Approximate nearest neighbor search using the Hierarchical Navigable Small World (HNSW) algorithm

Sebastian Björkqvist

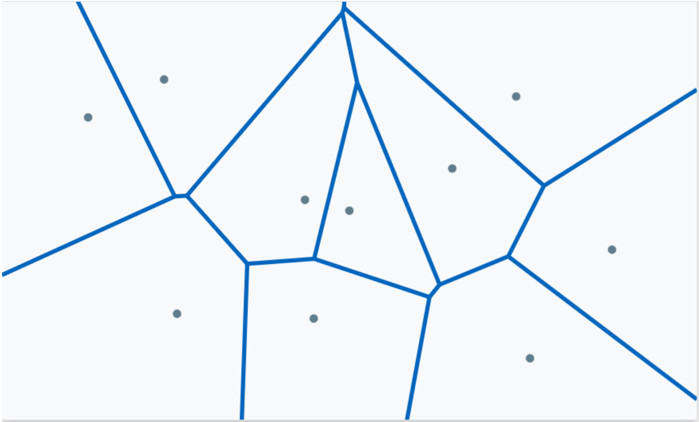
Lead AI Developer, IPRally

May 12, 2023

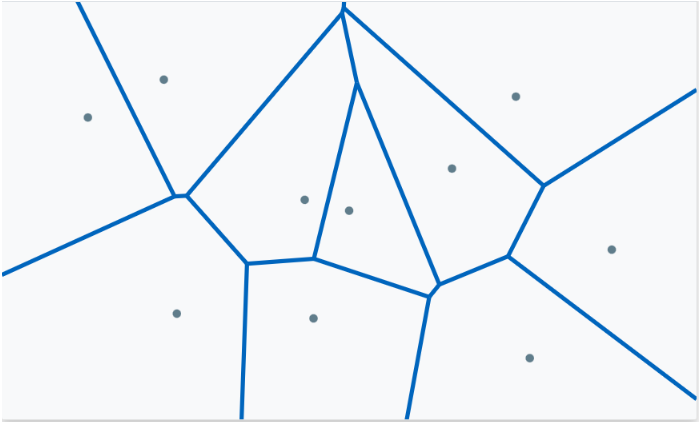
Outline

Voronoi diagram for a set of points

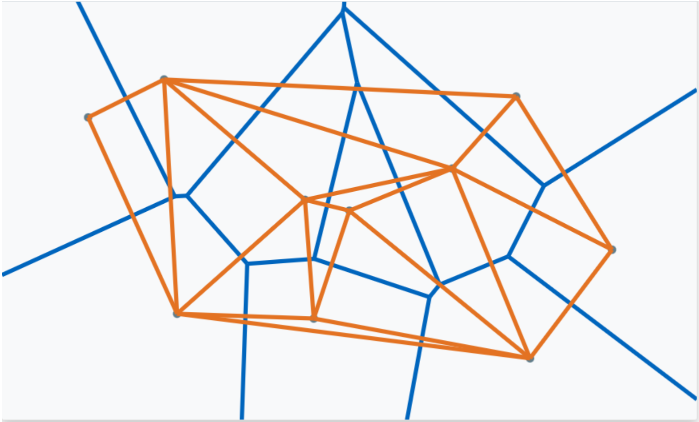
Voronoi diagram for a set of points



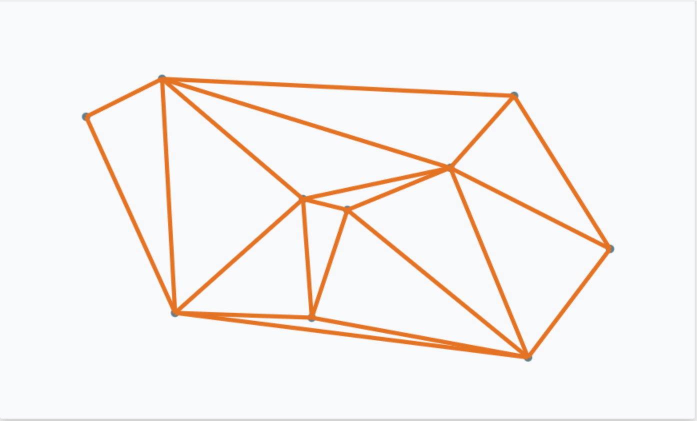
Voronoi diagram to Delaunay graph



Voronoi diagram to Delaunay graph



Delaunay graph



Greedy NN search algorithm

Greedy NN search algorithm

1. Select any graph node as entry node

Greedy NN search algorithm

1. Select any graph node as entry node
2. Calculate distance from query to current node and from query to all neighbors of current node

