Hierarchical Navigable Small World

Łukasz Jan Prus

24 Maj 2025

Streszczenie

Algorytm *Hierarchical Navigable Small World* (HNSW) to jedna z najbardziej wydajnych metod typu przybliżonego wyszukiwania najbliższych sąsiadów (*Approximate Nearest Neighbor* (ANN)). Jest ona wykorzystywana do szybkiego wyszukiwania podobieństw wektorowych (*vector similarity search*) w przestrzeniach wysokowymiarowych. W niniejszej pracy przedstawiono szczegółową charakterystykę struktury HNSW, zasadę jej działania oraz sposób implementacji. Celem pracy jest kompleksowe zaprezentowanie HNSW jako narzędzia o wysokim potencjale w analizie danych.

1 Wstęp teoretyczny

Wektoryzacja danych jest jedną z najczęściej stosowanych metod przetwarzania różnorodnych danych zapisanych w systemach komputerowych. Wraz z rozwojem technologii oraz zwiększaniem pojemności pamięci masowych komputerów, bazy wektorowe rosną zarówno pod względem liczebności (liczba wektorów liczona w miliardach), jak i wymiarowości (liczba wymiarów przestrzeni liczona w setkach).

W efekcie, tradycyjne algorytmy działające na tak liczebnych przestrzeniach wysokowymiarowych stają się niepraktyczne, ponieważ dokładne ich przeszukiwanie jest bardzo kosztowne obliczeniowo. Skutkuje to pojawieniem się nowego problemu algorytmicznego — wyszukiwania podobieństw wektorowych.

1.1 Wyszukiwanie podobieństw wektorowych *(vector similarity search)*

Przestrzeń wektorowa danych służy przede wszystkim do kategoryzacji danych. Podobne dane w poprawnie zbudowanej przestrzeni będą reprezentowane przez wektory, których poszczególne wymiary (cechy) różnią się nieznacznie. Oznacza to, że wektory o niewielkich odległościach[1] reprezentują dane o podobnych cechach i odwrotnie.
Znalezienie podobnych danych sprowadza się do wybrania, bądź stworzenia wektora z pożądanymi cechami i znalezieniu najbliższych (najbardziej podobnych) do niego wektorów. Metoda ta jest szczególnie skuteczna w bardzo zagęszczonych bazach danych, to znaczy istnieje wiele wektorów w relatywnie niewielkich odległościach od poszukiwanego wektora.

Formułując powyższy problem formalnie:

Definicja 1.1
Niech dana będzie przestrzeń wektorowa $\mathbb{R}^d$ oraz zbiór wektorów (baza danych) $\mathcal{D}$:

$$\mathcal{D} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\} \subset \mathbb{R}^d, \quad n, d \in \mathbb{N}$$

Dla zadanego zapytania $\vec{q} \in \mathbb{R}^d$ oraz określonej funkcji odległości $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geq 0}$, celem funkcji f realizującej wyszukiwanie podobieństw wektorowych jest znalezienie k wektorów z $\mathcal{D}$ najbardziej zbliżonych do $\vec{q}$, według tej funkcji odległości, gdzie:

$$k \in \mathbb{N} \wedge k \leq |\mathcal{D}|$$

$$f_k : \mathbb{R}^d \to \{\mathcal{S} \subseteq \mathcal{D} : |\mathcal{S}| = k\}$$

$$f_k(\vec{q}) = \underset{\substack{\mathcal{S} \subseteq \mathcal{D} \\ |\mathcal{S}|=k}}{\arg\min} \sum_{\vec{x} \in \mathcal{S}} d(\vec{q}, \vec{x})$$

W przestrzeniach niskowymiarowych bardzo dobrze sobie radzą algorytmy oparte na binarnych drzewach przeszukiwań, między innymi k-d tree (k-dimensional tree), bądź ball tree. Algorytmy te jednak nie radzą sobie z przestrzeniami wysokowymiarowymi. Skutkuje to nowym podejściem do problemu. Zamiast próbować znaleźć dokładnego najbliższego sąsiada, co jest obliczeniowo niepraktyczne, algorytmy starają się oszacować jego położenie. Takie rozwiązanie nie zawsze znajdzie najbliższe wektory, ale znalezienie innego „bliskiego" wektora jest wystarczająco dobre w większości przypadków.

[1] Metryka według której zdefiniowana jest funkcja odległości może być dowolna. Jej wybór jest zależny od sposobu wektoryzowania danych oraz pożądanych właściwości wyszukiwania.

