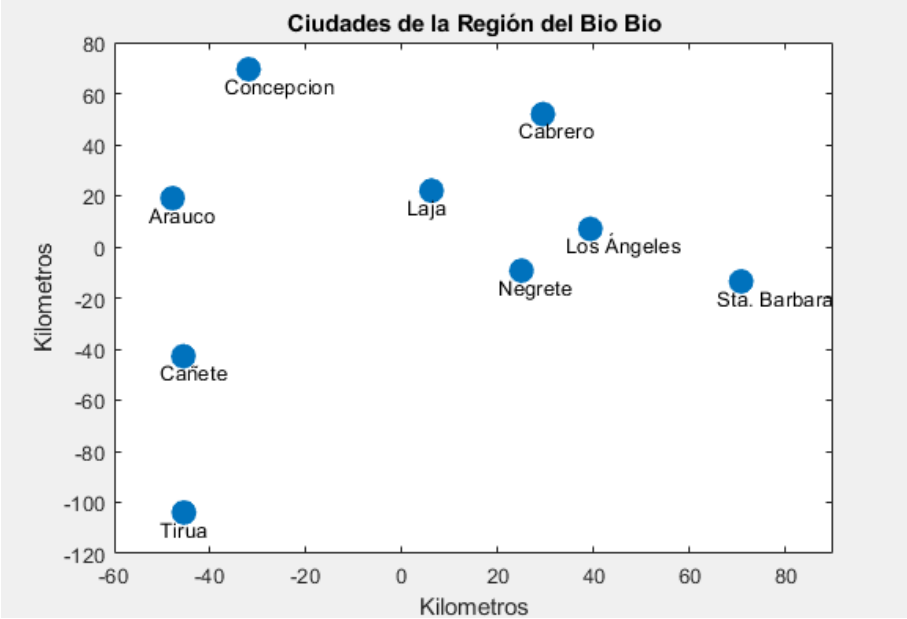
$$X = Q_{c_d}\,\Lambda_{c_d}^{\frac{1}{2}}$$

Siendo "c_d" la dimensión deseada para representar los datos. Se seleccionan los "m" mayores valores propios y sus correspondientes vectores propios como solución. Los valores propios negativos se consideran un error del algoritmo.

Un ejemplo clásico de las aplicaciones de MDS es determinar la localidad de las ciudades en base a las distancias entre estas. En la figura 2.6 a partir de un mapa se calcularon las distancias entre 9 localidades de la región del Bío Bío de manera de obtener una matriz de distancias. Una vez aplicado el algoritmo de MDS sobre la matriz, se puede observar que la distribución de puntos es similar a las localidades respectivas en el mapa.



(a) Mapa de la region del Bío-Bío



(b) Localidad de las Ciudades obtenida mediante MDS

Fig. 2.6: Ejemplo de aplicación de Multidimensional Scaling. Fuente: propia del autor.

Resumiendo, el algoritmo de MDS clásico es el siguiente:

1. A partir de una matriz de distancias D, se calcula la matriz de distancia al cuadrado, donde cada elemento de la matriz "D" se eleva al cuadrado.

$$D^{(2)} = [d_{ij}^2]$$

2. Se calcula la matriz "B" a partir del doble centrado de la matriz $D^{(2)}$.

$$B = -\frac{1}{2} C\, D^{(2)}\, C$$

3. Se calculan los valores propios de "B" y sus correspondientes vectores propios. Al tratarse de una matriz simétrica, los valores y vectores propios se pueden obtener a través de un algoritmo iterativo, como el método de Jacobi.

$$B = X X^T$$

$$X X^T = Q\, \Lambda\, Q^T$$

$$X_{c_d} = Q_{c_d}\, \Lambda_{c_d}^{\frac{1}{2}}$$

4. En este caso, se seleccionan los 2 mayores valores propios y se multiplica por las raíces de sus correspondientes vectores propios, obteniendo así la matriz "X" de coordenadas en un plano bidimensional.

$$X_2 = Q_2\, \Lambda_2^{\frac{1}{2}}$$

2.2.4. ISOMAP

El ISOMAP es un algoritmo de reducción dimensional, basado en MDS para representar un conjunto de datos en una menor dimensión (generalmente 2).

En la sección 2.2.3 se demostró que a partir de una matriz de distancias, se pueden obtener un conjunto de coordenadas que mantienen las distancias entre esos puntos. Una de las ventajas del algoritmo de MDS es que se puede aplicar una matriz que esté definida por cualquier tipo de distancia y no exclusivamente por la distancia euclídea. En ISOMAP se utiliza el algoritmo de MDS clásico en sobre una matriz de distancias geodésicas en vez de la distancia Euclídea entre puntos.

Desde el punto de vista geométrico, dada una superficie en un espacio tridimensional, la distancia geodésica corresponde a la longitud del menor recorrido posible sobre una superficie entre 2 puntos.

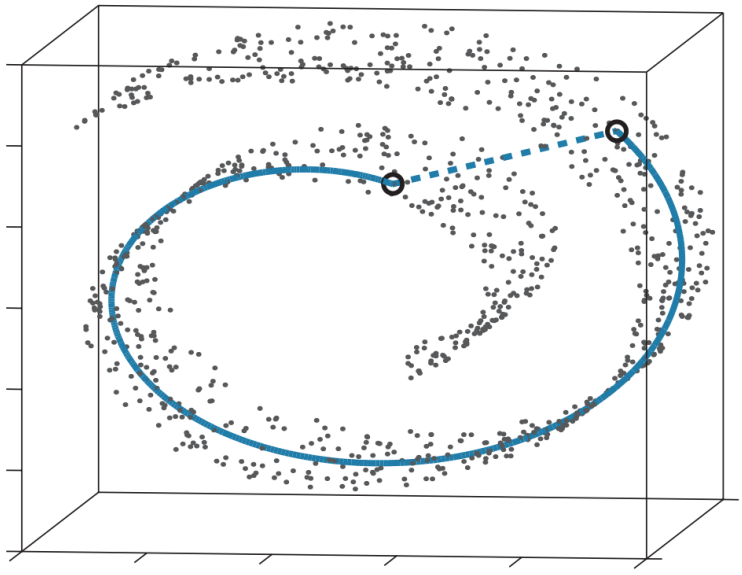


Fig. 2.7: Distancia euclídea y distancia geodésica. Fuente [3].

En la figura 2.7 se observa una representación gráfica de la distancia geodésica (línea continua) en comparación a la distancia Euclídea (linea punteada) entre 2 puntos aleatorios de un conjunto de datos. El dataset utilizado (´´´The swiss roll") es una nube de 1000 puntos distribuidos en una superficie en espiral.

De forma análoga a la matriz de distancias Euclídea, se desea obtener una matriz de distancias geodésicas, la cual posee las mismas propiedades de la matriz de distancia euclídea (simétrica, no negativa y de diagonal nula).

Desarrollar un algoritmo para analizar la morfología de una nube de puntos en un espacio tridimensional involucra un alto costo computacional. Por ello, en ISOMAP se opta por obtener una representación aproximada de la distancia geodésica mediante el análisis de grafos.

A partir de una matriz de distancias euclídeas, se obtiene una matriz de grafos. Esta corresponde a una matriz solo almacenará las distancias euclídeas de los pares de puntos que están más próximos entre sí (puntos conectados o puntos vecinos). A los pares de puntos que están más distantes, se les asigna un valor que representa que no están unidos (usualmente se usa infinito o un valor muy alto en comparación con la media).

Calculada la matriz de grafos, se calcula el trayecto más corto entre todos los pares posibles de puntos. En el caso de los pares de puntos distantes, el trayecto más corto será la suma de las distancias entre los puntos vecinos que unen estos puntos distantes. Se reemplazan los valores

de los pares de puntos no vecinos por el valor del trayecto más corto, obteniendo así una matriz de aproximaciones de distancias geodésicas.

A partir de una nube de puntos que definen una superficie en un espacio tridimensional, el algoritmo de ISOMAP que se aplica para representar esos datos en un plano bidimensional procede de la siguiente manera:

1. Se calcula la matriz de distancia D entre todos los puntos de la nube.
2. Por cada punto, se determinan los puntos vecinos. Esto se puede hacer mediante un algoritmo de los K-vecinos más cercanos (del inglés *K-Nearest Neighboor*, KNN) o mediante la selección de los puntos que estén dentro de un radio establecido.
3. Se obtiene la matriz G, la cual representa un grafo que contiene solo la distancia entre puntos vecinos. La distancia de puntos alejados (no vecinos) es desconocida, por lo que se le considera infinita (a nivel de informática se le asigna un valor muy grande en comparación con el resto de los datos).
4. A partir del grafo G, se calcula la longitud de los menores trayectos posibles entre puntos distantes, (generalmente se usa el algoritmo de Dijkstra). El resultado es la matriz D_G, la cual contiene las distancias geodésicas entre los puntos de la superficie.
5. Finalmente se aplica el algoritmo de MDS clásico en la matriz de distancias geodésicas D_G. Como se mencionó anteriormente, este algoritmo representará los puntos de la superficie tridimensional original en un plano bidimensional, manteniendo entre estos las distancias originales.

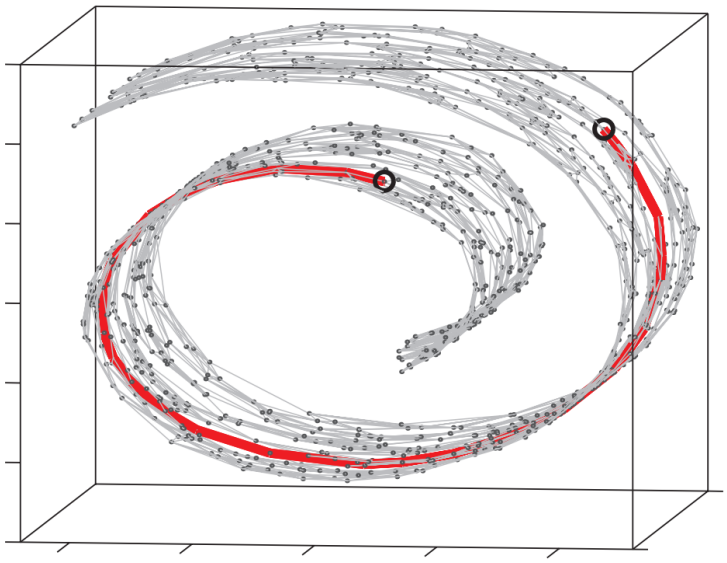


Fig. 2.8: Puntos vecinos y trayecto más corto. Fuente [3].

En la figura 2.8 se observa el resultado de aplicar el algoritmo de KNN (con K=7) para obtener los puntos vecinos de cada elemento del dataset. En rojo se observa el trayecto más corto entre los mismos puntos aleatorios vistos es la figura 2.7.

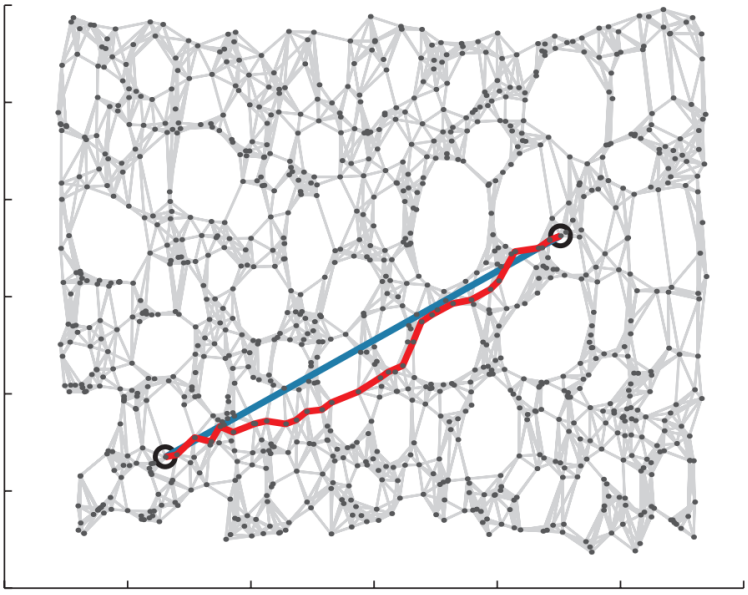
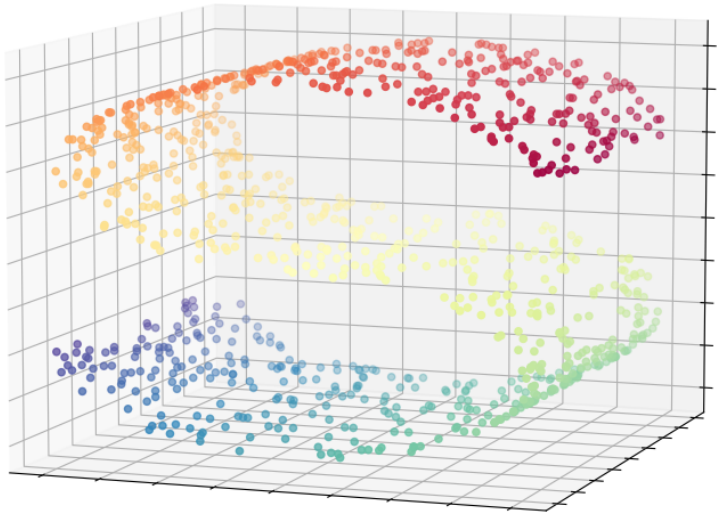


Fig. 2.9: Aplicación de ISOMAP en conjunto de datos. Fuente [3].

