Universidad Zaragoza

Trabajo Fin de Grado

Búsqueda aproximada por similitud mediante estructuras de grafos avanzadas

Autor

Miguel Ángel Royo Miguel

Directores

Daniel Huici Meseguer
Ricardo Julio Rodríguez Fernández

Grado en Ingeniería Informática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza
Septiembre de 2026

Agradecimientos

A Ricardo y Daniel, por guiarme durante todo el proceso.
A mi familia, por su apoyo incondicional.
A mis amigos, y en especial a aquellos que me han acompañado todos estos meses en el laboratorio.

Declaración de uso responsable de IA

En este trabajo se han utilizado los modelos Claude Sonnet 5, Claude Opus 4.8 y Claude Opus 5 para la generación de diagramas, la programación de algunos *scripts* y tests, y la depuración y detección de errores. Todo el contenido generado por esos modelos ha sido revisado y modificado por el autor, quien asume plena responsabilidad sobre el contenido íntegro de este trabajo, incluidas aquellas partes en cuya elaboración se haya hecho uso de las herramientas mencionadas.

Resumen

El crecimiento del volumen de datos digitales ha incrementado la complejidad de las investigaciones forenses, en las que es necesario examinar y comparar grandes cantidades de artefactos digitales para localizar evidencias relevantes. Con este propósito surge `APOTHEOSIS`, una herramienta que combina algoritmos de similitud aproximada con técnicas de búsqueda aproximada de vecinos más próximos, y que emplea un grafo jerárquico de pequeño mundo navegable (del inglés, *Hierarchical Navigable Small World*, HNSW) como estructura de indexación. HNSW presenta, sin embargo, limitaciones conocidas: la jerarquía de capas incrementa el tamaño del índice y el descenso voraz no garantiza un camino monótono hacia el elemento buscado, por lo que los recorridos de búsqueda pueden ser más largos de lo necesario.

Este trabajo extiende `APOTHEOSIS` con *Navigating Spreading-out Graph* (NSG), un grafo de una sola capa cuya regla de selección de aristas busca garantizar la existencia de caminos monótonos desde un único punto de entrada. Puesto que las implementaciones existentes de NSG presuponen datos vectoriales y métricas como la distancia euclídea, se ha desarrollado una implementación propia en Rust que permite comparar cualquier tipo de dato, requiriendo únicamente una función de similitud.

La evaluación se ha realizado sobre dos conjuntos de datos forenses reales, ficheros de código fuente y páginas de memoria de binarios de sistema de Windows, indexando hasta un millón de elementos. Los resultados muestran que, a igualdad de grado de salida, el tamaño del índice NSG es entre un 20 % y un 30 % menor que el de HNSW y ofrece tiempos de consulta inferiores, a costa de un tiempo de construcción mayor.

Abstract

The growth in the volume of digital data has increased the complexity of digital forensic investigations, in which large amounts of digital artifacts must be examined and compared in order to locate relevant evidence. It is with this purpose that `APOTHEOSIS` was developed, a tool that combines approximate similarity algorithms with approximate nearest neighbor search techniques, and that uses a *Hierarchical Navigable Small World* (HNSW) graph as its indexing structure. HNSW has, however, well-known limitations: the layered hierarchy increases the size of the index and the greedy descent does not guarantee a monotonic path towards the queried element, so search traversals may be longer than necessary.

This work extends `APOTHEOSIS` with the *Navigating Spreading-out Graph* (NSG), a single-layer graph whose edge selection rule aims to guarantee the existence of monotonic paths from a single entry point. Since existing NSG implementations assume vector data and metrics such as the Euclidean distance, a custom implementation has been developed in Rust that can compare any type of data, requiring only a similarity function.

The evaluation has been carried out on two real forensic datasets, source code files and memory pages of Windows system binaries, indexing up to one million elements. The results show that, for the same out-degree, the NSG index takes up between 20 % and 30 % less space than the HNSW one and offers lower query times, at the cost of a considerably higher construction time.

Índice

Capítulo 1

Introducción

1.1. Motivación

El crecimiento exponencial de los datos digitales ha incrementado la complejidad de las investigaciones forenses digitales. Los analistas deben examinar grandes cantidades de datos en cada fichero, aplicación, *log* de sistema o paquete de red con el objetivo de encontrar evidencias digitales (también llamadas artefactos digitales) relevantes para identificar la naturaleza, ámbito e impacto de un incidente de seguridad. Con la idea de ayudar en esta tarea surge `APOTHEOSIS` (*APproximaTe searcH systEm Of Similarity dIgeSts*) [1], una herramienta diseñada para identificar y comparar de forma eficiente similitudes en grandes conjuntos de datos.

`APOTHEOSIS` combina dos tecnologías inherentemente interconectadas, como son las técnicas de búsqueda aproximada y los algoritmos de similitud aproximada. Las técnicas de búsqueda aproximada se basan en encontrar elementos que se aproximen a una consulta dada, relajando la restricción de encontrar una coincidencia exacta [2]. De esta forma se consigue acelerar la búsqueda, lo cual resulta especialmente interesante cuando se trabaja con grandes conjuntos de datos. Los algoritmos de similitud aproximada permiten detectar similitudes entre artefactos digitales. Se entiende por artefacto cualquier secuencia de *bytes* (por ejemplo, un fichero) que posea una interpretación significativa [3]. Estos algoritmos pueden entenderse como una generalización de la búsqueda de objetos idénticos mediante *hashing* criptográfico, un enfoque muy común en áreas como la seguridad y el análisis forense: en vez de devolver una respuesta binaria $\{0, 1\}$ como resultado de una comparación, devuelven un valor en una escala continua, interpretado como una medida de similitud.

Actualmente, `APOTHEOSIS` emplea un grafo jerárquico de pequeño mundo navegable

(del inglés, *Hierarchical Navigable Small World*, HNSW) [4] como estructura de datos para organizar y buscar de forma eficiente en grandes conjuntos de datos. No obstante, HNSW presenta limitaciones conocidas: la estructura jerárquica introduce sobrecarga de memoria, y el elevado grado de salida de los nodos y la longitud de los caminos de búsqueda penalizan el tiempo de consulta [5].

1.2. Objetivos y metodología

El objetivo de este trabajo es extender la implementación existente de `APOTHEOSIS`, tratando de atajar las limitaciones de HNSW. Para ello, tras un estudio del estado del arte de la búsqueda aproximada de vecinos más próximos [6, 7] y el análisis de la implementación existente de `APOTHEOSIS`, se pretende integrar *Navigating Spreading-out Graph* (NSG) [5] como método alternativo de búsqueda aproximada. Finalmente, se realiza una evaluación del rendimiento de NSG frente a HNSW en diversos escenarios.

Este trabajo se ha llevado a cabo en el marco de unas prácticas extracurriculares en el grupo de investigación DisCo de la Universidad de Zaragoza, bajo la supervisión de los directores del trabajo e integrándose en las dinámicas de los miembros de dicho grupo.

1.3. Estructura del documento

Este trabajo se divide en seis capítulos y un anexo. En el Capítulo 2 se introducen los conceptos previos necesarios para entender la herramienta desarrollada y los experimentos realizados. El Capítulo 3 detalla el diseño, la implementación y la integración de NSG en `APOTHEOSIS`. El Capítulo 4 se dedica a la experimentación del sistema, así como al análisis y discusión de resultados. En el Capítulo 5 se hace un recorrido del estado del arte tanto de la búsqueda aproximada de los vecinos más próximos como de la indexación de resúmenes de similitud en análisis forense, detallando a su vez la aportación de este trabajo. Finalmente, en el Capítulo 6 se exponen las conclusiones del trabajo y se definen algunos aspectos y áreas a explorar de cara al futuro. De forma adicional, en el Anexo A se describe la distribución temporal del trabajo y la dedicación invertida mediante un diagrama de Gantt.

Capítulo 2

Conceptos previos

En este capítulo se introducen los conceptos necesarios para seguir el resto del documento. Se comienza por los algoritmos de similitud aproximada, y se continúa con el problema de la búsqueda aproximada de vecinos más próximos y con las estructuras de datos que se emplean para resolverlo.

2.1. Algoritmos de similitud aproximada

Una función *hash* o función resumen es una función que transforma cualquier sucesión finita de ceros y unos en una sucesión de n ceros y unos, con n fijo, y que cumple ciertas condiciones [8]. Formalmente, es una función

$$\mathcal{H} : \left\{\{0,1\}^j \mid j \in \mathbb{N}\right\} \to \{0,1\}^n.$$

Las condiciones a cumplir son las siguientes: tiene que ser fácil y poco costoso calcular la imagen, y debe ser muy difícil y costoso conociendo la imagen calcular una preimagen. Tiene que ser muy costoso encontrar dos elementos con la misma imagen (esta propiedad recibe el nombre de resistencia a colisiones), y también tiene que ser muy sensible a modificaciones, es decir, que cambiando un solo *bit* cambie de manera significativa el resumen. Las funciones *hash* juegan un papel importante en el campo de la seguridad informática, garantizando la seguridad e integridad de los datos. Por ejemplo, las firmas digitales, utilizadas para verificar la autenticidad e integridad de documentos digitales, hacen uso de este tipo de funciones. Así, las funciones *hash* permiten resumir los datos de entrada en una forma fácil de comparar, sin tener que comparar directamente los datos originales.

Los algoritmos de similitud aproximada [3] permiten detectar similitudes entre artefactos digitales, entendiendo por artefacto cualquier secuencia de *bytes* (por ejemplo,

un fichero) que posea una interpretación significativa. Estos algoritmos pueden entenderse como una generalización de la búsqueda de objetos idénticos mediante *hashing* criptográfico: en vez de devolver una respuesta binaria (0, si el *hash* de los objetos no coincide, o 1, si coincide), devuelven un valor en una escala continua, interpretado como una medida de similitud.

Los algoritmos de similitud aproximada constan de dos fases. En la primera, se genera un resumen o representación comprimida a partir del artefacto de entrada. Para ello, se identifican y extraen características de cada artefacto digital, entendiendo como característica cualquier valor que se pueda derivar de dicho artefacto. En la segunda fase, se comparan los resúmenes para producir una medida de similitud. Esta medida representa una estimación normalizada del número de características que coinciden entre los artefactos digitales que se están comparando (la comparación entre dos características siempre devuelve un valor binario $\{0, 1\}$ indicando si hay o no coincidencia).

Estos algoritmos son muy eficientes cuando se trabaja con grandes volúmenes de datos, siendo esto de interés en el campo del análisis forense, donde se deben examinar grandes cantidades de datos. `APOTHEOSIS` soporta dos de ellos:

- `ssdeep` [9], basado en *context triggered piecewise hashing*, que devuelve una puntuación de similitud entre 0 y 100. Su principal limitación es que hay pares de resúmenes que no se pueden comparar, en cuyo caso la puntuación es 0 con independencia del contenido.
- `TLSH` [10], basado en *locality sensitive hashing*, que devuelve directamente una distancia no acotada, en la que el 0 corresponde a elementos idénticos y los valores crecen conforme los artefactos difieren.

2.2. Búsqueda aproximada de vecinos más próximos

Sea $\mathcal{X}$ un conjunto de N elementos y sea δ una medida de disimilitud sobre él. Una medida de disimilitud es una función que mide el grado de desemejanza entre dos objetos, es siempre un valor no negativo, y cuanto mayor sea este, mayor será la diferencia entre los objetos. Un caso particular habitual en la literatura viene dado por $\mathcal{X} \subseteq \mathbb{R}^d$, y δ siendo la distancia euclídea. Dada una consulta q y un entero $k > 0$, la búsqueda de los k vecinos más próximos sobre $\mathcal{X}$ devuelve el conjunto $\mathcal{K} \subseteq \mathcal{X}$, con $|\mathcal{K}| = k$, tal que

$$\max_{p \in \mathcal{K}} \delta(p, q) \leq \min_{p \in \mathcal{X} \setminus \mathcal{K}} \delta(p, q).$$

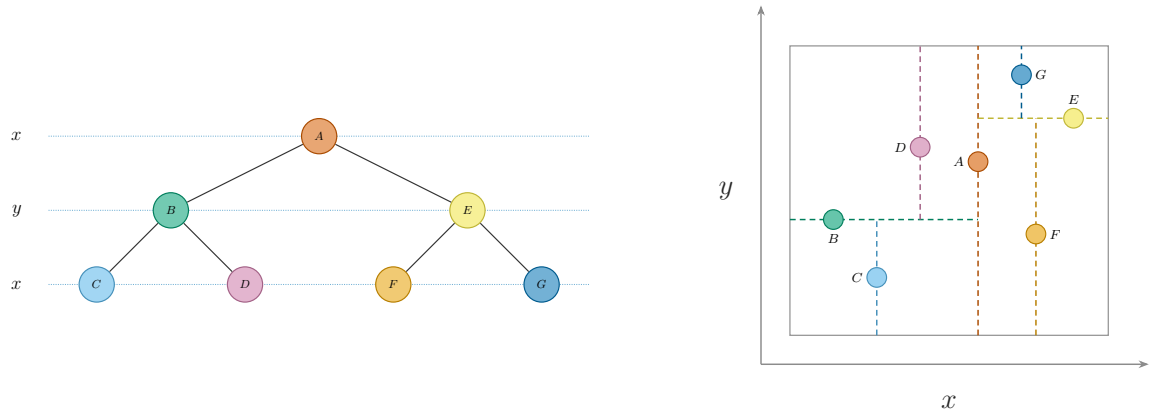
El coste de esta búsqueda es $\mathcal{O}(N)$, el cual se corresponde con calcular la disimilitud de cada punto con la consulta, y devolver los k puntos más similares a q. Por tanto, encontrar los k vecinos más próximos cuando N es grande resulta ser muy costoso en tiempo. En consecuencia, se recurre a la búsqueda aproximada de vecinos más próximos (del inglés, *Approximate Nearest Neighbors*, ANN), la cual relaja la restricción de exactitud, devolviendo k vecinos que están "cerca" de ser óptimos. Formalmente, si se denota por $\mathcal{K}'$ al conjunto formado por estos k vecinos aproximados, se define el *recall@k* del conjunto $\mathcal{K}'$ como $\frac{|\mathcal{K} \cap \mathcal{K}'|}{k}$. Así, el objetivo de la búsqueda ANN es maximizar el *recall* tratando de recuperar los resultados lo más rápido posible.

Se han propuesto distintos enfoques para hacer frente a este problema, entre los cuales destacan tres: los basados en estructuras jerárquicas como árboles [11, 12], los basados en *hashing* [2] y los basados en grafos. Estos últimos han despertado un reciente interés por su gran potencial [6, 7], y son el foco principal de este trabajo.

2.3. *k-d trees* y *VP-trees*

Un *k-d tree* o *k-dimensional tree* [11] es un árbol binario tal que cada nodo representa un punto en un espacio de k dimensiones. Cada nodo distinto de una hoja actúa como un hiperplano, dividiendo el espacio en dos particiones. Este hiperplano es perpendicular a un eje, el cual se corresponde con una de las k dimensiones. Existen distintas alternativas para la elección del eje al dividir, si bien la más común es ir recorriendo las distintas dimensiones en orden y, una vez que se agoten las k dimensiones, regresar a la primera dimensión y repetir el proceso. En la Figura 2.1 se muestra un ejemplo de *k-d tree* construido sobre siete puntos (nodo A a nodo G) de un espacio bidimensional. El nodo A actúa como una recta perpendicular al eje X que divide el espacio en dos particiones. Los nodos descendientes de A por la rama izquierda (derecha) se corresponden con los puntos que quedan a la izquierda (derecha) de la recta de corte. Los nodos hijos de A, B y E, definen rectas perpendiculares al eje Y que dividen cada una de las particiones anteriores en otras dos particiones. Este proceso se repite, alternando los ejes X, Y, hasta que se hayan recorrido todos los nodos del árbol.

La construcción de un *k-d tree* comienza con la elección de un eje. A continuación, se elige el punto que queda a la mitad a lo largo de dicho eje (la mediana) y se particiona el espacio, de forma que los puntos restantes quedan agrupados en dos subconjuntos en función de su posición relativa con respecto al hiperplano. Este punto elegido será la raíz del árbol. Este proceso se repite de forma recursiva, seleccionando un nuevo eje, eligiendo el nodo hijo izquierdo (derecho) del nodo raíz de entre los puntos que han



(a) k-d tree (b) Subdivisión del plano

Figura 2.1: Un *k-d tree* construido sobre siete puntos del plano.

quedado a la izquierda (derecha) del hiperplano, y dividiendo cada una de las regiones resultantes en otras dos.

