Hierarchical Navigable Small World (HNSW)

Estimated Reading Time: 30 minutes

What is HNSW?

Imagine you're trying to find a specific restaurant in a huge city. Instead of checking every single restaurant one by one, you use a smart navigation system such as Google Maps. You start with a zoomed-out view showing major highways, then gradually zoom in to see smaller streets, and finally arrive at your exact destination. This is essentially how the Hierarchical Navigable Small World (HNSW) algorithm works - but instead of finding restaurants, it finds similar data points in massive databases.

HNSW is a sophisticated graph-based search algorithm designed to quickly find items that are similar to what you're looking for. It's particularly powerful when dealing with what computer scientists call "high-dimensional data" - think of complex information like the meaning of sentences, characteristics of images, or patterns in music. The algorithm was originally described in a groundbreaking 2016 research paper by Yu. A. Malkov and D. A. Yashunin, titled Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs.

The Building Blocks: Understanding the Components

1. Small World Networks

Think about how you're connected to other people. You know your friends directly, and through them, you can reach their friends, and so on. Surprisingly, research shows that any two people in the world are connected by just six degrees of separation on average. This creates what scientists call a "small world" - a network where you can reach anyone relatively quickly.

Small-world networks have two key characteristics:

- **High Clustering Coefficient**: Nodes tend to form tight-knit groups with many connections between neighbors
- **Low Average Path Length**: Despite the clustering, you can reach any node from any other node in just a few steps

HNSW applies this same principle to data. Instead of people knowing people, data points are "connected" to similar data points, creating a network where you can navigate quickly from any point to any other point. This concept was first formalized by Duncan Watts and Steven Strogatz in their influential 1998 paper, and forms the mathematical foundation for many modern search algorithms.

2. Navigable Networks

In a navigable network, you don't just have random connections - you have smart connections that help you move in the right direction. It's like having road signs that point you toward your destination. In HNSW, each data point is connected to others in a way that helps guide searches toward the target.

The Navigable Small World (NSW) algorithm, which forms the foundation of HNSW, works through a process called **greedy routing**:

1. Start at an entry point in the graph
2. Look at all directly connected neighbors
3. Move to the neighbor that is closest to your target
4. Repeat until you can't get any closer

This greedy search has a polylogarithmic time complexity of O(log^k n), which is significantly faster than a naive linear search through all data points.

3. Hierarchical Structure

This is where HNSW gets really clever. Instead of having just one network, it creates multiple layers - like a skyscraper with different floors. The hierarchical concept is inspired by **skip lists**, a probabilistic data structure that maintains multiple levels of linked lists.

Here's how the layers work:

Top Floor (Layer): Has very few data points, but they're connected by "long-distance" links that can jump far across the data space. Think of these as express highways. Only about 1/2^L nodes appear at layer L, where the probability decreases exponentially.

Middle Floors: Have more data points with medium-distance connections. These are like main roads in a city. Each layer down includes more nodes with progressively shorter-range connections.

Ground Floor (Bottom Layer): Contains all the data points with short-distance connections. These are like the local streets where you find your exact destination. Every single data point in the dataset exists at layer 0.