1.2 Przybliżone wyszukiwanie najbliższych sąsiadów *(Approximate Nearest Neighbor)*

Zamiast skupiać się na znalezieniu dokładnie najbliższych sąsiadów do wektora, ANN decyduje się na zmniejszenie dokładności w zamian za znaczne przyspieszenie przeszukiwania. Najczęściej budują one struktury danych, dzięki którym szacują, gdzie powinien znajdować się najbliższy wektor, a następnie obliczają funkcję odległości tylko dla części wszystkich wektorów. Ponieważ algorytmy te oparte są na elementach probabilistycznych, ich wyniki są niedeterministyczne, co może być postrzegane jako zaleta — dla tego samego zapytania mogą zostać zwrócone różne wyniki. W praktyce oznacza to, że zamiast tworzyć nowe zapytanie, wystarczy je ponowić, otrzymując w ten sposób wachlarz różnych wektorów wynikowych.
Algorytmy ANN działają heurystycznie, zatem nie spełniają definicji 1.1, gdyż nie zawsze ich zbiór wynikowy zawiera wszystkie k najbliższych wektorów. Wprowadzono zatem nową miarę — tzw. recall. Recall to miara dokładności działania algorytmu dla danej przestrzeni, jest on opisany następującym wzorem

$$\text{recall} = \frac{|\mathcal{S} \cap \mathcal{A}|}{k}$$

Gdzie $\mathcal{S}$ to zbiór k najbliższych sąsiadów spełniający definicję 1.1, natomiast zbiór $\mathcal{A}$ to zbiór przybliżonych najbliższych sąsiadów zwrócony przez algorytm ANN. Istotne jest spostrzeżenie, że ze względu na niedeterministyczny charakter algorytmów ANN, wartość recall dla tego samego algorytmu i tej samej bazy danych może różnić się w kolejnych wywołaniach. Dlatego przy estymacji recall danego algorytmu zaleca się wykonanie wielu iteracji na zróżnicowanych próbkach przestrzeni wektorowej.

2 Hierarchical Navigable Small World

Hierarchical Navigable Small World (HNSW) to algorytm rozwiązujący problem wyszukiwania podobieństw wektorowych metodą przybliżonego wyszukiwania najbliższych sąsiadów (ANN). Wykorzystuje on wielopoziomową strukturę grafową opartą na właściwościach grafów małego świata (small-world graphs). Graf ten składa się z wielu warstw, gdzie każda kolejna wyższa warstwa zawiera coraz rzadsze połączenia między wierzchołkami, co pozwala na pokonanie problemu porzucania lokalnego optimum na rzecz znalezienia globalnego optimum.
Podczas wyszukiwania zapytanie rozpoczyna się na najwyższym poziomie, gdzie wykonywane jest heurystyczne przeszukiwanie wzdłuż krawędzi grafu. Następnie algorytm schodzi kolejno do niższych, gęstszych warstw, gdzie następuje dokładniejsze zawężenie obszaru poszukiwań. Takie podejście łączy efektywność przeszukiwania globalnego z precyzją lokalną, umożliwiając szybkie i dokładne znajdowanie przybliżonych najbliższych sąsiadów nawet na dużych i wysokowymiarowych zbiorach danych.

Algorytm jest zbudowany na wielu parametrach, które istotnie wpływają na jego działanie. Owe parametry są wymienione i opisane pokrótce dla wygody czytelnika:

d liczba wymiarów wektorów w przestrzeni danych

k liczba najbliższych sąsiadów zwracanych przez algorytm wyszukiwania

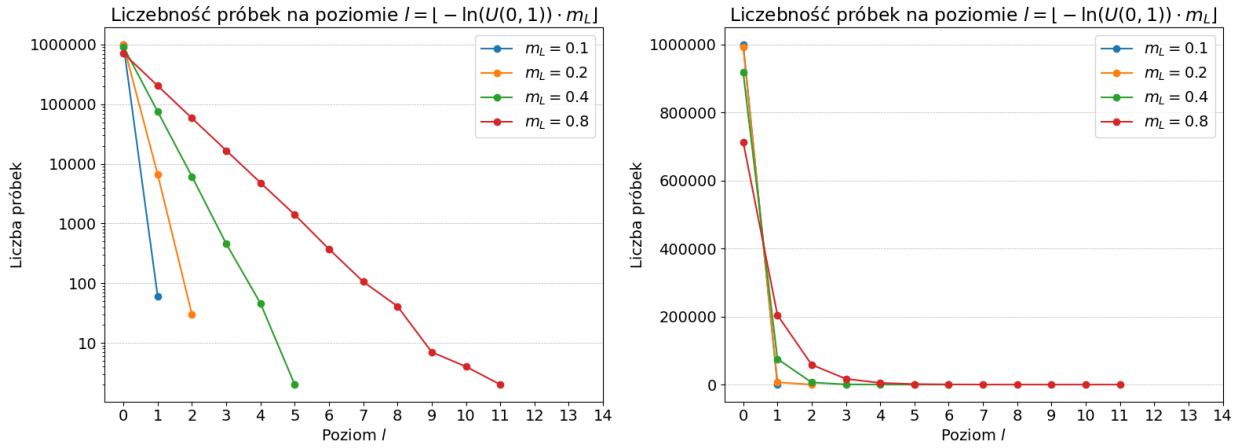
m_L parametr normalizujący używany do próbkowania poziomu hierarchii

$M_{\max}$ maksymalna liczba krawędzi wychodzących z wierzchołka na poziomach wyższych niż 0

$M_{0\,\max}$ maksymalna liczba krawędzi wychodzących z wierzchołka na poziomie 0 (bazowym)

`efConstruction` rozmiar listy kandydatów przeszukiwanych podczas dodawania nowego wierzchołka do grafu

`efSearch` rozmiar listy kandydatów przeszukiwanych podczas wyszukiwania k najbliższych wierzchołków do wektora zapytania



Rysunek 1: Rozkład liczebności punktów na danym poziomie w zależności od parametru normalizującego m_L, w skalach linowej i logarytmicznej. Liczba punktów - 1 000 000.

2.1 Losowanie warstwy

Pierwszym krokiem algorytmu Hierarchical Navigable Small World jest przypisanie każdemu punktowi poziomu (warstwy), do której będzie należał. Aby algorytm działał poprawnie liczba wektorów na wyższych warstwach musi być znacznie mniejsza niż liczba wektorów na warstwach niższych. W tym celu poziom l dla danego punktu losowany jest zgodnie z wykładniczym rozkładem prawdopodobieństwa:

$$l = \lfloor -\ln\left(\mathcal{U}\left(0,1\right)\right) \cdot m_L \rfloor \quad [4] \tag{1}$$

Gdzie m_L to parametr normalizujący, a $\mathcal{U}\left(0,1\right)$ to zmienna losowa o rozkładzie jednostajnym w przedziale $(0,1)$. Wartość parametru normalizującego ma wpływ na zachowanie struktury, zatem dobranie optymalnej wartości jest istotne. Typową wartością parametru jest $m_L = 1/\ln(M_{\max})$. [1]

Przedstawienie zależności rozkładu punktów na warstwach od parametru m_L obrazuje rysunek 1. Niewielkie zmiany parametru drastycznie zmieniają rozkład. Można wykazać, że dla większego m_L liczba poziomów w grafie będzie większa.

2.2 Budowa grafów lokalnych dla poziomów

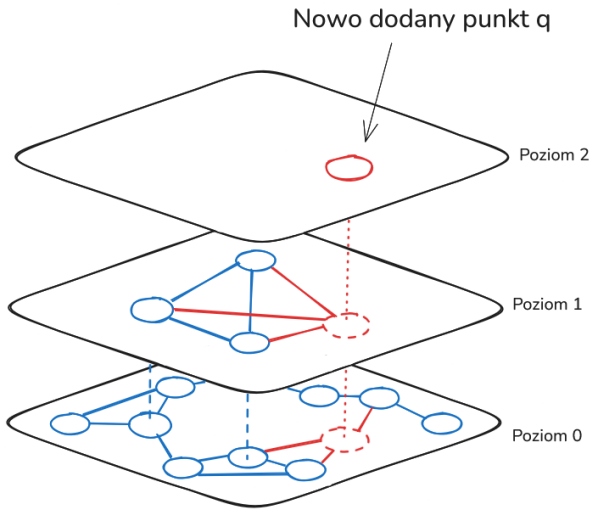
Ten etap algorytmu to najbardziej obliczeniowo czasochłonny element całej implementacji, zatem wszelkie starania dążące do optymalizacji HNSW powinny skupiać się na tym elemencie. Budowany graf to aproksymacja grafu Delaunay starający się minimalizować prawdopodobieństwo wystąpienia fałszywego globalnego minimum przy utrzymaniu jak najmniejszej liczby krawędzi między wierzchołkami [2]. Budowa tego grafu to sekwencyjna insercja elementów (Algorytm 1), gdzie dla każdego nowo stworzonego wierzchołka q znajdujemy `efConstruction` kandydatów na najbliższego sąsiada za pomocą wybranego algorytmu wyszukiwania, najczęściej greedy search (Algorytm 2). Następnie z listy kandydatów wybieranych jest M sąsiadów dla q, zazwyczaj jest to po prostu wybór M najbliższych sąsiadów względem q z podanego zbioru kandydatów, dlatego pomija się jego formalną definicję.
Warto zauważyć, że podczas przypisania wektorowi poziomu l, punkt ten występuje nie tylko w grafie warstwy l, ale także we wszystkich niższych warstwach $0 \ldots l-1$. Przedstawia to rysunek 2.

