Universidad de Cantabria

Facultad
de
Ciencias

Aplicación de redes neuronales de tipo grafo a la reconstrucción de vértices en el detector CMS del LHC

(Application of Graph Neural Networks to Vertex Reconstruction in the CMS Detector at the LHC)

Trabajo de Fin de Grado
para acceder al

GRADO EN FÍSICA

Autor: David Abad Riaño

Director: Pablo Martínez Ruiz del Árbol

20 de mayo de 2025

A mi familia y amigos, por su apoyo constante a lo largo de la carrera. A mis padres y hermano, por confiar siempre en mí y tenderme la mano en los momentos más importantes. A mi director, por su dedicación y apoyo durante la realización de este trabajo. Y a las personas nuevas que estuvieron a mi lado en los momentos más complicados y dejaron en mí una marca indeleble. Sin su ayuda, tal vez no estaría escribiendo esto ahora.

Resumen

En este trabajo de fin de grado, se estudia la posible aplicación de redes neuronales de tipo grafo (*Graph neural networks*, GNN) como alternativa al algoritmo clásico de reconstrucción de vértices en el detector CMS del LHC (*Large Hadron Collider*). Para ello, se ha desarrollado un programa en Python que simula los eventos de colisión protón-protón producidos en el LHC con las condiciones esperadas para la fase de Alta Luminosidad del LHC (*high Luminosity LHC*, HL-LHC) como los efectos de *pile up*. Para el estudio, se ha confeccionado una versión simplificada del algoritmo utilizado en CMS para la reconstrucción de vértices primarios a partir del algoritmo *k-means* y se ha desarrollado una red neuronal de tipo grafo que ha sido entrenada para agrupar trazas en vértices. Se han comparado ambos procesos en términos de eficiencia de reconstrucción y se han discutido las ventajas e inconvenientes del uso de este tipo de redes para tareas de clasificación en física de partículas.

Abstract

In this bachelor's thesis, the possible application of graph neural networks (GNN) as an alternative to the classical vertex reconstruction algorithm in the CMS detector at the LHC (Large Hadron Collider) is studied. For this purpose, a Python program has been developed to simulate proton-proton collision events produced at the LHC under the expected conditions for the High Luminosity LHC (HL-LHC) phase, including pile-up effects. For the study, a simplified version of the algorithm used in CMS for primary vertex reconstruction based on the k-means clustering method has been implemented, and a graph neural network has been designed and trained to cluster tracks into vertices. Both processes have been compared in terms of reconstruction efficiency, and the advantages and limitations of using this type of network for classification tasks in particle physics have been discussed.

Índice

1 Introducción

El LHC (*Large Hadron Collider*) en el CERN es el mayor acelerador de partículas jamás construido. Se trata de un anillo en el que se aceleran protones a velocidades cercanas a la de la luz para hacerlos colisionar, generando diferentes partículas. Con el fin de recolectar los datos de las colisiones, se construyen los detectores CMS, ATLAS, LHCb y ALICE. Estos detectores se utilizan para múltiples objetivos, desde el estudio del Modelo Estándar hasta la búsqueda de candidatos a la materia oscura. CMS y ATLAS por ejemplo, se usan con los mismos objetivos pero no tienen la misma estructura de imanes y utilizan soluciones técnicas diferentes. Este trabajo se va a centrar en las técnicas utilizadas por CMS (*Compact Muon Solenoid*) para la reconstrucción de trazas generadas por las partículas resultantes de las colisiones [1].

En el LHC se aceleran paquetes de protones a velocidades cercanas a la de la luz y se producen hasta 1000 millones de colisiones por segundo. A esto se lo conoce como luminosidad. La luminosidad es una magnitud que indica el número de colisiones que tienen lugar en un detector por cm^2 por segundo. Sin embargo, sería deseable aumentar el valor de esta magnitud en un factor de 5 o 7 con el fin de tener más capacidad recolectora de datos. De esta forma, se podrán estudiar fenómenos físicos poco frecuentes y se podrán tomar mediciones más exactas de los procesos ya observados. Para conseguir este hito, se está trabajando en el HL LHC (*High Luminosity Large Hadron Collider*), un proyecto que busca precisamente aumentar la luminosidad del acelerador [2] .

No obstante, al aumentar el número de colisiones por segundo, se intensifica más un problema ya existente en LHC, el apilado de colisiones, o más conocido como *pile up*. Debido a que no se puede forzar una sola colisión protón-protón, se aceleran los paquetes de protones mencionados anteriormente, generando un número muy elevado de colisiones. El problema es que estas colisiones son, prácticamente, simultáneas. En el caso de este trabajo, se utilizan datos simulados que corresponderían con los tomados en el detector CMS por lo que todos los vértices resultantes de las colisiones tendrán lugar en una región específica [3].

Para tratar de reducir el *pile up*, no solo se tendrá en cuenta la posición espacial donde sucedan las colisiones y se formen vértices, sino que también se tendrá en cuenta el tiempo en el que suceden dichas colisiones. De esta forma, se tendrá un grado de libertad más con el que se podrá diferenciar

de mejor manera a qué vértice pertenece cada traza generada por las partículas producto. Para lograr medir con precisión dicho tiempo, se instalará un nuevo detector que complementará a CMS, el MTD (MIP *Timing Detector*) [4].

El objetivo de este trabajo es estudiar la viabilidad de una red neuronal de tipo grafo como algoritmo de reconstrucción de vértices en el detector CMS. Para ello, se desarrolla un programa que simule los datos de colisiones tomados en CMS. Una vez obtenidos los datos a través de dicho simulador, se pretende replicar el algoritmo de CMS utilizado para el *clustering* de trazas, es decir, para asignar las trazas de las partículas resultantes a los vértices que las han generado, este proceso es conocido como *vertexing*. Tras haber obtenido el rendimiento del algoritmo de *clustering* que imita el de CMS, se desarrollará una red neuronal de tipo grafo para comparar la eficiencia de ambos métodos.

2 El upgrade de CMS

2.1. El LHC

En el laboratorio Europeo de Física de Partículas (CERN) se realizan las investigaciones más avanzadas en la física fundamental. Se encuentra situado en la frontera entre Francia y Suiza y cuenta con el LHC (*Large Hadron Collider*). Tal y como se ha mencionado anteriormente, el LHC es el mayor y más potente acelerador de partículas construido en la actualidad. Se trata de un anillo de aproximadamente 27 km de circunferencia construido bajo tierra, a 100 m de profundidad [1].

Con el objetivo principal de desentrañar los misterios de la física fundamental, cuenta con diferentes métodos para aumentar la energía de haces de hadrones hasta que alcanzan velocidades cercanas a la de la luz. Estos hadrones, habitualmente protones, son acelerados y colisionan entre sí al alcanzar una determinada energía, generándose más partículas en el proceso, entre las cuales se pueden manifestar nuevas nunca antes vistas.

2.1.1. Diseño del LHC y su funcionamiento

Cuando el LHC acelera haces de protones, no lo hace en el anillo principal. El proceso de aceleración es gradual, utilizando una serie de aceleradores hasta alcanzar la energía deseada. Una vez se han acelerado lo suficiente, los haces de protones son inyectados al anillo principal, donde se terminan de acelerar hasta alcanzar una energía de centro de masas de $\sqrt{s} = 13$ TeV [6]. En la figura 2.1 se muestra un esquema en el que aparecen los aceleradores del LHC.

Para guiar a las partículas a través del anillo, siguiendo una trayectoria circular, se utilizan un total de 9593 imanes electromagnéticos superconductores. Para conseguir que estos imanes funcionen correctamente, se necesita reducir la temperatura hasta los 1.9 K (-271.3 $^{\text{o}}$C) ya que, de esta forma, se alcanza el estado de superconducción. Para conseguir esto, se utiliza helio líquido.

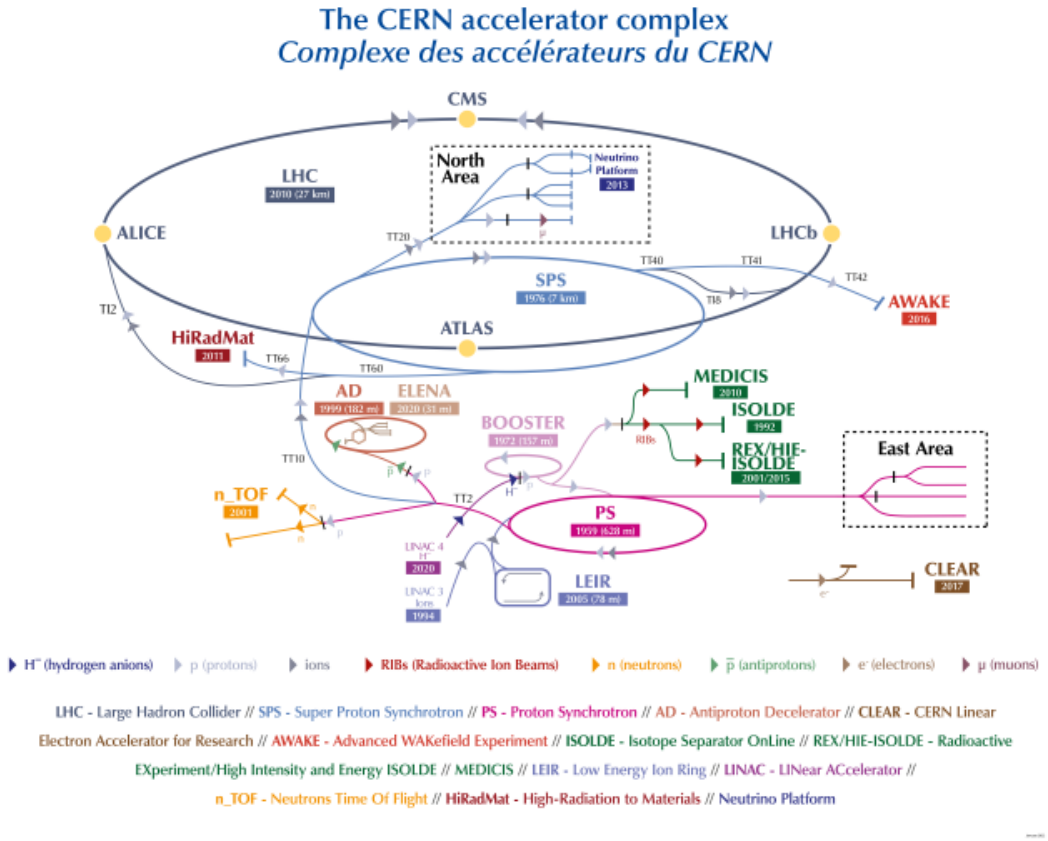

Figura 2.1: Esquema del complejo de aceleradores del CERN. Representa el LHC y diferentes aceleradores. [6]

2.1.2. Objetivos del LHC

El Modelo Estándar de partículas es la teoría más actual que describe las partículas fundamentales y sus interacciones. En el LHC se busca confirmar y estudiar de forma detallada las partículas e interacciones que describe dicho modelo. Por ejemplo, tras haber detectado y confirmado la existencia del bosón de Higgs, la partícula asociada a la atribución de masa, se busca medir propiedades como sus modos de desintegración.

No obstante, el LHC también busca nuevos eventos físicos que van más allá del Modelo Estándar. Por ejemplo, la búsqueda de partículas supersimétricas, una teoría que relaciona los bosones con los fermiones. En esta teoría, cada bosón contaría con un "compañero" fermión y viceversa. Estos compañeros serían más pesados que el bosón o fermión al que estén asociados. Por otro lado, también se busca dar una explicación a lo que es la materia y la energía oscura, pues la materia conocida solo representa un 4 % del total. Otra cuestión que queda sin resolver todavía es el porqué solo observamos materia en el universo, cuando debería haberse producido la misma cantidad de materia y antimateria al inicio de este.

2.2. CMS

Tras acelerar los protones del LHC a velocidades cercanas a la de la luz, estos se hacen colisionar en puntos estratégicos del anillo. En estos puntos se encuentran los detectores responsables de analizar dichos eventos. El Solenoide Compacto de Muones (CMS, por sus siglas en inglés) es uno de estos detectores. Fue construido con el fin de trabajar en el ámbito de la física de partículas a energías del orden de TeV [7].

La idea principal de CMS es ser capaz de identificar y medir las características de diferentes partículas con ayuda de las distintas partes que lo conforman. Se busca medir muones, electrones, fotones y hadrones con ayuda de las distintas partes que componen el detector: un sistema de trazado (*tracker*), un calorímetro electromagnético, un calorímetro hadrónico y un sistema de detección de muones.

Este detector tiene forma cilíndrica de, aproximadamente, 21.6 m de longitud y 14.6 m de diámetro y pesa alrededor de 12500 T. Todas las partes del detector se disponen en capas concéntricas alrededor del punto por el que pasan y colisionan los haces de protones. En la figura 2.2 se puede observar un esquema de la estructura de CMS.

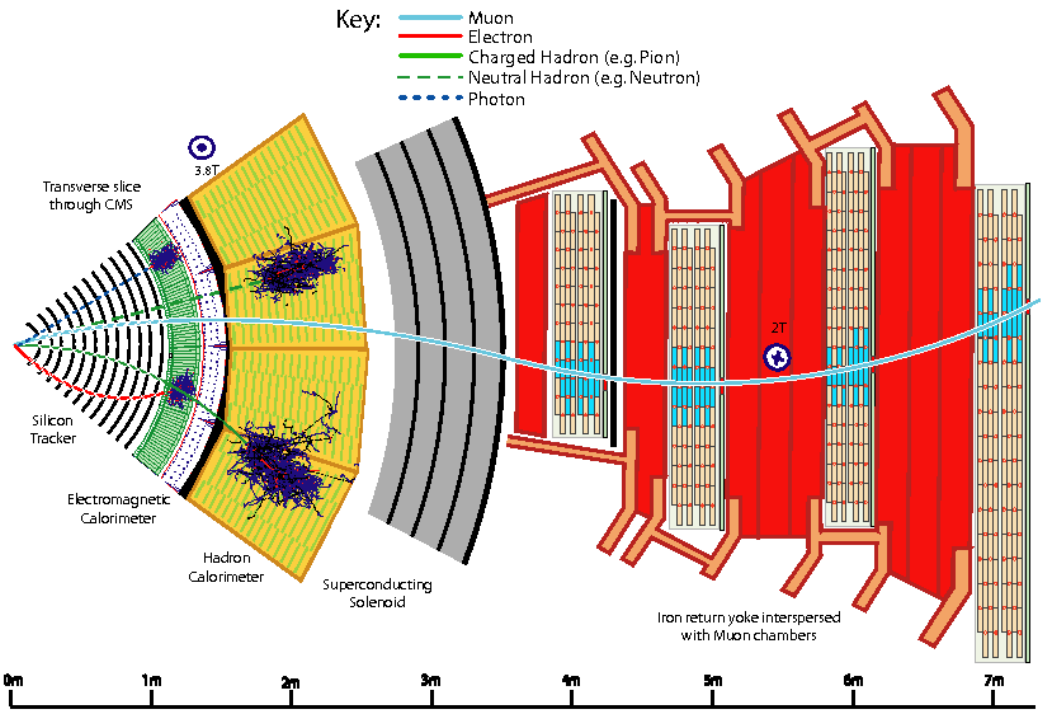

Figura 2.2: Esquema de la estructura de capas de CMS. Representa un corte transversal del detector CMS. [8]

2.2.1. Imán superconductor

Una de las partes que componen el detector CMS es el imán superconductor. Se trata de un imán solenoide, una bobina que genera un campo magnético uniforme en su interior de 3.8 T. Este

imán es de 12.5 m de diámetro y 3.15 m de radio libre interno, por lo que es hueco. Dentro del radio del imán, se colocan el *tracker* y los calorímetros electromagnético y hadrónico. Al estar estos detectores antes que la bobina, las partículas han de atravesar menor cantidad de material antes de ser medidas, por lo que la precisión de los datos tomados mejora [8]. En la figura 2.2, se puede apreciar cómo el solenoide superconductor encierra al *tracker* y a los calorímetros.