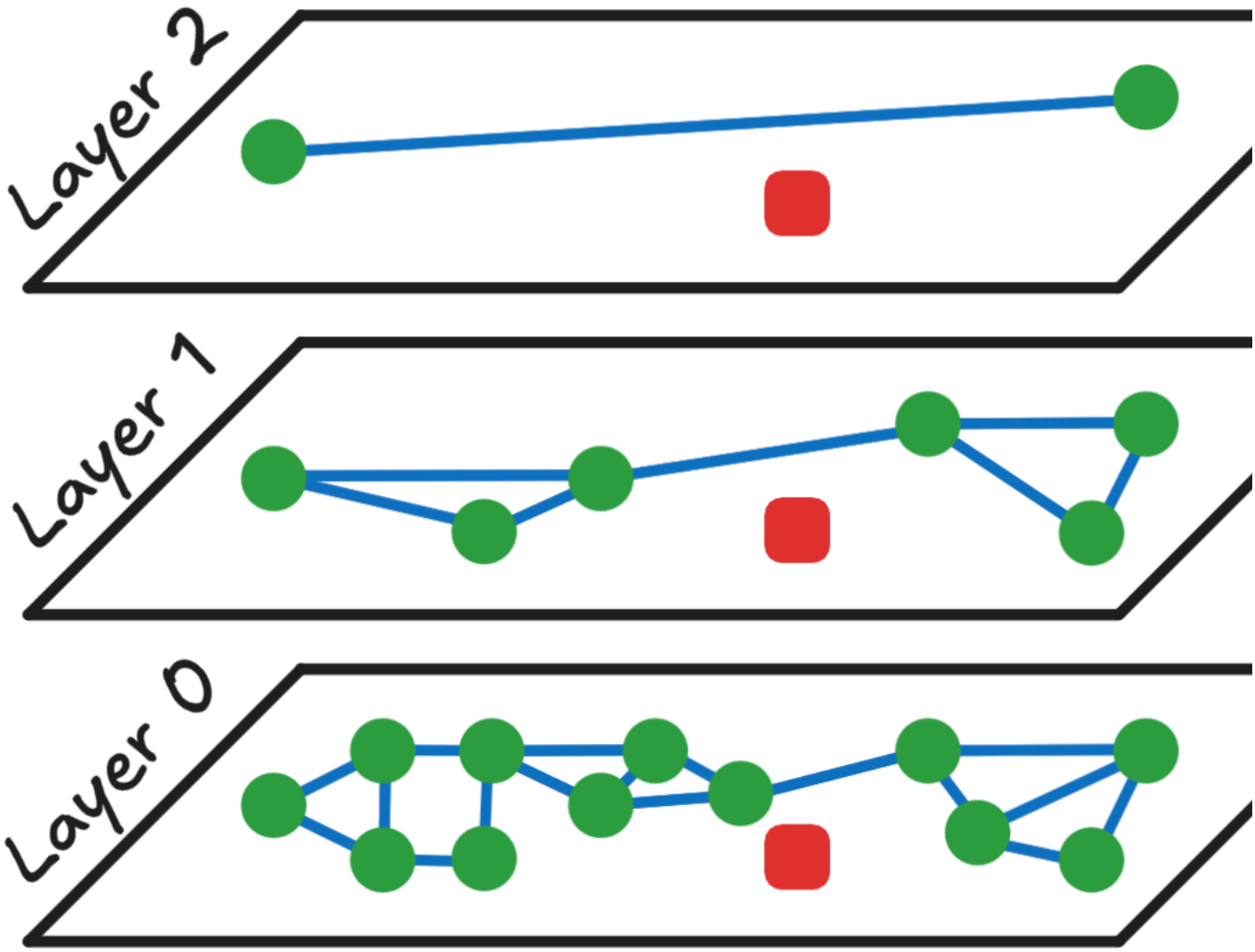
The layer assignment for each node is determined probabilistically using an exponentially decaying distribution, similar to how skip lists work.

How HNSW Works: A Step-by-Step Journey

Let's walk through how a search is performed using the HNSW index before diving into how the index is constructed:

Step 1: Start with an HNSW Index

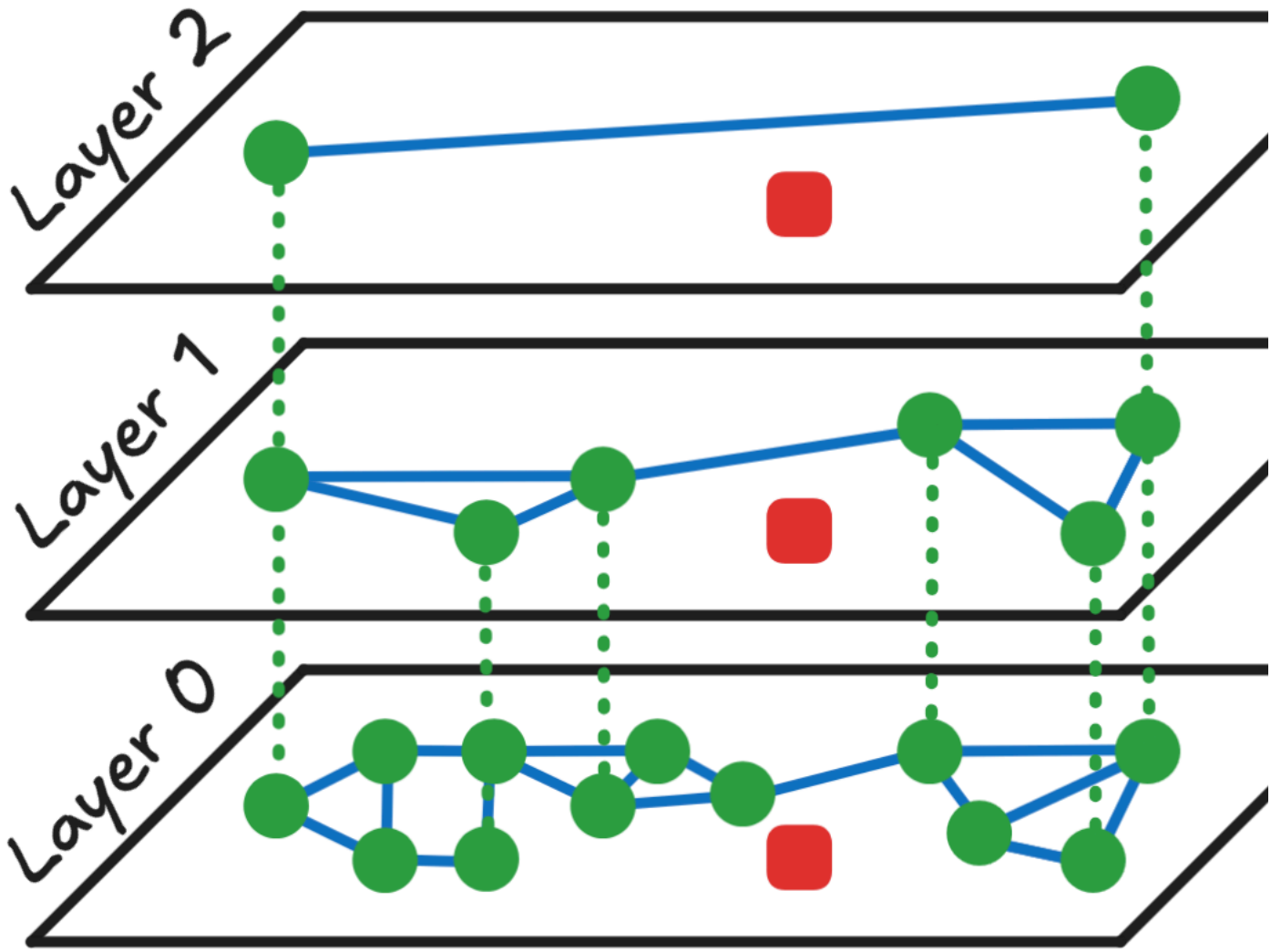
The diagram below illustrates a hierarchical navigable small world graph.



The bottom layer (`Layer 0`) contains all data points, shown as green circles, some of which are interconnected. The goal of the HNSW algorithm is to find the approximate nearest neighbor to a given query point, represented by the red rectangle. In a vector database context, the green circles represent data embeddings, and the red square represents the query embedding.

In this example, there are 12 data points. A brute-force search would require computing the distance (for example, cosine or L2) between the query and each of the 12 points. Instead, HNSW enables an approximate solution with fewer computations.