Algorytm 1 Dodanie nowego wierzchołka
INSERT($\texttt{HNSW}, q, M, M_{\max}, M_{0\max}$, $\texttt{efConstruction}, m_L$)
q – wektor, dla którego dodawany jest wierzchołek

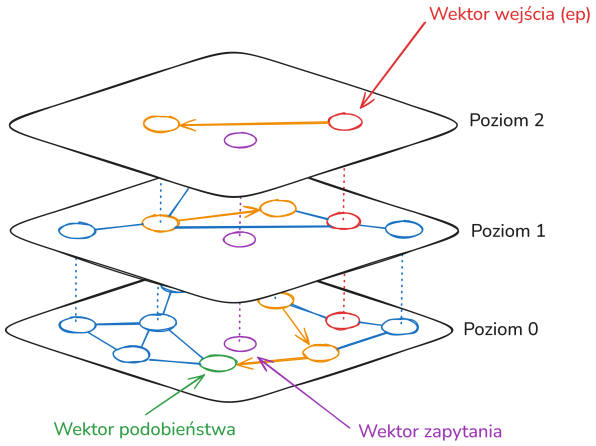
1: $ep \leftarrow$ punkt na najwyższej warstwie grafu HNSW
2: $L \leftarrow$ numer najwyższej warstwy w grafie HNSW
3: $C \leftarrow \varnothing$ {lista kandydatów}
4: $l \leftarrow \lfloor -\ln(\mathcal{U}(0,1)) \cdot m_L \rfloor$ {równanie (1)}
5: **for** $l_i = L, \ldots, l+1$ **do**
6: $C \leftarrow$ SEARCH-LAYER$(q, ep, ef = 1, l_i)$
7: $ep \leftarrow$ element najbliższy do q spośród C
8: **end for**
9: **for** $l_i = \min(L, l), \ldots, 0$ **do**
10: $C \leftarrow$ SEARCH-LAYER$(q, ep,$ $\texttt{efConstruction}, l_i)$
11: $N \leftarrow$ SELECT-NEIGHBORS(q, C, M, l_i)
12: **for** each n in N **do**
13: utwórz krawędź nieskierowaną między q a n w warstwie l_i
14: **if** $\deg(n) > M_{\max}$ {dla $l_i = 0$ użyj $M_{0\max}$} **then**
15: $nN \leftarrow$ zbiór sąsiadów wierzchołka n w warstwie l_i
16: $nN \leftarrow$ SELECT-NEIGHBORS(n, nN, M, l_i)
17: **end if**
18: **end for**
19: $ep \leftarrow C$ {ep zawiera teraz wszystkie punkty z C jako punkty wejścia}
20: **end for**
21: **if** $l > L$ **then**
22: ustaw punkt startowy ep w HNSW na q
23: **end if**

Algorytm 2 Wyszukanie kandydatów na sąsiadów metodą greedy search
SEARCH-LAYER$(\texttt{HNSW}, q, ep, ef, l_i)$
ef – liczba kandydatów do zwrócenia