Esta estructura de datos resulta de gran utilidad por su aplicación en la búsqueda de los vecinos más próximos a una consulta dada. No obstante, el uso de *k-d trees* para llevar a cabo este tipo de búsqueda resulta ineficiente cuando el número de dimensiones es muy alto, lo que se conoce como la maldición de la dimensionalidad [2]. Además, la construcción de un *k-d tree* requiere que los puntos pertenezcan a un espacio vectorial, de modo que cada elemento quede descrito por un vector de coordenadas y la partición pueda definirse comparando una única componente. Esta exigencia restringe su aplicabilidad, ya que en numerosos problemas los objetos no admiten una representación por coordenadas y lo único de lo que se dispone es de una función que mide la distancia entre pares de elementos.

Los *VP-trees* o *vantage-point trees*, propuestos por Yianilos [12], surgen precisamente para hacer frente a este problema. La principal ventaja de esta estructura de datos es que no presupone ninguna estructura vectorial sobre el conjunto de datos. Así, basta con que el dominio sobre el que se trabaja constituya un espacio métrico, esto es, un conjunto X dotado de una función de distancia

$$d\colon X \times X \to \mathbb{R}$$

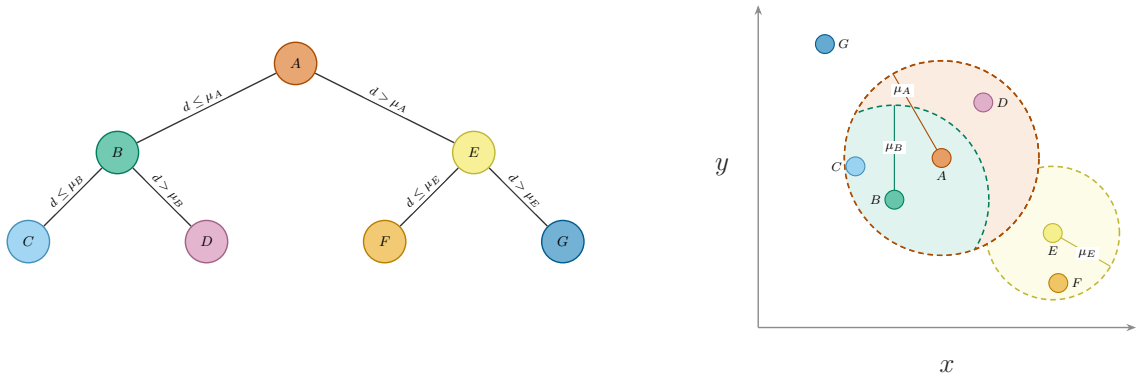
que sea no negativa, que se anule únicamente cuando sus dos argumentos coinciden, que sea simétrica y que satisfaga la desigualdad triangular

$$d(x,z) \leq d(x,y) + d(y,z), \qquad \forall\, x,y,z \in X.$$

De forma análoga a lo que ocurre con los *k-d trees*, cada nodo de un *VP-tree* divide el

espacio en dos. No obstante, en vez de utilizar coordenadas para realizar dicha división, se utiliza la distancia a un punto que recibe el nombre de punto de observación o *vantage point*. Los puntos cercanos al *vantage point* conforman el subespacio izquierdo/interior, mientras que los puntos lejanos conforman el subespacio derecho/exterior. De forma recursiva se acaba constituyendo un árbol binario.

La construcción de un *VP-tree* comienza seleccionando un punto del conjunto de puntos. Este punto recibe el nombre de *vantage point* y se corresponde con el nodo raíz del árbol. A continuación, se calcula la distancia de dicho punto al resto de puntos, que se reparten en dos subconjuntos, en función de si dicha distancia es inferior o no a un cierto umbral (que suele ser la mediana de las distancias calculadas). Finalmente, se repite el proceso de forma recursiva, eligiendo un punto de observación de entre los puntos del subconjunto interior (exterior), que será el hijo izquierdo (derecho) del nodo raíz, y dividiendo cada una de las particiones en otras dos particiones. En la Figura 2.2 se muestra un ejemplo de *VP-tree* construido sobre siete puntos de un espacio métrico.



(a) *VP-tree* (b) Subdivisión del espacio

Figura 2.2: Un *VP-tree* construido sobre siete puntos del espacio.

La búsqueda de los k vecinos más próximos a una consulta $q \in X$ se realiza mediante un recorrido en profundidad, manteniendo una cola de prioridad con los k mejores candidatos encontrados hasta el momento. Se denota por τ la distancia de q al peor de dichos candidatos, con $\tau = \infty$ mientras la cola no contenga k elementos. Al visitar un nodo con punto de observación p y umbral μ, se calcula $d(q, p)$ y se actualiza la cola si procede, lo que puede reducir el valor de τ. A continuación se decide qué subárboles merece la pena explorar. Los puntos x del subárbol interior satisfacen $d(p, x) \leq \mu$, de modo que la desigualdad triangular garantiza que

$$d(q, x) \geq d(q, p) - d(p, x) \geq d(q, p) - \mu,$$

y por tanto dicho subárbol puede descartarse por completo cuando $d(q, p) - \mu > \tau$. De

forma simétrica, los puntos del subárbol exterior cumplen $d(p,x) > \mu$, luego

$$d(q,x) \geq d(p,x) - d(q,p) > \mu - d(q,p),$$

y el subárbol exterior puede descartarse cuando $d(q,p)+\tau < \mu$. En caso de que ninguna de las dos condiciones se satisfaga, es necesario explorar ambas ramas. Conviene entonces visitar primero aquella en la que se encuentra la consulta (la interior si $d(q,p) \leq \mu$ y la exterior en caso contrario), ya que así se encuentran candidatos próximos cuanto antes, τ decrece con mayor rapidez y la poda de la rama restante resulta más probable.

2.4. NN-Descent

Un grafo k-NN (*k-nearest neighbor*) es un grafo $G_k = (X, E)$, donde el conjunto de vértices, X, son puntos $\{x_1, \ldots, x_N\}$ sobre los que es posible definir una función de disimilitud δ, y el conjunto de aristas viene dado por

$$E = \big\{(v,u) \mid v \in X,\ u \in B_k(v)\big\},$$

con $B_k(v)$ el conjunto de los k vecinos más próximos de $v \in X$. Es interesante notar que un grafo k-NN es un grafo dirigido. Todo vértice tiene grado de salida exactamente k, mientras que su grado de entrada es variable, ya que la relación de vecindad no es simétrica.

Puesto que la construcción de este grafo resulta irrealizable cuando N es muy grande, en la práctica se recurre a grafos k-NN aproximados, en los que cada vecindario $\widehat{B}_k(v)$ es una estimación de $B_k(v)$. Se sacrifica así la exactitud a cambio de un coste de construcción menor.

NN-Descent [13] es un algoritmo simple y eficiente para la construcción de grafos k-NN aproximados. Se apoya en el principio de que un vecino de un vecino es, con alta probabilidad, un vecino. Es decir, si se tiene una aproximación de los k vecinos más próximos para cada punto, $\widehat{B}_k(v)$, se puede mejorar iterativamente explorando los vecinos de cada punto de $\widehat{B}_k(v)$.

El algoritmo comienza asignando a cada $v \in X$ un conjunto $\widehat{B}_k(v)$ formado por k puntos elegidos al azar. En cada iteración, para todo vértice v se calcula la disimilitud entre cada vecino de cada vecino de v con el propio v. Si alguno de estos candidatos resulta más próximo que el peor vecino almacenado, se produce una actualización. Esta operación recibe el nombre de *local join*.

Sobre este esquema básico se aplica una serie de refinamientos. En primer lugar, en vez de restringir la exploración únicamente a $\widehat{B}_k(v)$, se realiza *local join* sobre el conjunto

$\overline{B}_k(v) = \widehat{B}_k(v) \cup R_k(v)$, con

$$R_k(v) = \left\{u \in X \mid v \in \widehat{B}_k(u)\right\}$$

el conjunto de vecinos inversos. De esta forma, la propagación deja de ser unidireccional, lo que mejora la convergencia del algoritmo. Por otra parte, en cada iteración solo se evalúan aquellos pares que incluyen, al menos, un vecino incorporado en la iteración anterior, evitando repetir comparaciones ya realizadas. Por último, se añade un criterio de parada anticipada: el proceso finaliza cuando el número de actualizaciones efectuadas en una iteración cae por debajo de εNk, con ε un umbral pequeño, lo que indica que el grafo se ha estabilizado.

La formulación práctica del algoritmo depende de cinco parámetros: el número K de vecinos que se conservan por nodo, el tamaño $L \geq K$ de la lista de candidatos que se mantiene durante el refinamiento, el número S de candidatos nuevos que se muestrean de dicha lista en cada iteración, la cota R sobre el número de vecinos inversos almacenados por nodo y el número T de iteraciones de refinamiento, que se fija de antemano cuando no se emplea el criterio de parada anticipada.

2.5. HNSW

Hierarchical Navigable Small World [4] es una estructura de datos basada en grafos que permite la indexación y búsqueda eficiente en conjuntos de datos de alta dimensionalidad. HNSW se fundamenta en dos estructuras de datos: las listas por saltos y los pequeños mundos navegables.

Una lista por saltos o *skip list* [14] es una estructura de datos probabilística que permite la inserción, búsqueda y eliminación en listas enlazadas de manera eficiente. La idea principal consiste en evitar recorrer todos los elementos de la lista de manera secuencial al operar sobre la lista. Para ello, se introducen niveles y saltos entre los elementos de la lista enlazada, permitiendo un recorrido más rápido al no tener que visitar cada elemento de la lista.

Su estructura es la que sigue. El nivel más bajo (nivel 0) se corresponde con la lista enlazada original ordenada. Con cada nivel subsiguiente aumenta el número de saltos de la lista de forma probabilística. Formalmente, un nodo del nivel i aparece en el nivel $i + 1$ con una probabilidad fija p. La búsqueda de un determinado elemento comienza en el nivel superior y compara el siguiente elemento de la lista con el valor a buscar. Si el elemento es menor o igual que el valor, el algoritmo avanza al siguiente elemento de la lista en el mismo nivel. Si no, la búsqueda desciende al nivel inmediatamente

inferior (con un mayor número de elementos), y se repite el mismo procedimiento. La búsqueda acaba en el nivel 0, encontrando el elemento buscado. Se puede demostrar que el coste esperado de la búsqueda es logarítmico en el número de nodos de la lista. En la Figura 2.3 se muestra un ejemplo de *skip list* con nodos $\{3, 6, 7, 9, 12, 17, 19\}$, y del procedimiento de búsqueda del nodo 9.

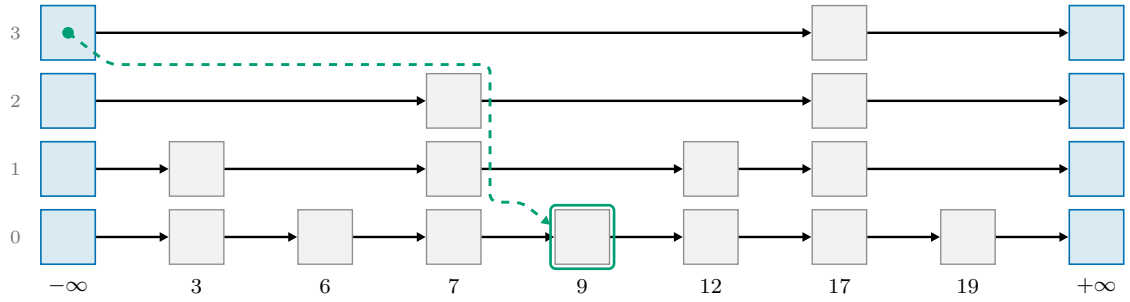


Figura 2.3: Ejemplo de búsqueda en una *skip list*.

Un pequeño mundo navegable (del inglés, *Navigable Small World*, NSW) [15] es un grafo que permite encontrar caminos de longitud polilogarítmica entre pares de nodos utilizando un enrutamiento voraz. La idea principal consiste en combinar en un mismo grafo dos tipos de enlaces: los enlaces cortos, que unen nodos próximos entre sí y aportan precisión en la búsqueda, y los enlaces largos, que unen nodos de regiones alejadas del espacio y aportan navegabilidad.

En la construcción de un pequeño mundo navegable, los elementos se insertan uno a uno en orden aleatorio, conectando cada elemento nuevo con los k elementos más cercanos de entre los ya insertados. De este modo, los enlaces creados en las primeras inserciones son largos, puesto que el grafo contiene todavía pocos elementos, y se van acortando conforme aumenta el número de elementos insertados.

La búsqueda comienza en un nodo de entrada elegido de forma aleatoria. En cada paso se calcula la distancia del nodo consulta a los vecinos del nodo actual y se avanza hacia el vecino que arroja la menor distancia. El procedimiento termina cuando ningún vecino está más cerca de la consulta que el nodo actual, esto es, cuando se alcanza un mínimo local, que no tiene por qué coincidir con el nodo más cercano a la consulta.

HNSW combina las dos estructuras anteriores para construir una estructura jerárquica de grafos que agiliza los saltos entre nodos y mejora la eficiencia y la precisión de la búsqueda.

HNSW consta de un conjunto de capas, cada una de las cuales es un grafo de pro-

ximidad. La capa 0 contiene todos los elementos del conjunto de datos, y cada capa superior contiene un subconjunto de los elementos de la capa inmediatamente inferior, de modo que un mismo elemento puede aparecer en varias capas. En las capas superiores hay pocos elementos y sus enlaces cubren distancias grandes, formando grafos ligeros y poco conectados. Conforme se desciende, las capas contienen más elementos, los enlaces son más cortos y los grafos, más densos.

Durante la búsqueda se entra en primer lugar a la capa superior a través del punto de entrada. En esta capa se realiza un recorrido del grafo tal y como se hace en NSW, hasta encontrar el mínimo local de dicha capa. Cuando se alcanza este mínimo, la búsqueda se desplaza a ese mismo elemento pero situado en la capa inferior, y desde él se vuelve a recorrer el nuevo grafo en busca de otro mínimo local. El proceso se repite hasta llegar a la capa 0. En la Figura 2.4 se muestra un ejemplo de búsqueda (nodo y caminos en naranja) en una estructura de datos HNSW.

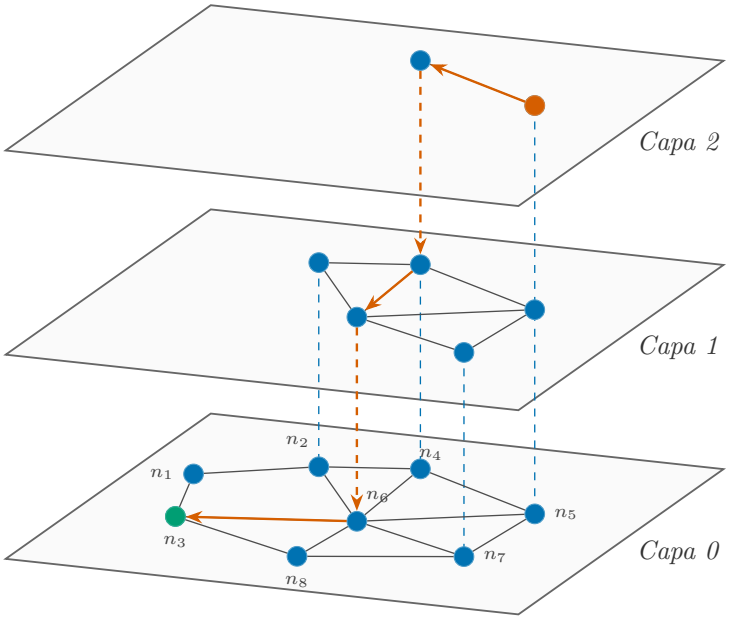


Figura 2.4: Ejemplo de una estructura de datos HNSW.

2.6. NSG

Navigating Spreading-out Graph (NSG) es una estructura de datos basada en grafos propuesta por Fu et al. [5] para la búsqueda de vecinos más próximos aproximados. NSG está formado por un único grafo, a diferencia de la estructura jerárquica de HNSW.

NSG parte del análisis del coste del recorrido voraz descrito en la Sección 2.5, común a la mayoría de los métodos basados en grafos. Este recorrido depende del número

de saltos necesarios hasta alcanzar la consulta y del coste de decidir por qué nodo debe continuar el recorrido, proporcional al número de vecinos a examinar. La idea principal de NSG es tratar de mejorar la eficiencia de la búsqueda y la escalabilidad de los métodos existentes basados en grafos atendiendo a cuatro aspectos derivados del análisis anterior: garantizar la conectividad del grafo, de forma que la consulta sea siempre alcanzable, disminuir el grado de salida medio del grafo, acortar el camino de búsqueda para reducir la complejidad temporal de la búsqueda y reducir el tamaño del índice para mejorar la escalabilidad.