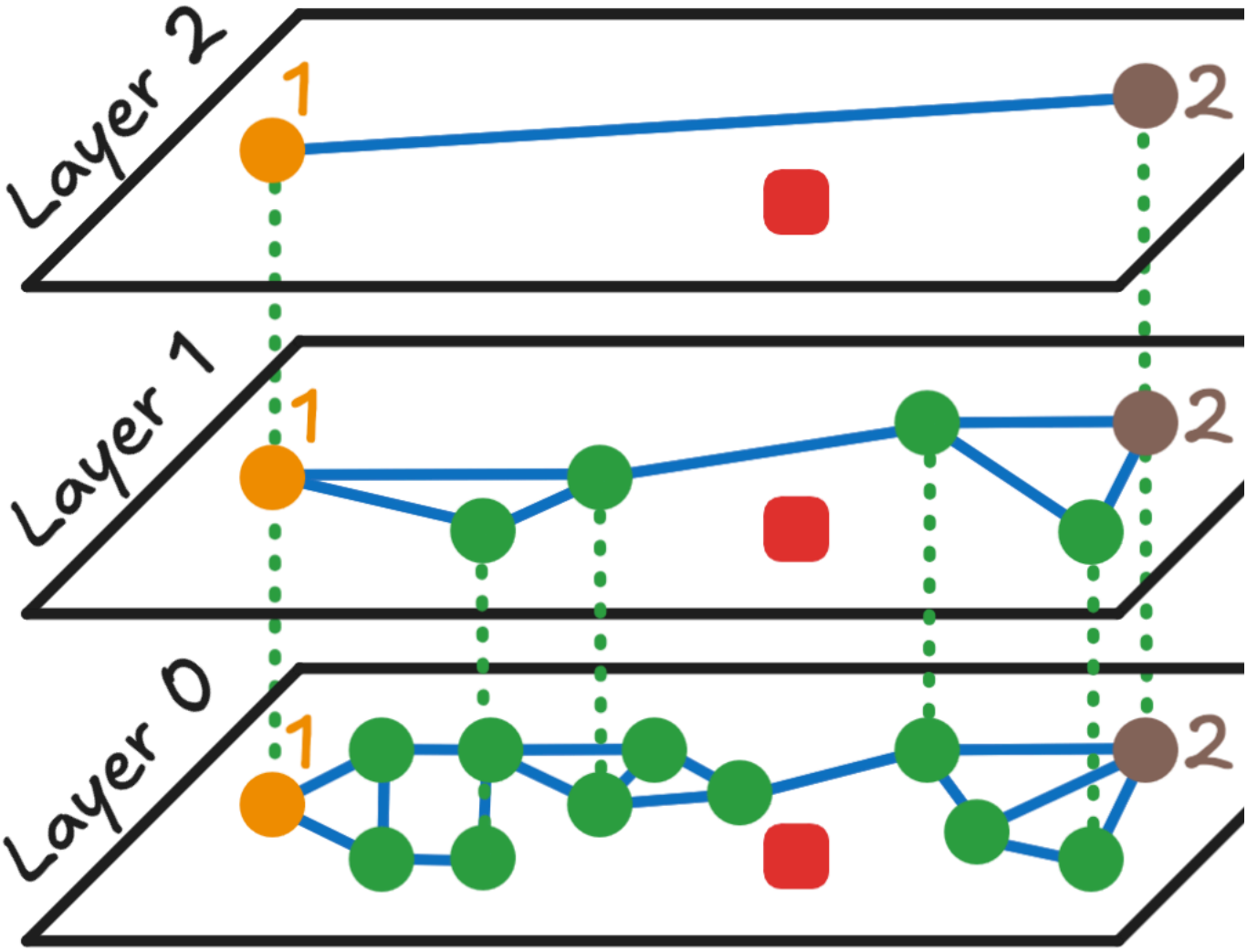
Some data points, including the query, are projected onto higher layers, with each successive layer containing fewer points. The following image shows how identical data points are connected across layers using dotted lines:



Note that connections in higher layers do not necessarily mirror those in lower layers.

Step 2: Enter the HNSW Index at a Random Point in the Top Layer

In the diagram below, the search begins at a randomly selected point in the top layer (`Layer 2`), marked as the orange point labeled `1`.