1: $v \leftarrow \{ep\}$ {Zbiór odwiedzonych wierzchołków}
2: $C \leftarrow \{ep\}$ {Zbiór kandydatów do rozwinięcia}
3: $N \leftarrow \{ep\}$ {Zbiór najlepszych znalezionych sąsiadów}
4: **while** $|C| > 0$ **do**
5: $c \leftarrow$ element najbliższy do q spośród C
6: $C \leftarrow C \setminus \{c\}$ {Usuń analizowany element z listy potencjalnych kandydatów}
7: $f \leftarrow$ element najdalszy z N względem q
8: **if** $\text{dist}(c, q) > \text{dist}(f, q)$ **then**
9: **break** {Nie znajdziemy już lepszych kandydatów}
10: **end if**
11: **for** każdy n w sąsiedztwie(c) w warstwie l_i **do**
12: **if** $n \notin v$ **then**
13: $v \leftarrow v \cup \{n\}$
14: $f \leftarrow$ element najdalszy z N względem q
15: **if** $|N| < ef$ **lub** $\text{dist}(n, q) < \text{dist}(f, q)$ **then**
16: $C \leftarrow C \cup \{n\}$
17: $N \leftarrow N \cup \{n\}$
18: **if** $|N| > ef$ **then**
19: $f \leftarrow$ element najdalszy z N względem q
20: $N \leftarrow N \setminus \{f\}$ {Usuń najgorszy element}
21: **end if**
22: **end if**
23: **end if**
24: **end for**
25: **end while**
26: **return** N



Rysunek 2: Wizualizacja dodania punktu q dla którego poziom l jest równy 2, do grafu `HNSW`



Rysunek 3: Przebieg przeszukiwania zapytania w strukturze HNSW od poziomu 2 do 0 dla $k = 1$. Wektor wejścia (ep) reprezentuje punkt początkowy. Wynik (zielony) wskazuje na najbliższego sąsiada do wektora zapytania (fioletowy). Kolorem żółtym została oznaczona droga przeszukiwania oraz odwiedzone wierzchołki.

2.3 Wyszukiwanie najbliższych sąsiadów

Interesującym aspektem algorytmu HNSW jest to, że wyszukiwanie zapytania przypomina procedurę dodawania nowego wierzchołka — wektor zapytania porusza się od najwyższej warstwy w dół, wykonując heurystyczne przeszukiwanie w każdej z warstw. Na najniższym poziomie (warstwie 0) zbierani są kandydaci, z których wybieranych jest k najbliższych sąsiadów. W odróżnieniu od procedury dodawania, zapytanie nie modyfikuje struktury grafu. (Algorytm 3)

Algorytm 3 Wyszukanie kandydatów na sąsiadów metodą approximate kNN
SEARCH-AkNN(`HNSW`, $q, k,$ `efSearch`)

1: $ep \leftarrow$ punkt na najwyższej warstwie grafu `HNSW`
2: $L \leftarrow$ numer najwyższej warstwy w grafie `HNSW`
3: $C \leftarrow \varnothing$ {lista kandydatów}
4: **for** $l_i \leftarrow L \ldots 1$ **do**
5: $C \leftarrow$ SEARCH-LAYER(`HNSW`, $ep, ef = 1, l_i$)
6: $ep \leftarrow$ element najbliższy do q spośród C
7: **end for**
8: $C \leftarrow$ SEARCH-LAYER-GREEDY(`HNSW`, $ep, ef =$ `efSearch`$, l_i = 0$)
9: **return** k elementów najbliższych do q spośród C

2.4 Analiza złożoności

2.4.1 Analiza złożoności obliczeniowej wyszukiwania

Złożoność obliczeniowa może być wyliczona przy założeniu, że graf budowany to graf Delaunay a nie jego przybliżenie, jednak jest ona wyznaczona empirycznie, co jest rozbieżne z zachowaniem HNSW przy wysokowymiarowych przestrzeniach. Według tej analizy złożoność obliczeniowa wynosi $O(log_2(N))$ [1]. W obecnym momencie analiza ta jest mało użyteczna, gdyż algorytm ten jest specyficznie używany dla wektorów wysokowymiarowych. Natura algorytmu, a w szczególności jego niedeterministyczność powodują, że dokładne matematyczne analizy złożoności są skomplikowane i niemarodajne (istnieją specjalne zbiory danych dla których złożoność wynosi $O(N)$) [5]. Zatem ocenę szybkości algorytmu najlepiej przeprowadzać symulacyjnie, co w większości przypadków jest zgodne z oszacowaniem teoretycznym $O(log_2(N))$ bądź $O(N^\alpha)$ dla $\alpha \in [0.2, 0.4]$ [1].