Caminos monótonos y redes de búsqueda monótonas

Sea X un conjunto finito de N puntos y δ una medida de disimilitud sobre él. Dados dos nodos $p, q \in X$ y un grafo G definido sobre X, un camino $v_1, v_2, \ldots, v_l$ de p a q en G (esto es, con $v_1 = p$, $v_l = q$ y $v_i v_{i+1} \in G$ para todo i) se dice *monótono* si

$$\delta(v_i, q) > \delta(v_{i+1}, q), \qquad \forall i = 1, \ldots, l-1,$$

es decir, si cada paso acerca el recorrido a su destino. Un grafo G es una *red de búsqueda monótona* (del inglés, *Monotonic Search Network*, MSNET) si para cualquier par de nodos $p, q \in X$ existe al menos un camino monótono de p a q.

De un MSNET se siguen tres propiedades. La primera es que es fuertemente conexo por definición, lo que resuelve el primero de los cuatro aspectos anteriores. La segunda es que sobre él el recorrido voraz encuentra un camino monótono hasta el destino sin necesidad de retroceder, a diferencia de lo que ocurre en un grafo de pequeño mundo navegable, donde el recorrido puede quedar atrapado en un mínimo local que no se corresponde con el nodo buscado. La tercera es que ese recorrido es corto: bajo la hipótesis de que los N puntos se distribuyen uniformemente en un subespacio de $\mathbb{R}^d$, la longitud esperada de un camino monótono entre dos nodos cualesquiera es

$$O\left(\frac{N^{1/d} \log N^{1/d}}{\Delta r}\right),$$

donde Δr es la menor diferencia entre las longitudes de los lados de cualquier triángulo no isósceles formado por puntos de X, una función decreciente de N.

Monotonic Relative Neighborhood Graph

Sea $B(p, r) = \{x \mid \delta(x, p) < r\}$ la bola abierta de centro p y radio r. Se define la *luna* determinada por dos puntos $p, q \in X$ como la región

$$\text{luna}_{pq} = B\big(p, \delta(p, q)\big) \cap B\big(q, \delta(p, q)\big),$$

esto es, el conjunto de puntos que están más cerca de p y de q de lo que p y q lo están entre sí. Un *Relative Neighborhood Graph* (RNG) [16] es un grafo no dirigido en el que la arista pq existe si y solo si $\text{luna}_{pq} \cap X = \emptyset$. Es decir, se elimina la arista más larga de todo triángulo posible sobre X. Sin embargo, esta condición es demasiado estricta, y es la causa de que el RNG no sea un MSNET.

A partir de esta idea, Fu et al. [5] proponen una regla de selección de aristas más permisiva que da lugar al *Monotonic Relative Neighborhood Graph* (MRNG), un grafo dirigido cuyo conjunto de aristas satisface que, para cualquier arista pq,

$$pq \in \text{MRNG} \iff \text{luna}_{pq} \cap X = \emptyset \;\text{ o }\; \forall r \in (\text{luna}_{pq} \cap X),\ pr \notin \text{MRNG}.$$

La definición es recursiva, e implica que los vecinos de un nodo deben seleccionarse del más cercano al más lejano. La arista pq solo se descarta cuando en la luna de p y q hay algún nodo r que ya ha sido seleccionado como vecino de p.

Cuando los candidatos se recorren ordenados por distancia creciente a p, la condición anterior se simplifica. Si r ya ha sido seleccionado antes de examinar q, entonces $\delta(p, r) \leq \delta(p, q)$ se cumple automáticamente, y la pertenencia $r \in \text{luna}_{pq}$ se reduce a comprobar que $\delta(r, q) < \delta(p, q)$. Esta es la formulación que se emplea en la práctica.

De esta definición se sigue que el MRNG es un MSNET, heredando las tres propiedades descritas anteriormente. Además, se puede probar que el grafo resultante sigue siendo muy disperso. El inconveniente de este grafo es su coste de construcción. Obtenerlo de forma exacta exige, para cada nodo, calcular su distancia a todos los demás, ordenarlos y aplicar sobre ellos la regla de selección, lo que resulta en un coste de $O(N^2 \log N + N^2 c)$, donde c es el grado medio de salida del MRNG. Este coste es inasumible cuando N es grande.

NSG como aproximación del MRNG

El proceso de construcción del índice NSG se desarrolla en cuatro etapas. En primer lugar, se construye un grafo k-NN aproximado (por ejemplo, mediante NN-Descent). En segundo lugar, se determina el *navigating node*, que se obtiene como medoide aproximado del conjunto. Esto es, se calcula el centroide de los datos, se emplea como consulta sobre el grafo k-NN aproximado y se toma el vecino más próximo devuelto por la búsqueda. Este nodo actúa como punto de partida para todas las búsquedas en el grafo. En tercer lugar, se seleccionan los vecinos de cada nodo v. Para ello, se toma v como consulta y se realiza una búsqueda voraz sobre el grafo k-NN aproximado partiendo del *navigating node*, registrando como candidatos todos los nodos visitados

durante el recorrido junto con los vecinos de v en el grafo k-NN. Sobre ese conjunto de candidatos, ordenado por distancia creciente a v, se aplica la regla de selección del MRNG, conservando a lo sumo m vecinos. Por último, se recorre el grafo resultante en profundidad desde el *navigating node*. Los nodos que no queden enlazados al árbol generado se conectan con su vecino más próximo aproximado dentro de él, repitiendo el proceso hasta que todos los nodos resulten alcanzables.

La idea principal de NSG es relajar el requisito de existencia de un camino monótono entre cualquier par de nodos. NSG solo garantiza la existencia de un camino monótono desde el *navigating node* a cualquier otro nodo, lo cual es suficiente porque la búsqueda siempre empieza desde el *navigating node*.

La búsqueda sobre un grafo NSG es el mismo recorrido voraz empleado en el resto de métodos basados en grafos, con dos particularidades: parte siempre del *navigating node* y, en lugar de avanzar a un único nodo en cada paso, mantiene una lista ordenada de candidatos de tamaño fijo $ef \geq k$, expandiendo en cada iteración el candidato aún no explorado más próximo a la consulta y truncando la lista a los ef mejores elementos. Con $ef = 1$ se tiene el descenso voraz de la Sección 2.5, que se detiene en el primer mínimo local. Valores mayores exploran una porción mayor del grafo y aumentan el *recall* a costa del tiempo de consulta.

Capítulo 3

Diseño e implementación

En este capítulo se expone en detalle el sistema desarrollado. Se comienza presentando `APOTHEOSIS` en su diseño original. A continuación, se detalla la implementación de NSG desarrollada, junto con las adaptaciones que ha sido necesario llevar a cabo para integrar la nueva estructura en `APOTHEOSIS`. La herramienta resultante se encuentra disponible en [17]. Finalmente, se discuten las implicaciones que tiene la sustitución de HNSW por NSG sobre la funcionalidad del sistema.

3.1. APOTHEOSIS

`APOTHEOSIS` [1, 18] es una herramienta desarrollada por el grupo de investigación DisCo de la Universidad de Zaragoza cuyo propósito es la identificación y comparación eficiente de similitudes en grandes conjuntos de datos. Su utilidad se pone de manifiesto en campos como la informática forense, donde la comparación de artefactos digitales similares facilita el análisis de evidencias que permiten determinar la naturaleza, el alcance y el impacto de un incidente de seguridad.

Como se ha descrito anteriormente, `APOTHEOSIS` combina dos tecnologías distintas pero inherentemente interconectadas, como son las técnicas de búsqueda aproximada y los algoritmos de similitud aproximada. La Figura 3.1 muestra una descripción de alto nivel de `APOTHEOSIS`. La herramienta se compone de dos estructuras de datos complementarias, un *radix tree* y una implementación propia de un HNSW, y hace uso de resúmenes de similitud para gestionar y comparar de forma eficiente los datos. El *radix tree* se utiliza para almacenar los resúmenes de similitud y recuperar de forma eficiente las coincidencias exactas. Por otra parte, HNSW se utiliza para llevar a cabo búsquedas aproximadas por similitud, ofreciendo un compromiso entre velocidad y precisión en grandes conjuntos de datos. El almacenamiento de los metadatos asociados a cada

elemento, como el nombre del fichero o el registro del que procede, se relega a una base de datos. Además, se puede acceder al sistema de forma remota a través de una API REST, lo que permite integrarlo en flujos de trabajo automatizados y realizar análisis de similitud en tiempo real.

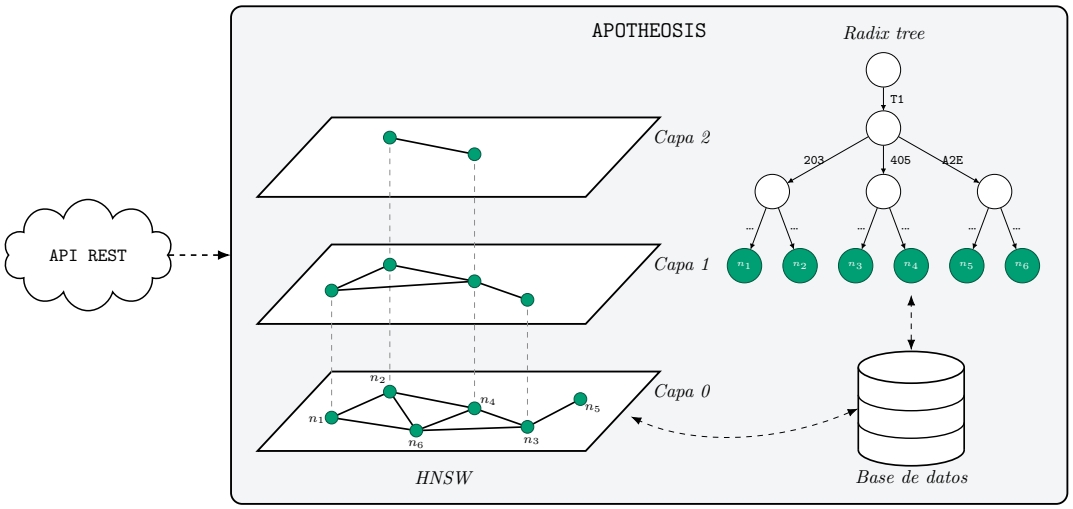


Figura 3.1: Descripción de alto nivel de `APOTHEOSIS`.

El sistema trabaja en dos fases: indexación y búsqueda. Durante la fase de indexación, se extraen los resúmenes de similitud a partir de los artefactos digitales que se pretenden indexar. Dichos resúmenes de similitud se almacenan en el *radix tree* para permitir búsquedas exactas sobre el índice. Al mismo tiempo, también se insertan en el grafo HNSW, creándose aristas entre nodos similares según las medidas de similitud calculadas. La búsqueda de un elemento comienza buscando una coincidencia exacta en el *radix tree*. En caso de no encontrarla, se realiza una búsqueda aproximada por similitud utilizando HNSW, devolviendo los k vecinos más próximos aproximados. El uso del *radix tree* permite recuperar de forma eficiente un elemento cuando la coincidencia es exacta, evitando llevar a cabo una búsqueda aproximada de vecinos más próximos innecesaria. Además, impide la inserción de elementos duplicados, evitando la degradación del *recall* de las búsquedas en el grafo HNSW.

HNSW presenta ciertas limitaciones, introducidas en el Capítulo 1. Estas limitaciones motivan la exploración de distintos algoritmos de búsqueda aproximada de vecinos más próximos.

3.2. Extensión de APOTHEOSIS con NSG

Después del estudio del estado del arte de la búsqueda aproximada de los vecinos más próximos, se ha optado por extender `APOTHEOSIS` con NSG como estructura de datos para llevar a cabo las búsquedas aproximadas por similitud. NSG es un grafo de una sola capa, a diferencia de la estructura jerárquica multicapa de HNSW, lo que se traduce en un menor tamaño del índice generado. Además, la regla de selección de aristas de NSG trata de que exista un camino monótono desde el *navigating node* hasta cualquier otro nodo del grafo, de modo que el recorrido voraz tiende a acercarse al objetivo en cada paso y a recorrer caminos más cortos, lo que se traduce en menores tiempos de búsqueda. En definitiva, NSG consigue mejorar, al menos teóricamente, los aspectos negativos de HNSW que se han comentado y que podían resultar problemáticos, especialmente cuando el número de elementos a indexar es muy grande.

3.2.1. Implementación de NSG

Se ha llevado a cabo una implementación propia de NSG en Rust, puesto que las implementaciones existentes de NSG trabajan con espacios vectoriales continuos y dependen de métricas de distancia como la distancia euclídea o la distancia del coseno. Como `APOTHEOSIS` trabaja con resúmenes de similitud, ha sido necesario adaptar la implementación de NSG de acuerdo con este requisito.

El algoritmo de búsqueda sobre el grafo NSG (Algoritmo 1) simplemente recorre el grafo y trata de alcanzar la consulta de forma voraz. La principal diferencia con el algoritmo original [5] es que la distancia entre dos nodos no se calcula sobre una representación vectorial, sino que se deriva de la comparación de sus resúmenes de similitud. En el caso de `TLSH`, la comparación devuelve ya una distancia y se emplea directamente, mientras que la puntuación s de `ssdeep` se convierte en distancia como $100 - s$, de modo que 0 corresponde a resúmenes idénticos y 100 al valor máximo.

La construcción del grafo NSG (Algoritmo 2) sigue el esquema planteado por Fu et al. [5], si bien se han introducido cambios necesarios para integrar NSG en la herramienta `APOTHEOSIS`. En primer lugar, en el algoritmo original se calcula el centroide de los datos y se emplea como consulta sobre el grafo k-NN aproximado, de forma que el vecino más próximo devuelto por la búsqueda se convierte en el *navigating node*. Puesto que el cálculo del centroide deja de tener sentido cuando los datos de entrada no son vectoriales, se sustituye por el cálculo del medoide. Formalmente, el medoide de un conjunto de puntos es el punto que minimiza la suma de las distancias a todos

Algoritmo 1: Búsqueda voraz sobre el grafo NSG.

Entrada: Grafo $\mathcal{G}$, punto de entrada p, consulta q, número de vecinos k, tamaño de la lista de candidatos ef

Salida: Los k nodos más cercanos a q hallados, y el conjunto E de todos los nodos visitados

```
1  Procedimiento SearchOnGraph(G, p, q, k, ef):
2      S = los primeros ef vecinos de p en G
3      mientras S.size() < ef hacer S.add(nodo aleatorio no visitado)
4      Ordenar S en orden ascendente de distancia a q
5      E = S
6      Marcar todos los nodos de S como no comprobados
7      mientras quede algún nodo sin comprobar en S hacer
8          p_i = el primer nodo sin comprobar de S; marcarlo como comprobado
9          para cada vecino w de p_i en G no visitado hacer  S.add(w); E.add(w)
10         Ordenar S en orden ascendente de distancia a q
11         si S.size() > ef entonces S.resize(ef)
12     fin
13     devolver los primeros k nodos de S, y E
```

los demás puntos del conjunto. Así, dado un conjunto de puntos $v_1, \ldots, v_n$, el medoide viene dado por

$$\arg \min_{v_i \in \{v_1, v_2, \ldots, v_n\}} \sum_{j=1}^{n} \delta(v_i, v_j).$$

Durante la construcción del grafo NSG tan solo se utiliza un subconjunto reducido de los datos de entrada para calcular dicho medoide. En concreto, dado un conjunto de entrada de tamaño N, se toma una muestra de aproximadamente 256 elementos recorriéndolo a intervalos regulares, esto es, uno de cada $\lfloor N/256 \rfloor$ elementos, y se calcula el medoide exacto sobre ella. Esto supone un número constante de comparaciones, independiente de N, frente a las $O(N^2)$ que exigiría el medoide exacto del conjunto completo. Dicho medoide se emplea después como consulta sobre el grafo k-NN aproximado, y el vecino más próximo devuelto por la búsqueda se toma como *navigating node*.

En segundo lugar, se ha generalizado la estrategia de selección de aristas de MRNG, descrita en la Sección 2.6, mediante un factor de relajación $\alpha \geq 1$. En su formulación, dado un nodo v y su lista de candidatos ordenada por distancia, un candidato p se descarta si existe un vecino ya seleccionado w tal que $\delta(w, p) < \delta(v, p)$. Ahora, la condición pasa a ser $\alpha \cdot \delta(w, p) < \delta(v, p)$, de manera que un candidato solo se descarta cuando el vecino ya seleccionado está sensiblemente más cerca de él que el propio v. Con $\alpha = 1$ se tiene exactamente la regla de MRNG, mientras que valores mayores hacen la poda más conservadora y producen un grafo más denso. Una relajación de

esta misma forma se emplea en [19] sobre una regla de poda análoga.