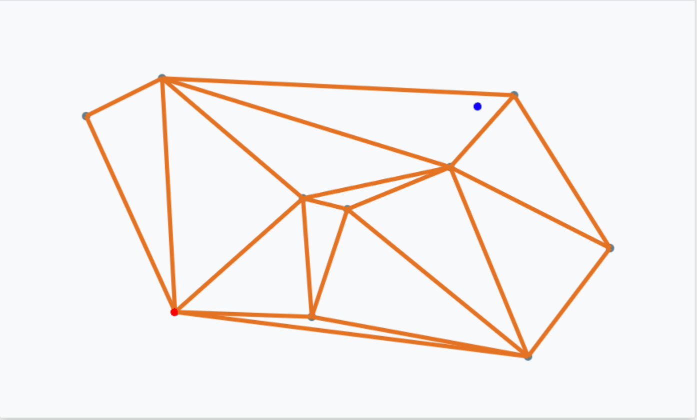
Greedy NN search algorithm

1. Select any graph node as entry node
2. Calculate distance from query to current node and from query to all neighbors of current node
3. Select neighbor with smallest distance to query as next node to visit

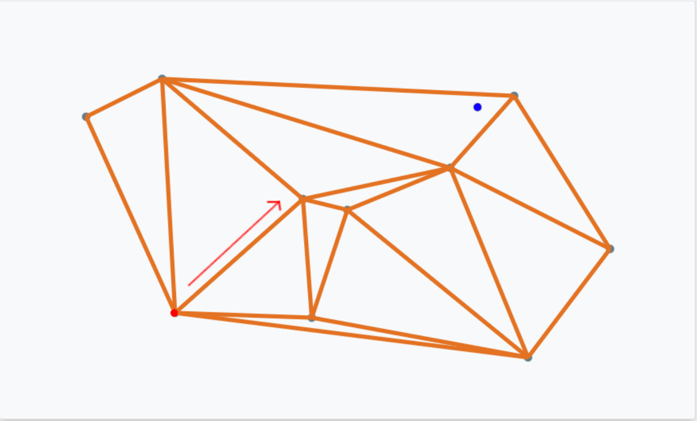
Greedy NN search algorithm

1. Select any graph node as entry node
2. Calculate distance from query to current node and from query to all neighbors of current node
3. Select neighbor with smallest distance to query as next node to visit
4. Repeat 2 and 3 until no neighbor is closer to query than the current node

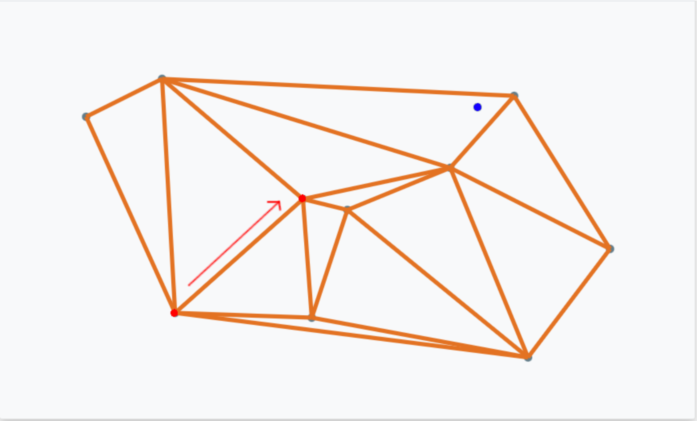
Greedy NN search start - Query and entry node