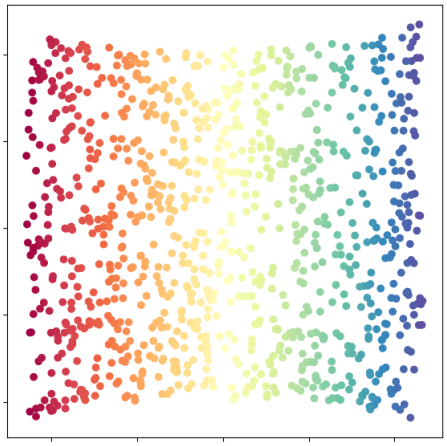
En la figura 2.9 se observa el resultado final de la aplicación de ISOMAP en el dataset "The swiss roll". Esta representación bidimensional es equivalente a "desenrrollar" el dataset, por lo que mantienen las distancias geodésicas. La línea azul corresponde a la distancia

geodésica entre los mismos puntos de la figura 2.7, mientras que la línea roja corresponde al trayecto más corto entre estos mismos pares de puntos.

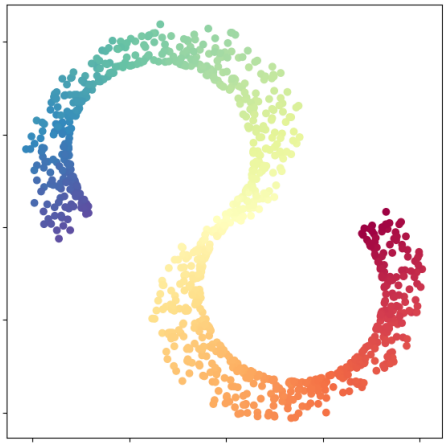
Cuando los datos se distribuyen en una superficie tridimensional, el algoritmo de ISOMAP genera una menor pérdida de información. Gracias al uso las distancias geodésicas, se conserva la geometría intrínseca de la superficie descrita por la nube de puntos, y los datos se distribuyen de mejor manera en el plano bidimensional en comparación al uso de MDS usando la distancia euclídea.



(a) Dataset Curva S (1000 puntos)



(b) Aplicación de isomap en Dataset



(c) Aplicación de MDS en Dataset

Fig. 2.10: Comparación de Isomap y MDS en dataset "Curva S".

Al tratarse de una aproximación, la precisión de la representación está dada por la cantidad

de vecinos de cada punto, pero el valor óptimo es dependiente de cada proceso y de la distribución de los puntos. En el caso del algoritmo de KNN la cantidad de vecinos está determinada por el valor de K. La cantidad inadecuada de puntos vecinos puede generar diferentes errores:

- Un valor alto de k, aumenta la cantidad de puntos vecinos y con ello, se pueden generar trayectos más cortos entre puntos no vecinos aproximándose mejor a la distancia geodésica. Pero según la morfología del conjunto de datos, puede generar como pares vecinos puntos que estén alejados superficialmente en el conjunto de datos. En este caso, la distancia obtenida se asemeja más a la distancia euclídea y no a la geodésica, por lo que la distribución bidimensional sería similar al uso de MDS usando distancias geodésicas. Esto se puede apreciar en la figura 2.11.

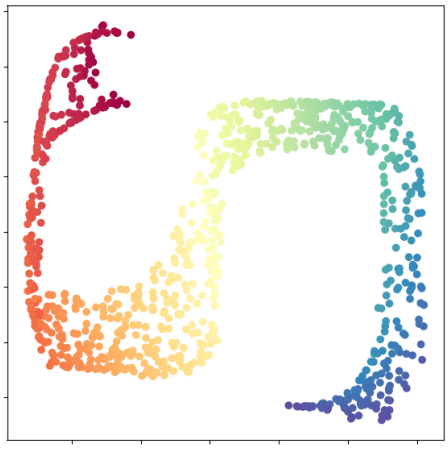


Fig. 2.11: Aplicación de Isomap con un alto valor de K

- Un valor de k es muy bajo, disminuye la cantidad de trayectos posibles entre pares de puntos, generando una mala aproximación de distancias geodésicas disminuyendo la dispersión de los datos en la representación bidimensional. Otro error debido a un k muy bajo, es que para una matriz de distancia se puede generar 2 o más grafos, lo que provocaría errores en los algoritmos de trayectoria.

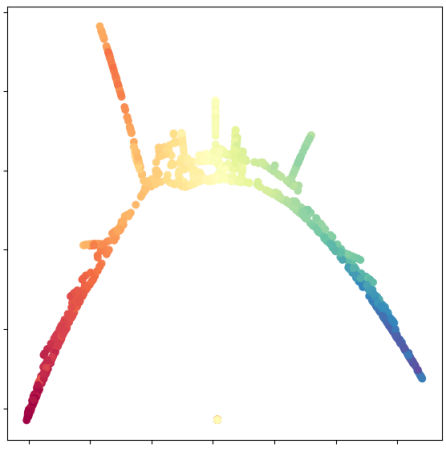


Fig. 2.12: Aplicación de Isomap con un valor bajo de K

2.2.5. Lenguajes de Programación

Al desarrollar un programa, primero se debe elegir el lenguaje de programación a utilizar. Un lenguaje de programación es un lenguaje formal, el cual le permite al programador redactar una serie de instrucciones (algoritmos) al computador. Cada lenguaje ofrece distintas características en términos de sintaxis del código y rendimiento del programa realizado.

En computación, un programa es una secuencia ordenada de instrucciones. A nivel del procesador, estas instrucciones corresponden a una secuencia de bits que le indican al sistema qué operaciones realizar, a cuáles bloques de memorias acceder y desde cuál fuente de entrada o salida recibirá o enviará la información. A esta secuencia de bits se le conoce como lenguaje de máquina y a pesar de que es posible desarrollar un programa directamente en este lenguaje, su sintaxis es de mayor complejidad, lo que dificulta su uso por parte de los programadores.

En programación, el término nivel de abstracción, hace referencia a la similitud o cercanía de una lenguaje de programación al lenguaje de máquina o desde otro punto de vista, está relacionado con la simplicidad de la sintaxis para el programador. Un lenguaje de programación de alto nivel de abstracción tiene una sintaxis más simple, pero el procesador realiza más procesos por cada instrucción descrita por el programador. Por otra parte, un lenguaje de bajo nivel de abstracción tiene una sintaxis más compleja para el programador, pero el sistema requiere de menos recursos para ejecutar cada instrucción. En base a lo mencionado, el lenguaje de máquina, sería el lenguaje con menor nivel de abstracción.

En la actualidad existen diversos lenguajes de programación de diversos niveles, cuyas dife-

rencias radican principalmente en su sintaxis. Además del nivel, hay que considerar el proceso de traducción del código descrito por el programador a lenguaje de máquina. Esta traducción se puede llevar a cabo por 2 procesos dividiendo a los lenguajes de programación en 2 grupos, los lenguajes compilados y los lenguajes interpretados.

Los lenguajes de programación compilados, como su nombre lo indica, requieren de un proceso de compilación. En dicho proceso, la traducción del código se realiza solo una vez, generando un programa ejecutable. Este programa ejecutable no almacena el código original, más bien almacena las instrucciones generadas por el compilador (en lenguaje de máquina) y estas son ejecutadas directamente por el procesador. Ejemplos de algunos lenguajes compiladores son C, C++ y Fortran.

Los lenguajes interpretados, a diferencia de los anteriores, realizan el proceso de traducción de cada instrucción durante la ejecución del programa. Este proceso le da más flexibilidad al desarrollador al momento de programar, como dividir un programa en secciones para ejecutarlas individualmente, ver y modificar las variables del programa, realizar pruebas rápidas del código sin tener que ejecutar el algoritmo completo, entre otras. El inconveniente del proceso de interpretación, es la necesidad de realizar dicho proceso cada vez que se ejecute el programa, por lo que requiere mantener el código fuente y ejecutarse siempre a través del intérprete. En este caso hay un proceso de traducción permanente, ralentizando la ejecución del programa. Ejemplos de estos son Python, Java, Ruby y MATLAB [12].

Además de los diversos lenguajes, también existen una diversidad de compiladores e intérpretes para cada lenguaje, con distintas optimizaciones según la secuencia de instrucciones. Esto genera diferencias en tiempos de ejecución de un mismo código descrito en cierto lenguaje usando distintos compiladores o intérpretes.

2.2.6. Computación Paralela

La mayoría de los equipos computacionales de la actualidad cuentan con procesadores multi-núcleo o multi-hebras. Comercialmente, los términos núcleos y hebras se describen como estructuras similares, pero son diferentes elementos del procesador los cuales abarcan el paralelismo de diferentes formas.

Un procesador multi-núcleo, posee más de un núcleo de procesamiento. Cada núcleo opera como un procesador independiente, lo que permite realizar cómputo de diferentes datos de

manera simultanea, mejorando el rendimiento de ciertos algoritmos. El uso de estas capacidades se le conoce como paralelización o computación paralela [13], mientras que la programación usando solo un núcleo se le conoce como programación secuencial. A veces cada núcleo tiene distintas arquitecturas, por lo que cada núcleo puede dedicarse a tareas específicas. En los equipos móviles, es común tener procesadores multi-núcleo, donde cada núcleo tiene diferentes consumos de potencia y velocidades, con el fin de optimizar el uso de la batería según las tareas en ejecución.

Por otro lado, en un procesador multi-hebra, el número de núcleos de procesamiento no es necesariamente variable. Cada hebra se puede interpretar como un flujo de entrada de instrucciones a un núcleo del procesador, por lo tanto en un procesador multi-hebra se pueden reorganizar las instrucciones de manera que en un núcleo existan menos tiempos de espera entre una instrucción y otra. A este proceso, informalmente se le denomina pseudo-paralelismo, ya que no se ejecutan diversos procesos de forma simultanea, pero la reorganización de las instrucciones, permite la ejecución de diversos procesos de forma más eficiente usando solo una hebra de procesamiento.

En la actualidad es común que los computadores comerciales sean multi-núcleo y multi-hebra a la vez.

A pesar de que la computación paralela ofrece una gran ventaja en términos de aumentar la capacidad computacional de un equipo, esta tiene restricciones en su uso y no puede aplicarse a cualquier algoritmo, lo que puede generar graves errores que no serán detectados por el programador.

La principal dificultad radica en el acceso a los recursos del sistema, no se puede acceder ni modificar una misma variable desde 2 instrucciones de manera simultanea, lo que conlleva a un error durante la ejecución del programa. Es necesario desarrollar un algoritmo considerando los momentos de acceso y modificación de las variables. Si para una variable x se requiere realizar primero la función f(x) y después la función g(x), se debe respetar el orden de esas operaciones en el proceso de paralelización. La falta de control de las operaciones podría alterar el orden de las funciones, realizando primero la función g(x) y posteriormente f(x), lo que podría generar errores no detectados por el programa, y continuar su ejecución con los datos incorrectos.

Los lenguajes de programación usados con mayor frecuencia no están diseñados para realizar programación paralela de forma nativa, sin embargo, se han diseñado bibliotecas que permiten el uso de la computación paralela, como la OpenMP para C [14].

3. Desarrollo

3.1. Objetivos

El objetivo de este estudio es mejorar el desempeño del procedimiento de filtrado del algoritmo de segmentación automática de materia blanca superficial (SWM) desarrollado por Mendoza (2021) [1].

Se espera que desarrollando algoritmos y funciones específicas en C, se obtenga un mayor control de las operaciones realizadas y del uso de la memoria. De esta manera lograr acelerar la ejecución del algoritmo no solo gracias cambio del lenguaje, sino que también a una menor cantidad de operaciones realizadas y a un acceso más eficiente a la memoria.

3.1.1. Objetivo general

Analizar y optimizar el procedimiento de filtrado de un algoritmo de segmentación automática de fibras cerebrales cortas en datos de tractografía probabilística, basado en un atlas de fascículos.

3.1.2. Objetivos específicos

- Analizar algoritmo filtrado de segmentación automática de fibras cortas existente y determinar partes de alto costo computacional.
- Evaluar las distintas zonas del algoritmo y determinar los mejores parámetros a utilizar para la base de datos disponible.
- Seleccionar las partes del código a optimizar, implementarlas en lenguaje C, y optimizarlas.
- Evaluar posibles zonas de paralelización.
- Desarrollar métricas para contrastar los resultados con los obtenidos por el algoritmo original.

3.2. Metodología

Se analizará el programa desarrollado por Mendoza 2021, [1], en busca de las secciones que demandan un mayor costo computacional e investigar posibles algoritmos alternativos en el desarrollo de sus procesos.

Este programa se puede dividir en 2 procesos: aplicación del clustering y filtrado de clusters, los cuales fueron desarrollados en C++y Python respectivamente.

En la ejecución del programa, se puede notar que el procedimiento de clustering tiene un tiempo de ejecución considerablemente menor que el procedimiento de filtrado. Este último, fue desarrollado en Python, el cual se considera es un lenguaje del tipo interpretado de muy alto nivel.

Al ser el algoritmo con mayor tiempo de ejecución, se optimizará el algoritmo de filtrado mediante distancias SSPD tanto para su uso con ISOMAPs como en el espacio tridimensional. Se desarrollará un algoritmo de filtrado equivalente al desarrollado por Mendoza 2021, [1] en C, el cual es un lenguaje de medio nivel del tipo compilado. Usando este lenguaje se espera que tener un mejor de los recursos de memoria y de las operaciones realizadas. Para el desarrollo de este algoritmo se utiliza principalmente el IDE "Code::Blocks" con el compilador MinGW-w64 y haciendo uso de la biblioteca "Open MP" para el desarrollo de computación paralela.

En primera instancia se desarrollará un archivo de encabezado (de ahora en adelante, encabezado) en C, que facilite el trabajo con fibras y permita la lectura y escritura de los archivos necesarios en el proyecto. Posteriormente se desarrollarán los encabezados específicos según las necesidades de cada filtro para realizar las operaciones aritmético-lógicas necesarias en la implementación del algoritmo.