La regla de MRNG por la que se descartan las aristas se basa en la desigualdad triangular, que garantiza que ningún camino indirecto sea arbitrariamente peor que la arista directa que se descarta. Los resúmenes de similitud no ofrecen tal garantía. `TLSH`, en particular, es únicamente una semimétrica: es simétrica, no negativa y se anula solo entre resúmenes iguales, pero no cumple la desigualdad triangular, sino una versión relajada de la forma $\delta(x, z) \leq \delta(x, y) + \delta(y, z) + c$, cuya constante c ha sido acotada formalmente en trabajos recientes [20]. En estas condiciones, la regla estricta puede descartar aristas cuyo rodeo asociado resulta más largo de lo que la propia regla presupone. En el caso de `ssdeep`, su distancia está acotada en $[0, 100]$ y toma valores enteros, por lo que los empates entre candidatos son frecuentes y la comparación $\delta(w, p) < \delta(v, p)$ que decide la poda se resuelve a menudo entre valores iguales o muy próximos. Además, el valor máximo es ambiguo, ya que no distingue entre dos artefactos muy distintos y dos cuyos resúmenes no son comparables. Por tanto, cabe esperar que la regla de selección de aristas se comporte peor sobre `ssdeep` que sobre `TLSH`.

Esta relajación se complementa con una segunda medida: las aristas descartadas no se pierden, sino que se conservan en una lista auxiliar y se emplean, en orden de distancia creciente, para completar el vecindario hasta el grado máximo m cuando la selección no lo ha alcanzado, técnica tomada de HNSW [4]. Así, se evita que la poda deje nodos con vecindarios muy por debajo de m, lo que degradaría la conectividad del grafo y, con ella, la calidad de la búsqueda.

3.2.2. Construcción del grafo k-NN aproximado

La construcción del índice NSG parte de un grafo k-NN aproximado que se refina mediante una estrategia de selección de aristas. El algoritmo utilizado para la construcción de este grafo k-NN aproximado es NN-Descent.

La calidad del grafo k-NN aproximado depende en buena medida de su inicialización. En una primera versión se empleó la inicialización aleatoria propuesta en el algoritmo original [13], en la que el vecindario inicial de cada nodo se compone de S nodos elegidos al azar. EFANNA [21] muestra que partir de una inicialización de mayor calidad reduce el número de iteraciones necesarias y mejora la precisión del grafo resultante, y para ello recurre a un bosque de *k-d trees* aleatorizados. Dicha estrategia no es aplicable en el caso de resúmenes de similitud, puesto que un *k-d tree* particiona el espacio seleccionando una dimensión y un umbral de corte, lo que exige una representación

vectorial de los resúmenes de similitud.

Se ha adaptado esta idea sustituyendo los *k-d trees* por un bosque de *VP-trees* aleatorizados [12], que únicamente requieren una función de distancia. Cada árbol se construye a partir de una permutación aleatoria del conjunto de nodos. En cada nivel se elige al azar un nodo del subconjunto como punto de referencia, se ordenan sus elementos según la distancia a dicho punto y se divide por la mediana, procediendo recursivamente hasta que un subconjunto contiene a lo sumo ℓ elementos. De la estructura resultante solo se conservan las hojas. Los nodos que comparten hoja se consideran candidatos mutuos, y la unión de las candidaturas obtenidas en los t árboles del bosque, una vez eliminados los duplicados, conforma el conjunto de candidatos de cada nodo. Dichos candidatos se ordenan a continuación según la distancia real al nodo y se conservan a lo sumo los L más próximos. Si un nodo no llega a reunir S candidatos, su lista se completa con nodos elegidos al azar hasta alcanzar ese mínimo, ya que NN-Descent parte de S candidatos nuevos por nodo en la primera iteración.

3.2.3. Integración en APOTHEOSIS

La Figura 3.2 muestra la arquitectura resultante. El sistema mantiene las dos estructuras de datos del diseño original: un *radix tree* para la búsqueda exacta y un grafo de proximidad para la búsqueda aproximada, sustituyendo el HNSW por un NSG. El *radix tree* sigue empleando los resúmenes de similitud como claves y, como valor, la posición del nodo correspondiente en el grafo. La Figura 3.3 muestra el diagrama de clases del sistema resultante, junto con todos sus componentes.

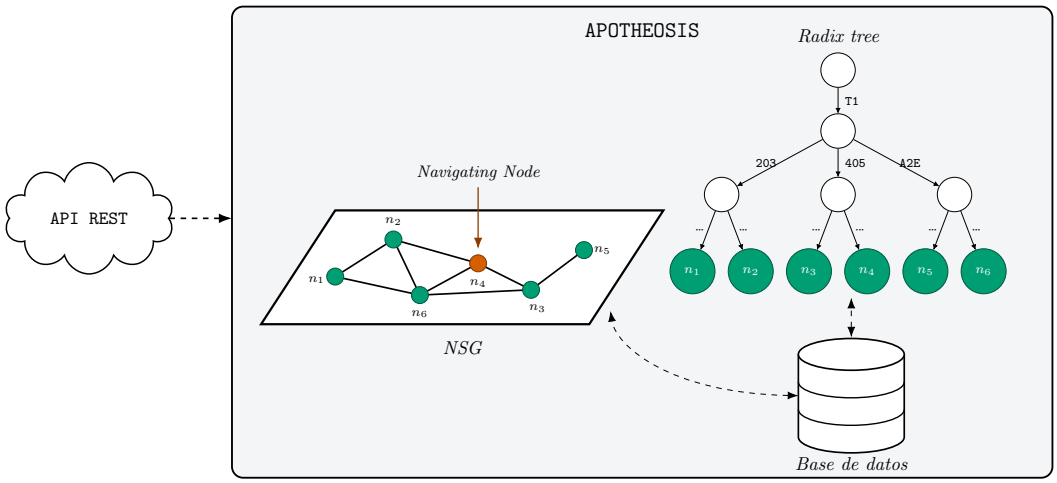


Figura 3.2: Descripción de alto nivel de `APOTHEOSIS` con NSG.

Sobre esta arquitectura se definen las dos operaciones principales del sistema. La in-

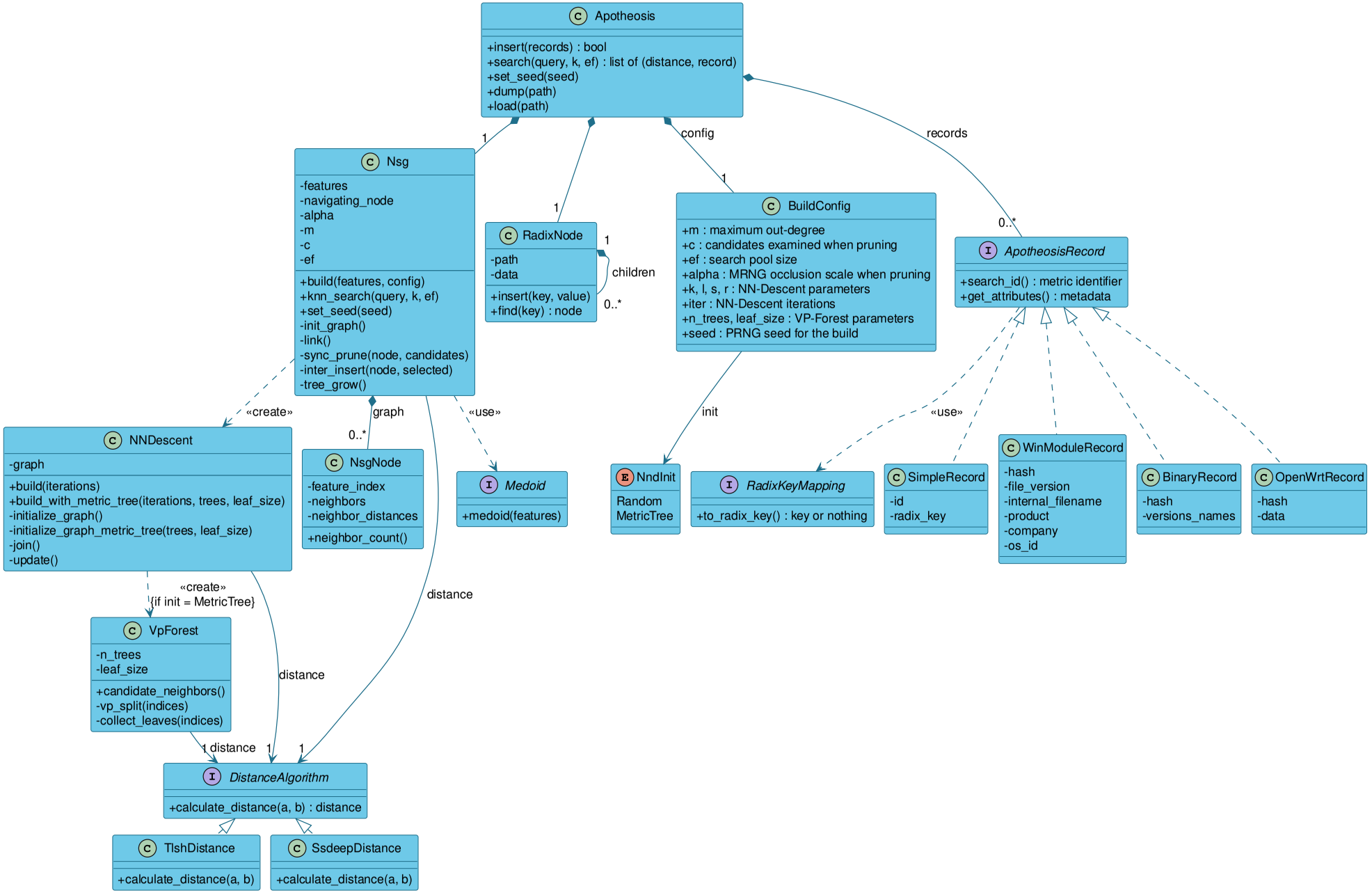


Figura 3.3: Diagrama de clases de `APOTHEOSIS` con NSG.

serción (Algoritmo 3) recorre el conjunto de nodos de entrada, descarta aquellos cuyo resumen ya se encuentra en el *radix tree* y construye el grafo NSG sobre los restantes. La búsqueda (Algoritmo 4) consulta el *radix tree* y recorre el grafo, incorporando al resultado el nodo idéntico a la consulta si existe y la búsqueda aproximada no lo ha recuperado. Sigue entonces teniendo sentido mantener esta doble estructura. En la inserción, el *radix tree* impide que un mismo resumen se incorpore dos veces al grafo, evitando la degradación del *recall* de las búsquedas. Del mismo modo, en la búsqueda, el *radix tree* garantiza que una coincidencia exacta nunca pase inadvertida, con independencia de la naturaleza aproximada del recorrido.

La sustitución de HNSW por NSG conlleva, no obstante, una pérdida de funcionalidad que conviene explicar. En la versión original, la inserción opera sobre un único nodo y el sistema admite además una operación de borrado, de modo que el modelo puede modificarse de forma dinámica. En NSG, la selección de aristas de cada nodo se realiza a partir de una búsqueda sobre el grafo k-NN completo, y el *navigating node* se obtiene como medoide aproximado de la totalidad del conjunto, por lo que ambos dependen globalmente de los datos disponibles en el momento de la construcción. Incorporar o eliminar un nodo exige, en consecuencia, reconstruir el índice por completo. El modelo resultante es, por tanto, estático. En concreto, se construye en una única operación y se serializa a disco, de manera que las consultas posteriores operan sobre un índice precalculado que se carga en memoria.

Finalmente, la Tabla 3.1 recoge los parámetros configurables del sistema, agrupados según la etapa en la que intervienen. Todos ellos, con la excepción de los relativos a la consulta, afectan únicamente a la construcción del índice.

Algoritmo 2: Construcción del grafo NSG.

Entrada: Conjunto de nodos V, cada nodo v con hash h_v; grado de salida máximo m; candidatos examinados c; tamaño de la lista de candidatos ef; factor de relajación α

Salida: Grafo NSG $\mathcal{G}$ con *navigating node* $\mathbf{n}$

1 **Procedimiento** `BuildNSG(`V`):`
2 $\mathcal{G}_k$ = `NNDescent(`V`)`; $\mathcal{G} = \mathcal{G}_k$
3 Calcular el medoide aproximado μ de un subconjunto reducido de V
4 r = nodo aleatorio
5 $\mathbf{n}$, _ = `SearchOnGraph(`$\mathcal{G}$, r, μ, 1, ef`)`
6 **para cada** *nodo* v *en* $\mathcal{G}$ **hacer**
7 _, E = `SearchOnGraph(`$\mathcal{G}$, $\mathbf{n}$, h_v, 1, ef`)`
8 Añadir a E los vecinos de v en $\mathcal{G}_k$
9 Ordenar E en orden ascendente de distancia a v, excluyendo v
10 $R_v = \emptyset$, $P = \emptyset$, $i = 1$ `//` pv `entra en conflicto con` pw `si` $\alpha \cdot \delta(h_w, h_p) < \delta(h_v, h_p)$
11 **mientras** $E \neq \emptyset$ **y** R_v.size() $< m$ **y** $i < c$ **hacer**
12 $p = E$.front(); E.remove(E.front()); $i = i + 1$
13 **si** *pv entra en conflicto con pw para algún* w *en* R_v **entonces** P.add(p) **en otro caso** R_v.add(p)
14 **fin**
15 Completar R_v hasta m con los nodos de P más cercanos a v
16 **fin**
17 **para cada** *arista* $v \to u$ *seleccionada* **hacer** añadir la arista inversa $u \to v$ a R_u
18 Podar a m, con el criterio de las líneas 9–15, los R_u que superen el grado
19 $N_{\mathcal{G}}(v) = R_v$ para todo nodo v en $\mathcal{G}$
20 T = árbol construido con las aristas de $\mathcal{G}$ desde $\mathbf{n}$ mediante DFS
21 **mientras** T *no contiene todos los nodos de* $\mathcal{G}$ **hacer**
22 Añadir una arista entre un nodo fuera de T y su vecino más cercano dentro de T (Algoritmo 1)
23 T = árbol reconstruido desde $\mathbf{n}$ mediante DFS
24 **fin**

Algoritmo 3: Inserción de un conjunto de nodos en un modelo `APOTHEOSIS`.

Entrada: Modelo `APOTHEOSIS` $\mathcal{M} = \{$*radix tree* $\mathcal{RT}$, NSG $\mathcal{G}\}$, conjunto de nodos W, cada nodo $w \in W$ con hash h_w

Procedimiento `InsertNodes`($\mathcal{M}$, W)**:**
- $V = \emptyset$ `// Nodos a insertar en el grafo NSG`
- **para cada** $w \in W$ **hacer**
  - *existente* = `SearchRadixTree`($\mathcal{RT}$, h_w)
  - **si** *existente* $\neq$ **nulo entonces**
    - `// El nodo ya existe, se evita la inserción redundante`
    - **continuar**
  - **fin**
  - `InsertRadixTree`($\mathcal{RT}$, h_w, w)
  - $V = V \cup \{w\}$
- **fin**
- $\mathcal{G}$ = `BuildNSG`(V)

Algoritmo 4: Búsqueda de los k vecinos más próximos en un modelo `APOTHEOSIS`.

Entrada: Modelo `APOTHEOSIS` $\mathcal{M} = \{$*radix tree* $\mathcal{RT}$, NSG $\mathcal{G}$ con *navigating node* $\mathbf{n}\}$, nodo de consulta q con hash h_q, número de vecinos $k \in \mathbb{N}_{>0}$, tamaño de la lista de candidatos *ef*

Procedimiento `KNNSearch`($\mathcal{M}$, q, k, *ef*)**:**
- *existente* = `SearchRadixTree`($\mathcal{RT}$, h_q)
- $\mathcal{R}$, _ = `SearchOnGraph`($\mathcal{G}$, $\mathbf{n}$, h_q, k, *ef*)
- **si** *existente* $\neq$ **nulo y** *existente* $\notin \mathcal{R}$ **entonces**
  - `// Coincidencia exacta no recuperada por la búsqueda aproximada`
  - Insertar $(0,$ *existente*$)$ al principio de $\mathcal{R}$
  - $\mathcal{R}$ = los primeros k elementos de $\mathcal{R}$
- **fin**
- **devolver** $\mathcal{R}$

Etapa	**Símbolo**	**Descripción**	**Valor por defecto**
Inicialización	*init*	Estrategia: aleatoria o bosque de *VP-trees*	VP
	t	Número de *VP-trees* del bosque	16
	ℓ	Tamaño máximo de hoja	32
NN-Descent	K	Vecinos conservados por nodo	50
	L	Tamaño de la lista de candidatos por nodo	400
	S	Candidatos nuevos muestreados por iteración	10
	R	Número máximo de vecinos inversos	200
	T	Iteraciones de refinamiento	10
Grafo NSG	m	Grado de salida máximo	16
	c	Candidatos examinados en la selección de aristas	500
	ef	Tamaño de la lista de candidatos en construcción	400
	α	Factor de relajación de la regla de selección	1,2
Consulta	k	Número de vecinos devueltos	—
	ef_{search}	Tamaño de la lista de candidatos	400
Global	*seed*	Semilla del generador pseudoaleatorio	42

Tabla 3.1: Parámetros configurables del sistema, agrupados según la etapa en la que intervienen.