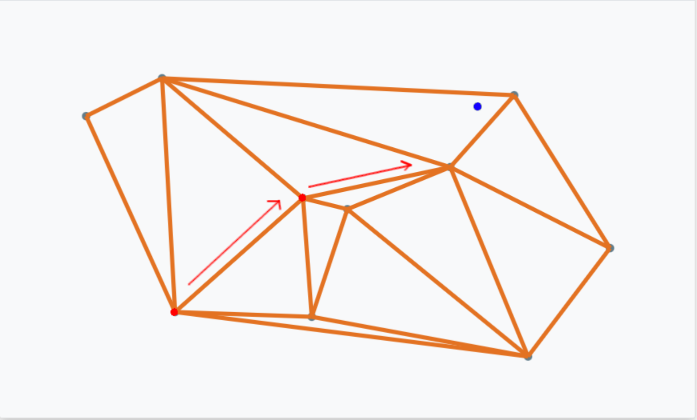
Greedy NN search - iteration



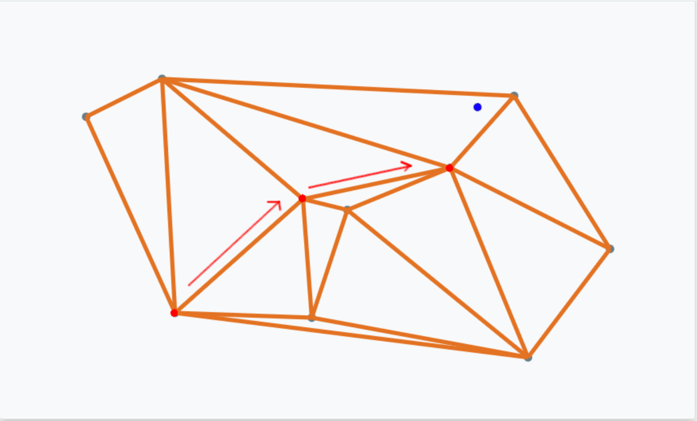
Greedy NN search - iteration



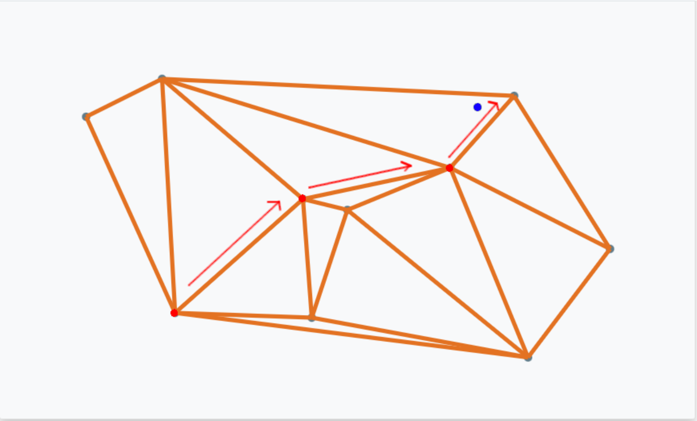
Greedy NN search - iteration



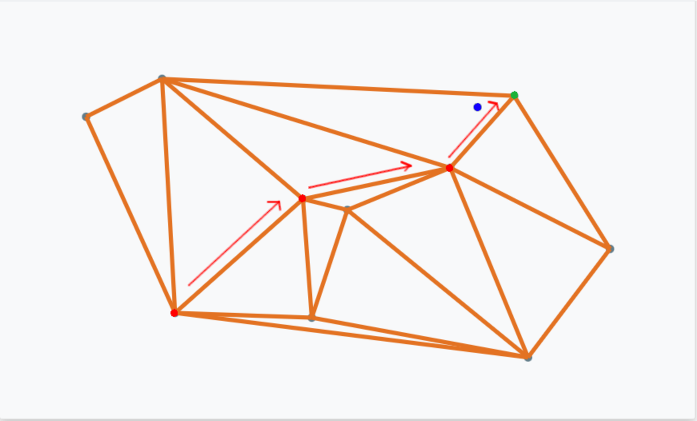
Greedy NN search - iteration



Greedy NN search - iteration



Greedy NN search done!