2.4.2 Analiza złożoności obliczeniowej budowania grafu

Budowa grafu to iteratywne dodawanie wierzchołków, które sprowadza się do wykonania K-ANN search dla każdego poziomu [1]. Średnia liczba poziomów przypadających na pojedynczy wektor odpowiada wartości oczekiwanej ze wzoru 1, powiększonej o 1. (Równanie 1 zwraca indeks a nie liczbę poziomów).

$$
\begin{gathered}
l \text{ to wartość skwantyfikowana} \\
\text{(przez użycie operatora zaokrąglenia w dół) zatem} \\
\mathbb{E}[l] = \sum_{i=0}^{\infty} i \cdot P(l = i) \\
\text{Z podstawowych przekształceń stochastycznych} \\
-\ln\left(\mathcal{U}\left(0,1\right)\right) \cdot m_L \sim Exp(1/m_L) \\
l = \lfloor Exp(1/m_L) \rfloor \\
\text{Zatem } P(l = i) \text{ przyjmuje postać} \\
P(i \leq Exp(1/m_L) < i+1) = \int_i^{i+1} \lambda e^{-\lambda x} dx \\
P(l = i) = \int_i^{i+1} m_L e^{-m_L x} dx = e^{-\frac{i}{m_L}} - e^{-\frac{i+1}{m_L}} \\
\text{Wartość oczekiwana przyjmuje postać} \\
\mathbb{E}[l] = \sum_{i=0}^{\infty} i \cdot (e^{-\frac{i}{m_L}} - e^{-\frac{i+1}{m_L}}) \\
\text{Średnia liczba poziomów} \\
\text{przypadająca na wektor wynosi} \\
\mathbb{E}[l] + 1 = 1 + \sum_{i=0}^{\infty} i \cdot (e^{-\frac{i}{m_L}} - e^{-\frac{i+1}{m_L}})
\end{gathered}
\tag{2}
$$

Powyższe równanie można porównać do złożoności obliczeniowej wyszukiwania [1], zatem skaluje się ona jako $O(N \cdot \log_2(N))$.

2.4.3 Analiza złożoności pamięciowej

Złożoność pamięciowa algorytmu HNSW to tak naprawdę rozmiar pamięci potrzebny na zapisanie zbudowanego grafu wielowarstwowego. Liczba połączeń dla elementów na poziomie bazowym to $M_{0\max}$ natomiast dla pozostałych $M_{\max}$. Wiadomo, że m_L to parametr normalizujący rozkład wykładniczy o parametrze 1, zatem m_L określa średni poziom wierzchołka w grafie HNSW. Zatem średnie zużycie pamięci przez wierzchołek to:

$$(M_{0\max} + M_{\max} \cdot m_L) \cdot \mathrm{B}$$

gdzie B to liczba bajtów poświęcona na zapisanie połączenia. Uwzględniając, że współcześnie liczebność wektorowej bazy danych jest liczona w dziesiątkach miliardów (zakładając górne ograniczenie 50 miliardów). Wartości $M_{\max}$ są zazwyczaj w okolicach od 6 do 48 [1] zatem potrzeba dla zapisania pojedynczego połączenia (50 miliardów kombinacji), 5 bajtów razy liczbę połączeń (od 12 do 96) co daje szacunkowo od 60 do 480 bajtów. (Nie licząc pamięci potrzebnej na zapisanie danych wektorów). Powyższe oszacowanie jest zbieżne z przeprowadzonymi symulacjami [1].

3 Wnioski

Algorytm HNSW stanowi jedną z najbardziej efektywnych metod wyszukiwania przybliżonych sąsiadów w dużych, wysokowymiarowych zbiorach danych. Dzięki wykorzystaniu grafów małego świata i hierarchicznej struktury, osiąga on kompromis między szybkością a dokładnością.

Potwierdzona efektywność HNSW czyni go wartościowym narzędziem w zastosowaniach takich jak analiza obrazu, systemy rekomendacyjne czy przetwarzanie języka naturalnego. Przyszłe prace mogą skupić się na automatyzacji doboru parametrów i integracji z systemami rozproszonymi.

Literatura

[1] Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, 2016.

[2] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

[3] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Ali Mousavi, Ihab F. Ilyas, Umar Farooq Minhas, Jeffrey Pound, and Theodoros Rekatsinas. High-throughput vector similarity search in knowledge graphs, 2023.

[4] NMSLIB Development Team. NMSLIB: Non-Metric Space Library. `https://github.com/nmslib/nmslib`, 2023. Accessed: 2025-05-24.

[5] Indyk Piotr and Xu Yujia. Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations. *arXiv preprint arXiv:2310.19126*, 2023.