Capítulo 4

Evaluación y discusión de resultados

En este capítulo se evalúa experimentalmente la implementación desarrollada. El objetivo es determinar en qué medida la sustitución de HNSW por NSG como estructura de indexación de `APOTHEOSIS` mejora el compromiso entre calidad de la búsqueda, tiempo de consulta, tiempo de construcción y tamaño del índice. La evaluación se articula en torno a cuatro preguntas: si la ausencia de jerarquía se traduce efectivamente en índices más compactos, cómo se compara el tiempo de construcción de ambos índices, si la calidad de la búsqueda a igualdad de tiempo de consulta mejora respecto a HNSW, y cómo se comportan ambas estructuras al escalar el número de elementos indexados. El capítulo describe primero el entorno de experimentación y los conjuntos de datos empleados, detalla a continuación las métricas y las configuraciones de parámetros evaluadas, y presenta por último el análisis de los resultados obtenidos.

4.1. Entorno de experimentación

El entorno seleccionado para realizar las pruebas consiste en un servidor propiedad de la Universidad de Zaragoza, denominado *valhalla*, equipado con dos procesadores Intel(R) Xeon(R) Silver 4216 a 2,10 GHz, que suman un total de 32 núcleos y 64 hilos. El sistema cuenta con 512 GB de memoria RAM y ejecuta un sistema operativo Debian GNU/Linux en su versión 12 (*bookworm*). La implementación se ha desarrollado en Rust (edición 2024) y compilado con `rustc` 1.94.0 en perfil *release*, con `codegen-units = 1`. Todas las mediciones de tiempo se han tomado sobre binarios compilados con esta configuración.

4.2. Conjuntos de datos

La evaluación se ha llevado a cabo sobre dos conjuntos de datos de naturaleza distinta. El primero está formado por ficheros de código fuente y permite valorar la calidad de la búsqueda sobre artefactos textuales. El segundo está formado por páginas de memoria de binarios de sistema de Microsoft Windows y, por su tamaño, permite valorar el rendimiento y la escalabilidad del índice sobre datos binarios. Ambos conjuntos fueron empleados en la evaluación original de `APOTHEOSIS` [18].

El primer conjunto de datos es el publicado por Ljubovic y Pajic [22], que recoge las entregas de prácticas de dos asignaturas introductorias de programación de la Universidad de Sarajevo (Bosnia y Herzegovina). El conjunto contiene un total de 43.792 ficheros de código fuente. Para cada uno de estos ficheros se calcula el resumen de similitud utilizando `ssdeep` y `TLSH`.

El segundo conjunto de datos está formado por módulos de sistema (bibliotecas de enlace dinámico, controladores del núcleo y ejecutables de sistema) recopilados de distintas versiones de 64 bits de Microsoft Windows, concretamente Windows 7, Windows 8.1, Windows Server 2012 R2, Windows Server 2016 y Windows Server 2019 [23]. Un módulo es la representación en el espacio de memoria de un proceso de un fichero ejecutable o de una biblioteca compartida, mientras que una página es el bloque de tamaño fijo (habitualmente 4096 bytes) en el que se divide la memoria virtual de un proceso. En total, el conjunto reúne aproximadamente 4,2 millones de páginas pertenecientes a unos 37.500 ficheros de sistema. Por cada una de las páginas, se guarda el resumen de similitud calculado con dos algoritmos distintos: `ssdeep` y `TLSH`.

4.3. Métricas y configuraciones

Se detallan a continuación las distintas métricas y configuraciones de los parámetros utilizadas para evaluar el desempeño, la eficiencia y la escalabilidad del sistema desarrollado.

La evaluación se organiza como un barrido de parámetros sobre cada combinación de conjunto de datos y algoritmo de similitud aproximada. En el caso del primer conjunto de datos, se indexan 42.000 resúmenes, y en el caso del segundo conjunto, se indexan 50.000, 100.000, 200.000, 500.000 y 1.000.000 resúmenes. En ambos casos se reservan 1000 consultas disjuntas del corpus indexado y se evalúan con $k = 1$ y $k = 10$, los dos valores habituales en la literatura del área. El *ground truth* se calcula por fuerza bruta

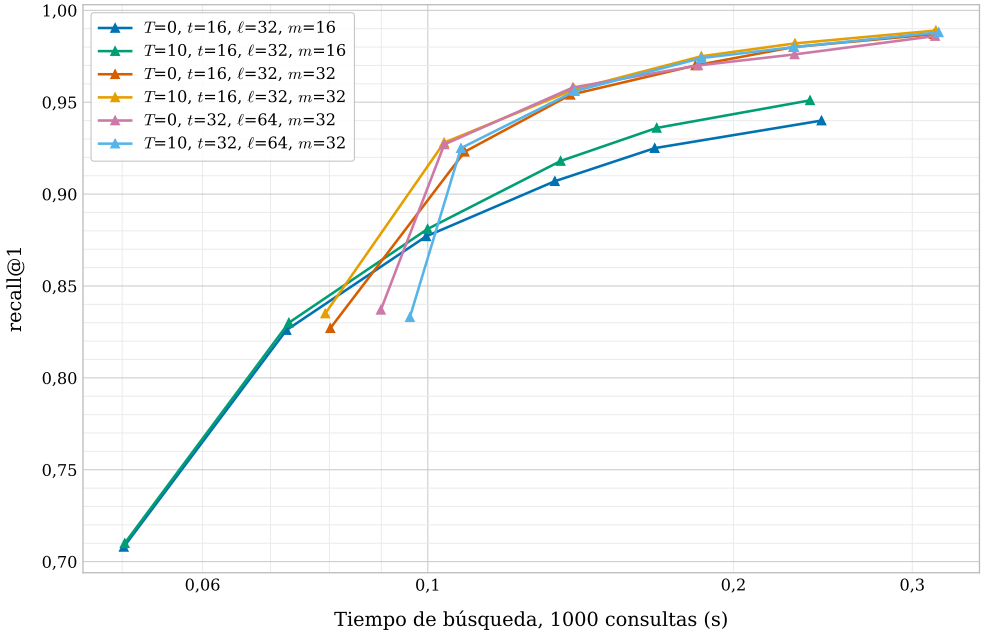
sobre el corpus indexado.

Los parámetros configurables de `APOTHEOSIS` con NSG vienen especificados en la Tabla 3.1. Los parámetros correspondientes al algoritmo NN-Descent se mantienen fijos en toda la evaluación, con $K = 50$ vecinos conservados por nodo, una lista de candidatos de tamaño $L = 400$, $S = 10$ candidatos nuevos muestreados por iteración y un máximo de $R = 200$ vecinos inversos. Estos son los valores recomendados en EFANNA [21], y las pruebas preliminares no mostraron mejoras apreciables en la calidad del grafo k-NN al modificarlos. El factor de relajación de la regla de selección de aristas se mantiene igualmente fijo en $\alpha = 1{,}2$, el valor por defecto del sistema, de modo que las diferencias observadas entre configuraciones no dependen de él. Las configuraciones evaluadas (resumidas en la Tabla 4.1) varían dos aspectos de interés. El primero es la inicialización del grafo k-NN aproximado: o bien aleatoria o bien haciendo uso del bosque de *VP-trees*. El segundo es el grado máximo de salida m del grafo NSG, por ser el parámetro que fija el tamaño del índice. El resto de parámetros de selección de aristas permanece fijo, con $c = 200$ candidatos examinados y una lista de candidatos de construcción de tamaño $ef = 64$. Ambos valores son inferiores a los que la Tabla 3.1 recoge como valores por defecto ($c = 500$ y $ef = 400$). Esto se debe a que las pruebas preliminares mostraron que reducirlos acorta apreciablemente el tiempo de construcción sin degradar la calidad del grafo resultante, lo que resulta especialmente relevante al indexar un millón de elementos.

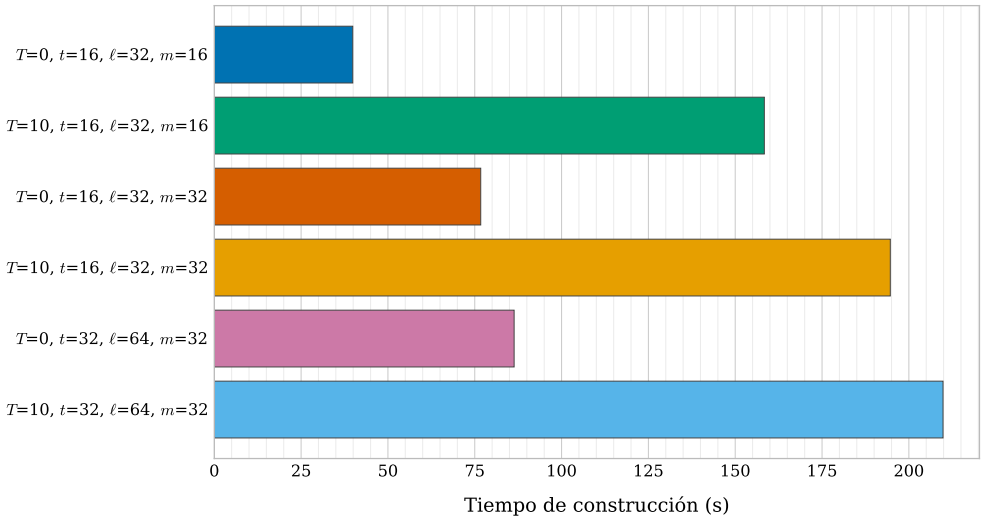
Configuración	**Inicialización**	$\boldsymbol{T}$	$\boldsymbol{t}$	$\boldsymbol{\ell}$	$\boldsymbol{m}$
N1	*VP-trees*	0	16	32	32
N2	aleatoria	10	—	—	32
N3	*VP-trees*	0	32	64	32
N4	*VP-trees*	0	16	32	16

Tabla 4.1: Configuraciones de construcción evaluadas para NSG. T es el número de iteraciones de NN-Descent, t y ℓ el número de árboles y el tamaño máximo de hoja del bosque de *VP-trees*, respectivamente, y m el grado máximo de salida.

Cabe destacar que en el caso de la inicialización con *VP-trees* no se aplica ninguna pasada de NN-Descent. Esto se debe a que experimentalmente se ha determinado que la calidad del grafo k-NN aproximado que produce esta inicialización ya es lo suficientemente buena, y refinarlo con NN-Descent apenas produce mejoras, si bien dispara el tiempo de construcción. La Figura 4.1 muestra esta experimentación sobre el conjunto de binarios de Windows con `TLSH` y 100.000 elementos indexados. Como se puede apreciar, las iteraciones de refinamiento apenas alteran la calidad de la búsqueda, mientras que el tiempo de construcción crece de forma apreciable con cada una de ellas.



(a) Calidad de la búsqueda



(b) Tiempo de construcción

Figura 4.1: Efecto de refinar con NN-Descent el grafo k-NN obtenido del bosque de *VP-trees*, sobre el conjunto de binarios de Windows con `TLSH` y 100.000 elementos indexados.

Puesto que se van a comparar los resultados obtenidos con el nuevo sistema con los obtenidos con HNSW, es necesario fijar una serie de configuraciones también para HNSW. Los parámetros configurables para HNSW son el grado máximo de salida en las capas superiores, M, el grado M_0 en la capa 0, el tamaño ef de la lista de candidatos durante la inserción y su equivalente ef_{search} durante la consulta. La Tabla 4.2 recoge las configuraciones evaluadas.

Configuración	M	M_0	ef
H1	16	32	64
H2	32	64	64
H3	32	64	128
H4	32	64	200

Tabla 4.2: Configuraciones de construcción evaluadas para HNSW.

La calidad de la búsqueda se mide mediante el *recall@k* definido en la Sección 2.2, y se presenta frente al tiempo empleado en resolver el conjunto de consultas. Ambos crecen a la vez al aumentar el tamaño de la lista de candidatos de consulta, $ef_{\text{search}} \in \{32, 64, 100, 150, 200, 300\}$, de manera que cada configuración no produce un único valor de *recall* sino una curva. Cuanto más arriba a la izquierda se sitúe la curva, mejores son los resultados. Se miden además el tiempo de construcción del índice y su tamaño en disco, este último sobre la representación serializada del grafo. Se busca con esto valorar la escalabilidad de ambas estructuras y, en particular, el sobrecoste en memoria que introduce la jerarquía de capas de HNSW.

4.4. Análisis de resultados

En la Tabla 4.3 se muestra el tamaño en disco del índice sobre el conjunto de ejercicios de programación para cada una de las configuraciones. En la Tabla 4.4 y en la Figura 4.2 se muestra el tamaño en disco del índice sobre el conjunto de binarios de Windows en función del número de nodos indexados.

El tamaño del índice crece linealmente con el número de nodos en ambas estructuras. A igualdad de grado de salida, el tamaño en disco del modelo `APOTHEOSIS` con NSG es en torno a un 30 % menor que el de la variante con HNSW con $M = 32$ (595,83 frente a 854,33 MiB con `TLSH` a un millón de nodos) y a un 23 % con $M = 16$ (474,01 frente a 611,35 MiB). Las cifras sobre `ssdeep` son prácticamente idénticas. El resultado más relevante es que el índice NSG de grado 32 ocupa incluso algo menos que el índice HNSW de grado 16, de modo que la ausencia de jerarquía permite duplicar el grado

de salida sin coste alguno en espacio.

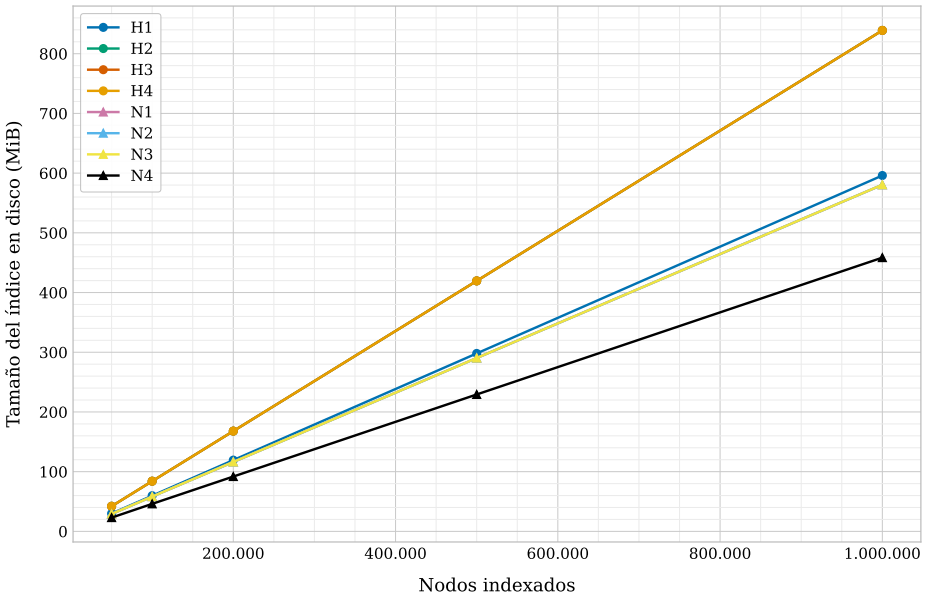
El tamaño depende únicamente del grado de salida. Las tres configuraciones de HNSW que comparten M y M_0 producen volcados de tamaño similar, independientemente del valor ef. Una situación similar ocurre con NSG, en la que ni la estrategia de inicialización ni el tamaño del bosque de *VP-trees* influyen prácticamente en el índice resultante. Conviene precisar que estas cifras corresponden al volcado completo, que almacena también los propios resúmenes de similitud, comunes a ambas estructuras.

Resumen	**Algoritmo**	**Configuración**	**Tamaño (MiB)**
`ssdeep`	HNSW	H1	11,93
		H2–H4	16,46
	NSG	N1	11,72
		N2	11,65
		N3	11,65
		N4	9,49
`TLSH`	HNSW	H1	11,46
		H2–H4	16,00
	NSG	N1	11,18
		N2	11,18
		N3	11,18
		N4	8,91

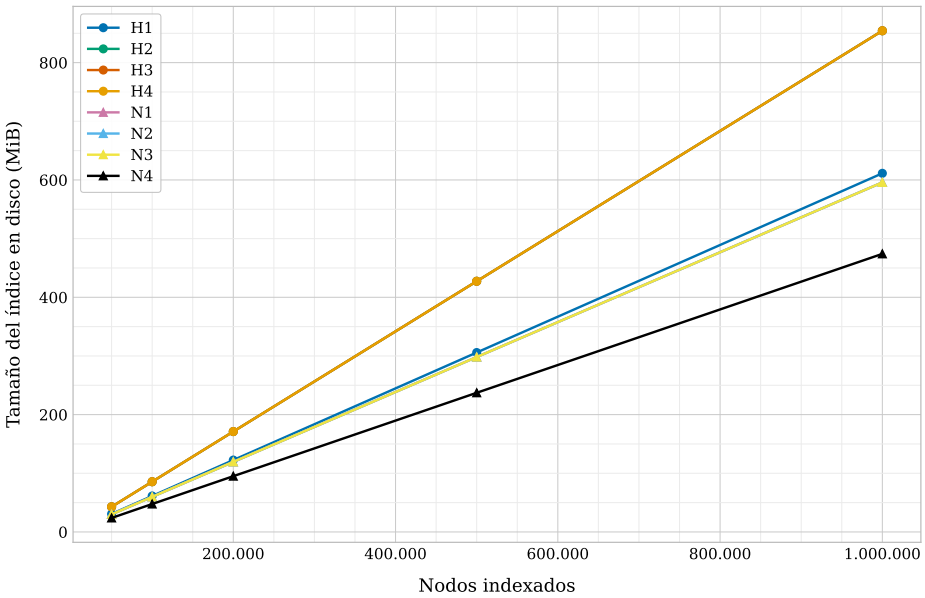
Tabla 4.3: Tamaño en disco del índice sobre el conjunto de ejercicios de programación, según las configuraciones de las Tablas 4.1 y 4.2.