Drawbacks

- Delaunay graph intractable to construct for large, high-dimensional data sets
- Greedy search might require a lot of steps if graph is large

Navigable small world (NSW) graph

Navigable small world (NSW) graph

- Small world graph

Navigable small world (NSW) graph

- Small world graph
  - Distance of two random nodes is $log\ N$, where N is the number of nodes in graph

Navigable small world (NSW) graph

- Small world graph
  - Distance of two random nodes is *log* N, where N is the number of nodes in graph
  - Neighbors of a given node are likely to be neighbors of another (clustering coefficient is high)

Navigable small world (NSW) graph

- Small world graph
  - Distance of two random nodes is $log\ N$, where N is the number of nodes in graph
  - Neighbors of a given node are likely to be neighbors of another (clustering coefficient is high)
- Navigability

Navigable small world (NSW) graph

- Small world graph
  - Distance of two random nodes is *log* N, where N is the number of nodes in graph
  - Neighbors of a given node are likely to be neighbors of another (clustering coefficient is high)
- Navigability
  - Greedy search algorithm has logarithmic scalability

Why is an NSW useful for nearest neighbor search?

Why is an NSW useful for nearest neighbor search?

- Logarithmic distance allows us to get anywhere in the graph quickly

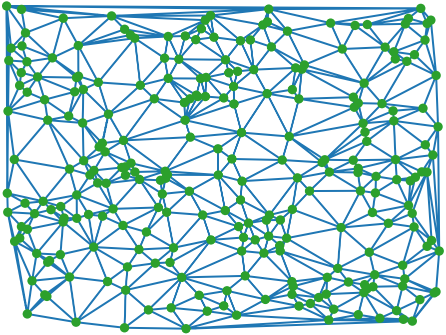
Why is an NSW useful for nearest neighbor search?

- Logarithmic distance allows us to get anywhere in the graph quickly
- Navigability ensures that the greedy algorithm finds the logaritmic path

Why is an NSW useful for nearest neighbor search?

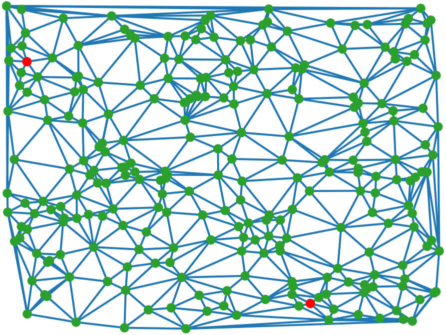
- Logarithmic distance allows us to get anywhere in the graph quickly
- Navigability ensures that the greedy algorithm finds the logaritmic path
- High clustering coefficient lets us zoom in on the actual correct node when we're in the right area

Making Delaunay graph navigable



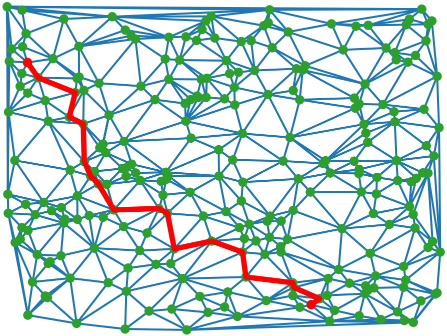
256 nodes

Making Delaunay graph navigable



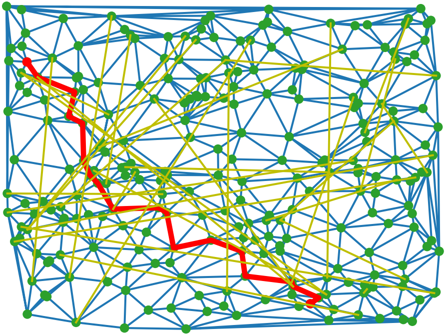
Start and end nodes in red

Making Delaunay graph navigable



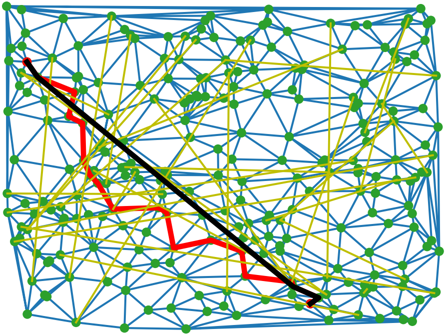
Length of path: 19

Making Delaunay graph navigable



32 random edges added

Making Delaunay graph navigable



Length of path: 5

Properties of NSW graph

Properties of NSW graph

- An NSW graph is not necessarily a Delaunay graph (or have one as a subgraph)

Properties of NSW graph

- An NSW graph is not necessarily a Delaunay graph (or have one as a subgraph)
- Thus the greedy algorithm doesn't always return the actual nearest neighbor

Properties of NSW graph

- An NSW graph is not necessarily a Delaunay graph (or have one as a subgraph)
- Thus the greedy algorithm doesn't always return the actual nearest neighbor
- Ok since we're doing approximate nearest neighbor search!