Las funciones y métodos utilizados de las bibliotecas de Python son multipropósito, por lo que existen diversos medios de verificación y prevención de errores, como la comprobación del tamaño de los arreglos, verificación y transformación de los tipos de datos [15]. Gracias a estos procesos, Python permite tener una sintaxis menos abstracta para el programador, facilitando el desarrollo de código en este lenguaje.

A pesar de las virtudes que presenta un lenguaje de programación de sintaxis más simple, la principal desventaja de este, es el aumento del costo computacional por cada línea de código descrita. Añadiendo además el constante proceso de interpretación en la ejecución del código,

provoca un coste computacional muy alto.

Al generar operaciones aritmético-lógicas en C, se optará por generar funciones dedicadas al trabajo con fibras y no multipropósito, evitando así añadir algoritmos de verificación, ya que las entradas y salidas de cada función será la adecuada.

Una vez analizado los códigos y la documentación de las librearías utilizadas en estos, se espera desarrollar un código de arquitectura más simple y que sea fácilmente reproducible en otro lenguaje como C o C++.

3.3. Análisis del programa

Como fue detallado anteriormente, el algoritmo de segmentación se basa en el algoritmo de FFClust, el cual fue implementado en C++. El proceso de segmentación tiene un tiempo de ejecución considerablemente menor en comparación a los procedimientos de posteriores de filtrado. Por esto, se opta por optimizar los algoritmos de filtrado de clusters.

Con el fin de medir el desempeño por el cambio de lenguaje, en el algoritmo diseñado en C se realiza un código análogo al de Python, evitando realizar mayores modificaciones para acelerar la ejecución de estos.

Posteriormente se realizan las optimizaciones correspondientes, con el fin de acelerar lo más posible el algoritmo original.

3.3.1. Filtro por distancias SSPD en espacio tridimensional

Como el filtrado por distancia D_{SSPD} se aplica en ambos algoritmos (En el espacio tridimensional y en el plano bidimensional en conjunto con ISOMAP), se comenzará por optimizar esta técnica.

El algoritmo para filtrar un clúster es el siguiente:

1. Se computa la distancia D_{SSPD}, cuyo proceso fue detallado en 2.2.2, entre todas las fibras del clúster, de manera de obtener una matriz de distancias.
2. Obtenida la matriz, se calcula la suma por filas de manera de tener un vector que con el

total de distancias de cada fibra con respecto a las otras. Al ser una matriz de distancias, se tendrá una matriz simétrica.

3. Se descartan las fibras cuyo total de distancias no pertenezcan a un percentil establecido.

La sección de mayor costo computacional en este proceso es el cálculo de matriz de distancias D_{SSPD}. Por lo que se realizan distintas técnicas de optimización para este procedimiento.

3.3.1.1. Cómputo de distancia euclídea

El procedimiento principal para computar las distancias SSPD es conocer la distancia euclídea entre 2 puntos, la cual corresponde a la longitud del segmento recto que ambos puntos.

El algoritmo original, escrito en Python, se utilizan las funciones de la biblioteca numpy [15] para la mayoría de los procedimientos matemáticos. Las funciones y métodos presentes en estas bibliotecas son de propósito general, por lo que existen en ellas una serie de algoritmos redundantes destinados a la comprobación de los tipos de datos, tamaño de los arreglos, etc.

En el caso de la lectura y escritura de datos relacionados con fibras, se tiene conocimiento del tipo de dato con el que se trabaja y la forma en que se almacena, por lo tanto en el desarrollo del algoritmo en C se crea una función dedicada al cálculo de la distancia euclídea para puntos en espacios bidimensionales y tridimensionales, sin mayor necesidad de algoritmos redundantes, con el fin reducir el coste computacional en este proceso base para el cálculo de distancias entre 2 puntos.

La euclídea entre los puntos A y B en 2 y 3 dimensiones respectivamente, se obtiene de la siguiente forma:

$$A \wedge B \in \Re^2 \ : \ \ A = \{a_1, a_2\} \ \wedge \ B = \{b_1, b_2\}$$
$$dist(A, B) := \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

$$A \wedge B \in \Re^3 \ : \ \ A = \{a_1, a_2, a_3\} \ \wedge \ B = \{b_1, b_2, b_3\}$$
$$dist(A, B) := \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2}$$

3.3.1.2. Cómputo de distancia punto-segmento

Como las fibras están definidas por vectores de puntos, la distancia punto segmento se debe calcular de forma vectorial y evitar el uso de la ecuación de la recta para reducir el computo innecesario.

Se sabe que la menor distancia de un punto a una recta es la distancia de dicho punto a su proyección en la recta. Al trabajar con segmentos de recta se debe verificar que la proyección del punto pertenezca al segmento de la recta. Si se siente un segmento S definida por los puntos $s1$ y $s2$ y se desea verificar que la proyección de un punto P pertenezca al segmento S, se debe calcular el parámetro u, el cual se obtiene a partir de los siguientes productos escalares.

$$\overline{\mathrm{S}} \cdot \overline{\mathrm{Ps1}}$$

$$\overline{\mathrm{S}} \cdot \overline{\mathrm{S}}$$

Por definición de producto escalar, se tiene que:

$$\overline{\mathrm{S}} \cdot \overline{\mathrm{Ps1}} = \|S\|\|Ps1\|cos(\alpha)$$

$$\overline{\mathrm{S}} \cdot \overline{\mathrm{S}} = \|S\|^2$$

Siendo $\|Ps_1\|cos(\alpha)$ el módulo de la proyección de $\overline{\mathrm{Ps1}}$ sobre $\overline{\mathrm{S}}$ ($Proy(Ps1, S)$). El valor de esta proyección pertenece al rango de los números reales, pero la zona de interés entre $0 < Proy(Ps1, S) < \|S\|$. Si $Proy(Ps1, S) \leq 0$ significa que $Proy(Ps1, S)$ está en el sentido opuesto al segmento $\overline{\mathrm{S}}$ y por lo tanto la proyección del punto P sobre el segmento $\overline{\mathrm{S}}$ se ubica antes de $s1$. Por otro lado, si $\|S\| \leq Proy(Ps1, S)$, significa que la proyección $Proy(Ps1, S)$ es de mayor longitud que el segmento $\overline{\mathrm{S}}$ y la proyección del punto P en el segmento $\overline{\mathrm{S}}$ se ubicará después de $s2$.

Al normalizar todo por el producto escalar de $\overline{\mathrm{S}}$ consigo mismo se obtiene el parámetro u:

$$u = \frac{\overline{\mathrm{S}} \cdot \overline{\mathrm{Ps1}}}{\overline{\mathrm{S}} \cdot \overline{\mathrm{S}}}$$

De manera que la proyección de P pertenece al segmento $\overline{\mathrm{S}}$ cuando $0 < u < 1$. Además el valor de u indica la ubicación relativa de la proyección del punto P en la recta $\overline{\mathrm{S}}$.

Usando la representación analítica del producto escalar, se tiene que:

$$\overline{\mathrm{A}} \cdot \overline{\mathrm{B}} = (A_x \cdot B_x) + (A_y \cdot B_y)$$

$$\overline{\mathrm{S}} := \{s2_x - s1_x \ , \ s2_y - s1y\}$$

$$\overline{\mathrm{Ps1}} := \{P_x - s1_x \ , \ P_y - s1y\}$$

$$u = \frac{(s2_x - s1_x) \cdot (P_x - s1_x) + (s2_y - s1_y) \cdot (P_y - s1_y)}{(s2_x - s1_x)^2 + (s2_y - s1_y)^2}$$

Como el valor de u es una referencia a la ubicación de la proyección de P en el segmento, se tiene que la ubicación del punto P' que corresponde a la proyección del punto P en la recta $\overline{\mathrm{S}}$ es:

$$P' = s1 + u \cdot \overline{\mathrm{S}}$$

$$P' := \{s1_x + u \cdot (s2_x - s1_x) \ , \ s1_y + u \cdot (s2_y - s1_y) \ \}$$

Finalmente se puede calcular la distancia euclídea entre el punto y su proyección, si es que esta se ubica en el segmento $\overline{\mathrm{S}}$.

Entonces la secuencia de pasos utilizada para calcular la distancia punto-segmento es:

1. Verificar que los puntos s1 y s2 definan un segmento, es decir, que tengan al menos un componente que sea diferente. Si s1 y s2 tienen los mismos componentes, ambos puntos tienen la misma ubicación y por lo tanto bastaría con calcular la distancia entre P y s1, ya que sería la misma que entre P y s2.

2. Si los puntos s1 y s2 definen un segmento, se debe verificar que la proyección P' pertenezca al segmento $\overline{\mathrm{S}}$ mediante el cálculo del parámetro u.

3. Si u es menor o igual a 0, la distancia punto segmento será la distancia euclídea entre P y s1.

4. Si u es mayor o igual a 1, la distancia punto segmento será la distancia euclídea entre P y s2.

5. Si u tiene un valor entre 0 y 1, se calcula la proyección P' y la distancia punto segmento será la distancia euclídea entre P y P'.

El procedimiento descrito está desarrollado de manera de realizar la menor cantidad de computo posible. Si bien, realizar la menor cantidad de computo puede ser una ventaja, se debe

considerar que el computador tiene direcciones y bloques de memoria, por lo que realizar estas operaciones de manera selectiva, puede ralentizar la ejecución del programa debido a mayores tiempos de acceso a los bloques de memoria.

Con el fin de solucionar el problema de acceso a la memoria, previo a este procedimiento se calcula una matriz de distancias entre todos los pares de puntos de las fibras a comparar y un vector que contiene las distancias entre puntos consecutivos de la fibra, así se conocen previamente los valores de $\overline{\text{Ps1}}$, $\overline{\text{Ps2}}$ y $\overline{\text{S}}$. En caso de requerir estos valores en el algoritmo, estos estarán disponibles en la matriz de distancias y en el vector, y los datos estarán ubicados en direcciones de memoria adyacentes o muy cercanas, disminuyendo el tiempo utilizado en buscar direcciones de memoria de las variables.

Al ser este el proceso inicial para la obtención de la distancia D_{SSPD}, se implementa como una función cuyas entradas son las coordenadas del punto p, los puntos s_1 y s_2 que definen al segmento $\overline{\text{S}}$, la longitud de $\overline{\text{Ps1}}$, $\overline{\text{Ps2}}$ y $\overline{\text{S}}$. La salida sera la distancia punto segmento entre el punto p y el segmento $\overline{\text{S}}$.

En los pseudocódigos mostrados a continuación se utilizará la indexación de vectores y matrices a partir del valor 1 y no del 0.

function PUNTO_SEGMENTO(p[ ], s1[ ], s2[ ], dps1, dsp2,ds)

*Las entradas **p, s1, s2** corresponden al punto p y los puntos extremos del segmento s respectivamente. Las entradas **dps1, dps2, ds** corresponden a las distancia desde p a los extremos del segmento s y a la distancia entre los extremos de s.*

if $s1 = s2$ **then** ▷ Si los extremos del segmento están en la misma ubicación

return $D_{ps} \leftarrow dps1$ ▷ Termina la ejecución de la función sin calcular u.

end if

$diff \leftarrow s2 - s1$

$u \leftarrow (p_x - s1_x) \cdot diff_x + (p_y - s1_y) \cdot diff_y$

$u \leftarrow u/(ds \cdot ds)$

if $u \leq 0$ **then** ▷ Si u es negativo se usa la distancia $dps1$

return $D_{ps} \leftarrow dps1$

else if $u \geq 0$ **then** ▷ Si u es mayor a 1 se usa la distancia $dps2$

return $D_{ps} \leftarrow dps2$

else ▷ En otro caso se calcula la proyección de p en la recta

$Pp \leftarrow s1 + u \cdot diff$

$D_{ps} \leftarrow distancia\ euclidea(p, Pp)$

return D_{ps}
end if
end function

3.3.1.3. Cómputo de distancia punto trayectoria

La distancia D_{pt} corresponde a la menor distancia D_{ps} desde un punto P entre todos los segmentos que conforman la trayectoria T. A partir de este proceso, el costo computacional será dependiente de la cantidad de puntos que conformen la trayectoria T.

Si una trayectoria T está descrita por m puntos, entonces la trayectoria T estará conformada por $m-1$ segmentos. Por lo tanto el procedimiento para calcular la distancia punto trayectoria desde un punto P a la trayectoria T es el siguiente:

1. Se calcula la distancia punto segmento del punto P al primer segmento de la trayectoria T y se almacena su valor en una variable D_{pt}, la cual finalmente contendrá el valor mínimo.
2. De forma iterativa se calcula la distancia punto segmento del punto P con el segmento siguiente y se almacena e una variable temporal.
3. Se compara el valor de la variable temporal con el valor de D_{pt}. Si la variable temporal es menor que el valor mínimo almacenado, se actualiza el nuevo valor D_{pt}.
4. Se repiten los pasos 2 y 3 hasta haber calculado la distancia D_{ps} de todos los $m-1$ segmentos de la trayectoria T. De esta forma se asegura tener siempre la mínima distancia D_{ps}.