Gracias a que los primeros detectores están en el interior del imán, se pueden reconstruir de forma más sencilla las trayectorias de partículas. Esto es debido a que no pierden energía atravesando el material y la conexión entre las trayectorias detectadas por el *tracker* y los cúmulos generados por los calorímetros en función de la energía es mejor. Estos cúmulos de energía se forman debido a que, al entrar en un campo magnético, las partículas cargadas (un electrón) se curvan, separándose de las partículas neutras (un fotón) simplificando el proceso de distinción entre partículas. En este caso, el poder de curvatura (La capacidad que tiene el campo magnético para curvar la trayectoria de una partícula) es de 4.9 T·m.

2.2.2. *Tracker* interno

El sistema *tracker* interno es un detector que también compone a CMS. Este dispositivo reconstruye de forma precisa las trayectorias de partículas cargadas generadas por los eventos de colisiones protón-protón, por lo que resulta crucial para el *vertexing* llevado a cabo [7]. Es de forma cilíndrica con un radio de 1.2 m, una longitud de 5.6 m y está construido completamente en silicio. El diseño del cilindro se divide en; la sección central de este, el "barril", que cuenta con tres capas de píxeles y 10 capas de detectores de tiras de silicio y en los dos extremos que cierran el cilindro, los *endcaps*, que cuentan con 2 capas de píxeles y 12 capas de detectores de tiras de silicio. [8]

Este detector cuenta con 16588 módulos de sensores de silicio divididos en 66 millones de píxeles y 9.6 millones de tiras de silicio. Las dimensiones de estos píxeles son de 150×100 μm^2 mientras que, las longitudes de las tiras de silicio, varían entre 80 y 180 μm. De esta forma, se consigue una alta resolución con la que se pueden distinguir trayectorias de partículas muy cercanas, como sucede en los *jets* de hadrones por ejemplo.

Sin embargo, debido a todas las capas del *tracker* y demás material asociado a este como cables o el sistema de refrigeración, las partículas se topan con una considerable cantidad de material. Esto provoca de forma inevitable pérdidas en la energía de las partículas que atraviesan el detector antes de que lleguen a los calorímetros. Un fotón que atraviese el *tracker* podría convertirse en un par electrón-positrón antes de llegar al calorímetro electromagnético, al igual que un electrón podría emitir un fotón por efecto *bremsstrahlung*. [8]

2.2.3. Calorímetro electromagnético

Otra de las secciones que compone a CMS es el calorímetro electromagnético. Se trata de un calorímetro diseñado para medir la energía de electrones y fotones. Su barril está compuesto por 61200 cristales de tungstato de plomo ($PbWO_4$) mientras que cada uno de los extremos del cilindro está compuesto de 7324 cristales [7]. La longitud de los cristales en el barril es de 23 cm mientras que los cristales de los extremos son de 22 cm [8]. Además, frente a los cristales e los extremos, se encuentra instalado un detector de *preshower*, que consta de un radiador de plomo seguido de tiras de sensores de silicio dispuestas en planos. [8]

La elección de tungstato de plomo para la construcción del calorímetro tiene una serie de ventajas gracias a su alta densidad. Entre estas ventajas, se puede destacar la rapidez de respuesta al centelleo, alta granularidad (alto número de sensores) y resistencia a la radiación, características importantes en el LHC [7].

En este calorímetro, cuando un fotón o un electrón atraviesa el plomo, da comienzo un fenómeno denominado ducha electromagnética. Esto sucede cuando partículas cargadas atraviesan un material denso, provocando la creación de fotones y pares de electrón-positrón. Gracias a la granularidad y al pequeño radio inicial de la ducha, se puede determinar su posición. [8]

2.2.4. Calorímetro hadrónico

El segundo calorímetro que compone a CMS es el calorímetro hadrónico. Se trata de un detector empleado en la medición de jets de hadrones y neutrinos cuya energía no es enteramente depositada en el calorímetro electromagnético así como partículas exóticas [7]. Este detector se encuentra entre el calorímetro electromagnético y la bobina solenoidal que genera el campo magnético. Esta hecho de capas de absorbente de latón y baldosas de centelleador plástico. [8].

2.2.5. Detectores de muones

La capa más externa de CMS está compuesta por el detector de muones. Este está compuesto de tres placas de acero con cuatro planos de detectores de muones intercalados. El objetivo de este detector, como su nombre indica, es la detección de muones y la medición de su energía. Para ello, se utilizan tres tipos de detectores; las cámaras de tubos de deriva , las cámaras de tiras catódicas y las cámaras de tiras resistivas. [8]

2.3. El upgrade de CMS

Desde 2027 hasta 2030, se va a realizar la tercera parada larga en el LHC. De esta forma, se prepara al acelerador y a los detectores para la fase de Alta Luminosidad del LHC (HL-LHC, por sus siglas en inglés). Como se ha explicado anteriormente, el proyecto HL-LHC busca aumentar la luminosidad, es decir, el número de colisiones por segundo por cm^2. De esta forma, se aumentan las probabilidades de medir procesos poco comunes o limitados dentro del Modelo Estándar o más allá de este. [9]

A lo largo del año 2016, la media del número de eventos que sucedían de forma simultánea provocando el *pile up* fue de 53 con máxima luminosidad instantánea. No obstante, en la fase de Alta Luminosidad del LHC, se esperan 140 eventos de media, alcanzándose incluso 200 en luminosidad máxima.

No obstante, para aprovechar este incremento de eventos, CMS también debe mejorarse. A esta fase se la conoce como Actualización de Fase 2 de CMS. Al aumentar el número de eventos, la radiación que se emite también aumenta, por lo que CMS tendrá que ser capaz de soportarlo. Además, al haber un aumento de los eventos, también aumenta el número de partículas y el *pile up* por lo que necesitará mayor granularidad en el detector, mayor ancho de banda para el manejo de datos y una mayor tasa de disparo, que define la velocidad a la que se toman datos. [9]

2.4. El MIP Timing Detector

Tal y como se ha explicado, uno de los problemas a la hora de analizar las colisiones protón-protón es el del apilamiento de eventos. Para solucionar esto, se va a implementar en CMS para la fase 2 del LHC el sistema *MIP Timing Detector*. El objetivo de este nuevo detector sería tomar medidas precisas del tiempo en el que suceden los eventos. De esta manera, se podrá hacer una separación entre dos eventos que sucedan en un mismo punto basándonos en el tiempo en el que sucede cada uno. El MTD permitirá separar temporalmente los eventos de colisiones con una resolución temporal de en torno a 30 ps. De esta forma, se facilitará la reconstrucción de vértices, mitigando el problema del *pile up*. [4]

El MTD está compuesto de dos partes, el barril o *Barrel Timing Layer* (BTL) y los extremos o *Endcap Timing Layer* (ETL). En la figura 2.3 se muestra un esquema del MTD con sus dos componentes principales. El BTL está ubicado en la zona central del MTD y utiliza cristales centelladores LYSO:Ce acoplados a fotomultiplicadores de silicio (SiPMs, por sus siglas en inglés). Estos SiPMs han demostrado la capacidad de alcanzar altas resoluciones temporales, inferiores a 30 ps, por lo que proporcionan una información temporal sumamente exacta de las partículas que atraviesan el barril. De esta forma, se consigue mejorar la asociación de trazas a los vértices.

Por otro lado, los ETLs están ubicados en los extremos del BTL y utilizan sensores de silicio con ganancia interna, los *Low Gain Avalanche Detectors* (LGADs) [5]. Estos sensores amplifican la señal detectada, por lo que permiten una gran resolución temporal en los extremos del detector, donde se acumula una mayor dosis de radiación. Es por esta acumulación de radiación que se usan los LGADs y no los SiPMs.

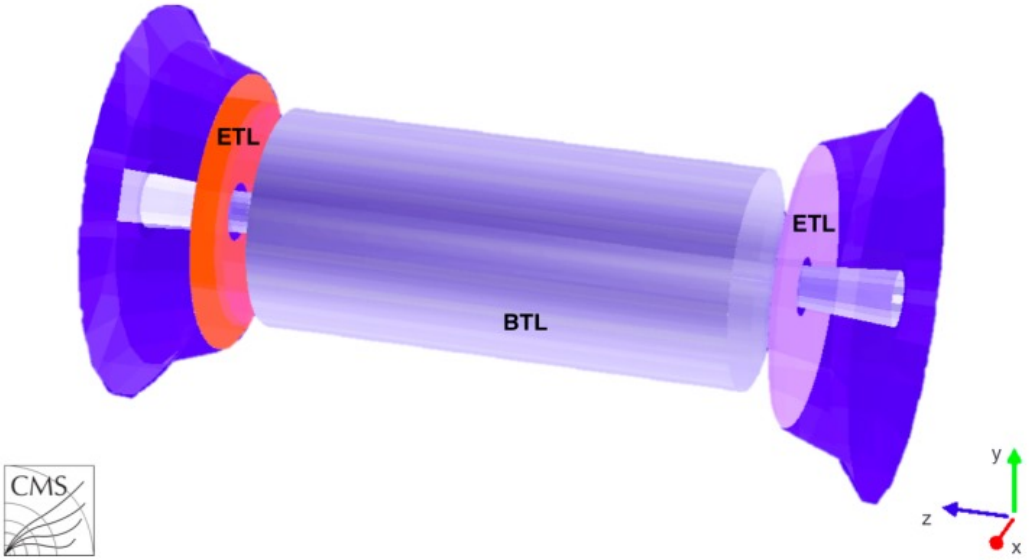

Figura 2.3: Esquema del MTD. Representa la estructura del *MIP Timing Detector* con sus dos componentes principales, el ETL y el BTL [4].

3 El algoritmo de vertexing de CMS

En este capítulo, se detalla la reconstrucción de vértices que tiene lugar en el experimento CMS del LHC. Se explicarán tanto los algoritmos de reconstrucción de trayectorias de partículas como la identificación de los vértices primarios, es decir, los vértices donde se han producido las colisiones protón-protón. Se describirá el proceso de reconstrucción local de impactos de partículas (*hits*) en los detectores, el proceso de reconstrucción de trazas y, finalmente, los métodos para la reconstrucción de vértices primarios a través del agrupamiento de trazas. Para este capítulo, se ha utilizado la información recogida en el artículo [10]. El algoritmo explicado en este artículo se centra únicamente en la coordenada espacial, aunque una versión que incluye también la coordenada temporal está disponible dada la próxima instalación del MTD. El algoritmo de dos coordenadas es similar al de una coordenada, así que, por simplicidad, el trabajo se centrará en este último caso.

3.1. Reconstrucción local de *hits*

El comienzo de todo el sistema de CMS es el análisis de las señales digitales recogidas por los detectores píxeles y las tiras de silicio del *tracker*.

3.1.1. Impactos en los píxeles

Para el análisis de las señales de los píxeles, se realiza una supresión de ceros, la cual consiste en recolectar solo los datos de los píxeles que superen un cierto umbral de carga (3200 electrones por píxel). Una vez obtenidos los datos, se forman agrupaciones de píxeles. Estos clústeres se realizan a partir de píxeles adyacentes y tienen que superar un umbral de carga de por lo menos 4000 electrones.

Tras haber formado las agrupaciones de píxeles, se emplean dos algoritmos diferentes para determinar la posición de estas; el *first-pass hit reconstruction*, utilizado para la semilla de traza, y el *Template-based hit reconstruction*, que se basa en la forma del clúster.

First-pass hit reconstruction

Este algoritmo es el primero que se pone en marcha, ya que se utiliza en las etapas tempranas de la reconstrucción de trazas como la generación de semillas y el reconocimiento de patrones. Su objetivo es dar una primera estimación del clúster de píxeles en función de dos coordenadas propias del sensor, la transversal (u) y la longitudinal (v).

En el caso de que el clúster solo abarque un píxel, se asociará el centro geométrico de este a la posición de la partícula. No obstante, si el clúster abarca varios píxeles, se proyecta el clúster sobre una de las coordenadas y se interpola la posición en dicha coordenada a partir de la carga relativa de los dos píxeles en los extremos. Luego se haría lo mismo para la otra coordenada.

Template-based hit reconstruction

Este es el algoritmo que se usa al final para refinar la posición de los *hits*. Para ello, emplean una serie de plantillas de distribuciones de carga esperadas creadas a partir de la simulación PIXELAV recreando el comportamiento de los sensores de píxeles. Estas plantillas almacenan distribuciones de carga diferentes según las condiciones en las que se dé el impacto. Para ello, se divide cada píxel en puntos más pequeños y finamente espaciados. De esta forma, se tienen plantillas para partículas que atraviesan el sensor en diferentes puntos de un mismo píxel.

Cuando se procesa un clúster medido experimentalmente, se compara con las diferentes plantillas que se tienen para analizar las coincidencias. La idea es encontrar la plantilla que mejor se adapte a la distribución de carga medida. Para determinar cuál es la plantilla que más se asemeja a dicha distribución, se minimiza una función que calcula la diferencia entre la distribución de carga experimental y la predicha por la plantilla. Entonces, la posición de impacto se estima a partir del punto de impacto dentro de un píxel de la plantilla que mejor se adapte al clúster experimental.

Por último, se pueden utilizar las muestras utilizadas para generar las plantillas para corregir el sesgo de las posiciones reconstruidas. Se aplica el algoritmo a estas muestras y, al conocer la posición verdadera del impacto, se determina dicho sesgo y se tiene en cuenta para cuando se aplique con datos reales.

3.1.2. Impactos en las tiras de silicio

Por otro lado, para el análisis de los datos recogidos en los detectores de tiras de silicio, se sigue un procedimiento similar al que se sigue con los píxeles. Se realiza la supresión de ceros y se forman clústeres a partir de una tira con una señal significativamente por encima del ruido que captan los detectores y se le añaden las tiras vecinas con carga significativa. Tras esto, se estima la posición del punto de impacto a partir del promedio de la posición de las tiras ponderado con su carga. Por último, se aplican correcciones debidas al efecto Lorentz y a la ineficiente recolección de carga por los sensores más gruesos.

3.2. Reconstrucción de trazas

Tras haber reconstruido los puntos de impacto de las partículas detectadas, se busca reconstruir las trazas para determinar las trayectorias seguidas por éstas. Para ello, CMS utiliza el software

Combinatorial Track Finder (CTF), que es una adaptación del filtro de Kalman combinatorio. La reconstrucción se da mediante múltiples iteraciones del CTF, un proceso conocido como *iterative tracking*.

La razón de realizar varias iteraciones del algoritmo es abordar la reconstrucción de trazas por partes. En las primeras iteraciones se detectan las trazas más fáciles de reconstruir, como las que tienen un alto momento transverso o las que han sido originadas en el punto de interacción nominal. Tras estas primeras iteraciones, se excluyen las trazas que han sido asociadas a trayectorias, facilitando así la reconstrucción del resto de trayectorias a partir de trazas con momento transverso más bajo o que provienen de vértices desplazados.

El software descrito emplea un total de 6 iteraciones y cada una de ellas atraviesa cuatro etapas diferentes.

3.2.1. Generación de semillas

El primer paso que da el software es la generación de semillas. Una semilla es la estimación inicial de diferentes parámetros que se hace de una trayectoria. Para ello, se utilizan dos o tres *hits* como punto de partida para reconstruir dichas trayectorias. De esta forma, se obtienen semillas que pertenecen a trayectorias reales. Para determinar los 5 parámetros necesarios para reconstruir las trayectorias, se necesitan mínimo 3 mediciones espaciales de la partícula o 2 mediciones, pero con una condición del punto de origen, es por esto que se usan generalmente dos o tres *hits*.

Tras esto, el software se centra en recolectar los datos de diferentes capas del detector según qué iteración se esté realizando. Para las primeras, se enfoca en trazas que hayan sido generadas cerca del punto de interacción, pues son las que tienen el momento transverso más elevado y son las más fáciles de reconstruir. Es en estas en las que se usan los tripletes de *hits*. En las iteraciones intermedias, se recuperan las trazas con bajo momento transverso o que hayan dejado solo dos *hits* en el detector, por lo que se usa este par y la restricción de vértice. Por último, el software trata de reconstruir las trazas más complicadas, habiendo descartado las trazas de las semillas generadas.

3.2.2. Búsqueda de trazas

Tras haber obtenido las semillas, el siguiente paso es reconstruir la trayectoria completa. A este proceso se lo conoce como búsqueda de trazas y consiste en extender la información inicial de una semilla añadiendo los *hits* que una partícula dejó en las siguientes capas del detector. Para conseguir esto de la forma más eficiente posible, se emplea el algoritmo filtro de Kalman. Este algoritmo va actualizando los parámetros de la trayectoria a medida que se añaden nuevos puntos de impacto.