Constructing NSW graph

Constructing NSW graph

- Goal: Construct a graph that has the Delaunay graph as a subgraph, but also has longer connections to make it navigable

Constructing NSW graph

- Goal: Construct a graph that has the Delaunay graph as a subgraph, but also has longer connections to make it navigable
- Approximation of Delaunay graph is sufficient

Constructing NSW graph

Constructing NSW graph

1. Randomize order of data points

Constructing NSW graph

1. Randomize order of data points
2. Add data point to graph

Constructing NSW graph

1. Randomize order of data points
2. Add data point to graph
3. Add edges from data point to its k nearest neighbors that are already present in the graph

Constructing NSW graph

1. Randomize order of data points
2. Add data point to graph
3. Add edges from data point to its k nearest neighbors that are already present in the graph
4. Repeat 2 and 3 until all data points have been added

Why does NSW graph creation algorithm work?

Why does NSW graph creation algorithm work?

- Adding enough nearest neighbor edges approximates Delaunay graph

Why does NSW graph creation algorithm work?

- Adding enough nearest neighbor edges approximates Delaunay graph
- The edges added for the early nodes give long-range connections, enabling navigability

kNN search using NSW graph

kNN search using NSW graph

- Instead of only finding the nearest neighbor, we keep track of k nearest neighbors

kNN search using NSW graph

- Instead of only finding the nearest neighbor, we keep track of k nearest neighbors
- To improve results we can redo the search m times from different start nodes

kNN search algorithm

kNN search algorithm

1. Select any graph node as initial candidate, initialize candidates priority queue with initial candidate, initialize empty result priority queue

kNN search algorithm

1. Select any graph node as initial candidate, initialize candidates priority queue with initial candidate, initialize empty result priority queue
2. Select from the candidates queue the element closest to q

kNN search algorithm

1. Select any graph node as initial candidate, initialize candidates priority queue with initial candidate, initialize empty result priority queue
2. Select from the candidates queue the element closest to q
3. Calculate distance from query to all neighbors of candidate

kNN search algorithm

1. Select any graph node as initial candidate, initialize candidates priority queue with initial candidate, initialize empty result priority queue
2. Select from the candidates queue the element closest to q
3. Calculate distance from query to all neighbors of candidate
4. Add to result and candidate queue all neighbors who are closer to query than the kth result in the queue

kNN search algorithm

1. Select any graph node as initial candidate, initialize candidates priority queue with initial candidate, initialize empty result priority queue
2. Select from the candidates queue the element closest to q
3. Calculate distance from query to all neighbors of candidate
4. Add to result and candidate queue all neighbors who are closer to query than the kth result in the queue
5. Repeat until step 2 returns a candidate that's further away than the kth result in the queue

NSW drawbacks

NSW drawbacks

- Greedy search may get stuck in local minimum

NSW drawbacks

- Greedy search may get stuck in local minimum
- Algorithm scales polylogarithmically in general (logarithmic scaling in both steps and degrees of nodes)

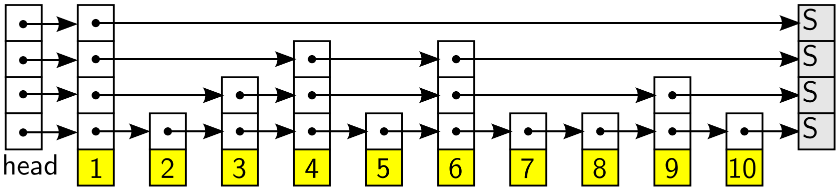
NSW drawbacks

- Greedy search may get stuck in local minimum
- Algorithm scales polylogarithmically in general (logarithmic scaling in both steps and degrees of nodes)
- Performance degrades on high-dimensional data

NSW drawbacks

- Greedy search may get stuck in local minimum
- Algorithm scales polylogarithmically in general (logarithmic scaling in both steps and degrees of nodes)
- Performance degrades on high-dimensional data
- Insertion order must be random