Del algoritmo descrito se puede deducir que es necesario calcular la distancia D_{ps} $m-1$ veces.

function PUNTO_TRAYECTORIA(p[ ], tray[ ][ ], m, m_dist[ ], t_dist[ ])

Entradas: ***p*** *corresponde al punto p,* ***tray*** *corresponde al conjunto de m puntos que definen una trayectoria y* ***m*** *es la cantidad de puntos que tiene la trayectoria. Las entrada* ***m_dist*** *corresponde a un vector con las m distancias desde p a los puntos de tray y* ***t_dist*** *es un vector que contiene las* $m-1$ *distancias consecutiva entre los puntos que conforman tray.*

$D_{pt} \leftarrow punto\_segmento(p, tray[1], tray[2], m\_dist[1], m\_dist[2], t\_dist[1])$

for $i \leftarrow 2, m-1$ **do**

$Temp \leftarrow punto\_segmento(p, tray[i], tray[i+1], m\_dist[i], m\_dist[i+1], t\_dist[i])$
if $D_{pt} \geq Temp$ **then**
$D_{pt} \leftarrow Temp$
end if
end for
return D_{pt}
end function

3.3.1.4. Cómputo de distancia entre trayectorias

Si se tiene una trayectoria T_1 con n puntos y una trayectoria T_2 de m puntos, se tiene que la distancia entre trayectorias de T_1 a T_2 es el promedio de las n distancias punto trayectoria de los puntos de T_1 a la trayectoria T_2.

El algoritmo implementado es el siguiente:

1. Se inicializa una variable acumuladora en 0 la cual cumple la función almacenar la sumatoria de las distancias punto trayectoria.
2. Se calcula la distancia punto trayectoria de forma iterativa e inmediatamente se suma su valor al acumulador.
3. Se repite el paso 2 hasta haber computado las n distancias y haberlas añadido al acumulador.
4. Finalmente se divide el acumulador por n para así obtener el promedio de las distancias.

function D_SPD(tray_1[ ][ ], tray_2[ ][ ], m_1, m_2, m_dist[ ][ ], t2_dist[ ])
Entradas: ***tray_1*** *y* ***tray_2*** *corresponden al conjunto de puntos que definen 2 trayectorias.* ***m_1*** *y* ***m_2*** *son la cantidad de puntos que tienen ambas trayectorias. Las entrada* ***m_dist*** *es una matriz de tamaño* $m\_1 \times m\_2$ *distancias entre los pares de puntos de ambas trayectorias y* ***t2_dist*** *es un vector que almacena la distancia consecutiva entre los* m_2 *puntos que conforman tray_2.*
$D_{SPD} \leftarrow 0$
for $i \leftarrow 1, m\_1$ **do**
$Temp \leftarrow punto\_trayectoria(tray\_1[i], tray\_2, m\_2, m\_dist[i], t2\_dist)$
$D_{SPD} \leftarrow D_{SPD} + Temp$

end for

$D_{SPD} \leftarrow D_{SPD}/m\_1$

return D_{SPD}

end function

3.3.1.5. Cómputo de distancia simétrica entre trayectorias

Corresponde al promedio entre ambas distancias entre trayectorias. Consecuencia de esto el computo de la distancia entre trayectorias (D_{SPD}) se realiza 2 veces y las operaciones que le anteceden se realizan múltiples veces en función de la cantidad de puntos que poseen ambas fibras.

Es en este proceso donde se computa la matriz de distancias entre los puntos de ambas fibras y los vectores de distancias entre puntos sucesivos de ambas fibras, de manera de que las funciones que le anteceden, principalmente la función "punto_segmento" realizan menos cómputo y las variables necesarias están ubicadas en direcciones de memoria más adyacentes.

function D_SSPD(tray_1[ ][ ], tray_2[ ][ ], m_1, m_2)

Entradas: ***tray_1*** *y* ***tray_2*** *corresponden al conjunto de puntos que definen 2 trayectorias.* ***m_1*** *y* ***m_2*** *son la cantidad de puntos que tienen ambas trayectorias.*

$M_{dist} \leftarrow 0_{m\_1 \times m\_2}$

for $i \leftarrow 1, m\_1$ **do**

for $j \leftarrow i+1, m\_2-1$ **do**

$M_{dist}[i,j] \leftarrow distancia\_euclidea(tray\_1[i], tray\_2[j])$

$M_{dist}[j,i] \leftarrow M_{dist}[i,j]$

end for

end for

$Mt_{dist} \leftarrow M'_{dist}$

for $i \leftarrow 1, m\_1-1$ **do**

$t1_{dist}[i] \leftarrow distancia\_euclidea(tray\_1[i], tray\_1[i+1])$

end for

for $i \leftarrow 1, m\_2-1$ **do**

$t2_{dist}[i] \leftarrow distancia\_euclidea(tray\_2[i], tray\_2[i+1])$

end for

$D1_{SPD} \leftarrow D\_SPD(tray\_1, tray\_2, m\_1, m\_2, M_{dist}, t2_{dist})$

$D2_{SPD} \leftarrow D\_SPD(tray\_2, tray\_1, m\_2, m\_1, Mt_{dist}, t1_{dist})$

$D_{SSPD} \leftarrow (D1_{SPD} + D2_{SPD})/2$
return D_{SSPD}
end function

3.3.1.6. Cómputo de matriz de distancias D_{SSPD}

Esta corresponde a una matriz con las distancias D_{SSPD} entre pares de fibras de un fascículo. Al ser una matriz de distancias, esta es una matriz cuadrada, simétrica, de diagonal 0 cuyos componentes triangulares serán mayores a 0.

En el algoritmo diseñado por Mendoza 2021 [1], este procedimiento calcula la matriz de distancia completa. De esta forma, si el clúster tiene M fibras, se calculan M^2 veces la distancia D_{SSPD} entre pares de fibras. Si se define CC_{SSPD} como el costo computacional del cómputo de una distancia D_{SSPD} entre un par de fibras, entonces el costo computacional del calculo de la matriz completa será:

$$M^2 \cdot CC_{SSPD} \tag{3.1}$$

Considerando las propiedades de la matriz de distancia, se puede computar solo un triangulo de la matriz y reutilizar sus valores para construir el otro triángulo. Por otro lado, como se sabe que la diagonal de la matriz son ceros, este valor se puede asignar directamente, evitando realizar el cálculo de la distancia de una fibra consigo misma.

Usando la misma definición de CC_{SSPD}, el costo computacional del computo de la matriz usando el método optimizado es :

$$\left(\sum_{k=1}^{M-1} k\right) \cdot CC_{SSPD}$$

En particular, a esa sumatoria se le conoce como la sumatoria triangular, y tiene una expresión matemática equivalente:

$$\frac{(M-1)^2 + (M-1)}{2} \cdot CC_{SSPD}$$

$$\frac{M^2 - M}{2} \cdot CC_{SSPD} \tag{3.2}$$

Si A corresponde al valor del costo computacional del algoritmo optimizado (3.2) y B es el costo computacional del algoritmo sin optimizar (3.1), entonces la proporción de operaciones

realizadas es:

$$A = \frac{M^2 - M}{2} \cdot CC_{SSPD}$$
$$B = M^2 \cdot CC_{SSPD}$$

$$\frac{A}{B} = \frac{M^2 - M}{2 \cdot M^2}$$

$$\frac{A}{B} = \frac{1 - \frac{1}{M}}{2}$$

Cada clúster tiene una cantidad variable de fibras. Como el objetivo de este algoritmo es filtrar las fibras ruidosas, se espera aplicar este procedimiento a los clúster que poseen mayor cantidad de fibras. Por lo tanto si la cantidad de fibras M es muy grande, se tiene que:

$$lim_{M \to +\infty} \frac{A}{B} = lim_{M \to +\infty} \frac{1 - \frac{1}{M}}{2}$$

$$lim_{M \to +\infty} \frac{A}{B} = 0.5 \tag{3.3}$$

De esta manera se estima que el costo de computacional del método optimizado para calcular la matriz tiende a ser la mitad del requerido al calcular las distancias D_{SSPD} de la matriz completa.

function MATRIZ_D_SSPD(bundle_1, n_fib)

Entradas: ***bundle*** *corresponde a una estructura que almacena un conjunto de fibras y la cantidad de fibras que posee.*

bundle.n_fib *Corresponde a la cantidad de fibras.*

bundle.fib[i] *Corresponde a las coordenadas de los puntos de la i-ésima fibra.*

bundle.fib[i].n_pt *Corresponde a la cantidad de puntos de la i-ésima fibra.*

$N \leftarrow bundle.n\_fib$
$MatD_{SSPD} \leftarrow [0]_{N \times N}$
for $i \leftarrow 1, N$ **do**
  $fib\_a \leftarrow bundle.fib[i]$
  for $j \leftarrow i + 1, N$ **do**
    $fib\_b \leftarrow bundle.fib[j]$
    $MatD_{SSPD}[i, j] \leftarrow D\_SSPD(fib\_A, fib\_b, fib\_A.n\_pt, fib\_b.n\_pt)$
    $MatD_{SSPD}[j, i] \leftarrow MatD_{SSPD}[i, j]$
  end for
end for
return $MatD_{SSPD}$

end function

Además de la optimización en el computo de una matriz triangular, uso de matrices a manera de registros de memoria y de optimizaciones menores realizadas en los algoritmos necesarios para obtener la distancia D_{SSPD}, se utilizó computación paralela en el llamado a la función "D_SSPD", reduciendo el tiempo de ejecución del algoritmo.

3.3.1.7. Selección de fibras

En este procedimiento se hace la suma por fila de la matriz de distancias D_{SSPD}. Este resultado refleja la suma de las distancias de una fibra respecto a todas las otras, lo que significa que los menores valores corresponden a las fibras que están más cercanas al resto y los valores más altos son las fibras más alejadas del resto.

Finalmente se seleccionan las fibras cuyas distancias no sobrepasen un percentil establecido y se descartan aquellas que posean una cantidad mayor.

3.3.2. Algoritmo de Multidimensional Scaling

En el área de Machine Learning, a veces son requeridas técnicas para reducir el tamaño de los datos de entrada. En el presente trabajo se utiliza este algoritmo para transformas las fibras (curvas) que de un espacio tridimensional a curvas un plano bidimensional. De esta manera, al obtener la distancia D_{SSPD}, se reduce la cantidad de cómputo necesaria por cada fibra.

En Python existe la biblioteca de sklearn la cual contiene los algoritmos de MDS e ISOMAP. Al no encontrar una implementación de MDS, se desarrolló una implementación del MDS clásico o PCoA que realiza los procedimientos descritos en la sección 2.2.3.

3.3.2.1. Computo de la Matriz de distancia al cuadrado

En este proceso se calcula la distancia entre todos los puntos que conforman la fibra. Al tratarse de una matriz de distancias, se realiza la misma optimización que la matriz de distancias D_{SSPD}. El algoritmo comienza con una matriz inicializada en 0, y se computan los datos del triángulo superior de la matriz, para reutilizar sus valores de forma simétrica en el triángulo inferior. Como la matriz está inicializada en 0, la diagonal no se modifica.

En el caso de elevar la matriz al cuadrado, se elevan al cuadrado solo los elementos de la sección triangular superior y se reutilizan sus valores en la sección triangular inferior.

function MATRIS_DISTANCIA(Tray[ ][ ], N)

Entradas: ***Tray*** *corresponde a una matriz que contiene las coordenadas de la curva en el espacio dimensional mayor y* ***N*** *corresponde a la cantidad de puntos que define a la curva.*

$D \leftarrow [0]_{N \times N}$
for $i \leftarrow 1, N$ **do**
for $j \leftarrow i + 1, N$ **do**
$D[i, j] \leftarrow distancia\_euclidea(Tray[i], Tray[j])$
$D[j, i] \leftarrow D[i, j]$
end for
end for
return $Dist\_Mat$
end function

function DISTANCIA_CUADRADO(D[ ][ ], N)

Entradas: ***D*** *corresponde a una matriz que contiene las distancias entre los pares de puntos de 1 curva. Esta matriz será cuadrada, simétrica y de diagonal 0 y* ***N*** *corresponde a la cantidad de puntos que define a la curva*

$D^{[2]} \leftarrow D$
for $i \leftarrow 1, N$ **do**
for $j \leftarrow i + 1, N$ **do**
$D^{[2]}[i][j] \leftarrow D[i][j] \cdot D[i][j]$
$D^{[2]}[j][i] \leftarrow D[i][j]$
end for
end forreturn $D^{[2]}$
end function

3.3.2.2. Doble centrado de la matriz de Distancia

En este proceso se calcula la matriz B, la cual es el resultado del doble centrado de la matriz de distancia al cuadrado $-\frac{1}{2} C\, D^{(2)}\, C$, cuyo proceso fue detallado en la sección 2.2.3.

El proceso de doble centrado consiste en multiplicar la matriz $D^{(2)}$ por C a la izquierda y a la derecha. La multiplicación $C\, D^{(2)}$ es el equivalente a restar a cada elemento el valor promedio

de su columna correspondiente. Sucede de forma análoga con las filas al multiplicar $D^{(2)}\,C$.

El proceso para realizar esta multiplicación para una matriz $D^{(2)}$ de tamaño $n \times n$ es el siguiente:

- Se debe crear la matriz C correspondiente. Esta matriz solo tiene 2 valores, uno en su diagonal y otro en su parte triangular superior e inferior.

$$diag(C) = \frac{n-1}{n}$$

$$triang(C) = -\frac{1}{n}$$

- Se realiza el producto matricial $C\,D^{(2)}$.
- Se debe realizar el producto entre la matriz resultante del paso anterior, con la matriz de centrado $[C\,D^{(2)}]\,C$.
- Cada elemento de la matriz resultante se multiplica por $-\frac{1}{2}$.

El alto costo computacional de este procedimiento está asociado a las 2 multiplicaciones de matrices, por lo que se desarrolló un algoritmo alternativo evitando la multiplicación de matrices que conlleva al mismo resultado.