En la Tabla 4.5 se muestra el tiempo de construcción del índice sobre el conjunto de ejercicios de programación. Como se puede observar, el modelo de `APOTHEOSIS` con NSG requiere más tiempo de construcción en comparación con la variante con HNSW. Esta diferencia se acentúa en el caso de los resúmenes producidos por `ssdeep`. Esto puede deberse a que en la construcción de NSG se lleva a cabo un número elevado de comparaciones (tanto en la creación del grafo k-NN aproximado como en la propia selección de aristas de NSG), y a que la comparación de dos resúmenes `ssdeep` es más cara que la de dos resúmenes `TLSH`. La diferencia se aprecia en el cálculo del *ground truth* por fuerza bruta, que realiza exactamente el mismo número de comparaciones con ambos algoritmos. Así, en el caso del conjunto de binarios de Windows, sobre el millón de nodos indexados y las mil consultas, el cálculo del *ground truth* con `ssdeep` tarda 18,82 segundos, frente a los 1,12 segundos con `TLSH`.

En la Figura 4.3 se muestra el tiempo de construcción del índice sobre el conjunto de binarios de Windows frente al número de nodos indexados. Nótese la escala logarítmica



(a) `ssdeep`



(b) `TLSH`

Figura 4.2: Tamaño en disco del índice frente al número de nodos indexados sobre el conjunto de binarios de Windows.

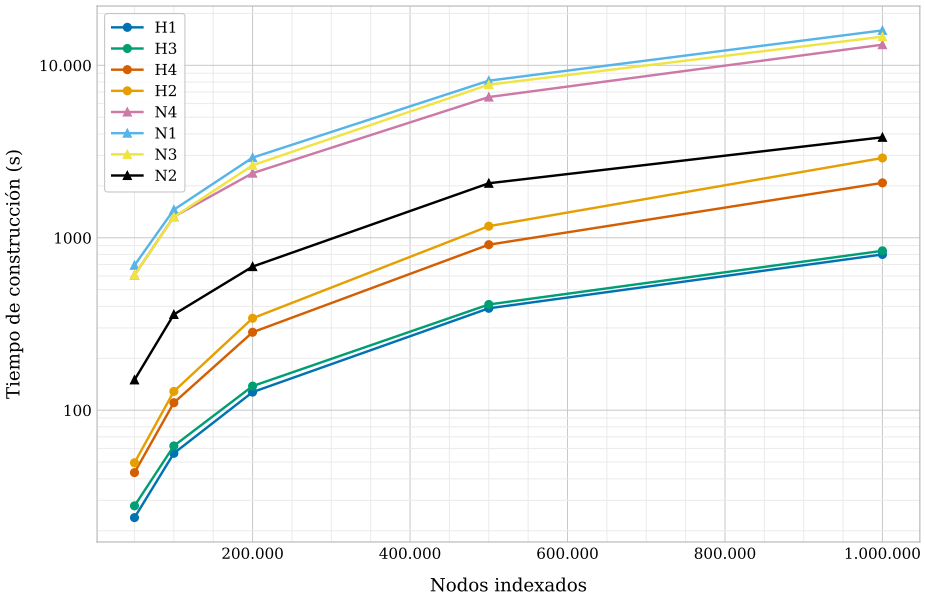
			Nodos indexados				
Resumen	**Algoritmo**	**Config.**	**50.000**	**100.000**	**200.000**	**500.000**	**1.000.000**
`ssdeep`	HNSW	H1	29,92	59,78	119,35	298,03	596,02
		H2–H4	42,07	84,06	167,93	419,51	839,00
	NSG	N1	29,14	58,22	116,23	290,30	580,50
		N2	29,14	58,22	116,23	290,29	580,48
		N3	29,14	58,23	116,23	290,30	580,50
		N4	23,04	46,07	91,82	229,28	458,45
`TLSH`	HNSW	H1	30,72	61,34	122,53	305,85	611,35
		H2–H4	42,87	85,62	171,11	427,32	854,33
	NSG	N1	29,94	59,79	119,42	298,12	595,83
		N2	29,94	59,79	119,42	298,12	595,83
		N3	29,94	59,79	119,41	298,11	595,81
		N4	23,85	47,60	95,05	237,21	474,01

Tabla 4.4: Tamaño en disco del índice, en MiB, sobre el conjunto de binarios de Windows, según las configuraciones de las Tablas 4.1 y 4.2.

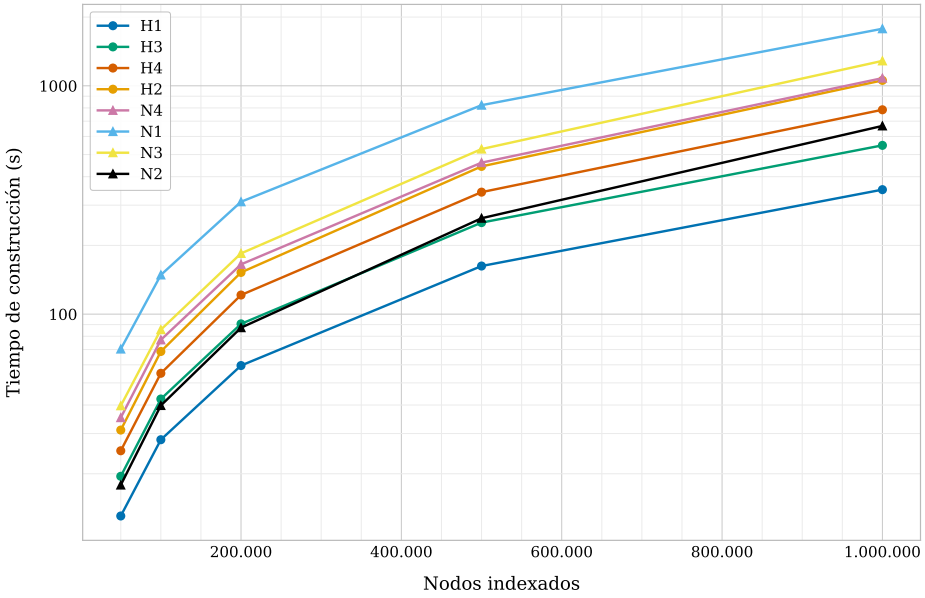
del eje vertical. El tiempo de construcción crece de forma próxima a lineal con el número de nodos en ambas estructuras. Con `TLSH`, NSG cuesta aproximadamente el doble que HNSW a igualdad de grado de salida. Con `ssdeep`, en cambio, todas las curvas de NSG quedan por encima de todas las de HNSW, con una diferencia próxima a un orden de magnitud. La razón es el número de comparaciones, como se ha comentado en el caso del conjunto de datos de ejercicios de programación.

Dentro de NSG, sobre el conjunto de binarios de Windows, la inicialización mediante el bosque de *VP-trees* resulta más barata que la aleatoria seguida de $T = 10$ iteraciones de NN-Descent, y el bosque con más árboles y hojas de mayor tamaño ($t = 32$, $\ell = 64$) es el más caro, como cabría esperar.

En la Figura 4.4 se muestran el *recall*@1 y el *recall*@10 frente al tiempo de búsqueda sobre el conjunto de ejercicios de programación. Con `ssdeep` los resultados obtenidos por NSG en cualquiera de sus configuraciones superan claramente a los obtenidos por HNSW, y el comportamiento es idéntico para ambos valores de k. Con `TLSH` y $k = 1$ ambos índices alcanzan un *recall* de 1,0, si bien NSG lo hace antes, en torno a los 0,04 segundos, frente a los 0,12 segundos que tarda HNSW. Cabe destacar el caso de `TLSH` con $k = 10$, en el que ninguna de las variantes es capaz de alcanzar un *recall* de 0,65. Una explicación plausible de este comportamiento es la presencia de empates en las distancias calculadas con `TLSH`. Los elementos que se indexan son ficheros de código fuente con estructura similar, lo que hace posible que varios elementos queden a la misma distancia de la consulta, de forma que el *ground truth* no queda unívocamente



(a) `ssdeep`



(b) `TLSH`

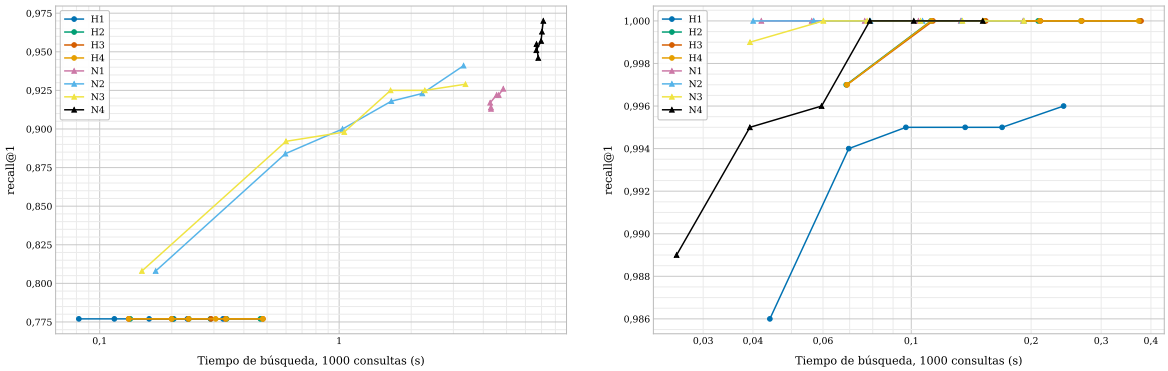
Figura 4.3: Tiempo de construcción del índice frente al número de nodos indexados sobre el conjunto de binarios de Windows.

Resumen	Algoritmo	Configuración	Tiempo (s)
`ssdeep`	HNSW	H1	5,30
		H2	8,08
		H3	9,53
		H4	11,09
	NSG	N1	241,03
		N2	241,71
		N3	206,26
		N4	115,19
`TLSH`	HNSW	H1	4,56
		H2	6,78
		H3	8,60
		H4	10,61
	NSG	N1	12,25
		N2	22,11
		N3	13,33
		N4	6,16

Tabla 4.5: Tiempo de construcción del índice sobre el conjunto de ejercicios de programación, según las configuraciones de las Tablas 4.1 y 4.2.

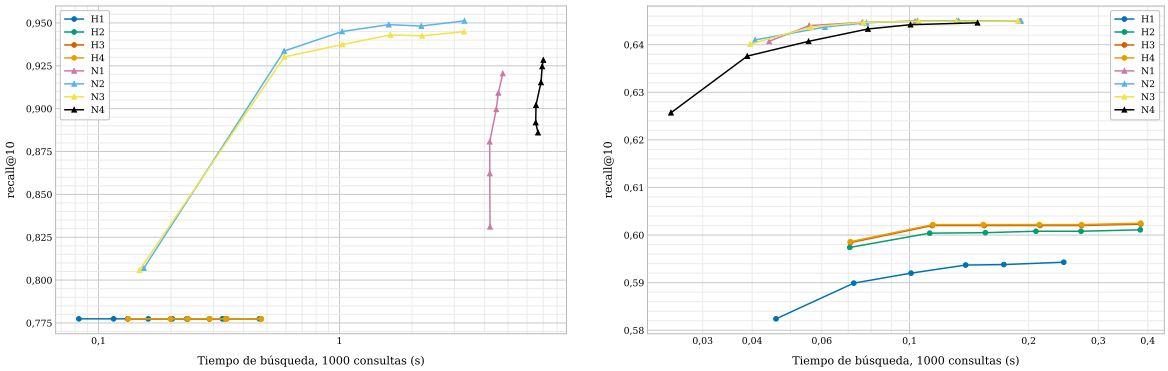
determinado.

Las Figuras 4.5 a 4.9 muestran el *recall*@1 y el *recall*@10 frente al tiempo de búsqueda sobre el conjunto de binarios de Windows. En estas figuras el eje horizontal, que recoge el tiempo de búsqueda, está en escala logarítmica. Se observa cómo el *recall* se degrada conforme crece el número de nodos indexados, lo que obliga a aumentar ef_{search} al escalar, y con ello el tiempo de consulta. Esta degradación parece afectar por igual a ambas estructuras.



(a) `ssdeep`, $k = 1$

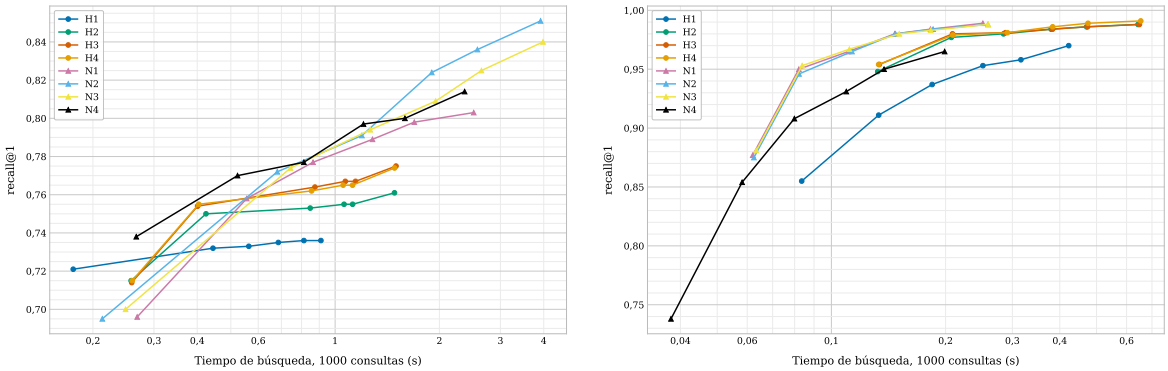
(b) `TLSH`, $k = 1$



(c) `ssdeep`, $k = 10$

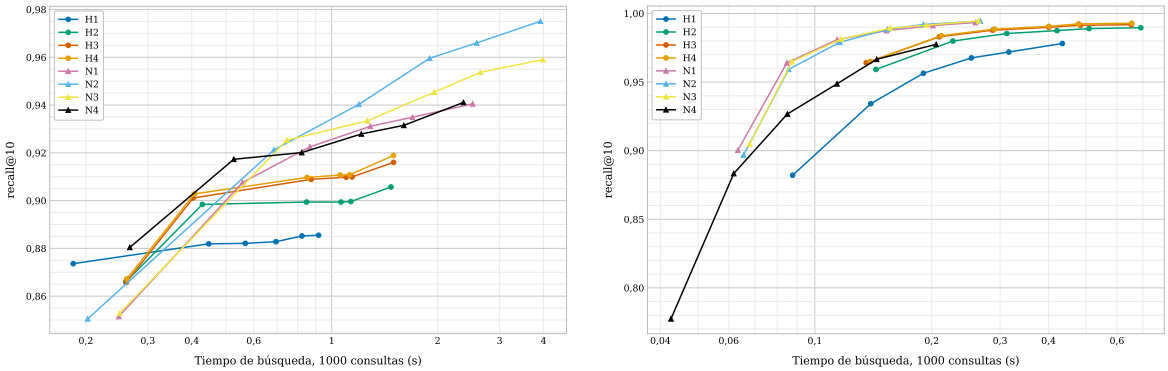
(d) `TLSH`, $k = 10$

Figura 4.4: Recall@k frente al tiempo de búsqueda sobre el conjunto de ejercicios de programación, con 42.000 resúmenes indexados.