Inspiration: Skiplist



https://en.wikipedia.org/wiki/Skip_list

Idea: Combine NSW and skipping

Idea: Combine NSW and skipping

- NSW enables finding the approximate nearest neighbors

Idea: Combine NSW and skipping

- NSW enables finding the approximate nearest neighbors
- Skipping allows zooming in to the correct area quickly and reliably

Idea: Combine NSW and skipping

- NSW enables finding the approximate nearest neighbors
- Skipping allows zooming in to the correct area quickly and reliably
- The zoom-in property is accomplished by a hierarchical construction, like in skiplists

HNSW diagram

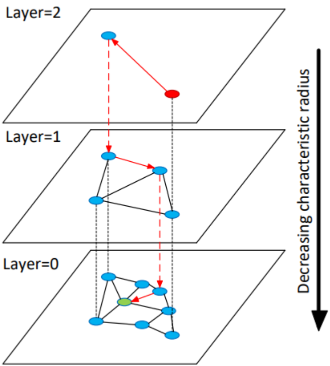


Fig. 1. Illustration of the Hierarchical NSW idea. The search starts from an element from the top layer (shown red). Red arrows show direction of the greedy algorithm from the entry point to the query (shown green).

Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs (Malkov et al.) `https://arxiv.org/abs/1603.09320`

kNN search using HNSW

kNN search using HNSW

1. Find nearest neighbor to query in top layer using greedy search algorithm, starting from any node

kNN search using HNSW

1. Find nearest neighbor to query in top layer using greedy search algorithm, starting from any node
2. Continue downwards to next layer, run greedy search starting from nearest neighbor found in previous layer

kNN search using HNSW

1. Find nearest neighbor to query in top layer using greedy search algorithm, starting from any node
2. Continue downwards to next layer, run greedy search starting from nearest neighbor found in previous layer
3. Repeat previous step until bottom layer is reached

kNN search using HNSW

1. Find nearest neighbor to query in top layer using greedy search algorithm, starting from any node
2. Continue downwards to next layer, run greedy search starting from nearest neighbor found in previous layer
3. Repeat previous step until bottom layer is reached
4. Run kNN algorithm on bottom layer (like when using NSW)

Construction of HNSW index

Construction of HNSW index

1. Randomly select (using exponential decay) the maximal layer l the new data point should be inserted in

Construction of HNSW index

1. Randomly select (using exponential decay) the maximal layer l the new data point should be inserted in
2. Find the nearest neighbor in the layer $l + 1$ using greedy search

Construction of HNSW index

1. Randomly select (using exponential decay) the maximal layer l the new data point should be inserted in
2. Find the nearest neighbor in the layer $l+1$ using greedy search
3. For each layer $l, \ldots, 0$, connect node to M nearest neighbors (found using kNN algorithm)

Construction of HNSW index

1. Randomly select (using exponential decay) the maximal layer l the new data point should be inserted in
2. Find the nearest neighbor in the layer $l+1$ using greedy search
3. For each layer $l, \ldots, 0$, connect node to M nearest neighbors (found using kNN algorithm)
    - After adding nearest neighbors to node, prune connections from neighbors if number exceeds M

Hyperparameters for HNSW

Hyperparameters for HNSW

1. M - amount of neighbors to connect to in each layer when constructing index (called *max-links-per-node* in Vespa)

Hyperparameters for HNSW

1. *M* - amount of neighbors to connect to in each layer when constructing index (called *max-links-per-node* in Vespa)
2. *efConstruction* - amount of neighbors to explore in the kNN search when constructing index (*neighbors-to-explore-at-insert* in Vespa)

Hyperparameters for HNSW

1. *M* - amount of neighbors to connect to in each layer when constructing index (called *max-links-per-node* in Vespa)
2. *efConstruction* - amount of neighbors to explore in the kNN search when constructing index (*neighbors-to-explore-at-insert* in Vespa)
3. *ef* - amount of neighbors to explore in the kNN search during inference (*exploreAdditionalHits* in Vespa)

Improvements compared to NSW

Improvements compared to NSW

- Less risk to get stuck in local minima due to long range edges being used first

Improvements compared to NSW

- Less risk to get stuck in local minima due to long range edges being used first
- Logarithmic scalability of search due to hierarchical structure