La multiplicación $C\,D^{(2)}$ es equivalente a restar el promedio por columna a cada elemento de $D^{(2)}$, por lo tanto se hace el cómputo del promedio de cada columna y se restará su valor a los elementos correspondientes a cada columna. Después se realizará el mismo procedimiento por filas.

function DOBLE_CENTRADO(D[ ][ ], N)

Entradas: ***D*** *Corresponde a una matriz que contiene las distancias al cuadrado entre los pares de puntos de una curva. Esta matriz será cuadrada, simétrica y de diagonal 0 y* ***N*** *corresponde a la cantidad de puntos que define a la curva*

$P_{col} \leftarrow [0]_{1\times N}$
$P_{fil} \leftarrow [0]_{1\times N}$
$B \leftarrow D$
for $i \leftarrow 1, N$ **do**
  for $j \leftarrow 1, N$ **do**
    $P_{col}[i] \leftarrow P_{col}[i] + B[j][i]$
  end for

```
        P_col[i] ← P_col[i]/N
        for j ← 1, N do
            B[j][i] ← B[j][i] − P_col[i]
        end for
    end for
    for i ← 1, N do
        for j ← 1, N do
            P_fil[i] ← P_fil[i] + B[i][j]
        end for
        P_fil[i] ← P_fil[i]/N
        for j ← 1, N do
            B[i][j] ← B[i][j] − P_fil[i]
        end for
    end for
    for i ← 1, N do
        for j ← 1, N do
            B[i][j] ← B[i][j] · (−0.5)
        end for
    end for
     return B
end function
```

En este caso para obtener una aproximación en el ahorro del costo computacional, se cuenta la cantidad de operaciones elementales realizadas ($O_{dbl.C}$). Se considera como una operación elemental la aplicación de la suma, resta, multiplicación y división entre 2 operandos. Si hay más de 2 operandos, se agrupan las operaciones en pares respetando las preferencias operacionales.

Multiplicar por la matriz de centrado, es realizar la multiplicación de 2 matrices cuadradas de tamaño N. Por como está definida la multiplicación matricial, para obtener un elemento de la matriz resultante se realizan N multiplicaciones y $N-1$ adiciones. Al ser N^2 elementos el total de operaciones para toda la matriz es es N^3 multiplicaciones y $N^2(N-1)$ adiciones, por lo tanto el número de operaciones en una multiplicación matricial es.

$$O_{Mult} = N^3 + N^2(N-1)$$

$$O_{Mult} = N^2(2N-1)$$

En el proceso de doble centrado se realizan 2 multiplicaciones matriciales (por la insquierda

y por la derecha) y una multiplicación escalar a cada elemento de la matriz resultante, siendo N^2 multiplicaciones. Por lo tanto el total de operaciones en el doble centrado es.

$$O_{dbl.C} = N^2(2N-1) + N^2(2N-1) + N^2$$

$$O_{dbl.C} = N^2(4N-1) \tag{3.4}$$

Al realizar el algoritmo optimizado, para obtener una fila o columna de la matriz resultante se requiere $N-1$ adiciones y 1 división, por lo que para computar la matriz resultante completa serán $N(N-1)$ multiplicaciones y N divisiones.

$$O_{Opt} = N(N-1) + N$$

$$O_{Opt} = N^2$$

De manera análoga al caso anterior, este procedimiento debe aplicarse 2 veces (por columnas y por filas) y posteriormente se debe multiplicar por un escalar N^2 veces. Por lo tanto el total de operaciones en el doble centrado optimizado es.

$$O_{dbl.C.Opt} = N^2 + N^2 + N^2$$

$$O_{dbl.C.Opt} = 3N^2 \tag{3.5}$$

Haciendo una proporción de los costes operacionales obtenidos (3.4 y 3.5) se puede obtener una cantidad aproximada del ahorro de costo computacional. En necesario resaltar que este es un cálculo aproximado y el ahorro de tiempo dependerá de la arquitectura de cada procesador para realizar determinadas operaciones.

$$\frac{O_{dbl.C.Opt}}{O_{dbl.C}} = \frac{3N^2}{N^2(4N-1)}$$
$$\frac{O_{dbl.C.Opt}}{O_{dbl.C}} = \frac{3}{4N-1}$$

En la proporción obtenida tendiendo el valor de N al infinito, se puede determinar que mientras mayor sea la cantidad de puntos, mas despreciable es la cantidad de operaciones realizada por el algoritmo optimizado en comparación a la multiplicación matricial.

$$lim_{N\to+\infty}\frac{O_{Opt}}{O_{Mult}} = lim_{N\to+\infty}\frac{3}{4N-1}$$

$$lim_{N\to+\infty}\frac{O_{Opt}}{O_{Mult}} = 0$$

Este trabajo se realiza con fibras de 21 puntos, por lo que el ahorro de cómputo es sobre el 96 %. En la tabla 3.1 se puede observar el coste computacional aproximado de ambos algoritmos y una comparación porcentual del ahorro de computo del algoritmo optimizado.

Puntos por fibra	Multiplicación de matrices	Algoritmo optimizado	Ahorro de cómputo
5	475	75	84.21 %
10	3 900	300	92.31 %
15	13 275	675	94.92 %
20	31 600	1 200	96.20 %
25	61 875	1 875	96.97 %
30	107 100	2 700	97.48 %

Tabla 3.1: Cantidad de operaciones elementales de $-\frac{1}{2}C\,D^{(2)}\,C$ en función de la cantidad de puntos de cada fibra.

3.3.2.3. Descomposición en valores y vectores propios

En la sección 2.2.3 se demostró que la matriz de coordenadas se obtiene a partir de la descomposición de B en sus valores y vectores propios.

$$X_m = Q_m \lambda_m^{\frac{1}{2}}$$

Considerando que $B = XX^T$, se concluye que B es una matriz simétrica real, por lo que se puede calcular los valores propios mediante el algoritmo de Jacobi o también conocido como método de diagonalización.

La base del método de Jacobi es el uso de una matriz de rotación P, la cual al multiplicar a la matriz B, hace nulo sus elementos no diagonales.

$$P = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \cdots & \cdots & \cdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & cos(\theta)_{ii} & \cdots & sen(\theta)_{ij} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \cdots & \ddots & \cdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & -sen(\theta)_{ji} & \cdots & cos(\theta)_{jj} & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \cdots & \cdots & \cdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Hacer la operación $P\,B\,P^T$ se le define realizar la rotación de B. Por cada rotación realizada se hacen nulos algunos términos no diagonales. De esta manera, el resultado de realizar suficientes rotaciones será una matriz diagonal, cuyos valores no nulos corresponden a los valores propios, con los cuales se pueden determinar sus correspondientes vectores propios.

En la aplicación del algoritmo usualmente el proceso tiende a ser muy largo o no alcanza a hacer nulos todos los elementos no diagonales, por lo que se debe establecer un criterio de detención del proceso iterativo. En el algoritmo clásico es establecer que el valor absoluto de algún elemento no diagonal de la matriz, no debe pasar cierto umbral establecido.

En este estudio se utilizó una adaptación de la implementación en C de este algoritmo diseñada por John Burkart [16]. A diferencia del algoritmo clásico, en este algoritmo se realizó las modificaciones propuestas por Rutishauser [17]. En esta modificación se aprovecha la propiedad simétrica y se realizan los cálculos solo la sección triangular superior, mientras que la sección triangular inferior se utiliza para recuperar la matriz original.

Al ser un proceso iterativo con criterio de detención a través de un umbral no se puede estimar con precisión su coste operacional, ya que dependerá del tamaño de la matriz y el valor de los elementos que compongan a esta. En "Handbook Series Linear Algebra", escrito por Rutishauser [17], se detalla el algoritmo de Jacobi modificado. El método de Jacobi convencional es el siguiente.

1. Se debe determinar los índices i y j del mayor elemento presente fuera de la diagonal. Al ser una matriz simétrica, basca con buscar por solo 1 de las secciones triangulares.

2. Determinado los índices i y j, se procede a calcular θ.

$$\theta = \frac{1}{2}\, tan^{-1}\left(\frac{2\, b_{ij}}{b_{ii} - b_{jj}}\right)$$

3. Obtenido θ calcula el valor de $cos(\theta)$ y $sin(\theta)$.

4. Obtenido el valor de $cos(\theta)$ y $sen(\theta)$ se realiza el producto matricial entre $P\,B$. Ya que la matriz P es similar a la matriz identidad, se puede notar que el resultado del producto solo modificará las filas i y j de la matriz B.

$$P\,B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{n1} \\ b_{21} & b_{22} & \cdots & b_{n2} \\ \vdots & \vdots & \cdots & \vdots \\ c\, b_{i1} + s\, b_{j1} & c\, b_{i2} + s\, b_{j2} & \cdots & cc\, b_{in} + s\, b_{jn} \\ \vdots & \vdots & \cdots & \vdots \\ c\, b_{j1} - s\, b_{i1} & c\, b_{j2} - s\, b_{i2} & \cdots & cc\, b_{jn} - s\, b_{in} \\ \vdots & \vdots & \cdots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c = cos(\theta) \qquad s = sen(\theta)$$

5. Obtenido $P\,B$ se realiza la el producto matricial $(P\,B)P^T$ cuya estructua es similar al producto anterior, pero modifica las columnas i y j.

Finalmente el procedimiento completo de la implementación del algoritmo Multidimensional Scaling para una curva es el siguiente:

function MDS(Tray[ ][ ], N)

Entradas: ***Tray*** *corresponde a una matriz que contiene las coordenadas de la curva en el espacio dimensional mayor y* ***N*** *corresponde a la cantidad de puntos que define a la curva.*

$D \leftarrow Matris\_Distancia(Tray, N)$

$D^{[2]} \leftarrow Distancia\_Cuadrado(D, N)$

$B \leftarrow Doble\_Centrado(D^{[2]}, N)$

$X \leftarrow Vectores\_propios(B, N)$

$Q \leftarrow Valores\_Propios(B, N)$

$X \leftarrow X \cdot \sqrt{Q}$

return X

end function

3.3.3. Algoritmo de ISOMAP

Este algoritmo hace uso de Multidimensional Scaling para generar una representación bidimensional de un conjunto de datos. La diferencia es el uso de las distancias geodésicas entre los puntos de la fibra, con el fin de mantener la geometría intrínseca de esta.

Para conocer las distancias geodésicas, se debe generar un grafo que represente los puntos cercanos de la fibra y calcular las trayectorias más cortas entre puntos distantes.

3.3.3.1. Algoritmo de K-Nearest Neighboor

Obtenida la matriz D de distancias entre los puntos de la fibra, se debe determinar los puntos vecinos. En este caso se utiliza un algoritmo de ordenamiento por selección (del inglés *Selection Sort*, SSort) modificado.

Para ordenar un vector de forma ascendente, el algoritmo SSort recorre un vector hasta llegar al menor valor y lo intercambia con el que esta en la primera posición, después realiza lo mismo con el segundo menor valor y así sucesivamente hasta ordenar el vector completamente.

La modificación realizada es usar el parámetro k para establecer cuantas veces máximo se repite el proceso de intercambio, de manera de ordenar solo los primeros k elementos de interés. Junto con estos intercambios, se hacen los intercambios de los respectivos índices de cada valor. De esta manera se tiene una lista con k índices correspondientes a los menores k elementos de un vector.

Cada elemento d_{ij} de la matriz D representa la distancia del punto i al punto j en una fibra. Por lo tanto, los k menores elementos de una fila i, serán los k puntos más cercanos al punto i.

Este proceso se repite por cada fila de la matriz D, obteniendo como resultado una matriz de tamaño $n \times k$ donde cada fila corresponde a los índices de los menores k elementos de D, siendo estos los índices de los puntos vecinos.

3.3.3.2. Matriz de grafo

Una vez que se tiene un listado de los puntos vecinos, se procede a diseñar un grafo. Gráficamente un grafo corresponde a un conjunto de nodos unidos por aristas. Cuando estas aristas

poseen alguna magnitud, se dice que es un grafo con peso. Como el grafo solo representa conexiones, la longitud de las aristas no tienen restricciones geométricas, por ejemplo se pueden unir 3 nodos cuyas 3 longitudes de aristas no formen un triángulo.

A nivel matricial, un grafo con peso esta representado por una matriz G la cual tiene como elementos las distancias de los puntos vecinos (la longitud de las aristas) y un valor muy alto (usualmente denotado como infinito) para los puntos que no están conectados.

Teniendo los índices j de los puntos más cercanos a cada punto i se puede generar una matriz G la cual tiene solo los valores d_{ij} y d_{ji} de la matriz D, mientras que el resto de los valores son reemplazados por infinito.

3.3.3.3. Matriz de distancia geodésicas

Obtenido el grafo, se calcula la ruta más corta entre todos los pares de puntos. En este caso se utiliza el algoritmo de Dijkstra el cual devuelve la longitud del trayecto más corto entre 2 vértices dentro de un grafo. El algoritmo es el siguiente:

1. Se marcan todos los nodos del grafo como “no visitados”.
2. Se le asigna a cada nodo una “distancia tentativa” cuyo valor inicial es infinito para todos los nodos, excepto el primero, que será 0, representando la distancia consigo mismo. El primer nodo se marca como nodo actual.
3. Desde el nodo actual se calcula la distancia a sus vecinos y se suma a la distancia acumulada actual. Si alguna de las sumas tiene una una distancia menor a la distancia tentativa se actualiza el nuevo valor de la distancia tentativa correspondiente a dicho nodo, en el caso contrario, se mantiene el valor.
4. Una vez realizado el análisis anterior con todos los vecinos se marca el nodo actual como “visitado”, de manera que no vuelva a ser analizado por el algoritmo.
5. Se selecciona un nuevo “nodo actual” que corresponde al nodo con menor “distancia tentativa” y se repite el proceso desde el paso 3. Este paso asegura encontrar la menor ruta hasta el “nodo actual”.
6. El algoritmo se detiene cuando todos los nodos fueron visitados.

Cabe destacar que este algoritmo funciona cuando la matriz del grafo G esta definida correctamente. La matriz del grafo es generada en función del valor de k. Un valor muy bajo de k puede generar puntos vecinos aislados, por lo que la matriz G describe más de un grafo y el algoritmo de Dijkstra no podría encontrar una solución.