Este ciclo de búsqueda de *hits* y actualización de parámetros se repite capa por capa desde el interior hasta el exterior del detector. La reconstrucción de una trayectoria cesa en el momento en el que se alcanza el exterior del *tracker* o en el momento en el que se acumulan muchos *hits* fantasma, que son puntos de impacto con poca certeza de que pertenezcan a la trayectoria que se está reconstruyendo. El ciclo también puede parar cuando el momento transverso estimado cae por debajo de un umbral, ya que partículas con momento transverso bajo son complicadas de seguir.

Por último, cuando este ciclo de búsqueda de interior a exterior cesa, se puede iniciar un ciclo contrario, de exterior a interior. De esta forma, se pueden recuperar posibles *hits* de capas internas

que se hayan omitido, mejorando la calidad de las trazas reconstruidas.

3.2.3. Ajuste final de trazas

Tras haber reconstruido las trayectorias de las partículas a partir de un grupo de *hits* que, en principio, corresponden a una partícula, se sigue un procedimiento de ajuste. Para ello, se vuelve a utilizar el filtro Kalman, ajustando de esta forma los parámetros de la trayectoria reconstruida y un proceso de suavizado. Es cierto que la búsqueda de trazas ya ha estimado estos parámetros, pero es posible que estos estén sesgados debido a que se parte de condiciones iniciales o a no haber utilizado de forma eficiente toda la información, pues solo estaba disponible de forma completa al finalizar la reconstrucción.

El proceso de ajuste comienza aplicando nuevamente el filtro Kalman en el punto de impacto más interno de la trayectoria. Desde ahí, avanza actualizando la estimación de los parámetros pasando por cada uno de los *hits*. Una vez llega al final, se pone en marcha el proceso de suavizado, que es también un filtro Kalman. No obstante, al realizar este segundo proceso, se hace desde el *hit* más externo de la trayectoria, y avanza hacia el interior. Al realizar estos dos procesos, que se ejecutan en direcciones contrarias, se estiman los parámetros de forma mucho más exacta.

Una vez realizados los dos procedimientos, se procede a buscar puntos de impactos incorrectamente asociados a la trayectoria. Estas asignaciones incorrectas pueden deberse a ruido electrónico, *hits* de trazas cercanas o a radiación. Para la identificación de estos impactos mal asociados, se utilizan dos métodos. Uno se basa en el residuo entre la posición del *hit* y la predicción de la traza, descartándose el punto cuando la discrepancia es alta. El segundo analiza si la distribución de carga de los píxeles cuadra con la trayectoria seguida. Al eliminarse estos puntos de impacto, se vuelven a aplicar los dos procedimientos mencionados hasta que no se encuentren más *hits* mal asignados o hasta que la traza no cumpla con unos criterios de calidad.

3.2.4. Selección de trazas

Por último, una vez se han reconstruido y ajustado las trazas, se procede con la selección de trazas. Este paso es sumamente importante debido a la cantidad de partículas que se detectan. Al detectarse un número alto de partículas en el LHC, se pueden generar trazas falsas. Estas son trazas reconstruidas que no corresponden a ninguna partícula o que están mal reconstruidas. Para ello, se aplican una serie de criterios que disminuyen el número de estas trazas, tratando de no disminuir la eficiencia de las trazas bien reconstruidas.

Entre los criterios que se emplean para esta selección, se pueden destacar el número de capas que tienen *hits* asociados a la traza, la calidad de ajuste de la traza y la compatibilidad de las trazas con que se originen en un vértice de interacción primario. Debido al *pile up*, puede ocurrir que haya varios vértices, por lo que se tomarán en cuenta todos. Para optimizar el proceso, los criterios de selección se ajustan en función del momento transverso y del número de capas con puntos de impacto, ya que, cuantos más *hits*, mayor calidad de trazas.

3.3. Reconstrucción de vértices primarios

Una vez reconstruidas las trazas, estas se utilizan para la reconstrucción de vértices primarios. Estos vértices son en los que tienen lugar los eventos de colisión protón-protón. Este proceso consta de 3 pasos: la selección de trazas, el agrupamiento de trazas y el ajuste de vértices.

3.3.1. Selección de trazas

La primera fase para la reconstrucción de vértices es la selección de trazas de partículas. Estas han sido previamente reconstruidas mediante los procesos descritos en la sección anterior. El objetivo de esta fase es filtrar las trazas reconstruidas, seleccionando solo las que provienen de colisiones protón-protón y descartando las que provengan de otras vías, como decaimientos de partículas de vida larga.

Para conseguir esta selección, se aplican diferentes criterios. En primer lugar, se analiza la distancia de mínima aproximación de la traza al centro del beam spot. Para que la traza sea válida, dicha magnitud ha de tener una significancia de menos de 5σ. En segundo lugar, el número de *hits* que se hayan detectado para una traza. En concreto, ha de haber *hits* en al menos dos de las capas del detector de píxeles, y, en total, ha de haber 5 *hits* entre el detector de píxeles y el detector de tiras de silicio. De esta forma, se garantiza que la partícula se ha medido con suficiente precisión. Otro criterio a tener en cuenta es la calidad del ajuste (χ^2). El valor máximo que puede tener una traza es de $\chi^2 < 20$.

Por último, señalar que no se le impone ninguna condición al momento transverso de la traza. De esta forma, se consigue una alta eficiencia en la reconstrucción de los vértices.

3.3.2. Agrupamiento de trazas

La segunda fase de la reconstrucción de vértices, tras haber seleccionado a las trazas que podrían haber sido generadas a partir de un vértice primario, es la agrupación. Debido al *pile up*, se sabe que las trazas seleccionadas no tienen porqué venir del mismo vértice. Es por ello que se realiza un proceso de *clustering*, para identificar el vértice que tienen un grupo de trazas en común. Para realizar este proceso, CMS emplea un algoritmo denominado *Deterministic Annealing* (DA).

Deterministic Annealing

Este algoritmo actúa buscando de forma dinámica cuáles son las mejores agrupaciones de trazas. Para ello, se utiliza como medida de similitud la Distorsión, que consiste en la suma de distancias cuadráticas de las trazas al centroide candidato. Esto hace que el método sea comparable al algoritmo *k-means*, el cual es más sencillo de implementar y que se detallará en el próximo capítulo. Uno de los parámetros que utiliza es una temperatura T ficticia para controlar la resolución de la agrupación. Este algoritmo, a diferencia de otros algoritmos de *clustering*, no asigna una traza a un solo vértice, sino que asigna a una traza diferentes probabilidades de que pertenezca a distintos vértices. El algoritmo trata de minimizar una función F que depende, entre otras cosas, de la posición de la traza y de las probabilidades de que una traza pertenezca a un vértice. Las probabilidades de

una traza para los diferentes vértices (ρ_{ik}) se derivan de la minimización de F. ρ_{ik} depende exponencialmente de T, la temperatura ficticia que modula la resolución. A altas temperaturas, ρ_{ik} es similar para todos los vértices, por lo que la traza no estaría bien definida para un clúster. No obstante, al bajar T, las agrupaciones se definen más, por lo que es posible asignar un vértice a la traza.

En cuanto a cómo se maneja la temperatura ficticia T, el algoritmo empieza con un valor de T muy alto y con un solo vértice común a todas las trazas.Esta T se iría disminuyendo gradualmente y, a su vez, se minimiza la función F, reajustando sus parámetros y actualizando ρ_{ik}. Cuando T alcanza un punto crítico, el vértice inicial se divide en dos ya que es más favorable para el sistema. De esta forma, el algoritmo DA determina de forma dinámica el número ideal de vértices. Este proceso de enfriamiento y división de vértices continúa hasta que se alcanza una temperatura T_{min}. Es importante definir correctamente esta magnitud, pues una T_{min} muy alta podría provocar que no se encontrase el número de vértices apropiado, mientras que, para una T_{min} baja, se podría dividir un vértice cuando realmente solo existe uno. Tras alcanzar T_{min}, la temperatura se enfría hasta $T = 1$, pero sin que se produzcan más divisiones de vértices. De esta forma, se afinan las asignaciones ρ_{ik}, haciendo que las trazas tiendan a asociarse a un solo vértice.

Este algoritmo no descarta *outliers*, trazas que no pertenecen a ningún vértice real primario. Para solucionar esto, por debajo de T_{min} se añade un término a la función F que actúa como filtro, infraponderando trazas que estén a más de 4σ del vértice más cercano.

Finalmente, se hace un filtrado de vértices. Este conserva vértices que tengan al menos dos trazas que sean incompatibles con cualquier otro vértice.

Al añadirse la información temporal gracias al subsistema MTD, el algoritmo DA requiere de una suposición inicial del tipo de partícula. Al no conocerse, se asume la hipótesis de masa de un pión cargado, pues son los hadrones más abundantes en los eventos de colisión protón-protón. A partir de esta suposición, se calcula el tiempo esperado de cada traza y se realiza la primera estimación de los vértices en el espacio y en el tiempo. Una vez obtenidos los vértices preliminares, se vuelve a aplicar el algoritmo DA. Gracias a las medidas tomadas por el MTD del tiempo, se puede ajustar la primera hipótesis de masa comparando el tiempo esperado con el medido realmente. De esta forma, se mejora la asignación de trazas a los vértices, permitiendo una mejor reconstrucción de estos.

3.3.3. Ajuste de vértices

Tras haber agrupado las trazas en vértices mediante el algoritmo DA, el último paso sería la refinación de las posiciones de los vértices. Para ello, CMS utiliza un *adaptative vertex fitter*. Este ajustador sirve para tener en cuenta que una traza tal vez no pertenezca realmente al vértice al que ha sido asignada y también las diferentes incertidumbres de todas las trazas.

Este ajustador asocia a cada traza un peso que representa el grado de pertenencia al vértice. Tras esto, se estima una posición inicial del vértice basándose, por ejemplo, en la ponderación de la posición de sus trazas y calcula los nuevos pesos de las trazas en función de la nueva posición. Esto se repite hasta que los pesos de las trazas y la posición del vértice se estabilicen. No obstante, este último paso no se va a tratar en el trabajo posteriormente.

4 Algoritmo de CMS en un entorno de simulación

En este capítulo, se explica con detalle el proceso llevado a cabo para realizar un simulador de eventos de colisión protón-protón en Python [11]. Estos eventos tratan de replicar las colisiones que tienen lugar en el LHC. Posteriormente, se analizarán los datos obtenidos por medio de algoritmos de agrupamiento. Para la agrupación, se ha implementado el algoritmo de *k-means*, un método de aprendizaje no supervisado, con el fin de reconstruir los vértices de las colisiones y agrupar las trazas de las partículas resultantes. El código utilizado se encuentra en el archivo simulador.ipynb, disponible en el repositorio del proyecto [12].

4.1. Generación de datos simulados

Se ha desarrollado un programa que simula los eventos protón-protón que suceden en el LHC y que son detectados por el *tracker* de CMS. Para ello, se asigna un valor al número de vértices que se van a simular siguiendo una distribución de Poisson (Ecuación 4.1) centrada en $\lambda = 200$, que es el valor máximo esperado de *pile-up* en el HL-LHC. Este es el número de colisiones que se podrían llegar a dar de forma simultánea tras la mejora del LHC. [9]

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{4.1}$$

En la figura 4.1, se muestra una gráfica en la que se compara la curva teórica de una distribución de Poisson con un histograma que representa la frecuencia de un número determinado de vértices. Para ello, se ha ejecutado la simulación un total de $N = 10000$ veces y se han obtenido los diferentes valores del número de vértices obtenidos.

Se han asignado los valores típicos de media y desviación estándar a los vértices. Debido a los perfiles y forma de los paquetes de protones que entran en colisión, los vértices producidos siguen

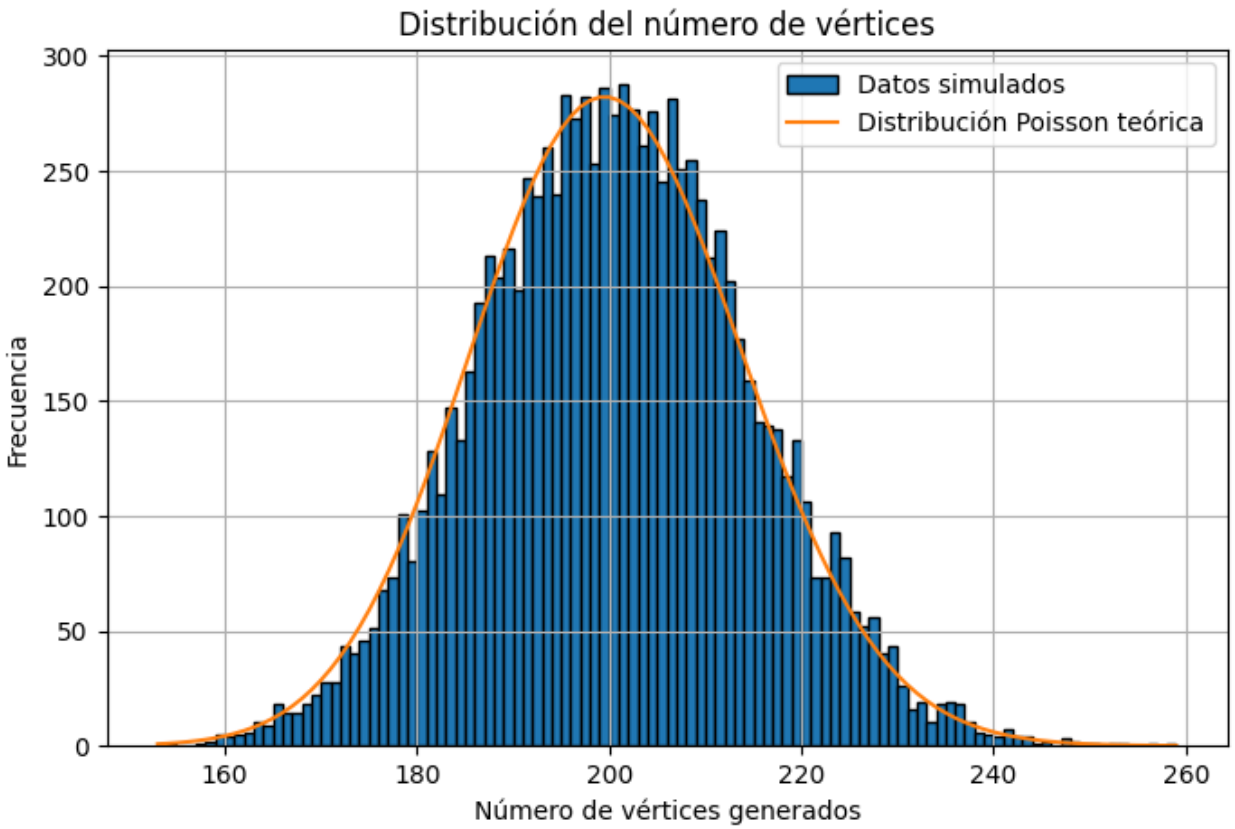

Figura 4.1: Distribución del número de vértices generados en las simulaciones. El histograma muestra los datos simulados, mientras que la línea naranja representa la distribución de Poisson teórica con valor esperado $\lambda = 200$.

una distribución aproximadamente gaussiana tanto en la coordenada espacial z como en la coordenada temporal t. El promedio de las distribuciones para ambas coordenadas es $\mu_z = 0$ cm y $\mu_t = 0$ ps. Por otro lado, sus dispersiones adquieren valores típicos; $\sigma_z = 5$ cm y $\sigma_t = 200$ ps. Con estos valores, se han asignado las coordenadas espacial y temporal a cada vértice siguiendo una distribución normal.

Por otro lado, el programa de simulación también asigna a cada vértice generado un cúmulo de trazas que representan las partículas generadas a partir de la colisión protón-protón que formó dicho vértice. Para determinar el número de trazas asociadas a cada vértice, se ha vuelto a utilizar una distribución de Poisson, esta vez centrada en $\lambda = 20$.

En la figura 4.2 se muestra una gráfica en la que se compara la curva teórica de una distribución de Poisson con un histograma que representa la frecuencia del número de trazas por vértice. Para ello, se ha ejecutado, de igual manera que para la figura 4.1, la simulación un total de $N = 10000$ veces y se han obtenido los diferentes valores del número de trazas por vértice.