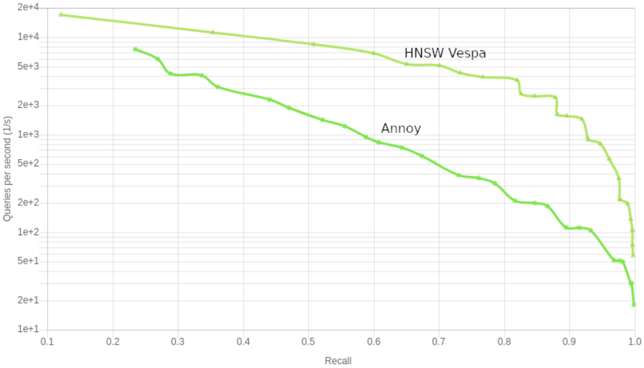
Improvements compared to NSW

- Less risk to get stuck in local minima due to long range edges being used first
- Logarithmic scalability of search due to hierarchical structure
- Better performance on high-dimensional data

Improvements compared to NSW

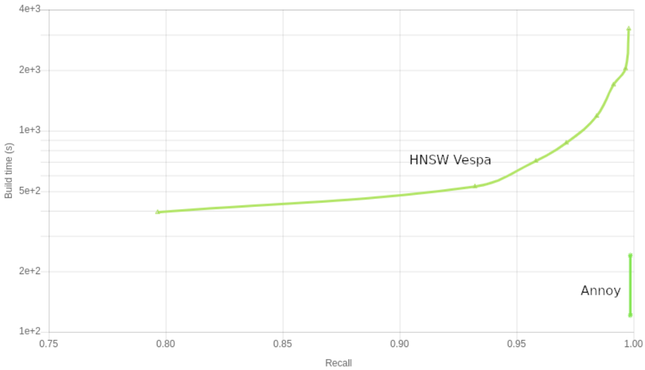
- Less risk to get stuck in local minima due to long range edges being used first
- Logarithmic scalability of search due to hierarchical structure
- Better performance on high-dimensional data
- Insertion order can be anything - randomization happens automatically during index construction

Recall vs queries per second (up and to the right is better)



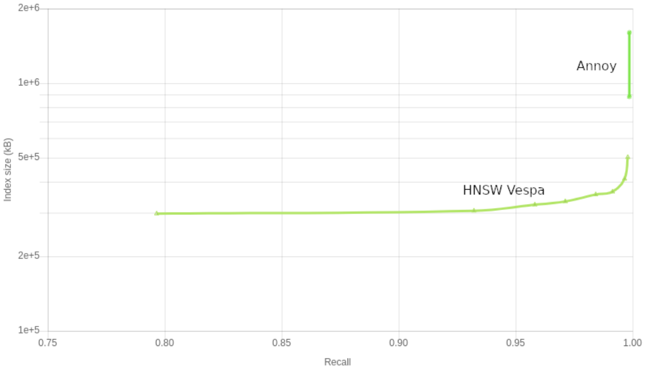
`https://ann-benchmarks.com/nytimes-256-angular_10_angular.html`

Recall vs build time (down and to the right is better)



`https://ann-benchmarks.com/nytimes-256-angular_10_angular.html`

Recall vs index size (down and to the right is better)



https://ann-benchmarks.com/nytimes-256-angular_10_angular.html

Summary

Summary

- HNSW algorithm combines navigable small world graphs with idea from skiplists to improve performance

Summary

- HNSW algorithm combines navigable small world graphs with idea from skiplists to improve performance
- Performs well on high-dimensional data

Summary

- HNSW algorithm combines navigable small world graphs with idea from skiplists to improve performance
- Performs well on high-dimensional data
- Supports adding new vectors without rebuilding graph from scratch

References

- *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs (Malkov et al.)* `https://arxiv.org/abs/1603.09320`
- *Approximate nearest neighbor algorithm based on navigable small world graphs (Malkov et al.)* `https://doi.org/10.1016/j.is.2013.10.006`
- *Voronoi diagrams—a survey of a fundamental geometric data structure (Aurenhammer)* `https://dl.acm.org/doi/10.1145/116873.116880`
- *Hierarchical Navigable Small Worlds (HNSW) (Pinecone blog)* `https://www.pinecone.io/learn/hnsw/`