(a) `ssdeep`, $k = 1$

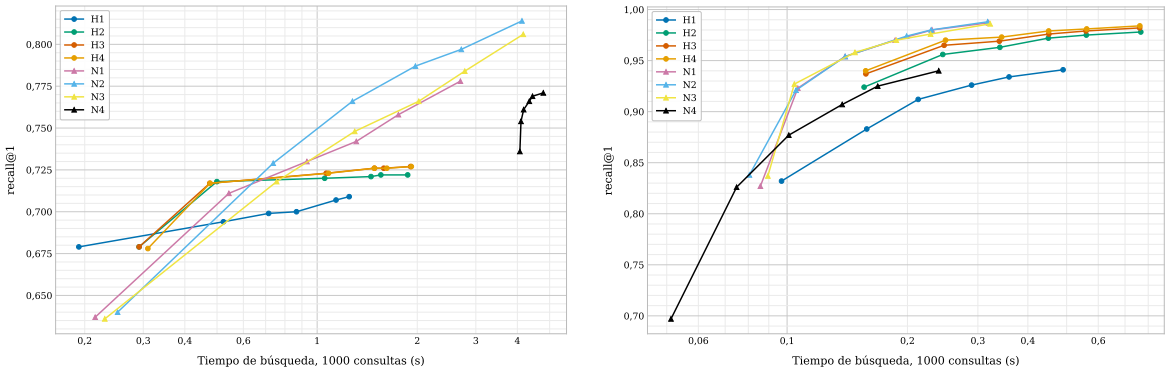
(b) `TLSH`, $k = 1$



(c) `ssdeep`, $k = 10$

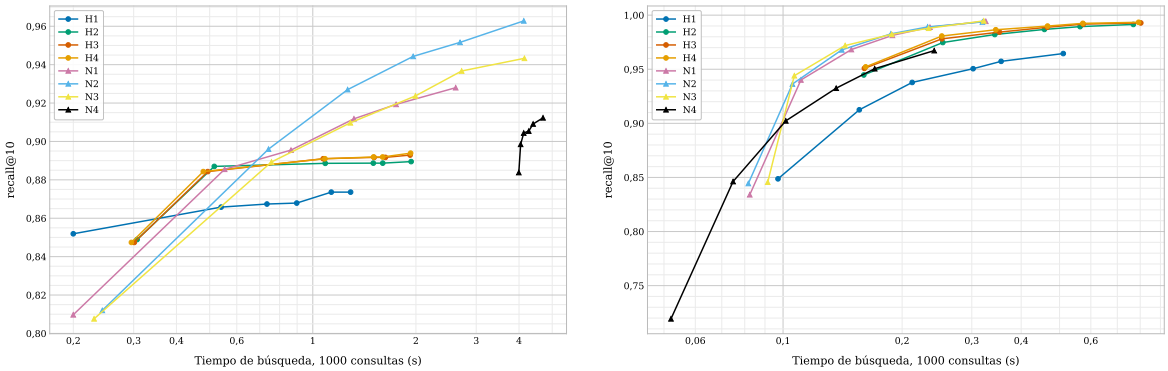
(d) `TLSH`, $k = 10$

Figura 4.5: Recall@k frente al tiempo de búsqueda sobre el conjunto de binarios de Windows, con 50.000 resúmenes indexados.



(a) `ssdeep`, $k = 1$

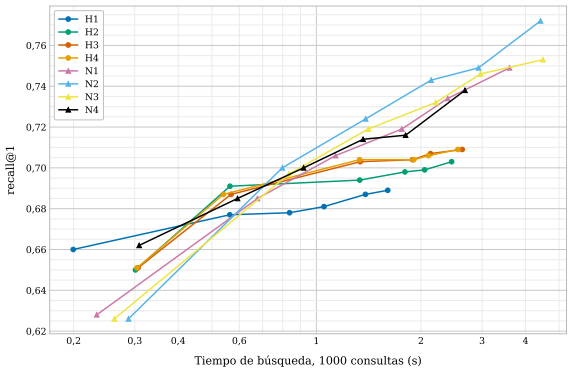
(b) `TLSH`, $k = 1$



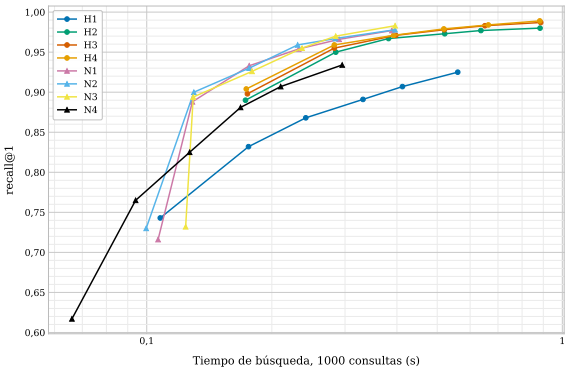
(c) `ssdeep`, $k = 10$

(d) `TLSH`, $k = 10$

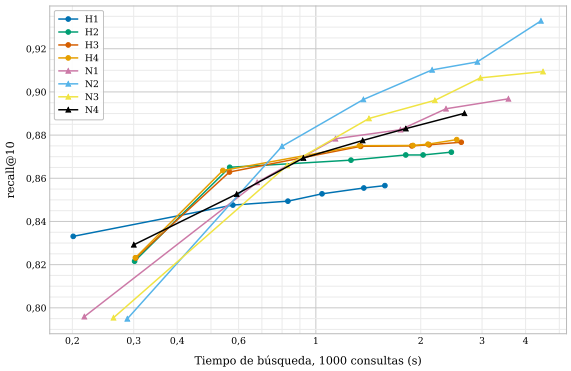
Figura 4.6: Recall@k frente al tiempo de búsqueda sobre el conjunto de binarios de Windows, con 100.000 resúmenes indexados.



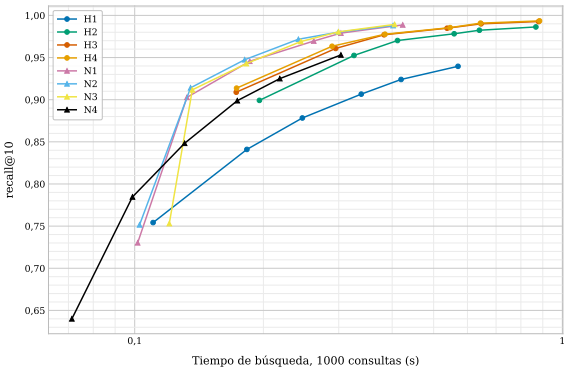
(a) `ssdeep`, $k = 1$



(b) `TLSH`, $k = 1$

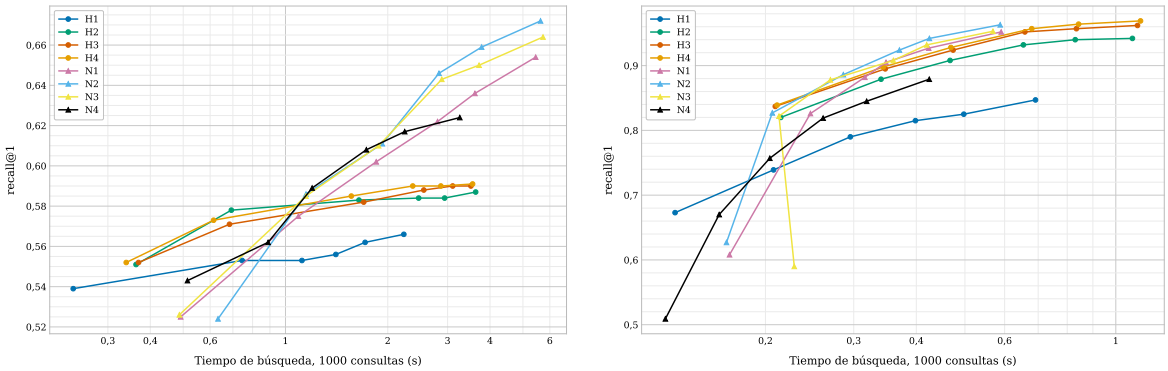


(c) `ssdeep`, $k = 10$



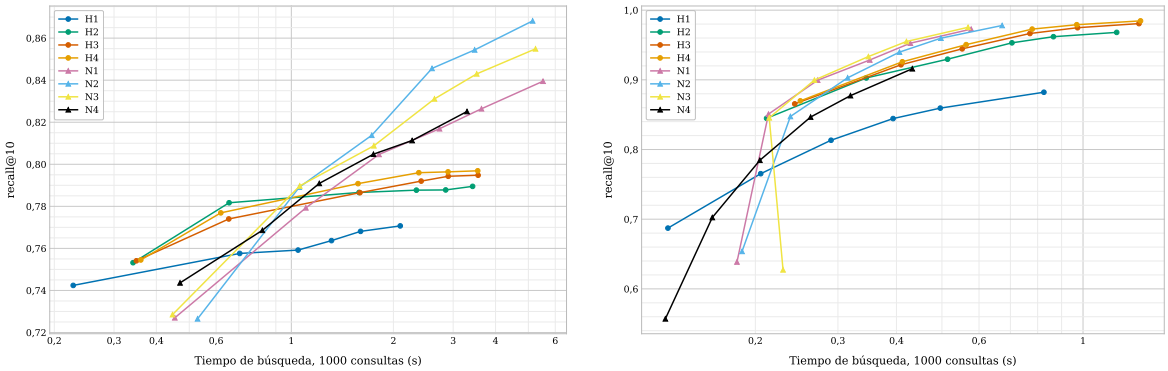
(d) `TLSH`, $k = 10$

Figura 4.7: Recall@k frente al tiempo de búsqueda sobre el conjunto de binarios de Windows, con 200.000 resúmenes indexados.



(a) `ssdeep`, $k = 1$

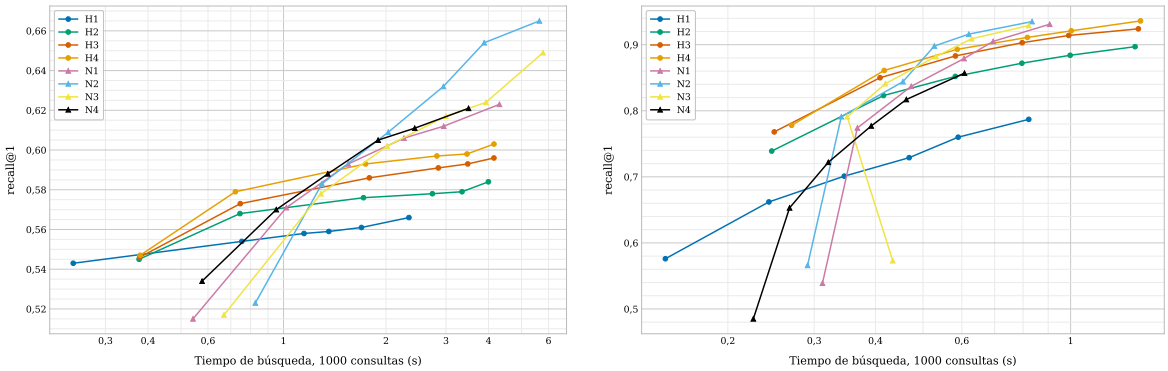
(b) `TLSH`, $k = 1$



(c) `ssdeep`, $k = 10$

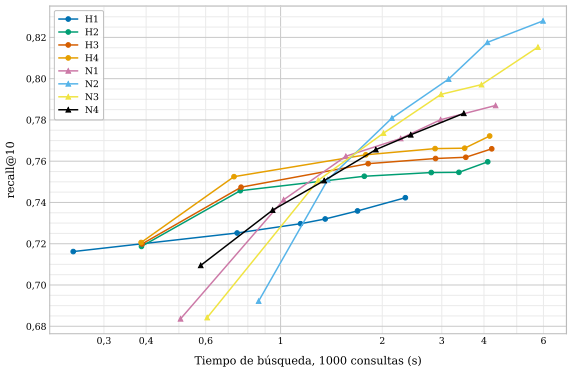
(d) `TLSH`, $k = 10$

Figura 4.8: Recall@k frente al tiempo de búsqueda sobre el conjunto de binarios de Windows, con 500.000 resúmenes indexados.

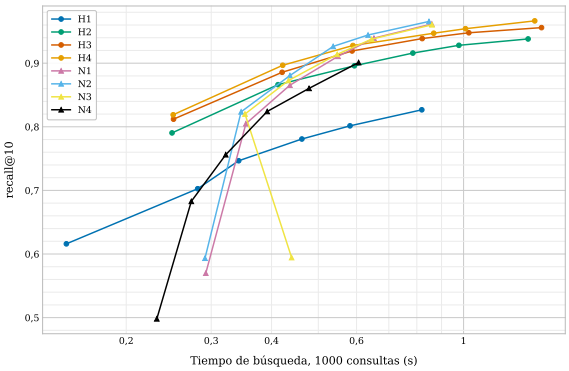


(a) `ssdeep`, $k = 1$

(b) `TLSH`, $k = 1$



(c) `ssdeep`, $k = 10$



(d) `TLSH`, $k = 10$

Figura 4.9: Recall@k frente al tiempo de búsqueda sobre el conjunto de binarios de Windows, con 1.000.000 de resúmenes indexados.

El análisis realizado confirma solo en parte los resultados esperados de forma teórica. Se ha conseguido reducir el tamaño de los modelos al eliminar la sobrecarga de memoria derivada de la estructura jerárquica de HNSW. En cuanto al tiempo de consulta, NSG es la única de las dos estructuras que alcanza los valores más altos de *recall*, y los alcanza además en menos tiempo, si bien HNSW resulta más rápido cuando no se exigen valores altos de recall y el número de nodos indexados es grande. Sin embargo, los tiempos de construcción del índice con NSG son muy superiores a los del índice con HNSW, un compromiso a tener en cuenta. Por último, cabe destacar el buen desempeño obtenido por NSG pese al hecho de que los datos indexados no son métricos.

Capítulo 5

Estado del arte

Este trabajo se sitúa en la intersección de dos líneas de investigación que han evolucionado de forma prácticamente independiente. Por un lado, la búsqueda aproximada de vecinos más próximos, desarrollada sobre todo en el ámbito de las bases de datos y la recuperación de información, y orientada a representaciones vectoriales densas. Por otro, la indexación de resúmenes de similitud, desarrollada en el ámbito del análisis forense digital. Este capítulo revisa estas dos líneas y sitúa la aportación de este trabajo respecto a ellas.

5.1. Grafos de proximidad para la búsqueda aproximada

Los métodos propuestos para la búsqueda aproximada de vecinos más próximos se agrupan habitualmente en tres familias: los basados en estructuras jerárquicas que particionan el espacio, como *k-d trees* [11], los basados en *hashing* [2] y los basados en grafos de proximidad. Durante la última década, esta última familia se ha consolidado como la que ofrece mejores compromisos entre precisión y tiempo de consulta, dominando la literatura experimental del área [6, 7].

Entre los métodos basados en grafos se pueden distinguir dos tipos. Por un lado, aquellos que parten de un grafo k-NN aproximado que después se refina mediante alguna estrategia de selección de aristas. NN-Descent [13] proporciona el mecanismo de construcción, EFANNA [21] mejora su inicialización mediante un bosque de árboles aleatorizados y NSG [5] introduce la poda basada en la teoría de los MRNG, grafos que garantizan caminos monótonos entre cualquier par de nodos, garantía que NSG relaja al camino desde el *navigating node*. NSSG [24] continúa esta línea sustituyendo la regla

de MRNG por una condición angular: en lugar de decidir en función de las longitudes de las aristas, se consideran en conflicto dos aristas de salida de un mismo nodo cuyo ángulo es menor que un umbral dado. En este caso, se descarta la más larga, de manera que las aristas de cada nodo quedan repartidas de forma uniforme en todas las direcciones. La propuesta surge tras identificar dos limitaciones de NSG: la ausencia de garantías teóricas cuando la consulta no pertenece al conjunto indexado, y la reducida densidad del grafo, que alarga los recorridos de búsqueda. Sin embargo, la evaluación independiente de Wang et al. [7] llega a la conclusión opuesta sobre la superioridad de una regla frente a la otra. Dicha evaluación ilustra hasta qué punto los resultados de esta área dependen del conjunto de datos y del ajuste de parámetros.

Por otro lado, se encuentran los grafos de pequeño mundo navegable. NSW [15] construye el grafo insertando los elementos uno a uno y conectándolos con los más próximos ya insertados, y HNSW [4] añade sobre esa idea una jerarquía de capas inspirada en las listas por saltos. Vamana, el índice empleado por DiskANN [19], combina elementos de ambos: adopta el esquema de construcción de NSG pero generaliza la regla de poda mediante un factor de relajación α. Trabajos recientes han mostrado que dicho parámetro debe ajustarse con cuidado, ya que valores mayores conservan más vecinos de salida y encarecen tanto el índice como el recorrido [25].

Es importante recalcar que, si bien NN-Descent se presenta explícitamente para medidas de similitud genéricas, tanto los índices de grafo revisados como sus evaluaciones experimentales asumen datos vectoriales y medidas de disimilitud como la distancia euclídea, la distancia del coseno o el producto interno. Los conjuntos de prueba habituales (SIFT, GIST, GloVe, Deep) [26] son colecciones de descriptores o de *embeddings*. La aplicabilidad de estos métodos a datos que no admiten una representación por coordenadas queda, por tanto, fuera del alcance de los trabajos revisados.