Para las trazas, también se han asignado los valores típicos de desviación estándar que se suelen medir en CMS. Para el caso de la coordenada espacial, z, se tiene una desviación estándar $\sigma_z = 0.02$ cm. Mientras que, para la coordenada temporal t, se tiene una desviación estándar $\sigma_t = 35$ ps.

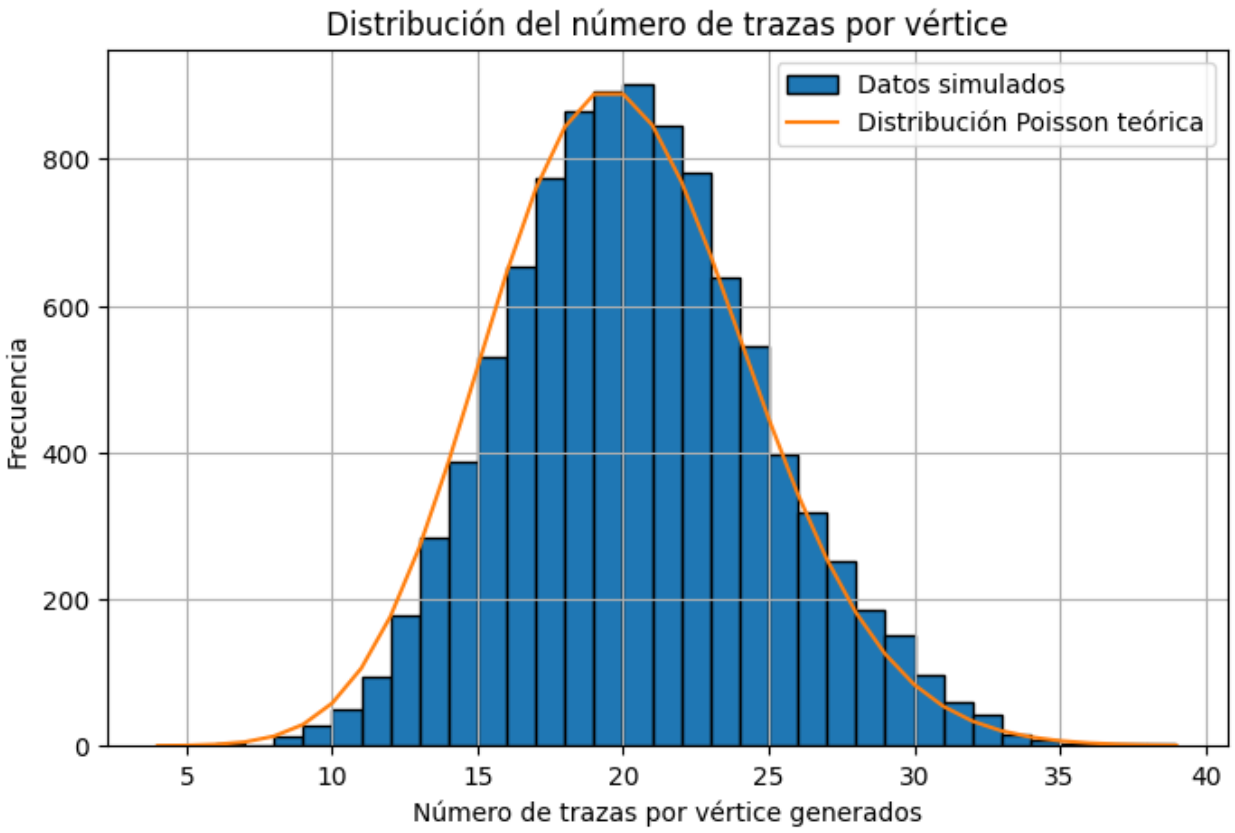

Figura 4.2: Distribución del número de trazas por vértice generadas en las simulaciones. El histograma muestra los datos simulados, mientras que la línea naranja representa la distribución de Poisson teórica con valor esperado $\lambda = 20$.

Con estos valores, se han asignado las coordenadas espacial y temporal a cada traza siguiendo una distribución normal.

En las figuras 4.3 y 4.4, se muestran dos gráficas en las que se representa la distribución del tiempo, t, y la distribución de la coordenada espacial z de las diferentes trazas. En este caso, solo ha sido necesario ejecutar una vez el simulador de datos para obtener los resultados deseados.

Por otra parte, a cada traza generada por el programa, se le asocia un momento. Para obtener el valor del momento de cada partícula, se utiliza una exponencial decreciente con un parámetro de escala $\lambda^{-1} = 0.5$ que tiene la siguiente forma:

$$f(x) = 2e^{-2x} \tag{4.2}$$

Además, también se realizará un truncamiento en 0.2, es decir, cualquier momento obtenido por debajo de un valor de 0.2 GeV, será descartado y se calculará otro. Esto se debe a que CMS no tiene una resolución tan alta como para medir momentos por debajo de dicho valor.

En la figura 4.5, se muestra una gráfica de la distribución del momento. Para este caso, se ha calculado el momento un total de $N = 10000$ veces y se han representado los valores obtenidos.

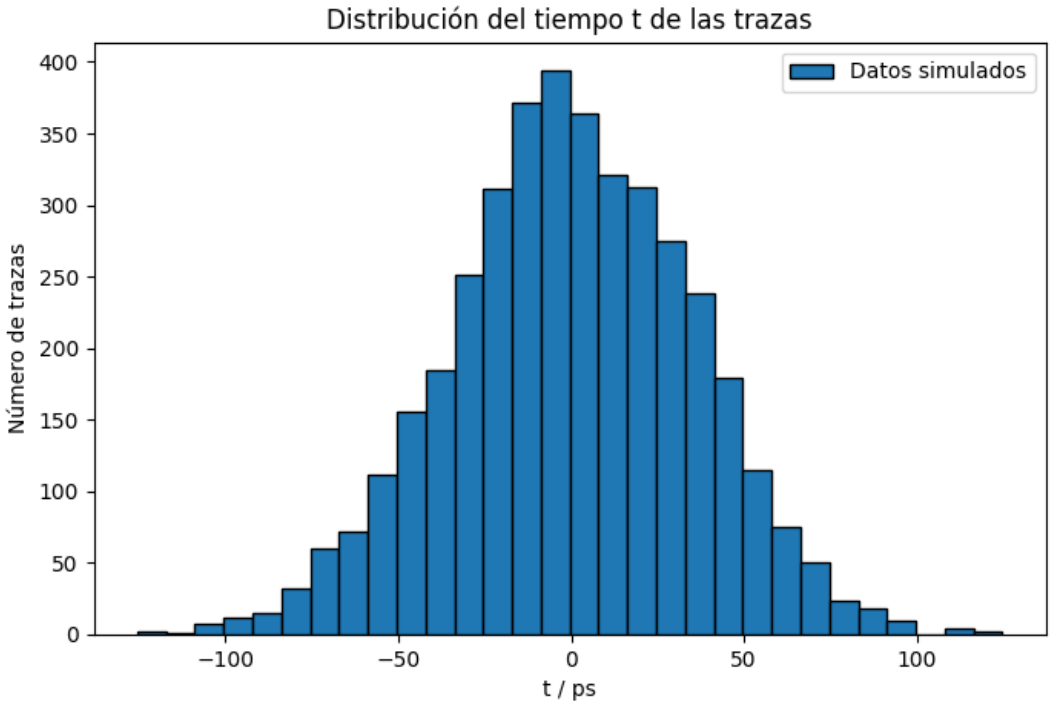

Figura 4.3: Histograma que muestra la distribución del tiempo t de las trazas obtenidas a partir de los datos simulados.

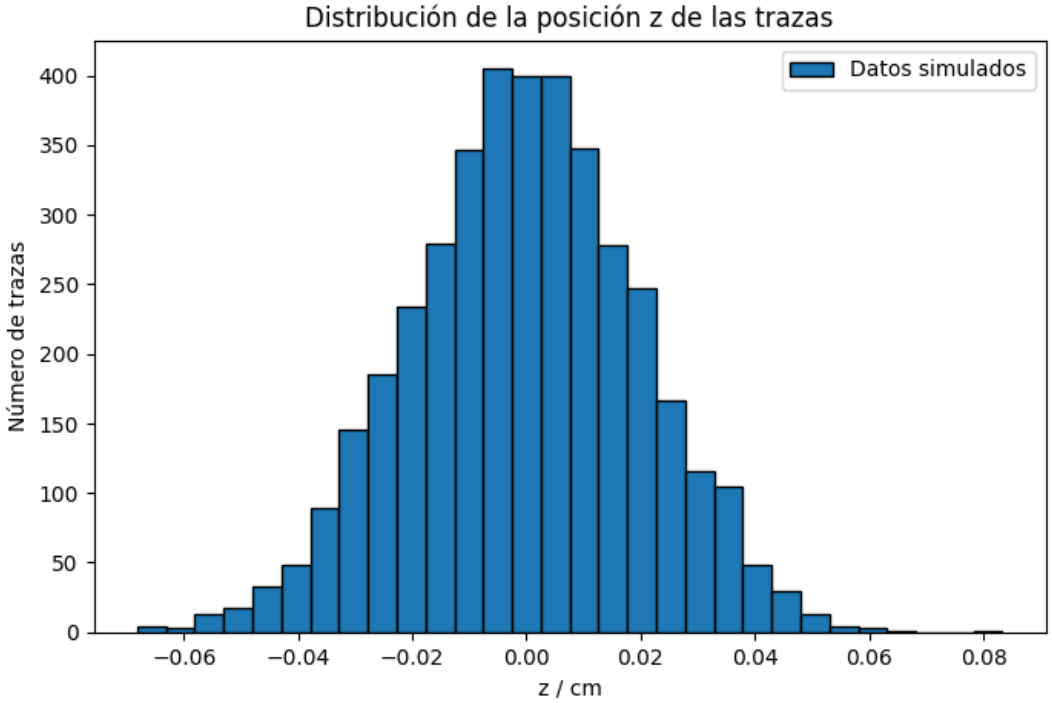

Figura 4.4: Histograma que muestra la distribución del tiempo z de las trazas obtenidas a partir de los datos simulados.

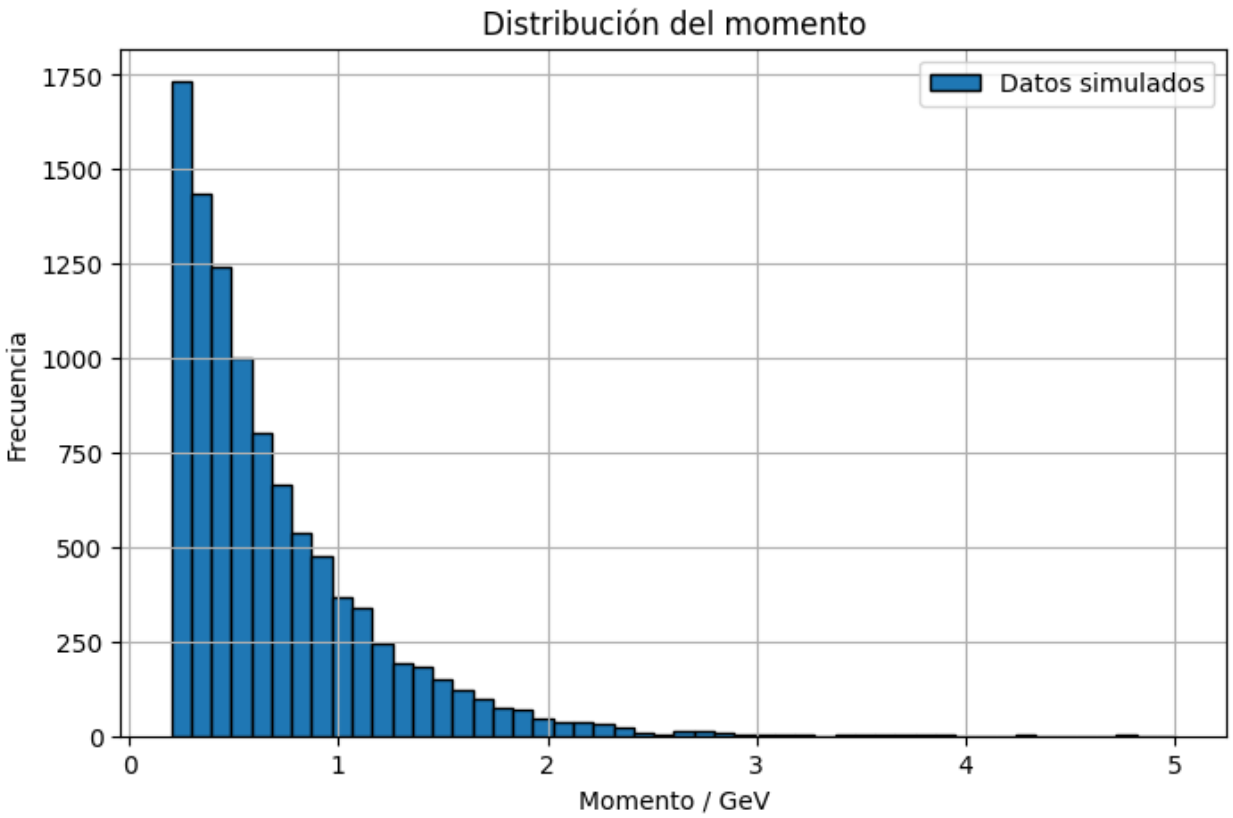

Figura 4.5: Distribución del momento obtenida a partir de los datos simulados.

En el detector CMS, el MTD es el dispositivo encargado de medir el tiempo en el que las partículas atraviesan sus sensores. No obstante, este tiempo no corresponde directamente con el tiempo en el que se produjo la interacción de la que se creó la partícula medida. Para estimar el tiempo en el que esta se generó, es necesario realizar una propagación inversa del detector en el que se ha medido la partícula hasta el punto de interacción. Este proceso depende del momento y tipo de partícula, pues su velocidad varía con la masa. En esta simulación, se genera directamente el tiempo de producción de las partículas en el vértice. Sin embargo, este tiempo es diferente para cada una de las partículas, por lo que se asume que el tiempo generado es de piones cargados y se calculan los tiempos para las posibles partículas que puedan darse en este tipo de eventos. En este caso, se simplificará el problema y se tendrán en cuenta solo dos posibles partículas más, los kaones y nuevos protones.

Para calcular los tiempos correspondientes a los kaones y protones, se utilizan las siguientes ecuaciones, que han sido deducidas a partir de la relatividad especial:

$$t_{\text{kaón}} = t_{\text{pión}} \cdot \frac{\sqrt{p^2 + m^2_{\text{kaón}}}}{\sqrt{p^2 + m^2_{\text{pión}}}} \tag{4.3}$$

$$t_{\text{protón}} = t_{\text{pión}} \cdot \frac{\sqrt{p^2 + m^2_{\text{protón}}}}{\sqrt{p^2 + m^2_{\text{pión}}}} \tag{4.4}$$

Tal y como se muestra en las ecuaciones 4.4 y 4.3, los tiempos de cada tipo de partícula dependen del momento de esta, p, de la masa del pión, $m_{\text{pión}}$, y de la masa de la propia partícula, $m_{\text{kaón}}$ o $m_{\text{protón}}$.

Por último, se asoció a cada traza un tipo de partícula en función de la probabilidad de encontrar dicha partícula en los eventos sucedidos en el LHC. Las probabilidades utilizadas fueron: un 60 % para los piones, un 20 % para los kaones y un 18 % para los protones. El 2 % de partículas restantes no se tuvo en cuenta ya que se trata comúnmente de electrones y muones que se detectan en otros subsistemas de CMS que no son el *tracker*. Estas son las proporciones que se encuentran comúnmente en el LHC.

4.2. Asignación inicial de partículas

En el experimento CMS, se necesitan reconstruir de forma precisa los vértices de las colisiones protón-protón. No obstante, no se conoce el tipo de partícula al que corresponde cada traza generada, por lo que, para realizar el algoritmo de *clustering* en el entorno de la simulación, se toma como hipótesis inicial que todas las trazas corresponden a piones cargados. La razón por la que se asume esta asignación como primera hipótesis es que los piones son el tipo de partículas cargadas más abundantes que se producen en las colisiones protón-protón que tienen lugar en el LHC.