Si la matriz G está mal definida y describe más de un grafo, al generar el arreglo de las "distancias tentativas" en algún momento del proceso, tendrán todos los valores de este en infinito, sin embargo quedan nodos no vecinos por visitar, pero al ser todas las distancias tentativas infinitas no se puede seleccionar un nuevo nodo y el algoritmo se detiene en un ciclo infinito.

Para solucionar este problema, se diseñó un algoritmo que detecta cuando todas las distancias tentativas son infinitas. En caso de este evento, el algoritmo de Dijkstra se detiene y se ejecuta nuevamente el algoritmo de KNN con un valor más alto de k para forzar la generación de un único grafo por matriz.

Una vez completado el algoritmo de la ruta más corta, se genera la matriz G_D la cual tiene la longitud de las trayectorias más cortas entre todos los pares de puntos vecinos que conforman la fibra, lo que corresponde a una aproximación de la Matriz de Distancias geodésicas entre los puntos de la fibra.

3.3.3.4. Aplicación de Algoritmo de MDS

Una vez obtenida la matriz G_D se le aplica el algoritmo de MDS clásico para generar una representación bidimensional de cada fibra.

Debido al proceso de doble centrado, se genera una representación de la fibra en un plano bidimensional centrado en el origen. El diseño del algoritmo de distancias D_{SSPD} permite medir similitudes entre curvas en base a su morfología y proximidad. Al realizar la transformación bidimensional, todas las fibras estarán próximas entre si, con distinta morfología. En este caso distancia D_{SSPD} no mide la distancias entre fibras, mas bien miden su similitud respecto a todas las fibras del fascículo.

Además de aplicar el algoritmo de ISOMAP para transformar las fibras a un plano bidimensional, también se aplicó la implementación de MDS directamente sobre las fibras, obteniendo otros resultados. Las diferencia en ambos procedimientos es que en el plano bidimensional el algoritmo de ISOMAP estira las fibras, mientras que en MDS se obtiene una proyección.

4. Resultados

Se realizan la ejecución de los distintos códigos de filtrado y se contrastan los resultados con el algoritmo original diseñado por Mendoza. Para la ejecución de los scripts de Python se ejecutó Python 3.8.10 a través de "Windows subsystem for Linux", un sistema que permite ejecutar la consola de Linux de forma nativa en Windows, en este caso la consola corresponde a un Sistema Ubuntu 20.04. Los códigos fuentes desarrollados en este trabajo se encuentran disponibles en https://github.com/mr-pineda/MT_UdeC_filtrado_fibras

Con el fin de medir el desempeño de los algoritmos, se dividieron los fascículos en 5 grupos que contienen distintas cantidades de fibras (500, 1000, 2000, 4000, 7600). Cada grupo contiene 10 fascículos sin filtrar (archivos .bundles y sus correspondientes .bundlesdata). Se midieron sus tiempos de ejecución y se estableció un tiempo promedio por cantidad de fibras. De esta forma, por cada implementación se tiene 5 tiempos de ejecución diferentes.

En la ejecución del código se determina que el proceso de mayor costo computacional es la obtención de la matriz de distancias D_{SSPD}, por ello, para medir el tiempo de ejecución de algoritmo de Python de 2000 fibras o más, se ejecuta su código de forma parcial imprimiendo el tiempo de cómputo de una fila de la matriz y el tiempo promedio por fila. Como las operaciones realizadas son las mismas en cada línea, se multiplica este tiempo promedio por el total de fibras del fascículo, obteniendo así el tiempo aproximado que requiere el algoritmo de Python para realizar el filtrado.

Se realizaron 7 implementaciones de C. La primera corresponde a una transcripción del algoritmo de Python sin realizar optimizaciones en el cálculo de la matriz D_{SSPD}, con el objetivo de verificar cuanto se acelera el el algoritmo por el cambio de lenguaje. Las 6 implementaciones restantes son del algoritmo optimizado con distintas cantidades de hebras del procesador utilizadas. El procesador del equipo utilizado en este trabajo es multi-núcleo con una hebra por núcleo, por lo tanto, el uso de múltiples hebras corresponde al uso de múltiples núcleos.

El equipo utilizado para la ejecución de este trabajo corresponde al equipo otorgado por el Proyecto FONDECYT 1190701 el cual posee las siguientes características:

- CPU: intel Core i5-8600K

    - Frecuencia base: 3.6 GHz
    - 6 núcleos y 6 hebras
- RAM: 16 Gb. (2666 MHz)
- Disco Duro: Toshiba HDWH110
    - Tipo: HDD (magnético)
    - Capacidad: 1 Tb

Se define una métrica llamada aceleración $acc(ALG)$, la cual es la relación entre los tiempos de ejecución de las implementaciones A y B (T_B y T_A) de cierto algoritmo ALG.

$$\frac{T_B}{T_A} = acc(ALG)_{[A][B]}$$

Donde $acc(ALG)_{[A][B]}$ es un índice adimensional que indica cuántas veces es la implementación A más rápida que B. Si $acc(ALG)_{[A][B]}$ es mayor a 1, significa que A es más rápido que B, si acc_{AB} es igual a 1 A y B tienen la misma velocidad y si $acc(ALG)_{[A][B]}$ es menor a 1, entonces A es más lento que B.

4.1. Ejecución de Algoritmo de distancia SSPD en un espacio tridimensional

En la tabla 4.1 se pueden observar las diferencias de tiempo de ejecución entre el algoritmo implementado en Python, la implementación no optimizada de C y las implementaciones optimizadas de C en programación secuencial y paralela.

Para determinar cuanto se acelera el algoritmo por el cambio de lenguaje, se normaliza los tiempos de ejecución de la implementación de C sin optimizar por los tiempos de ejecución de la implementación en Python ($acc(SSPD\ 3D)_{[C\ no\ opt.][Python]}$). A partir de lo mencionado, se obtiene la tabla 4.2.

Teniendo como base los tiempos de computo de la implementación en C no optimizada, se obtiene la aceleración de las implementaciones de C optimizadas, en programación secuencial ($acc(SSPD\ 3D)_{[C\ secuencial][C\ no\ opt.]}$) y paralela ($acc(SSPD\ 3D)_{[C\ x\ hebras][C\ no\ opt.]}$), obteniendo así la tabla 4.3

Implementación	Número aproximado de Fibras				
	500	**1 000**	**2 000**	**4 000**	**7 600**
Python	17m 50s	1h 11m	4h 40m	16h 3m	2d 16h
C (no opt.)	1.3s	5.4s	20.4s	1m 20s	4m 57s
C (secuencial)	0.65s	2.6s	10.2s	40s	2m 30s
C (2 hebras)	0.5s	2s	7.9s	29s	1m 53s
C (3 hebras)	0.4s	1.5s	5.7s	21s	1m 24s
C (4 hebras)	0.3s	1.2s	4.5s	17s	1m 7s
C (5 hebras)	0.27s	1.04s	3.89s	14s	57s
C (6 hebras)	0.24s	0.89s	3.34s	12s	49s

Tabla 4.1: Tiempos de filtrado por SSPD 3D para diferentes cantidades de fibras.

Implementación	Número aproximado de Fibras				
	500	**1 000**	**2 000**	**4 000**	**7 600**
C (no opt.)	785	789	824	722	786

Tabla 4.2: Aceleración filtrado por "SSPD 3D" de C sin optimizar respecto a Python.

Implementación	Número aproximado de Fibras				
	500	**1 000**	**2 000**	**4 000**	**7 600**
C (secuencial)	2.00	2.08	2.00	2.00	1.98
C (2 hebras)	2.6	2.7	2.58	2.76	2.63
C (3 hebras)	3.25	3.60	3.58	3.81	3.54
C (4 hebras)	4.33	4.50	4.53	4.71	4.43
C (5 hebras)	4.76	5.19	5.24	5.46	5.23
C (6 hebras)	5.42	6.02	6.10	6.38	6.07

Tabla 4.3: Aceleración filtrado por "SSPD 3D" de C optimizado respecto a C sin optimizar.

De la tabla 4.3, se observa que la implementación optimizada el es doble de rápido que la implementación sin optimizar, corroborando el ahorro de computo estimado en la ecuación 3.3 de

la sección 3.3.1.6. De esto, se deduce que el algoritmo optimizado en programación secuencial es en promedio 1500 veces más rápido que su implementación en Python, lo que se puede observar en la gráfica 4.1. Además se observa una tendencia de aumento lineal de la aceleración en función de la cantidad de hebras utilizadas.

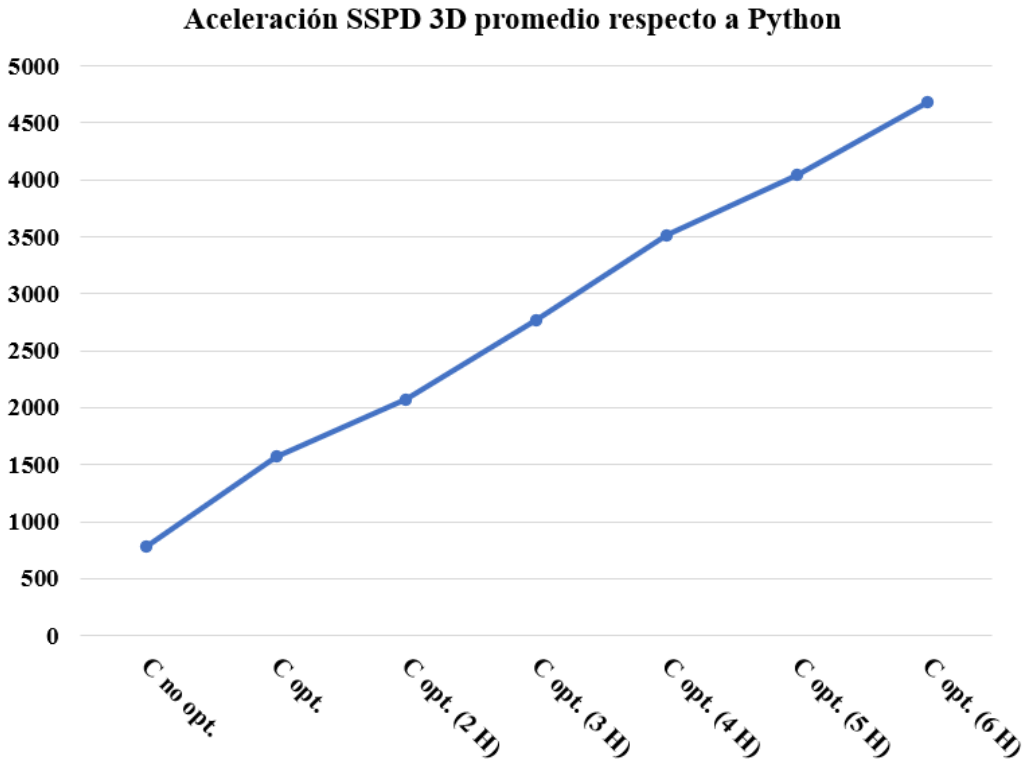


Fig. 4.1: Aceleración promedio de diversas implementaciones de SSPD 3D en C respecto a Python.

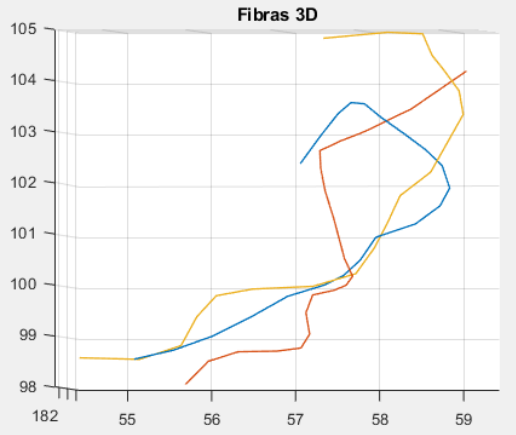
Aplicando una regresión potencial al algoritmo de C optimizado en conjunto con los datos obtenidos de las aceleraciones, se puede obtener una función que estime el tiempo de filtrado en segundos del algoritmo en función del número de fibras (f) y hebras (h) utilizadas.

$$T_{SSPD\ 3D}(f,h) = \frac{2.69 \cdot 10^{-6}}{1.25^{h-1}} \cdot f^{1.99} \quad [segundos]$$

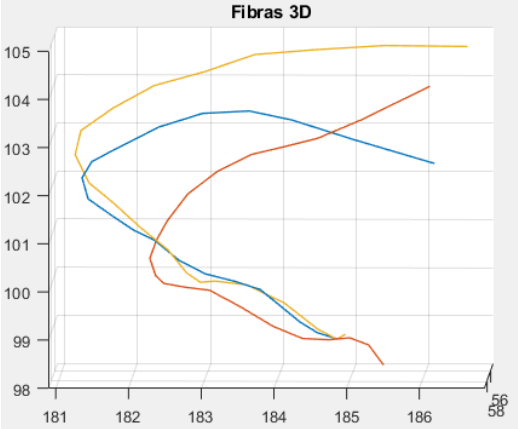
4.2. Ejecución de algoritmos de reducción dimensional

En el presente proyecto, a modo de contrastar resultados, se realiza la reducción dimensional mediante 2 algoritmos MDS y ISOMAP, obteniendo diferentes representaciones bidimensionales

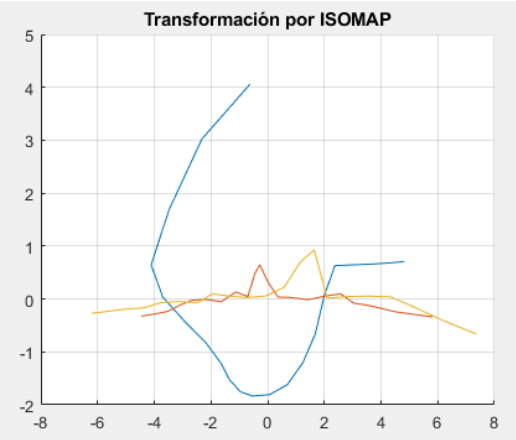
de las mismas fibras, las cuales se pueden apreciar en la imagen 4.2. Recordar que el algoritmo de ISOMAP "estira" los datos en el plano bidimensional, mientras que el algoritmo de MDS los "proyecta". Por esto, es que el algoritmo de MDS conserva más la curvatura original de las fibras en el plano bidimensional.