5.2. Indexación de resúmenes de similitud en análisis forense

El uso de algoritmos de similitud aproximada en análisis forense se remonta a `ssdeep` [9] y `sdhash` [27], a los que siguieron `TLSH` [10] y otros. El NIST fijó la terminología del área [3] y Martín-Pérez et al. [28] propusieron una clasificación sistemática de estos algoritmos junto con un análisis de los ataques conocidos. Breitinger et al. [29] formularon explícitamente el problema de buscar de forma eficiente sobre una base de datos de resúmenes. Las soluciones propuestas, como F2S2 [30], los árboles de filtros

de Bloom [31, 32] y las variantes recogidas en [33], resuelven un problema de filtrado, devolviendo candidatos en lugar de una ordenación de los k elementos más próximos.

`APOTHEOSIS` [1, 18], por su parte, es uno de los primeros sistemas que aplica un índice de grafo moderno a resúmenes de similitud, combinando un *radix tree* para la búsqueda exacta con una implementación propia de HNSW para la aproximada. Sus autores señalan entre las limitaciones del sistema que HNSW es solo una de las alternativas posibles y que una evaluación más profunda de otros métodos ayudaría a determinar cuál se adapta mejor a cada problema. Este trabajo puede entenderse como una respuesta parcial a esa cuestión abierta.

5.3. Aportaciones

La aportación de este trabajo se sitúa en la intersección de estas dos líneas. La idea fundamental ha sido adaptar uno de los métodos basados en grafos (diseñados y evaluados sobre datos vectoriales), NSG, para que pueda operar sobre resúmenes de similitud. En concreto, se ha desarrollado una implementación de NSG que no presupone ninguna estructura vectorial sobre el conjunto de datos y que únicamente requiere una función de similitud entre pares de elementos, y se ha integrado en `APOTHEOSIS` como alternativa a HNSW.

Por último, se ha discutido qué implicaciones tiene aplicar la regla de selección de aristas de NSG, sustentada en la desigualdad triangular, sobre resúmenes de similitud que no son métricos, y se ha introducido en ella un factor de relajación que trata de compensarlo. La evaluación se ha llevado a cabo sobre dos conjuntos de datos forenses reales, en lugar de los corpus de descriptores y *embeddings* habituales en la literatura.

Capítulo 6

Conclusiones y trabajo futuro

Este trabajo extiende la herramienta `APOTHEOSIS` con NSG, una estructura alternativa de búsqueda aproximada de vecinos más próximos basada en grafos. Dicha estructura se ha adaptado de forma que el único requisito para los datos de entrada es la existencia de una función de similitud entre pares de elementos. Su integración en `APOTHEOSIS` ha supuesto, no obstante, un cambio importante en su funcionamiento, dado que el modelo ya no puede modificarse de forma dinámica (esto es, insertar o eliminar un nodo implicaría la reconstrucción del modelo completo).

Los resultados de la evaluación son positivos. Se ha conseguido reducir el tamaño del índice, eliminando la sobrecarga de memoria derivada de la estructura jerárquica de HNSW. En cuanto al tiempo de consulta, NSG alcanza valores de *recall* más altos que HNSW y en menos tiempo, si bien HNSW resulta más rápido cuando no se exigen valores altos de *recall* y el número de nodos indexados es grande. Por otra parte, se ha observado que en general los tiempos de construcción del índice NSG son muy superiores a los del índice HNSW. Este compromiso entre tiempo de búsqueda y tamaño del índice, por un lado, y tiempo de construcción, por otro, puede resultar de interés en función del caso de uso de la herramienta `APOTHEOSIS`.

La evaluación muestra además que NSG mantiene un buen desempeño pese a que los resúmenes de similitud empleados no constituyen una métrica. Como trabajo futuro, se plantea cuantificar hasta qué punto influye este hecho, para lo cual sería necesario estudiar el efecto del factor de relajación α, fijo en toda la evaluación, y extender el estudio a otros algoritmos de búsqueda aproximada de vecinos más próximos.

Finalmente, y de cara a mejorar la integración de NSG en `APOTHEOSIS`, sería interesante explorar estrategias que permitan la inserción o eliminación de nodos en el índice de forma dinámica. Una posibilidad es seguir un enfoque similar al que adopta Fresh-

DiskANN [34] sobre Vamana, en el que las inserciones y eliminaciones se resuelven localmente sobre el grafo, reconstruyendo el índice por completo únicamente al superar un cierto umbral de degradación.

Bibliografía

[1] D. Huici, R. J. Rodríguez y E. Mena, «APOTHEOSIS: An efficient approximate similarity search system,» *SoftwareX*, vol. 29, pág. 102 016, 2025, ISSN: 2352-7110.

[2] P. Indyk y R. Motwani, «Approximate nearest neighbors: towards removing the curse of dimensionality,» en *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC '98)*, New York, NY, USA: Association for Computing Machinery, 1998, págs. 604-613, ISBN: 0897919629.

[3] F. Breitinger, B. Guttman, M. McCarrin, V. Roussev y D. White, «Approximate Matching: Definition and Terminology,» National Institute of Standards and Technology, NIST Special Publication 800-168, 2014.

[4] Y. A. Malkov y D. A. Yashunin, «Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs,» *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, n.º 4, págs. 824-836, abr. de 2020, ISSN: 0162-8828.

[5] C. Fu, C. Xiang, C. Wang y D. Cai, «Fast approximate nearest neighbor search with the navigating spreading-out graph,» *Proc. VLDB Endow.*, vol. 12, n.º 5, págs. 461-474, ene. de 2019, ISSN: 2150-8097.

[6] W. Li et al., «Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement,» *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, n.º 8, págs. 1475-1488, 2020.

[7] M. Wang, X. Xu, Q. Yue e Y. Wang, «A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search,» *Proc. VLDB Endow.*, vol. 14, n.º 11, págs. 1964-1978, 2021, ISSN: 2150-8097.

[8] A. J. Menezes, P. C. van Oorschot y S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996, ISBN: 0-8493-8523-7.

[9] J. Kornblum, «Identifying almost identical files using context triggered piecewise hashing,» *Digital Investigation*, vol. 3, págs. 91-97, 2006, The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06), ISSN: 1742-2876.

[10] J. Oliver, C. Cheng e Y. Chen, «TLSH — A Locality Sensitive Hash,» en *2013 Fourth Cybercrime and Trustworthy Computing Workshop (CTC)*, IEEE, 2013, págs. 7-13.

[11] J. L. Bentley, «Multidimensional binary search trees used for associative searching,» *Communications of the ACM*, vol. 18, n.º 9, págs. 509-517, 1975, ISSN: 0001-0782.

[12] P. N. Yianilos, «Data structures and algorithms for nearest neighbor search in general metric spaces,» en *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1993, págs. 311-321.

[13] W. Dong, M. Charikar y K. Li, «Efficient k-nearest neighbor graph construction for generic similarity measures,» en *Proceedings of the 20th International Con-*

ference on World Wide Web, ép. WWW '11, Hyderabad, India: Association for Computing Machinery, 2011, págs. 577-586, ISBN: 9781450306324.

[14] W. Pugh, «Skip lists: a probabilistic alternative to balanced trees,» *Communications of the ACM*, vol. 33, n.º 6, págs. 668-676, 1990, ISSN: 0001-0782.

[15] Y. Malkov, A. Ponomarenko, A. Logvinov y V. Krylov, «Approximate nearest neighbor algorithm based on navigable small world graphs,» *Information Systems*, vol. 45, págs. 61-68, 2014, ISSN: 0306-4379.

[16] G. T. Toussaint, «The relative neighbourhood graph of a finite planar set,» *Pattern Recognition*, vol. 12, n.º 4, págs. 261-268, 1980, ISSN: 0031-3203.

[17] M. Royo Miguel. «Implementación de NSG e integración en APOTHEOSIS.» Fork de `https://github.com/reverseame/APOTHEOSIS`. dirección: `https://github.com/844487/APOTHEOSIS/tree/nsg`

[18] D. Huici, R. J. Rodríguez y E. Mena, «An extensible and scalable system for hash lookup and approximate similarity search with similarity digest algorithms,» *Forensic Science International: Digital Investigation*, vol. 53, pág. 301 930, 2025, DFRWS USA 2025 — Selected Papers from the 25th Annual Digital Forensics Research Conference USA, ISSN: 2666-2817.

[19] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnaswamy y R. Kadekodi, «DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node,» en *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[20] J. Gonzalez, «If at first you don't succeed, trie, trie again: Correcting TLSH scalability claims for large-dataset malware forensics,» *Forensic Science International: Digital Investigation*, vol. 53, pág. 301 922, 2025.

[21] C. Fu y D. Cai, *EFANNA: An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph*, 2016. arXiv: `1609.07228 [cs.CV]`. dirección: `https://arxiv.org/abs/1609.07228`

[22] V. Ljubovic y E. Pajic, «Plagiarism Detection in Computer Programming Using Feature Extraction From Ultra-Fine-Grained Repositories,» *IEEE Access*, vol. 8, págs. 96 505-96 514, 2020, ISSN: 2169-3536.

[23] D. Huici y R. J. Rodríguez, «A dataset of Windows system binaries and similarity digests for enhanced forensic analysis,» *Data in Brief*, vol. 62, pág. 111 993, 2025, ISSN: 2352-3409.

[24] C. Fu, C. Wang y D. Cai, «High Dimensional Similarity Search With Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, n.º 8, págs. 4139-4150, 2022.

[25] S. Yang et al., «Revisiting the Index Construction of Proximity Graph-Based Approximate Nearest Neighbor Search,» *Proc. VLDB Endow.*, vol. 18, n.º 6, págs. 1825-1838, 2025.

[26] M. Aumüller, E. Bernhardsson y A. Faithfull, «ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms,» *Information Systems*, vol. 87, pág. 101 374, 2020, ISSN: 0306-4379.

[27] V. Roussev, «Data Fingerprinting with Similarity Digests,» en *Advances in Digital Forensics VI*, K.-P. Chow y S. Shenoi, eds., ép. IFIP Advances in Information and Communication Technology, vol. 337, Springer Berlin Heidelberg, 2010, págs. 207-226.

[28] M. Martín-Pérez, R. J. Rodríguez y F. Breitinger, «Bringing order to approximate matching: Classification and attacks on similarity digest algorithms,» *Forensic Science International: Digital Investigation*, vol. 36, pág. 301 120, 2021, ISSN: 2666-2817.

[29] F. Breitinger, H. Baier y D. White, «On the database lookup problem of approximate matching,» *Digital Investigation*, vol. 11, n.º Supplement 1, S1-S9, 2014, Proceedings of the First Annual DFRWS Europe, ISSN: 1742-2876.

[30] C. Winter, M. Schneider e Y. Yannikos, «F2S2: Fast forensic similarity search through indexing piecewise hash signatures,» *Digital Investigation*, vol. 10, n.º 4, págs. 361-371, 2013, ISSN: 1742-2876.

[31] F. Breitinger, C. Rathgeb y H. Baier, «An Efficient Similarity Digests Database Lookup — A Logarithmic Divide & Conquer Approach,» *Journal of Digital Forensics, Security and Law*, vol. 9, n.º 2, págs. 155-166, 2014.

[32] D. Lillis, F. Breitinger y M. Scanlon, «Hierarchical Bloom Filter Trees for Approximate Matching,» *Journal of Digital Forensics, Security and Law*, vol. 13, n.º 1, págs. 81-96, 2018.

[33] V. H. G. Moia y M. A. A. Henriques, «Similarity Digest Search: A Survey and Comparative Analysis of Strategies to Perform Known File Filtering Using Approximate Matching,» *Security and Communication Networks*, vol. 2017, pág. 1 306 802, 2017.

[34] A. Singh, S. Jayaram Subramanya, R. Krishnaswamy y H. V. Simhadri, *FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search*, 2021. arXiv: `2105.09613 [cs.IR]`. dirección: `https://arxiv.org/abs/2105.09613`

Índice de Figuras

Índice de Tablas

Anexos

Anexo A

Planificación y dedicación

La Tabla A.1 recoge el desglose de horas dedicadas por fase y tarea. Las tareas se agrupan en seis fases: gestión y seguimiento, que incluye reuniones con los directores y la planificación del trabajo, formación y estudio previo, con la toma de contacto con Rust y el estudio de los algoritmos de búsqueda ANN, así como el estado del arte, análisis y diseño, que incluye el análisis de la implementación existente de `APOTHEOSIS`, implementación, pruebas y evaluación, y documentación.

En total se han invertido 432 horas, de las cuales unas 350 corresponden al periodo de prácticas, entre abril y septiembre, lo que explica que se supere la referencia de 300 horas asociada a los 12 créditos ECTS del Trabajo Fin de Grado.

La Figura A.1 muestra la distribución temporal de las tareas. Cada columna se corresponde con una semana, identificada por el día del mes en el que comienza.

Fase	**Tarea**	**Horas**	**%**
Gestión y seguimiento	Reuniones con los directores	13	
	Planificación y organización	7	
	Subtotal	*20*	4,6
Formación y estudio previo	Aprendizaje del lenguaje Rust	20	
	Búsqueda ANN	36	
	NSG, MRNG y NN-Descent	34	
	Subtotal	*90*	20,8
Análisis y diseño	Análisis de `APOTHEOSIS`	22	
	Diseño de la integración de NSG	23	
	Subtotal	*45*	10,4
Implementación	Grafo NSG y algoritmo de búsqueda	32	
	NN-Descent	15	
	Bosque de *VP-trees*	12	
	Integración en `APOTHEOSIS`	14	
	Subtotal	*73*	16,9
Pruebas y evaluación	Entorno y conjunto de datos	14	
	Ejecución de los experimentos	90	
	Análisis de resultados y gráficas	45	
	Subtotal	*149*	34,5
Documentación	Redacción de la memoria	55	12,7
Total		**432**	**100,0**

Tabla A.1: Desglose de la dedicación por fase y tarea.

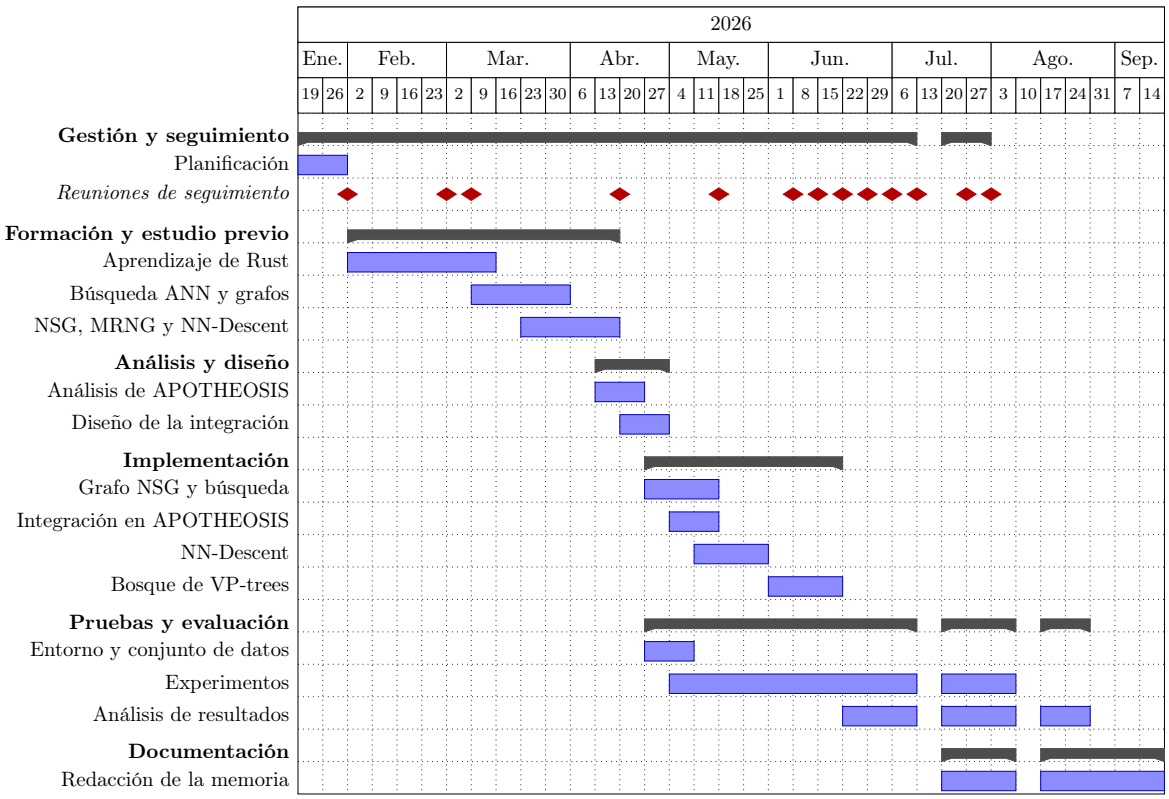


Figura A.1: Diagrama de Gantt del trabajo realizado.