Tras hacer la suposición de que todas las partículas resultantes son piones cargados, los detectores de CMS son capaces de reconstruir las trayectorias en todo el espacio. Para aplicar el algoritmo de agrupamiento, solo es interesante el punto de la trayectoria espacial de menor distancia al haz de protones. Más concretamente, lo que realmente hace falta para llevar a cabo este proceso es la coordenada z, que es la que indica en qué zona del haz se ha medido la partícula, es decir, la coordenada longitudinal. Es por ello que, a la hora de simular los datos, no se han simulado las coordenadas x e y de cada traza.

En las figuras 4.6, 4.7 y 4.8, se muestran tres gráficas en las que se encuentran representados los vértices generados y sus trazas asignadas. Cada una de las representaciones se corresponde con diferentes simulaciones llevadas a cabo para un número diferente de vértices.

La figura 4.8 corresponde con el caso con el que se va a trabajar, es decir, al que se le va a aplicar el algoritmo de *clustering* desarrollado.

4.3. Algoritmo de *k-means* para el *vertexing*

El método que se utiliza en el simulador para realizar el agrupamiento de las trazas se denomina *k-means*. Este algoritmo es de *clustering* iterativo basado en centroides, por lo que la agrupación de los datos se lleva a cabo en función de la cercanía entre ellos, definiéndose de este modo un centroide cuya posición sería la media de las posiciones de las trazas agrupadas entre sí. Dicho algoritmo es de aprendizaje no supervisado, por lo que, a la hora de pasarle los datos para entrenar el modelo, en este caso las trazas, se le pasan sin estar etiquetados. Al contrario que el algoritmo utilizado en CMS, el *Deterministic Annealing*, este algoritmo de clasificación es exclusivo, lo que significa que un dato asignado a un grupo, o clúster, no puede pertenecer a ningún otro más. Para utilizar el algoritmo, se escalaron los datos simulados debido a la diferencia de magnitudes entre la coordenada

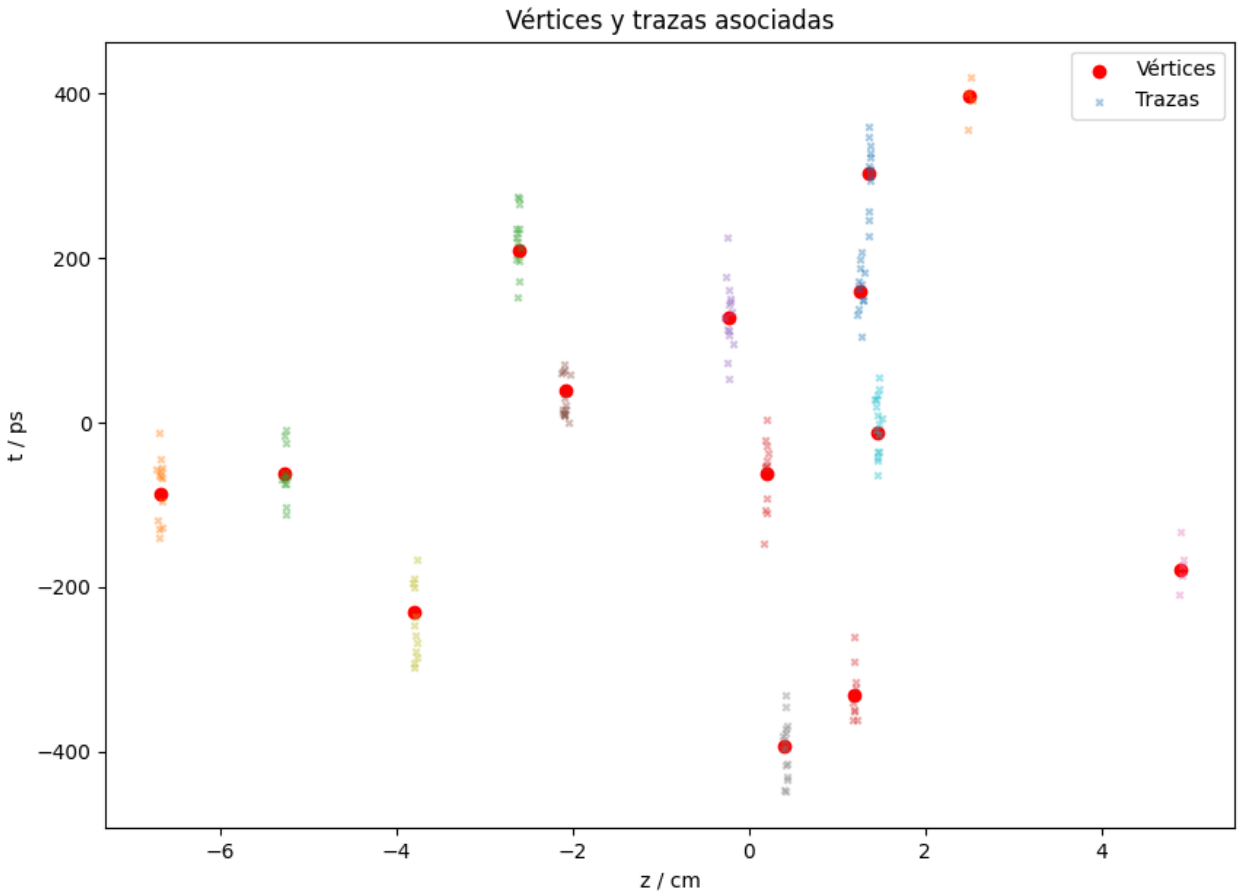

Figura 4.6: Representación en el plano (z, t) de los vértices reconstruidos (puntos rojos) y las trazas asociadas (cruces de colores). El valor esperado de la distribución de Poisson utilizado para generar el número de vértices es $\lambda = 10$. Número de vértices generados: 14

espacial y la temporal. Esto se hizo para dar el peso estadístico adecuado a cada tipo de dato. Para realizar el escalado, se dividieron los datos temporales entre la desviación estándar temporal de las trazas y se repitió el proceso para los datos espaciales con la desviación estándar en la coordenada z.

4.3.1. Funcionamiento de la agrupación

El algoritmo *k-means* es un proceso iterativo en el que se asigna un número de centroides y se agrupan los datos en función de la distancia euclidiana a dicho punto. Es decir, agrupa cada dato por cercanía a un centroide en concreto. En cuanto al número de centroides, o k, este puede variar, por lo que, a mayor número de puntos, menor será la cantidad de datos asociados a este y viceversa.

Para que el proceso iterativo del algoritmo comience, lo primero que hay que hacer es asignar un valor de k. Tras esto, el algoritmo asigna unos valores iniciales para los centroides. El primer centroide se escoge al azar entre todos los datos. El siguiente centroide se escoge entre los puntos de datos restantes. Para ello, se calcula la distancia de dichos puntos y se elige uno de ellos como centroide. La probabilidad de que un punto salga elegido es proporcional a la distancia calculada. Una vez se han asignado todos los centroides, comienza el proceso iterativo que recalcula la posición

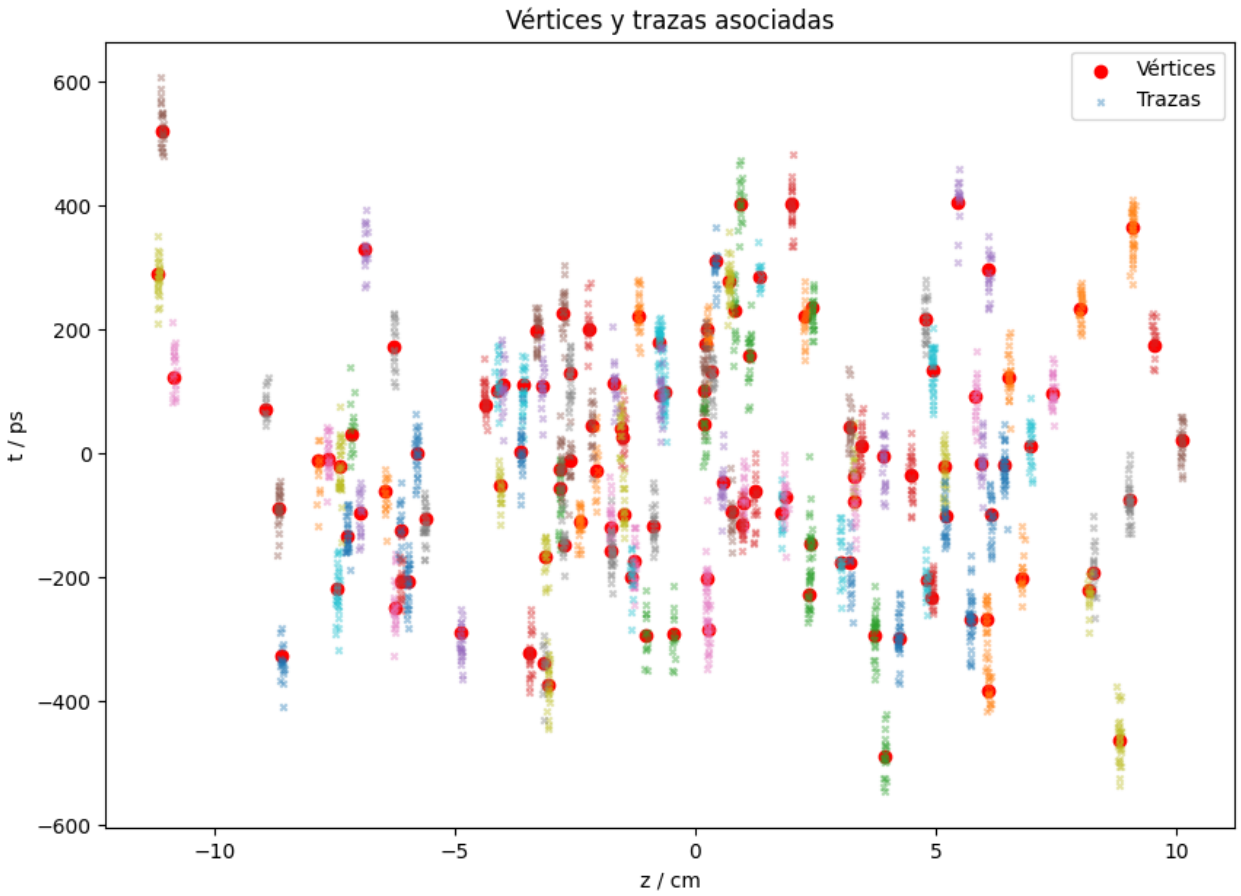

Figura 4.7: Representación en el plano (z, t) de los vértices reconstruidos (puntos rojos) y las trazas asociadas (cruces de colores). El valor esperado de la distribución de Poisson utilizado para generar el número de vértices es $\lambda = 100$. Número de vértices generados: 123

de los centroides en función de la media de las posiciones de los datos asignados en cada clúster. Una vez los valores de los centroides convergen y se estabilizan, el proceso se detiene [13].

4.3.2. Eficiencia del *vertexing*

Tras haber aplicado el algoritmo de *k-means* para realizar el *clustering* de las trazas de las partículas resultantes, se hizo un filtrado de los centroides que han sido reconstruidos correctamente en función de los vértices reales. De esta forma, se pudo calcular la eficiencia del algoritmo a la hora de agrupar las trazas mediante un modelo de aprendizaje no supervisado similar al utilizado en CMS.

Para realizar el filtrado de centroides reconstruidos correctamente, se calculó para cada centroide la distancia a todos los vértices reales. Estos datos se dispusieron en forma de matriz y se obtuvo el valor mínimo absoluto de la matriz para encontrar la primera asociación centroide-vértice correcta. Tras esto, se eliminaron los datos de dicho centroide y vértice y se repitió el proceso de forma iterativa hasta que el valor mínimo de la matriz de distancias superó un umbral de 3σ. De esta forma, se obtuvieron los centroides reconstruidos correctamente y los vértices de colisiones protón-protón que representan.

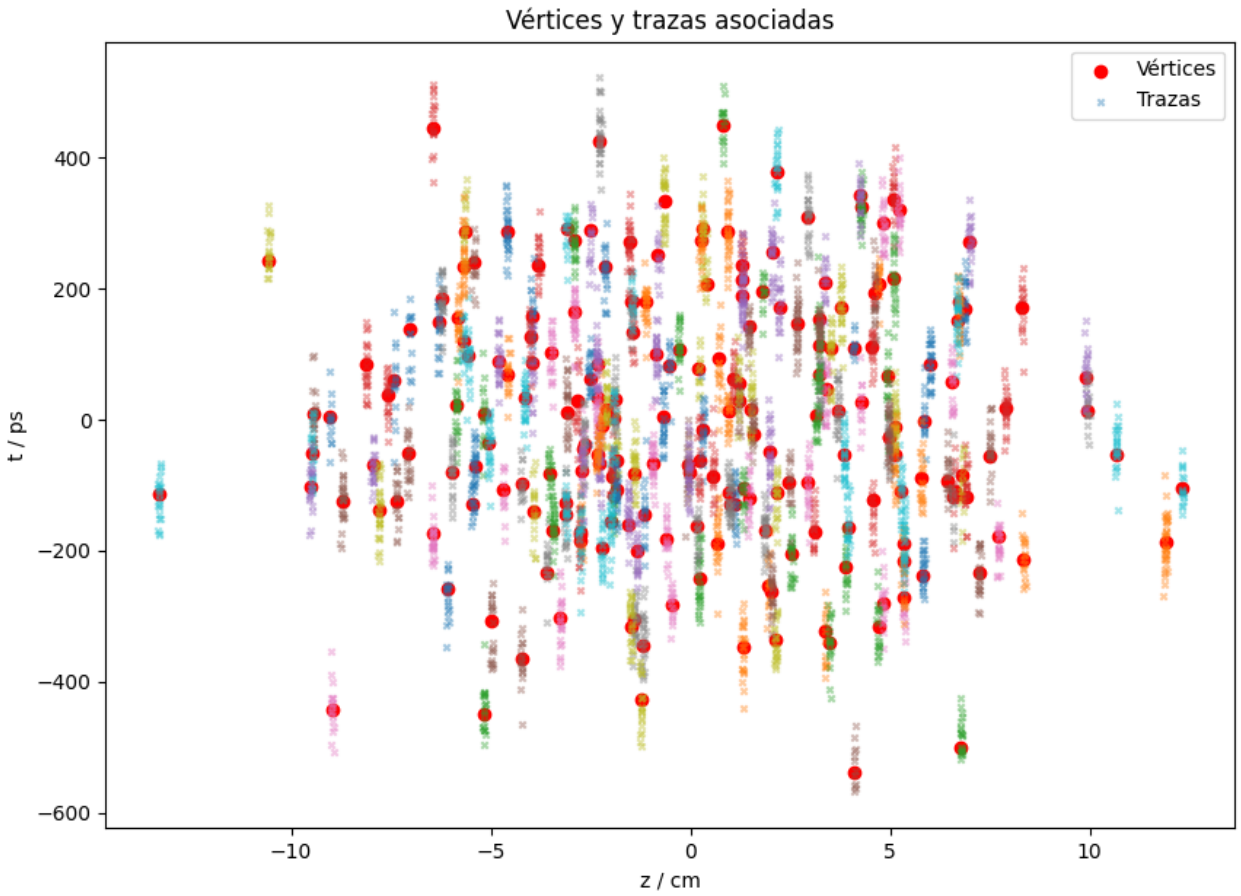

Figura 4.8: Representación en el plano (z, t) de los vértices reconstruidos (puntos rojos) y las trazas asociadas (cruces de colores). El valor esperado de la distribución de Poisson utilizado para generar el número de vértices es $\lambda = 200$. Número de vértices generados: 216

4.3.3. Eficiencia de la asignación de trazas

Tras haber analizado la eficiencia del *vertexing*, se eliminaron los centroides que no fueron bien reconstruidos y las trazas que se les asociaron. Una vez hecho esto, se compararon los datos reales de las trazas asociadas a los vértices de las colisiones protón-protón con las trazas asignadas a los centroides reconstruidos correctamente y se calculó la eficiencia total.

4.3.4. Resultados del algoritmo *k-means*

Tras haber realizado todo el proceso de agrupamiento mediante *k-means* y eliminar los datos que no cumplían con las condiciones impuestas para considerarse correctamente reconstruidos, se obtuvieron las eficiencias tanto de la reconstrucción de vértices como de la asignación de trazas a dichos vértices. En la figura 4.9, se muestra una representación de los centroides correctamente reconstruidos por el algoritmo de *clustering* utilizado junto con los vértices con los que se corresponden, así como las trazas asociadas a estos.