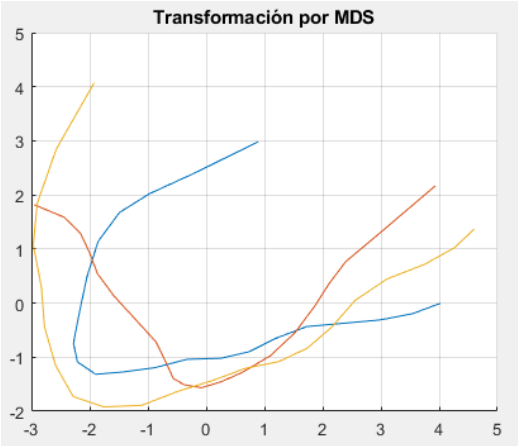
(a) Fibras en espacio tridimensional (vista 1)



(b) Fibras en espacio tridimensional (vista 2)



(c) Transformación por ISOMAP



(d) Trandformación por MDS

Fig. 4.2: Comparación de técnicas de reducción dimensional.

En relación al cómputo de ambos algoritmos, la implementación diseñada no es paralelizada, por lo que el tiempo de cómputo de las reducciones dimensionales es independiente del número de hebras utilizadas y dependerá solamente de la cantidad de fibras de cada fascículo, siendo el algoritmo de MDS el de menor tiempo de cómputo.

Implementación	Número aproximado de Fibras				
	500	**1 000**	**2 000**	**4 000**	**7 600**
Python (ISOMAP)	0.875s	1.604s	2.830s	5.338s	10.237s
C (ISOMAP)	0.087s	0.172s	0.344s	0.664s	1.328s
C (MDS)	0.074s	0.15s	0.298s	0.586	1.141s

Tabla 4.4: Tiempos de cómputo de algoritmos de reducción dimensional.

4.3. Ejecución de Algoritmo SSPD en espacio bidimensional

En este caso se utilizan 2 algoritmos de reducción bidimensional, ISOMAP y MDS con el fin de contrastar resultados. Para ambos casos se utiliza el mismo algoritmo de filtrado (SSPD 2D), por lo que el tiempo de de ejecución del algoritmo de filtrado es independiente de la técnica de reducción dimensional utilizada. En base a lo mencionado, la información mostrada en la tabla 4.5, corresponde a los tiempos de filtrado sin considerar el tiempo de reducción dimensional.

Implementación	Número aproximado de Fibras				
	500	**1 000**	**2 000**	**4 000**	**7 600**
Python	10m 13s	41m	2h 29m	9h 43m	1d 13h
C (no opt.)	1.15s	4.56s	17.53s	1m 11s	4m 4s
C (secuencial)	0.58s	2.30s	8.75s	35,65s	2m 4s
C (2 hebras)	0.44s	1.75s	6.63s	26,03s	1m 33s
C (3 hebras)	0.33s	1.30s	4.94s	18,60s	1m 9s
C (4 hebras)	0.26s	1.04s	3.90s	15.37s	54.56s
C (5 hebras)	0.22s	0.86s	3.23s	12.74s	45.61s
C (6 hebras)	0.19s	0.77s	2.80s	10.87s	39s

Tabla 4.5: Tiempos de Algoritmo de filtrado por SSPD 2D para distintas cantidades de fibra.

La notable reducción de tiempo en el filtrado bidimensional en Python, reduce la relación de tiempos de ejecución entre la implementación de Python y C sin optimizar , lo que conlleva a tener una menor aceleración de C respecto a Python ($acc(SSPD\ 2D)_{[C\ no\ opt.][Python]}$). Esto

se ve reflejado en la tabla 4.6. Por otro lado, obteniendo las aceleraciones de las implementaciones optimizadas de C respecto a la de C sin optimizar ($acc(SSPD\ 2D)_{[C\ secuencial][C\ no\ opt.]}$ y $acc(SSPD\ 2D)_{[C\ x\ hebras][C\ no\ opt.]}$), se obtiene la tabla 4.7, la cual muestra resultados similares a la obtenida en el filtrado por SSPD 3D (tabla 4.3).

Implementación	**Número aproximado de Fibras**				
	500	**1 000**	**2 000**	**4 000**	**7 600**
C (no opt.)	532	539	511	492	553

Tabla 4.6: Aceleración de filtrado por SSPD 2D de C sin optimizar respecto a Python.

Implementación	**Número aproximado de Fibras**				
	500	**1 000**	**2 000**	**4 000**	**7 600**
C (secuencial)	2.00	2.08	2.00	2.00	1.98
C (2 hebras)	2.60	2.70	2.58	2.76	2.63
C (3 hebras)	3.25	3.60	3.58	3.81	3.54
C (4 hebras)	4.33	4.50	4.53	4.71	4.43
C (5 hebras)	4.76	5.19	5.24	5.46	5.23
C (6 hebras)	5.42	6.02	6.10	6.38	6.07

Tabla 4.7: Aceleración filtrado por SSPD 2D de C optimizado respecto a C sin optimizar.

Calculando las aceleraciones promedio de las diversas implementaciones de C se obtiene la gráfica 4.3, la cual muestra una tendencia similar a la gráfica 4.1, pero en menor magnitud.

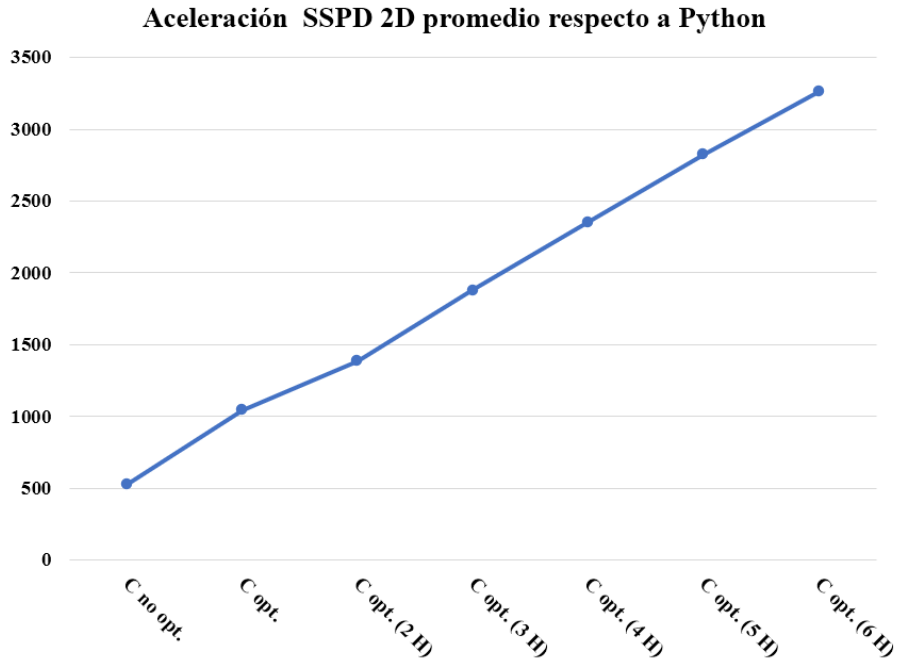


Fig. 4.3: Aceleración promedio de SSPD 2D en diversas implementaciones de C respecto a Python.

Aplicando una regresión potencial a los tiempos del algoritmo SSPD bidimensional en conjunto con una regresión lineal en los tiempos de ISOMAP, se obtiene una función que estime el tiempo de filtrado del algoritmo en función del número de fibras (f) y hebras (h) utilizadas.

$$T_{SSPD\ 2D}(f,h) = \frac{2.713 \cdot 10^{-6}}{1.26^{h-1}} \cdot f^{1.974} + 2 \cdot 10^{-4} f - 7.1 \cdot 10^{-3} \ \ [segundos]$$

4.4. Comparación entre filtro tridimensional y bidimensional

Calculando la relación entre los tiempos de ejecución del filtrado tridimensional y el procedimiento de reducción dimensional en conjunto con el filtrado bidimensional, se puede obtener la tabla 4.8 en la cual se observa el porcentaje de ahorro de tiempo de cómputo al usar el filtrado SSPD bidimensional en conjunto con el algoritmo de ISOMAP. De esta relación, se puede observar que en Python, la implementación de SSPD 2D tiene un ahorro de tiempo promedio del 42 %.

Implementación	Número aproximado de Fibras				
	500	1 000	2 000	4 000	7 600
Python	40 %	42 %	47 %	39 %	42 %
C (no opt.)	5 %	12 %	12 %	10 %	17 %
C (secuencial)	-2 %	5 %	11 %	9 %	17 %
C (2 hebras)	-6 %	4 %	12 %	8 %	16 %
C (3 hebras)	-4 %	2 %	7 %	8 %	16 %
C (4 hebras)	-16 %	-1 %	6 %	6 %	17 %
C (5 hebras)	-13 %	1 %	8 %	8 %	17 %
C (6 hebras)	-16 %	-4 %	6 %	8 %	18 %

Tabla 4.8: Porcentaje de ahorro de tiempo de ejecución de ISOMAP en comparación a SSPD 3D.

Por otro lado, en C se observa que a mayor cantidad de hebras y menor cantidad de fibras por fascículo, el ahorro de tiempo de cómputo disminuye, llegando a valores negativos, lo que significa que dicha implementación para un fascículo con esa cantidad de fibras es más lenta que la misma aplicación en el caso tridimensional.

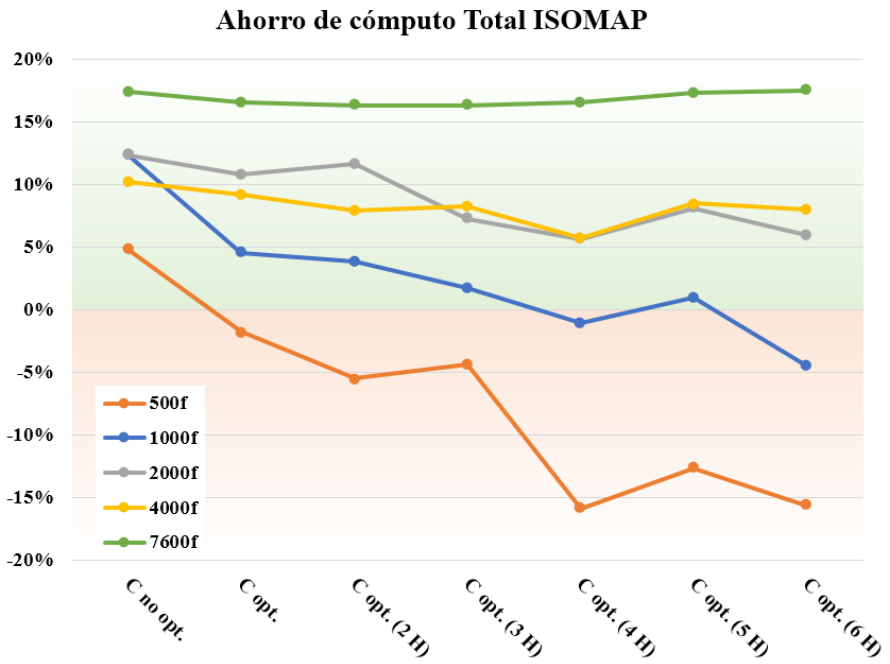


Fig. 4.4: Porcentaje de ahorro de tiempo en algoritmo de ISOMAP.

En la gráfica 4.4 se observa el ahorro de tiempo de cómputo al usar el algoritmo de ISOMAP en conjunto con el algoritmo de SSPD bidimensional en comparación al tiempo de cómputo del algoritmo SSPD tridimensional. En el algoritmo de MDS se obtienen resultado similares.

Esta deficiencia en el ahorro de tiempo ocurre debido a que el proceso de reducción dimensional no es paralelizado. Como el tiempo de reducción dimensional no disminuye en función del número de hebras, el tiempo total será el tiempo de filtrado bidimensional (SSPD 2D) que disminuye en función de las hebras más un tiempo constante correspondiente a la reducción dimensional, mientras que en el caso tridimensional, al existir solo el algoritmo de filtrado (SSPD 3D), su tiempo disminuye de forma proporcional al número de núcleos.

Calculando el porcentaje de tiempo de la reducción dimensional en comparación al tiempo total requerido por la reducción dimensional en conjunto con el filtrado, se observa que a mayor cantidad hebras utilizadas, mayor es la proporción de tiempo en la reducción dimensional, siendo el caso más extremo el filtrado de 500 fibras usando las 6 hebras, con fracción tiempo de cómputo de la reducción dimensional sobre el 30 % del tiempo total en el caso de ISOMAP y sobre el 27 % en el caso de MDS.