En este caso, se simularon un total de $N_{\text{total}} = 216$ vértices. No obstante, el número de centroides que se han generado y que han sido asignados a un vértice real, cumpliendo las condiciones

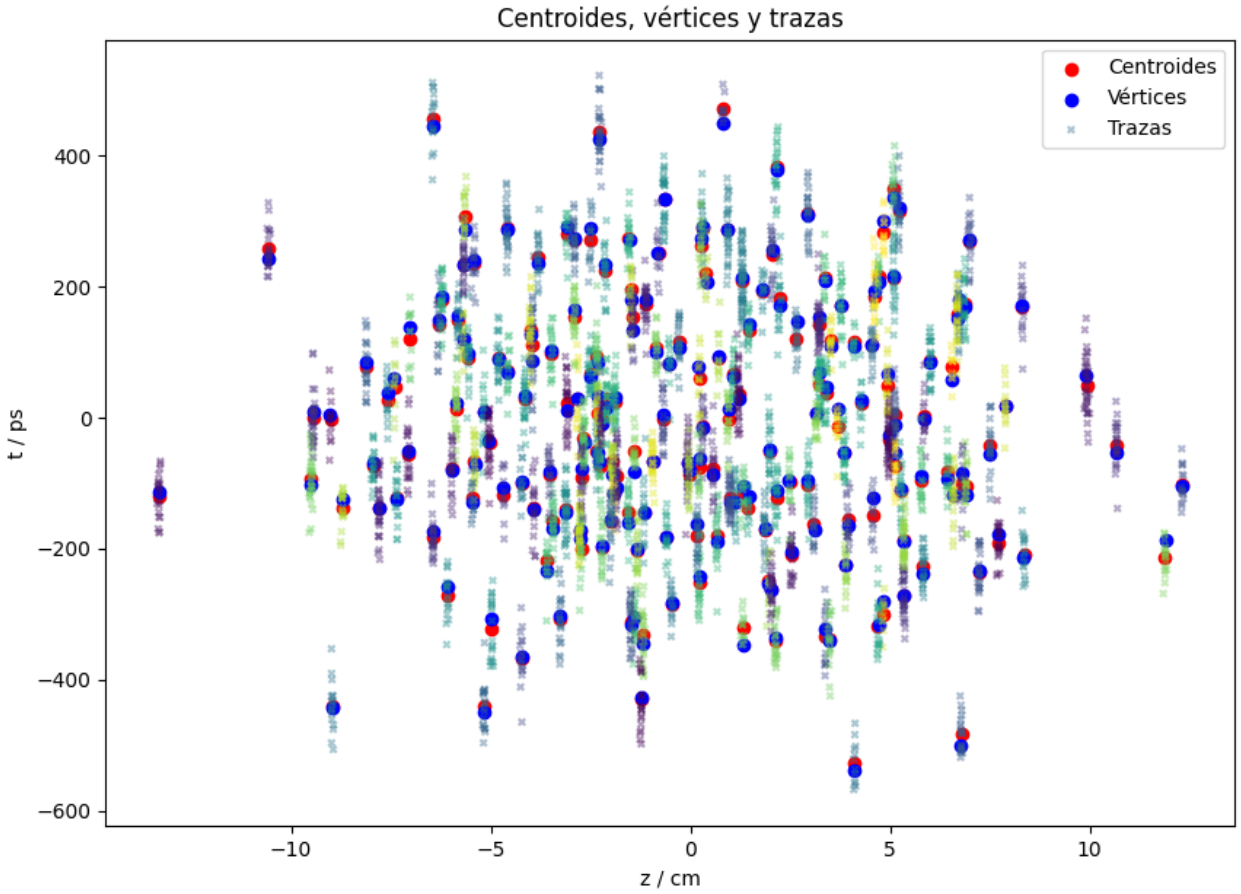

Figura 4.9: Distribución en el plano (z, t) de las trazas (cruces de colores), los vértices reales (puntos azules) y los centroides calculados mediante el algoritmo de k-means (puntos rojos). Cada clúster de trazas se asocia a un centroide estimado y este se corresponde con un vértice real.

impuestas, es de $N_{\text{correctos}} = 195$.

En cuanto a la eficiencia de las trazas, el simulador generó un total de $N_{\text{total}} = 4323$. No obstante, al eliminar 21 centroides debido a que no se correspondían con ningún vértice, se han eliminado las trazas que se habían asociado a estos. Por lo que, el número de trazas del que se parte para calcular la eficiencia es de $N_{\text{total}} = 3926$. Tras haber comparado las trazas asociadas a los centroides reconstruidos con las trazas de los vértices con los que se corresponden, se obtiene un valor de $N_{\text{correctos}} = 3637$ trazas correctamente asociadas.

Se ha realizado este proceso un total de $N = 10000$ para obtener las eficiencias de la reconstrucción de vértices. En la figura 4.10 se muestra la distribución de la eficiencia de los centroides bien reconstruidos. Se ha obtenido una eficiencia de $\eta = 0.83 \pm 0.03$, es decir, una eficiencia del $(83 \pm 3)\,\%$.

Por otra parte, en la figura 4.11 se muestra la distribución de la eficiencia de las trazas correctamente asociadas. Se ha obtenido una eficiencia de $\eta = 0.946 \pm 0.012$, es decir, una eficiencia del $(94.6 \pm 1.2)\,\%$.

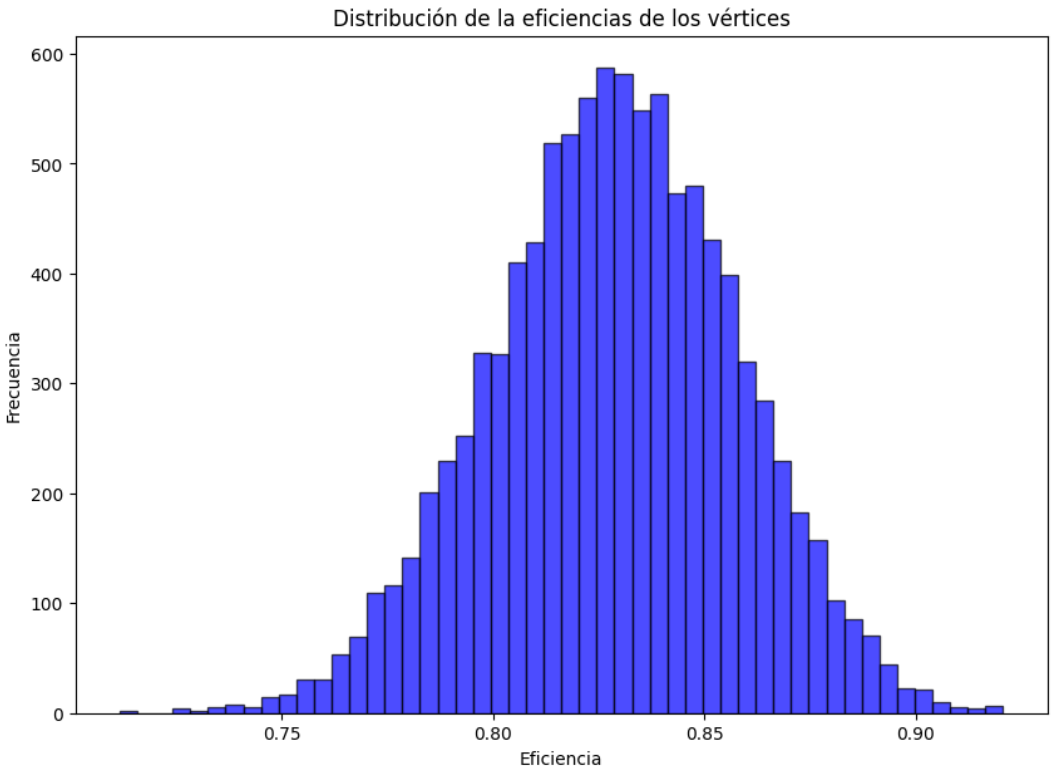

Figura 4.10: Distribución de la eficiencia de la reconstrucción de vértices

4.4. Corrección del tiempo en los centroides

Una vez se realizó toda la reconstrucción y filtrado de centroides, se corrigió la coordenada temporal t de estos. Esto es debido a que, al realizar toda la simulación y al aplicar todos los algoritmos, se supuso que todas las trazas pertenecían a piones, por lo que se utilizó el tiempo asociado a esta partícula para cada una de ellas. Sin embargo, se sabe que esto no es así. Para realizar la corrección temporal, se calculó la distancia de cada traza a su centroide con cada uno de los tres tiempos y se seleccionó el tiempo que minimizaba la distancia. Tras esto, se calculó el promedio temporal de las trazas de cada clúster y se le asoció su centroide.

Al igual que para el cálculo de las eficiencias de las trazas y los centroides mostrados en las figuras 4.10 y 4.11, se ha realizado este proceso un total de $N = 10000$ para obtener las eficiencias de la reconstrucción de vértices a partir de los centroides corregidos. En la figura 4.12 se muestra la distribución de la eficiencia de los centroides bien reconstruidos. Se ha obtenido una eficiencia de $\eta = 0.74 \pm 0.03$, es decir, una eficiencia del $(74 \pm 3)\,\%$.

No obstante, tras la corrección en la coordenada temporal, la eficiencia de la reconstrucción de los centroides disminuyó. No se sabe con certeza la razón de esta disminución.

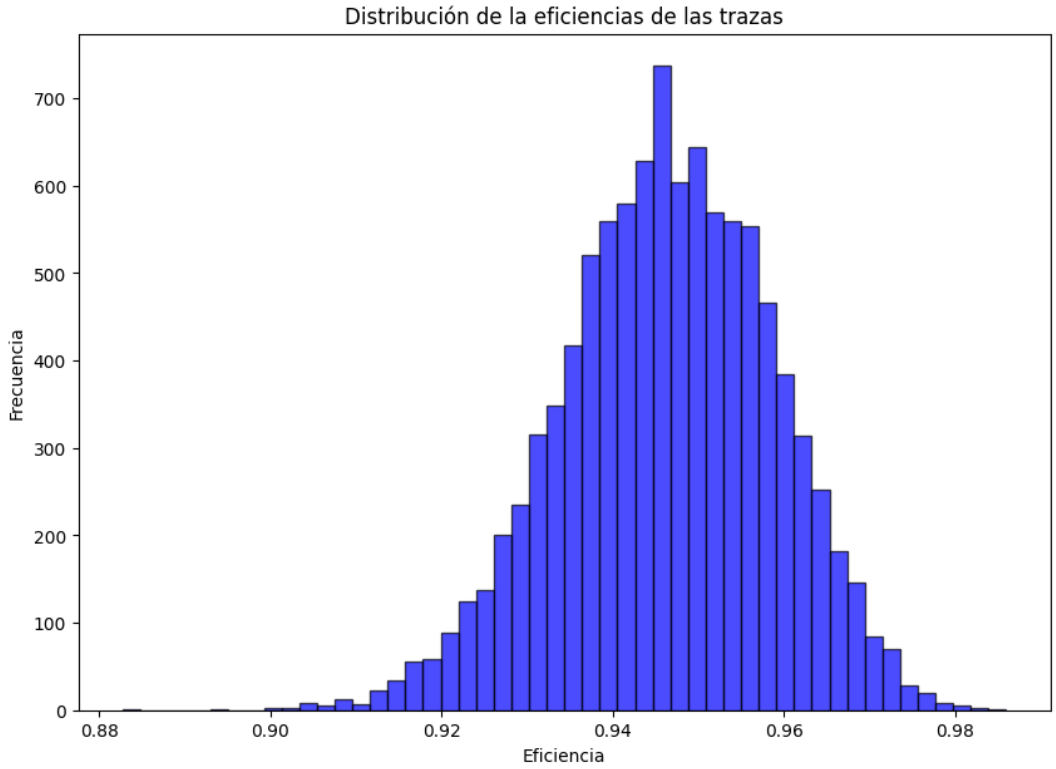

Figura 4.11: Distribución de la eficiencia de la asignación de trazas

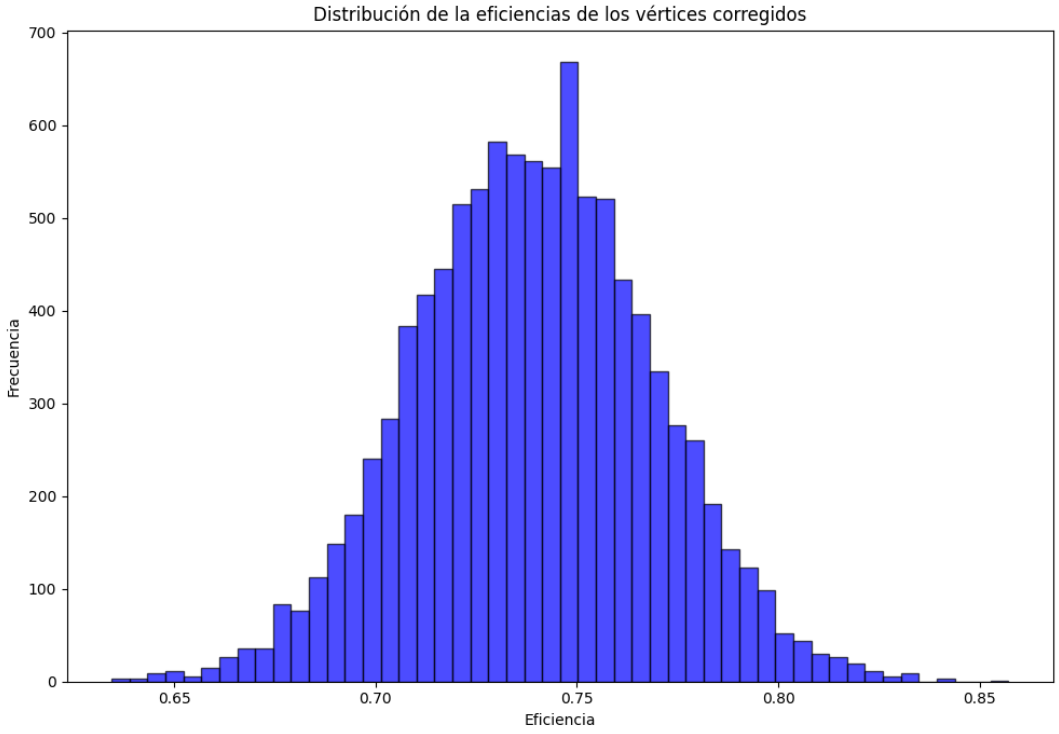

Figura 4.12: Distribución de la eficiencia de la reconstrucción de vértices a partir de los centroides corregidos

5 Redes neuronales de tipo grafo

Las redes neuronales de tipo grafo son un tipo de red neuronal que se especializan en el procesamiento de datos en forma de grafo. Un grafo está formado por un cúmulo de nodos y aristas, donde los nodos representan las entidades u objetos, mientras que las aristas representan las relaciones entre las entidades. La diferencia principal con respecto a otro tipo de redes neuronales es que estas trabajan sobre grafos, no sobre vectores, matrices o tensores. Es gracias a trabajar con esta estructura que estas redes son capaces de, no solo aprender a partir de las características de las entidades, sino de entender también la forma de relacionarse dichos objetos entre sí, permitiendo un aprendizaje más completo. Para redactar este capítulo, se ha utilizado la información recogida en el libro [16] y en el artículo [17].

5.1. Redes neuronales

Dentro del aprendizaje automático (*maching learning*), hay un tipo de modelos computacionales conocidos como redes neuronales. El objetivo de estas redes es aprender a asociar unos datos de entrada que se les den a las salidas que se buscan. Es decir, son algoritmos a los que se tiene que entrenar como si de una red neuronal biológica se tratara. De esta forma, estas redes consiguen desentrañar patrones y relaciones complejas entre los datos.

La unidad más simple de las redes neuronales son las neuronas o nodos. Estas son las que se encargan de hacer las operaciones básicas. Primero, se les asocian varios pesos ($n + 1$, siendo n el número de dimensiones de la entrada que recibe) y calculan la suma ponderada de los datos de entrada con sus pesos para, después, sumar un término de sesgo. Después, las neuronas aplican una función de activación no lineal para que la red aprenda patrones complejos.