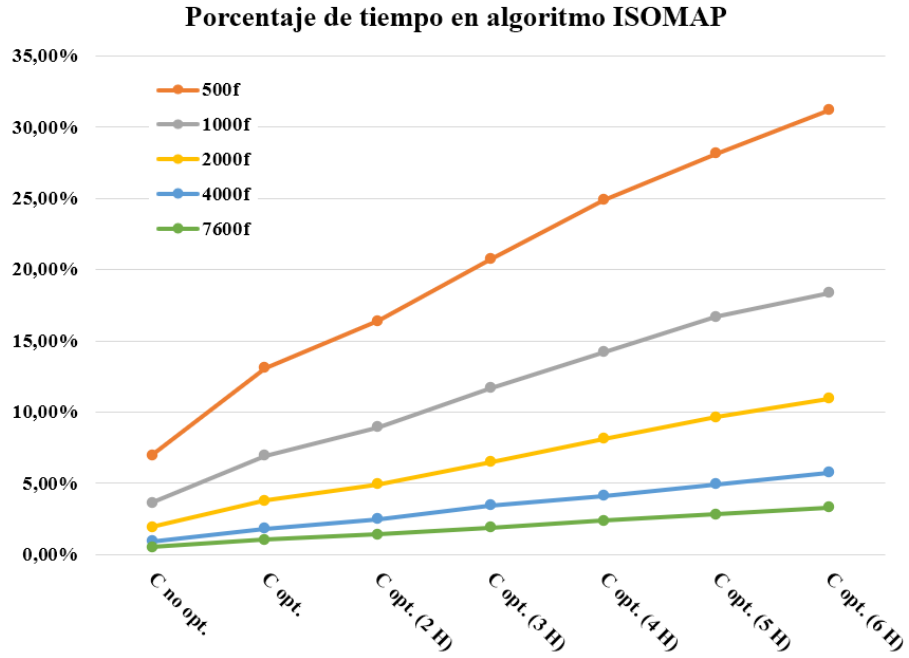


Fig. 4.5: Porcentaje de tiempo ocupado por la reducción dimensional a través ISOMAP.

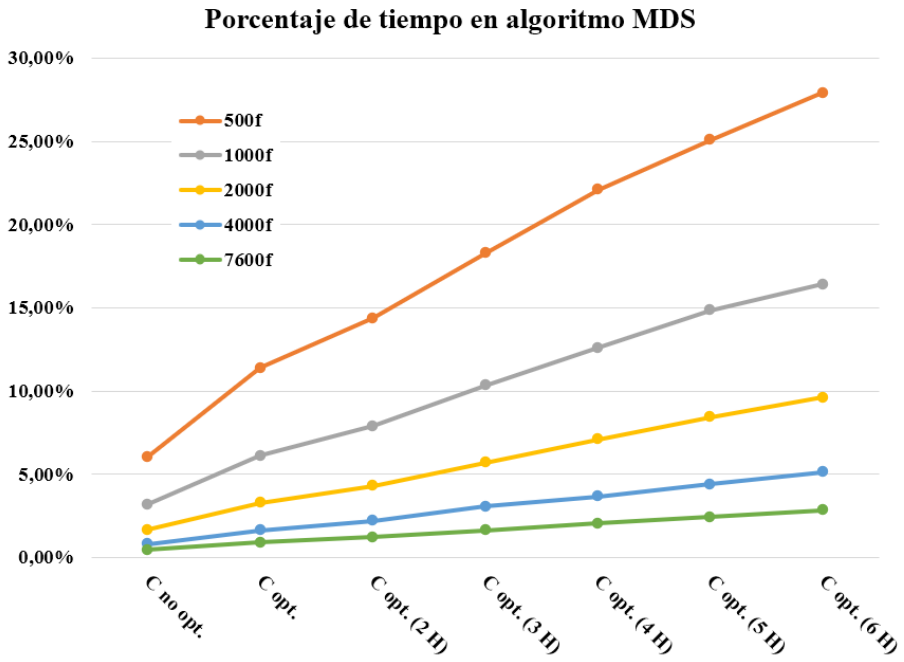
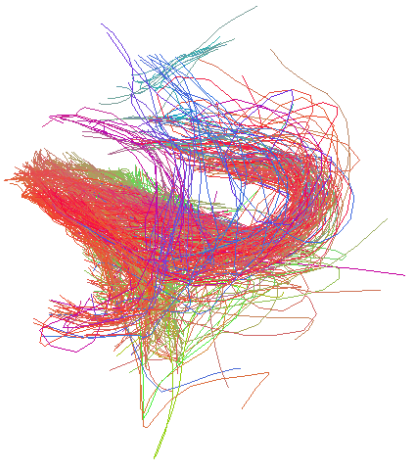


Fig. 4.6: Porcentaje de tiempo ocupado por la reducción dimensional a través MDS.

4.5. Visualización de resultados

En la figura 4.7 se pueden observar los resultados de cada técnica, observando que existen leves diferencias entre el algoritmo de Python y el algoritmo de C. A través de un algoritmo de comparación entre los fascículos obtenidos por ambas implementaciones (Python y C), se determina que existe una diferencia cercana al 2 % en la cantidad de fibras filtradas. Esta desigualdad, se atribuye a una diferencia en el cómputo del percentil de distancias. En el algoritmo de Python se utiliza la función *percentile* de la biblioteca *numpy* para computar los percentiles de los datos. Por defecto, esta función realiza un proceso de interpolación de los percentiles obtenidos, por lo que este puede ser diferente al percentil real, el cual fue implementado en el algoritmo en C.

Se puede observar que el filtrado por ISOMAP produce fascículos más ruidosos. Esto se debe a las transformaciones producidas por el algoritmo de MDS. Al trasladar todas las fibras a un único plano, se pierde información y además se generan cambios de la posición relativa original de la fibra. Como consecuencia, el algoritmo de MDS puede provocar que fibras que eran similares en el espacio tridimensional sean diferentes en el espacio bidimensional y vice versa.



(a) Fascículo sin filtrar.

(b) Filtrado por MDS (C)

(c) Filtrado por ISOMAP (Python)

(d) Filtrado por ISOMAP (C)

(e) Fltrado por SSPD 3D (Python)

(f) Filtrado por SSPD (C)

Fig. 4.7: Comparación de las distintas técnicas de filtrado en C y Pyhon

5. Conclusión

5.1. Discusión

De los tiempos de ejecución obtenidos, se puede notar que la implementación diseñada en C del algoritmo de filtrado en el espacio bidimensional es más rápida que el filtrado en el espacio tridimensional, pero genera fascículos más ruidosos.

Al comparar los tiempos de computo de ambos algoritmos de filtrado, el mayor ahorro ocurre en el filtrado del fascículo de mayor tamaño, siendo de 51 segundos en la implementación de C sin optimizar y 25 segundos en C optimizado utilizando la implementación secuencial. En el resto de la base de datos la disminución de tiempo será aún menor al poseer menores cantidades de fibras.

Al analizar los tiempos de filtrado en conjunto con el tiempo utilizado en la reducción dimensional, se observa que a mayor cantidad de hebras utilizadas, el proceso de reducción dimensional adquiere un mayor porcentaje de tiempo en relación al proceso de filtrado, provocando que en C, en programación paralela, el filtrado bidimensional demore más que el filtrado tridimensional en los fascículos con menores cantidades de hebras

En vista de lo mencionado, la disminución del tiempo de cómputo obtenida en la implementación diseñada en c de filtrado bidimensional, es despreciable en relación a la homogeneidad o calidad de los fascículos obtenidos mediante el filtro de fibras en el espacio tridimensional. Además al trabajar con múltiples fascículos, se desconoce la cantidad de hebras que estos pueden tener, por lo que si en un conjunto de datos existen múltiples fascículos pequeños, el filtrado bidimensional requerirá más tiempo de cómputo.

5.2. Conclusiones

Se cumplió el objetivo general, logrando generar un programa de filtrado de fibras en C el cual es más rápido que su implementación en Python. En su versión optimizada usando computación secuencial, este es aproximadamente 1000 veces más rápido que su versión de

Python en el filtrado bidimensional y 1500 veces más rápido en el filtrado tridimensional.

Los resultados obtenidos en programación multi-hebra muestran una tendencia lineal de aumento de la velocidad en relación a la cantidad de hebras utilizadas, sin embargo, el equipo utilizado cuenta con un límite de 6 hebras, lo que limita realizar proyecciones más precisas de cómputo con una mayor cantidad de hebras, o cómo se comportaría el algoritmo usando procesadores con más de una hebra por núcleo.

Si bien Python es un lenguaje de programación menos eficiente en términos de costo computacional, su sintaxis simplificada, ayudó el desarrollo de una versión preliminar de los algoritmos de filtrado, siendo la versión de Python una base para desarrollar un algoritmo equivalente en otros lenguajes de programación de más bajo nivel.

5.3. Trabajo futuro

Si bien se cumple con el objetivo esperado, se puede estudiar en mayor profundidad la programación paralela y generar una implementación que aproveche aún más el paralelismo, como por ejemplo paralelizar los procesos de reducción dimensional, para disminuir la fracción de tiempo utilizada por estos procedimientos evitando que al trabajar con pocas fibras el filtrado bidimensional sea más lento que el filtrado tridimensional

A diferencia de Python, C tiene un control manual de los recursos de memoria utilizados, por lo que se puede estudiar mejor este algoritmo, de manera de hacer más eficiente el proceso de lectura de los datos.

En relación a la calidad de los fascículos obtenidos, la distancia SSPD en el espacio tridimensional, resulta ser una métrica adecuada para el filtrado de fibras, por lo que se podría estudiar su viabilidad para ser utilizado en un algoritmo de segmentación automática por similitud de fibras, la cual podría generar fascículos más homogéneos sin aplicar técnicas de filtrado.

En este trabajo se desarrolló una librería en C para facilitar la lectura y escritura de fibras (pares de archivos ".bundles" y ".bundlesdata"). Actualmente se continúa trabajando en el desarrollo de esta, con el objetivo facilitar futuros trabajos de este tipo en este lenguaje de medio nivel.

Glosario

dMRI Resonancia Magnética por Difusión (del inglés *diffusion Magnetic Resonance Imaging*) . . . 8

DTI Imagen por Tensor de Difusión (del inglés *Diffusion Tensor Imaging*) . . . 8

ISOMAP Mapeo Isométrico (del inglés *Isometric Mapping*) . . . 2

KNN K-Vecinos Más Cercanos (del inglés *K-Nearest Neighboor*) . . . 23

MDS Escalamiento Multidimensional (del inglés *Multidimensional Scaling*) . . . 5

MRI Imagen de Resonancia Magnética (del inglés *Magnetic Resonance Imaging*) . 1

PCoA Análisis de Coordenadas Principales (del inglés *Principal Coordinate Analisys*) 15

ROI Regiones de Interés (del inglés *Regions of Interest*) . . . 11

SPD Distancia Entre Trayectorias (del inglés *Segment-Path distance*) . . . 13

SSPD Distancia Simétrica Entre Trayectorias (del inglés *Symmetrized Segment-Path distance*) . . . 2

SWM Materia Blanca Superficial (del inglés *Superficial White Matter*) . . . 2

WM Materia Blanca (del inglés *White Matter*) . . . 7

Referencias

[1] C. N. Mendoza Sánchez, "Mejora de segmentación automática de fascículos de materia blanca superficial en datos de tractografías probabilísticas", Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Biomédico, 2021.

[2] (2021). "HCP Young Adult, 1200 subjects data release", [Online]. Available: https://www.humanconnectome.org/study/hcp-young-adult/document/1200-subjects-data-release.

[3] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction", *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[4] I. Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*, 3^{rd} ed. Springer Science & Business Media, 2017.

[5] P. Bese, D. Guillouet, J.-M. Loubes, and F. Royer, "Review and perspective for distance based clustering of veicle trayectories", *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3306–3317, 2016.

[6] C. E. Roman Godoy, "Estudio de fibras de la materia blanca superficial en datos del Proyecto Conectoma Humano", Ph.D. dissertation, Universidad de Concepción, 2021. [Online]. Available: http://repositorio.udec.cl/jspui/handle/11594/5382.

[7] A. Vázquez, N. López-López, A. Sánchez, J. Houenou, C. Poupon, J.-F. Mangin, C. Hernández, and P. Guevara, "FFClust: Fast fiber clustering for large tractography datasets for a detailed study of brain connectivity", *NeuroImage*, vol. 220, p. 117 070, 2020, ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2020.117070. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811920305565.

[8] I. M. Goicovish Dinamarca, "CEFFC: CUDA ENHANCED FAST FIBER CLUSTERING", 2020.

[9] T. I. Fortoul van der Goes, *Histología y Biología Celular*, 3^{rd} ed. McGraw Hill Medical, 2017.

[10] J. H. Martin, *Neuroanatomía texto y atlas*, 4^{th} ed. McGraw Hill Medical, 2013.

[11] A. H. Ropper, M. H. Samuels, J. P. Klein, and S. Prasad, *Adams y Victor. Principios de neurología*, 11^{th} ed. McGraw Hill Medical, 2019.

[12] A. Marzal Varó, P. García Sevilla, and I. Gracia Luengo, *Introducción a la Programación en Python 3*, 1^{st} ed. D - Universitat Jaume I. Servei de Comunicació i Publicacions, 2014.

[13] R. Pandey and N. Badal, "Understanding the role of parallel programming in multi-core processor based systems", ICACSE-2019, Apr. 2019. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3350311.

[14] (2021). "OpenMP 5.1 Reference Guide", [Online]. Available: https://www.openmp.org/wp-content/uploads/OpenMPRefCard-5.1-web.pdf.

[15] (2021). "numpy.linalg.norm (Documentación y Código Fuente de la biblioteca Numpy)", [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html.

[16] (2021). "JACOBI_EIGENVALUE - Eigenvalues and Eigenvectors of a Symmetric Matrix (C implementation)", [Online]. Available: https://people.sc.fsu.edu/~jburkardt/c_src/jacobi_eigenvalue/jacobi_eigenvalue.html.

[17] H. Rutishauser, "Handbook Series Linear Algebra The Jacobi Method for Real Symmetric Matrices *", 2005.