Los pesos y los sesgos que se les asignan a las neuronas son parámetros que varían. Esta variación es la que constituye el proceso de aprendizaje de la red, que va ajustando estos parámetros hasta obtener la salida de datos deseada. Para que se dé este proceso de aprendizaje, las neuronas se disponen en capas. En la figura 5.1 se muestra un ejemplo de estructura típica de una red neuronal. La primera es la de entrada, la que recibe los datos. Tras esta, están las capas ocultas, a través de

las cuales los datos se ven alterados. En estas capas, cada neurona se encuentra conectada a todas las neuronas de la capa anterior, por lo que recolecta toda la información que contienen. Por último, se encuentra la capa de salida, que es la que genera los resultados finales.

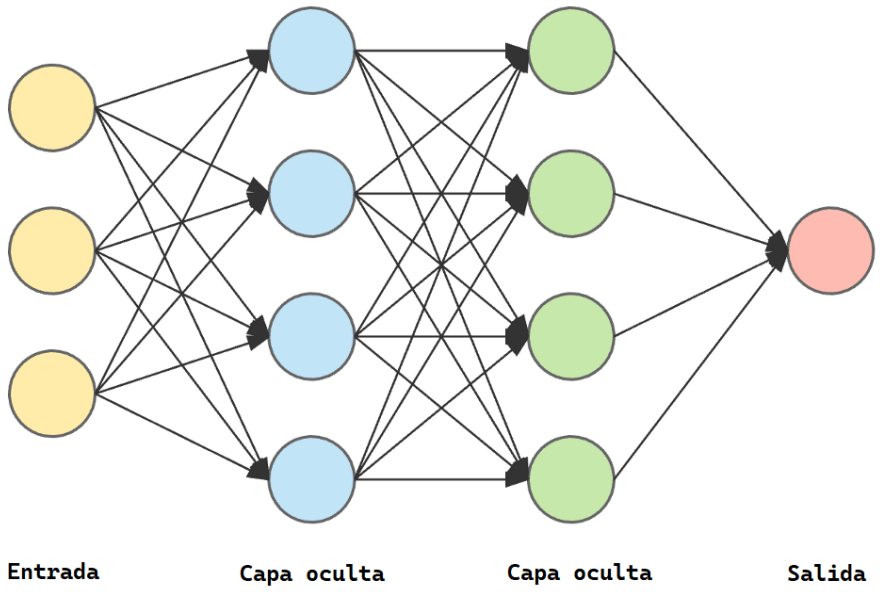

Figura 5.1: Estructura típica de una red neuronal. En este caso, consta de una capa de entrada, dos capas ocultas y una capa de salida [14]

El proceso de aprendizaje consiste en ajustar los parámetros de las neuronas de tal forma que la diferencia entre los valores reales de un conjunto de datos y las predicciones realizadas por la red sea mínima. Para cuantificar esta diferencia, se utiliza la función de pérdida (*Loss function*). Lo primero que hace la red es pasar los datos de entrada por todas las capas para generar una predicción. Luego, se utiliza el algoritmo de *Backpropagation*, que calcula el gradiente de la función de pérdida con respecto a los pesos y sesgos de las neuronas para, posteriormente, modificarlos con el fin de minimizar la función de pérdida. Por último, se utiliza un algoritmo de optimización, habitualmente el Descenso de Gradiente, que indica cómo actualizar los parámetros de las neuronas.

Todo este proceso se repite de forma iterativa un número determinado de veces. A medida que se repite, la red aprende y mejora cada vez más sus predicciones.

5.2. Grafos

Un grafo, tal y como se ha explicado, es un conjunto de nodos y aristas en los que los nodos representan las entidades que se quieren tratar (en el caso de este trabajo serían las partículas) mientras que las aristas conectan pares de vértices, es decir, indican las conexiones entre las entidades. Un grafo se denota como $G = (V, E)$ donde V es el conjunto de vértices y E el conjunto de aristas. Por otro lado, se puede definir el orden del grafo, n, a partir del número de nodos de este y el tamaño del grafo, m, a partir del número de aristas.

5.2.1. Direccionalidad en grafos

Dentro de los grafos, se puede hacer una distinción entre ellos en función de si son grafos no dirigidos o grafos dirigidos.

Por un lado, en los grafos no dirigidos las aristas conectan pares no ordenados de vértices. Esto quiere decir que, si se tiene vértice A y un vértice B, entonces $(A, B) = (B, A)$. En la figura 5.2 se puede observar un ejemplo de grafo no dirigido. Este tipo de grafos es bidireccional, por lo que no hay definido un inicio o un final de la conexión entre nodos.

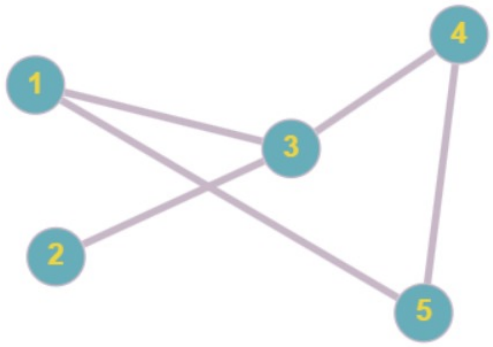

Figura 5.2: Ejemplo de grafo no dirigido [15]

Por otro lado, en los grafos dirigidos, las aristas conectan pares ordenados de vértices. Volviendo al par de nodos A y B del ejemplo anterior, si una arista es representada como (A, B), implica que el vértice de inicio es A y el del final es B. En este caso, $(A, B) \neq (B, A)$ ya que la dirección es relevante. En la figura 5.3 se puede observar un ejemplo de grafo dirigido.

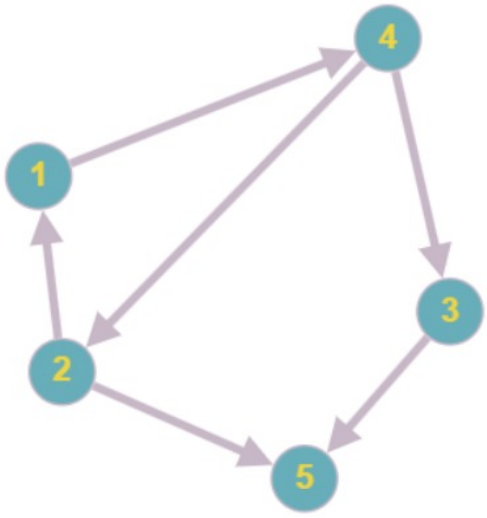

Figura 5.3: Ejemplo de grafo dirigido [15]

5.3. Redes neuronales de tipo grafo

Dentro del ámbito de las redes neuronales, existe un tipo específico de red denominada red neuronal de tipo grafo (GNN, por sus siglas en inglés). Como ya se ha explicado, la característica principal de este tipo de redes es que los datos que manejan tienen una estructura de tipo grafo. Este tipo de datos, con una estructura basada en relaciones y conexiones entre nodos, es más complejo pues no tiene una estructura regular. Esto provoca que las redes neuronales convencionales no sean capaces de trabajar con grafos, ya que éstas reciben los datos de entrada organizados en tablas o matrices.

Dado que las redes neuronales clásicas no son capaces de operar eficientemente con este tipo de datos, se utilizan las GNN. Una GNN recibe como entrada un grafo en el que los datos se encuentran distribuidos en los nodos y aristas, y genera una salida de datos con la misma estructura. Esto quiere decir que el grafo resultante cuenta con la misma conectividad que el de entrada. No obstante, los *embeddings*, que son representaciones vectoriales de las características de los nodos y aristas del grafo, se modifican progresivamente.

5.3.1. *Passing message neural netowrk*

El mecanismo de procesamiento de datos que siguen este tipo de redes neuronales, las GNN, es el de *passing neural message netowrk*. Este consiste en transformar gradualmente la representación de un nodo en función de las características de sus nodos adyacentes. Lo primero que sucede al llevarse a cabo este proceso es que cada nodo recoge las características de sus nodos adyacentes. Cada vecino calcula un "mensaje" a partir de sus propias características, las del nodo que recibirá el mensaje y las de la arista que los conecta. Una vez todos los vecinos de un nodo han calculado estos mensajes, se agrupan todos en un mismo vector a partir de una función de agregación. Esta función de agregación podría ser, por ejemplo, el sumatorio de todos los mensajes. Por último, la representación o *embedding* del nodo se modifica combinando la agrupación de mensajes de sus vecinos con su *embedding* anterior. Para realizar esta combinación, se utiliza una función de actualización.

5.3.2. Composición de capas

El proceso de *Passing message neural network* descrito, toma en cuenta únicamente los nodos adyacentes al que se busca modificar, es decir, los primeros vecinos. No obstante, se pueden agregar más capas a la red con el objetivo de recolectar más datos para un mismo nodo. Por ejemplo, en una segunda capa, cada nodo incorporaría a su representación la información de sus primeros vecinos, los cuales han incorporado a su vez la información de sus propios nodos adyacentes. Es decir, la representación actualizada del nodo incluye la información de nodos de hasta dos aristas de distancia. Por lo que, para un número N de capas, un nodo actualiza su *embedding* a partir de la información de nodos que estén a N aristas de distancia.

6 Aplicación de una red neuronal de tipo grafo al vertexing de CMS

En este capítulo, se explica con detalle el proceso llevado a cabo para desarrollar una red neuronal de tipo grafo (GNN, por sus siglas en inglés). Esta red es la alternativa propuesta en este trabajo para la reconstrucción de vértices que tiene lugar en el experimento CMS del LHC. La red neuronal ha sido desarrollada en Python [11] y utiliza los datos generados por la simulación de eventos de colisión protón-protón descrita en el capítulo 4. Para ello, se ha utilizado la librería *PyTorch* disponible en Python [16]. Más concretamente, se ha utilizado el paquete de *PyTorch Geometric* [17]. El código de esta red neuronal se encuentra en el archivo GNN.ipynb, disponible en el repositorio del proyecto [12].

6.1. Generación de datos

Tal y como se ha indicado, se sigue el mismo procedimiento que el que se ha detallado en el capítulo 4 para la simulación de datos. No obstante, los datos generados en este caso son para introducirlos en una red neuronal de tipo grafo. Como ya se ha explicado en el capítulo 5, este tipo de redes trabaja con datos en estructura de grafos, por lo que se han de adaptar los eventos simulados a este tipo de topología. Para ello, se generan las diferentes variables que se le van a presentar como datos de entrada a la red: los atributos de los nodos, las aristas del grafo, los atributos de las aristas y las etiquetas de los nodos.

Atributos de los nodos

Esta variable representa las características de los nodos. En este caso, cada nodo representa una traza, por lo que las características de este son la coordenada espacial z y las tres coordenadas temporales t de cada una de las partículas que se tienen en cuenta en el simulador de eventos: piones, kaones y protones. Al igual que el resto de variables que requiere la red, esta es un tensor. Cada fila de este corresponde a un nodo, mientras que cada columna, a una característica.

Aristas del grafo

Esta variable representa la conectividad del grafo, es decir, indica cómo están conectados los nodos. Define las aristas del grafo que conectan diferentes nodos como pares de índices. Para conectar dos nodos (trazas), estos han de estar a una distancia relativa por debajo de 3σ entre ellos. Al basarse en un criterio de cercanía, un par de nodos (A, B) que se conecten porque A esté cerca de B también va a cumplirse de forma inversa, ya que B también estará cerca de A. Esto quiere decir que se cumple la relación $(A, B) = (B, A)$ por lo que el grafo representado es no dirigido.

Atributos de las aristas

Este tensor contiene los atributos de las aristas. Es decir, contiene información extra acerca de la relación entre dos nodos que están conectados. En este caso, se usa como atributo la distancia inversa entre dos nodos. De esta forma, cuanto menor sea la distancia entre estos, mayor será el atributo que los una. Es una forma de modular la importancia de la información que transmita un nodo vecino.

Etiquetas de los nodos

En este último tensor se recogen las etiquetas de los nodos. Estas representan la salida esperada de cada nodo utilizado a la hora de entrenar la red neuronal. En este caso, es el índice del vértice primario al que corresponde cada traza. De esta forma, se busca que la red sea capaz de clasificar los nodos (trazas) en función de a qué vértice primario corresponden.

6.2. Función de pérdida

Una vez se generan los datos siguiendo la estructura necesaria para ser introducidos en una red neuronal de tipo grafo, se desarrolló la función de pérdida o *loss function*. Tal y como se ha explicado en el capítulo 5, la función de pérdida cuantifica en cierta forma el error cometido por la predicción de la red en comparación con el resultado esperado real en un grupo de datos de entrenamiento. A partir de la cuantificación de dicho error, la red es capaz de aprender, ajustando los pesos de los nodos con el fin de minimizar esta función, obteniéndose resultados más precisos.

En este caso, se ha desarrollado una función de pérdida que clasifica las trazas en grupos en donde todas han sido generadas a partir del mismo vértice primario. La función busca este agrupamiento de trazas que realmente formen un clúster a la vez que trata de separar las salidas de la red para trazas que pertenezcan a otro clúster.

Para ello, la función ha sido desarrollada de tal forma que el modelo minimice la diferencia entre la salida de diferentes nodos de un mismo grupo y maximice la de nodos de diferente grupo. Para ello, se define dentro de la propia función dos términos de pérdida diferentes: el que fuerza la cohesión entre trazas del mismo clúster y el que obliga a que la salida de la red para trazas de distintos clústeres sea diferente.

Pérdida intra-clúster

Este término fuerza a los nodos de un mismo clúster a que se les asocie la misma salida.

$$L_{\text{intra}} = \frac{1}{N_{\text{same}}} \sum_{(i,j)\in C} \|output_i - output_j\|^2 \tag{6.1}$$

En esta ecuación, $output_i$ es la salida de la red neuronal para el nodo o traza i, C es el conjunto de pares de índices de nodos que pertenecen a un mismo clúster y N_{same} es el número total de pares de nodos del grupo.

Pérdida inter-clúster

Este término provoca que los nodos de diferentes grupos sean asociados a salidas diferentes.

$$L_{\text{inter}} = \frac{1}{\frac{1}{N_{\text{different}}} \sum_{(i,j)\in D} \|output_i - output_j\|^2} \tag{6.2}$$

Al igual que en la anterior ecuación, $output_i$ es la salida de la red neuronal para el nodo o traza i. Sin embargo, D es el conjunto de pares de nodos de diferentes grupos y $N_{different}$ es el número total de pares de nodos diferentes.

Finalmente, el término de pérdida total de la función es:

$$L_{total} = L_{inter} + L_{intra} \tag{6.3}$$

6.3. Red neuronal de tipo grafo

Tras haber generado los datos y desarrollado la función de pérdida, se diseñó e implementó la red neuronal de tipo grafo. La función principal de esta red es transformar las características asociadas a cada traza en representaciones numéricas con el fin de facilitar la búsqueda de relaciones o patrones.

Como ya se ha explicado antes, al usarse una GNN, esta toma como entrada un grafo que representa la conexión entre trazas. Cada nodo representa una traza y las relaciones entre ellas se describen a través de las aristas, ponderadas en base a la distancia inversa. Cada nodo viene caracterizado por 4 componentes: la coordenada espacial y las tres coordenadas temporales.

La red neuronal desarrollada consta de tres capas, cada una con un número diferente de neuronas. La primera de las tres capas es la que recibe los datos de entrada. Esta recibe vectores de entrada de 4 dimensiones y los transforma en vectores de dimensión 32. Tras esto, se aplica una función de activación, en este caso la función ReLU, para evitar la linealidad y mejorar la capacidad de aprendizaje. Los *embeddings* generados pasan a la segunda capa de neuronas, que transforma los vectores de 32 dimensiones en vectores de 16 y se vuelve a aplicar la función de activación. Finalmente, los datos pasan por la tercera y última capa, en la que se reduce el vector de 16 dimensiones a una sola dimensión. Este último *embedding* de una dimensión es la salida de la red y es el que se utiliza para calcular la pérdida.

La razón por la que la salida de la GNN es de una sola dimensión es el intento de facilitar el proceso de *clustering* posterior. Se busca asignar una coordenada a cada nodo cuyo valor coincida con el resto de nodos que pertenezcan al mismo vértice primario. De esta forma, cada grupo de

trazas estaría caracterizado por un valor numérico diferente, facilitando el proceso de agrupamiento mediante algoritmos como el explicado anteriormente, el *k-means*.

6.3.1. Entrenamiento de la red

Una vez se estructuró la red neuronal, se entrenó para obtener salidas para cada nodo que ayudasen con el proceso de reconstrucción de vértices. Para ello, se le pasaron a la red 10 conjuntos de datos de entrenamiento, a partir de los cuales esta ajustaría sus parámetros para minimizar la función de pérdida.

Los datos de entrenamiento atraviesan toda la arquitectura de capas de la red múltiples veces. A medida que estos datos atraviesan la GNN, se va actualizando la información de cada nodo en función de la suya propia y la de sus vecinos. Esto sucede en cada capa hasta que los datos llegan a la capa de salida, donde sufren la última modificación antes de obtenerse la salida o *output* de la red para cada nodo. En cada iteración, la red varía sus parámetros en función del error cometido en la asignación final a la hora de agrupar trazas. Este error lo cuantifica la función de pérdida.

6.3.2. Reconstrucción de vértices

Una vez la red terminó de entrenarse con los conjuntos de datos que se le suministraron, se generó un nuevo *set* de datos con el fin de comprobar la eficiencia del modelo para la reconstrucción de vértices. Para ello, se introdujo este nuevo *set* en el modelo ya entrenado y, una vez este asignó cada nodo una representación numérica, se utilizó de nuevo el algoritmo de *clustering*, *k-means*, descrito en el capítulo 4 con el fin de agrupar las salidas de la red.

Los vértices reales están representados en dos dimensiones, z y t, mientras que la salida de la red es de una dimensión que no tiene por qué corresponderse con las coordenadas espacial y temporal. Es por ello que al haber aplicado el algoritmo *k-means* y haber obtenido los centroides de las trazas, estos no se pueden comparar con los vértices directamente, ya que están representados en diferentes dimensiones y coordenadas.

6.3.3. Eficiencia de la red neuronal

Para comparar los vértices reales con los centroides reconstruidos, se hizo la media de las posiciones espacial y temporal de cada grupo de trazas asignadas a un centroide. De esta forma, se obtuvo un centroide para cada clúster en las dos dimensiones y sistema de coordenadas en los que están los vértices reales y se pueden comparar.

Tal y como se hizo para el procedimiento detallado en el capítulo 4, se generó una matriz en la que se recogían las distancias entre vértices y centroides. A partir de estas distancias, se filtraron los centroides bien reconstruidos imponiendo una distancia máxima umbral. De esta forma, se obtuvieron pares de centroide-vértice que estuviesen a una distancia inferior a 3σ.

6.3.4. Resultados de la red neuronal de tipo grafo

Tras haber realizado todo el procedimiento descrito y haber obtenido los centroides bien reconstruidos en dos dimensiones, se calculó la eficiencia de esta reconstrucción. En la figura 6.1 se muestra una representación de los centroides en dos dimensiones bien reconstruidos y los vértices reales.

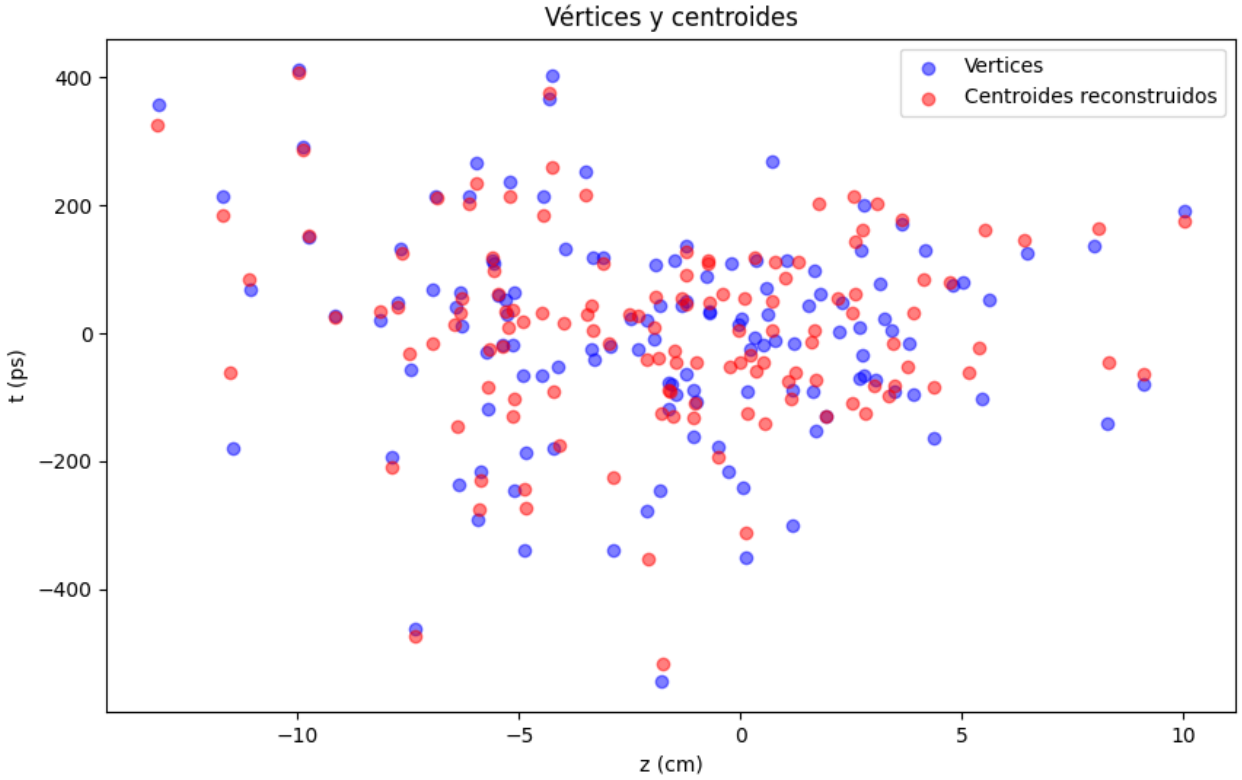

Figura 6.1: Distribución en el plano (z, t) de los centroides reconstruidos correctamente y de los vértices que encajan con estos.

En el caso de la figura 6.1, se simularon un total de $N_{total} = 199$ vértices. Sin embargo, el número de centroides generados que han sido asignados a un vértice real, pasando por los filtros impuestos, es de $N_{correctos} = 130$.

Se ha calculado la eficiencia de la GNN repitiendo el proceso un total de $N = 1000$. En la figura 6.2 se muestra la distribución de la eficiencia de los centroides bien reconstruidos por la red neuronal. Se ha obtenido una eficiencia de $\eta = 0.69 \pm 0.04$, es decir, una eficiencia del $(69 \pm 4)\,\%$

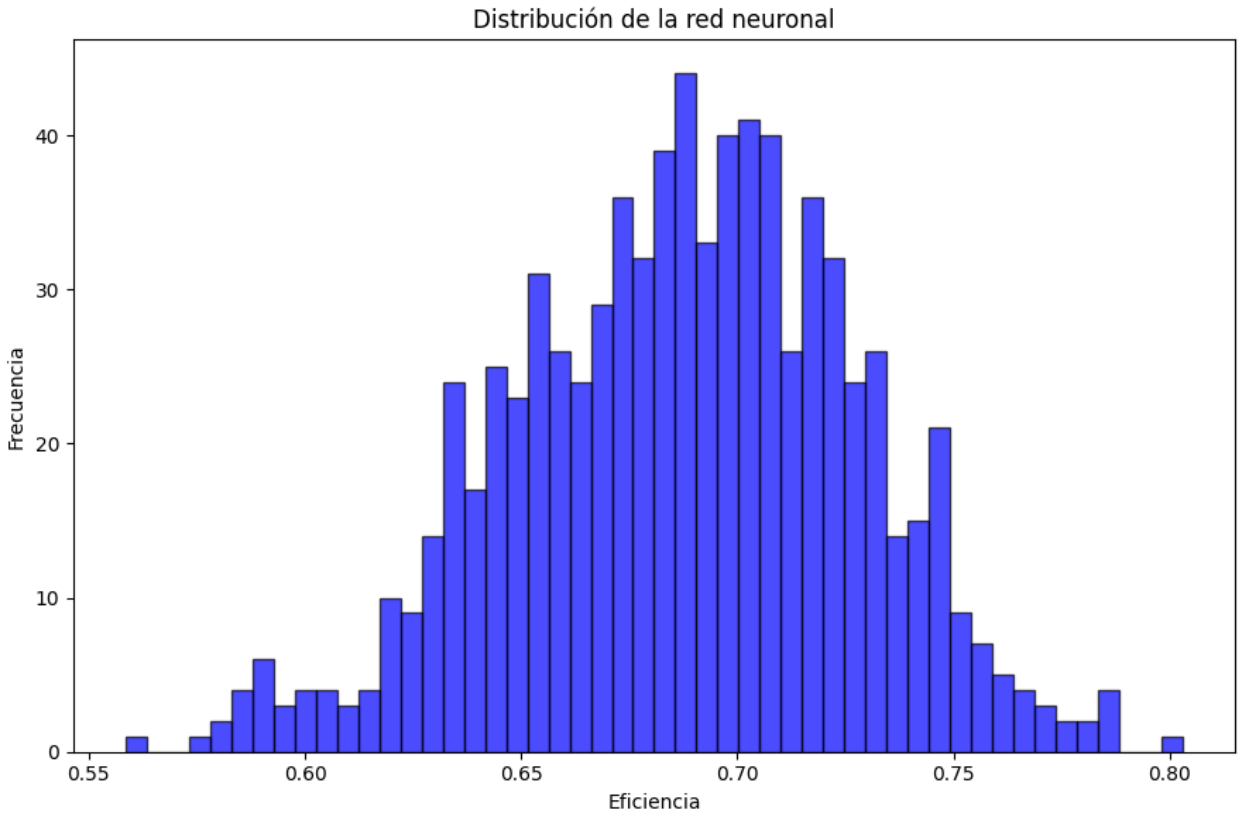

Figura 6.2: Distribución de la eficiencia de la reconstrucción de vértices mediante la red neuronal de tipo grafo

7 Conclusiones

En este trabajo, se ha abordado el problema de la reconstrucción de vértices primarios de los eventos protón-protón que se miden en el detector CMS, en el LHC. Este proceso se lleva a cabo mediante las trazas medidas en uno de los subsistemas del detector, el *tracker*. Concretamente, se ha desarrollado un programa que simula los datos recogidos en este y, posteriormente, se ha realizado una réplica simplificada del algoritmo que se emplea en CMS para la reconstrucción de vértices. Finalmente, se desarrolló una red neuronal de tipo grafo (GNN) para comparar el rendimiento de esta con el modelo clásico.

A partir del algoritmo simplificado de CMS, se ha establecido la eficiencia base que se busca mejorar mediante la red neuronal. La red desarrollada representa las trazas de partículas que se buscan agrupar como un conjunto de nodos que conforman un grafo donde las relaciones entre estos vienen dadas por aristas con diferentes pesos en función de la cercanía. El modelo ha sido entrenado con una función de pérdida diseñada específicamente para agrupar trazas que provengan de un mismo vértice a la vez que separa las que provienen de distintos.

Una vez la red fue entrenada con varios conjuntos de datos de prueba, se evaluó la eficiencia de la misma sobre nuevos datos. A la hora de comparar los resultados, la red neuronal desarrollada obtuvo una eficiencia media del $(69 \pm 4)\,\%$ frente al $(83 \pm 3)\,\%$ del algoritmo clásico simplificado de CMS. Cabe destacar que la eficiencia se ha definido como el porcentaje de centroides correctamente reconstruidos en cada evento, dando por válido aquel cuya distancia al vértice al que correspondiese fuese menor que 3σ.

Tal y como se observa en los resultados obtenidos, la eficiencia de la red neuronal desarrollada es inferior a la del algoritmo clásico simplificado. Esto puede haberse debido a diversos factores. Por ejemplo, un modelo de GNN demasiado simple o una cantidad de datos de entrenamiento demasiado limitada. También es posible que la función de pérdida desarrollada no sea la más óptima para el entrenamiento. O podría ser la falta de técnicas de *clustering* específicas para agrupar de forma correcta la salida de la red.

No obstante, pese a que la eficiencia de la red neuronal desarrollada haya sido inferior, los resul-

tados muestran que es capaz de aprender a agrupar trazas. Esto podría indicar que, en un futuro, se pueda desarrollar una GNN con mayores capacidades de *clustering*, aumentando su complejidad, ajustando más la función de pérdida para que sea más eficiente y aumentando la cantidad de datos de entrenamiento.

Este trabajo abre las puertas a nuevas vías de investigación sobre el uso de aprendizaje automático para problemas de *clustering* y reconstrucción de vértices en física de partículas.

Referencias

[1] CERN Collaboration. (2008). *The CERN Large Hadron Collider: Accelerator and Experiments*. 3(S08001). https://jinst.sissa.it/LHC/LHCmachine/2008$_J$$INST_{3S}$08001.*pdf*

[2] I. Béjar Alonso et al. (eds.), *High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report*, CERN Yellow Reports: Monographs, CERN-2020-010, CERN, 2020. https://cds.cern.ch/record/2749422/files/127-117-PB.pdf

[3] Villatoro, F. R. (2012, 25 de julio). El pile-up (apilado de colisiones) en el LHC del CERN. *La Ciencia de la Mula Francis (Naukas.com)*. Recuperado el [20 de mayo de 2025], de https://francis.naukas.com/2012/07/25/el-pile-up-apilado-de-colisiones-en-el-lhc-del-cern/

[4] CMS Collaboration. (2019, 19 de marzo). *A MIP Timing Detector for the CMS Phase-2 Upgrade* (CERN-LHCC-2019-003, CMS-TDR-020). CERN. https://cds.cern.ch/record/2667167/files/CMS-TDR-020.pdf

[5] Tao Yang, Ben Sekely, Yashas Satapathy, Greg Allion, Philip Barletta, Carl Haber, Steve Holland, John F. Muth, Spyridon Pavlidis, Stefania Stucci. (2024, 11 de septiembre). *Characterization of 4H-SiC Low Gain Avalanche Detectors (LGADs)*. arXiv:2408.12744. https://arxiv.org/pdf/2408.12744

[6] CERN. (s.f.). *CERN accelerating science. Facts and figures about the LHC*. https://home.cern/resources/faqs/facts-and-figures-about-lhc

[7] CMS Collaboration. (2008, 14 de agosto). *The CMS experiment at the CERN LHC*. https://iopscience.iop.org/article/10.1088/1748-0221/3/08/S08004/pdf

[8] CMS Collaboration. (2017, 6 de octubre) *Particle-flow reconstruction and global event description with the CMS detector*. https://iopscience.iop.org/article/10.1088/1748-0221/12/10/P10003/pdf

[9] CMS Collaboration. (2017, 1 de julio) *The Phase-2 Upgrade of the CMS Tracker*. https://cds.cern.ch/record/2272264/files/CMS-TDR-014.pdf

[10] CMS Collaboration. (2014, 16 de octubre) *Description and performance of track and primary-vertex reconstruction with the CMS tracker*. https://oar.princeton.edu/bitstream...

[11] Python Software Foundation. (2023). Python Language Reference, version 3.12. https://www.python.org

[12] David Abad Riaño, *TFG-vertexing: Cuadernos Jupyter del Trabajo de Fin de Grado*, GitHub, 2025. https://github.com/33david333/TFG-vertexing

[13] Scikit-learn Developers. (2024, 5 de junio). *sklearn.cluster.KMeans — scikit-learn 1.4.2 documentation*. Recuperado el [9 de junio de 2025]. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[14] Pablo Huet. (2023, 13 de abril). Qué son las redes neuronales y sus aplicaciones. Recuperado el [9 de junio de 2025]. https://openwebinars.net/blog/que-son-las-redes-neuronales-y-sus-aplicaciones/

[15] *Graph Online*. Recuperado el [9 de junio de 2025]. https://graphonline.top/en/

[16] Stevens, E., Antiga, L., y Viehmann, T., *Deep Learning with PyTorch: Build, Train, and Tune Neural Networks Using Python Tools*, Manning Publications, 2020. https://www.manning.com/books/deep-learning-with-pytorch

[17] Fey, M., & Lenssen, J. E. (2019). Fast Graph Representation Learning with PyTorch Geometric. Recuperado el [7 de junio de 2025]. https://arxiv.org/abs/1903.02428