Step 3: Perform Greedy Search in the Current Layer

This step involves the following substeps:

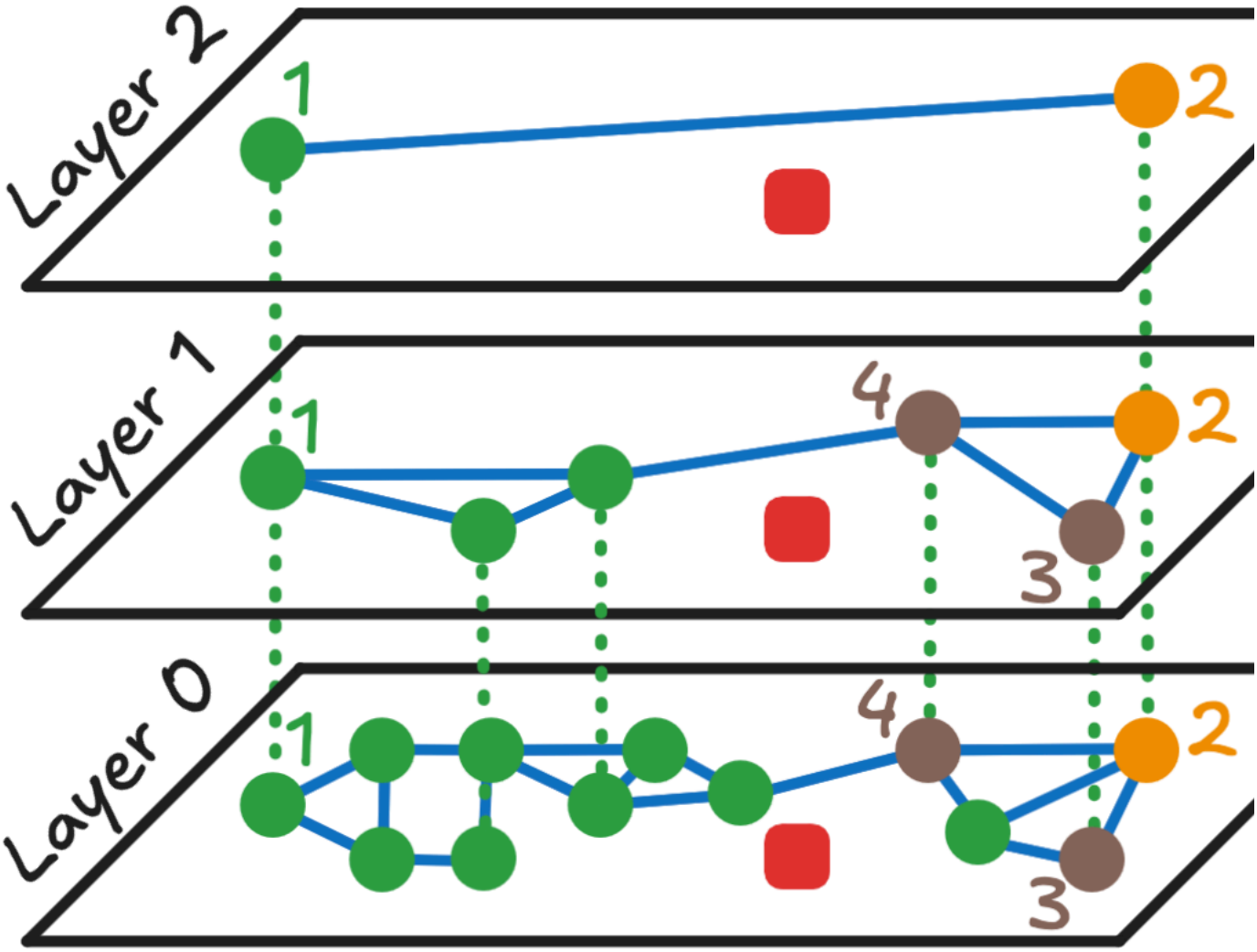
- **3.1**: Compute the distance between the entry point and the query.
- **3.2**: Compute distances between the query and all neighbors of the entry point in the current layer.
- **3.3**: Identify the point with the smallest distance to the query.
- **3.4**: If the closest point is the entry point, proceed to Step 4.
- **3.5**: If the closest point has no unexplored neighbors, proceed to Step 4.
- **3.6**: Otherwise, set the closest point as the new entry point and repeat from Step 3.1.

In this example:

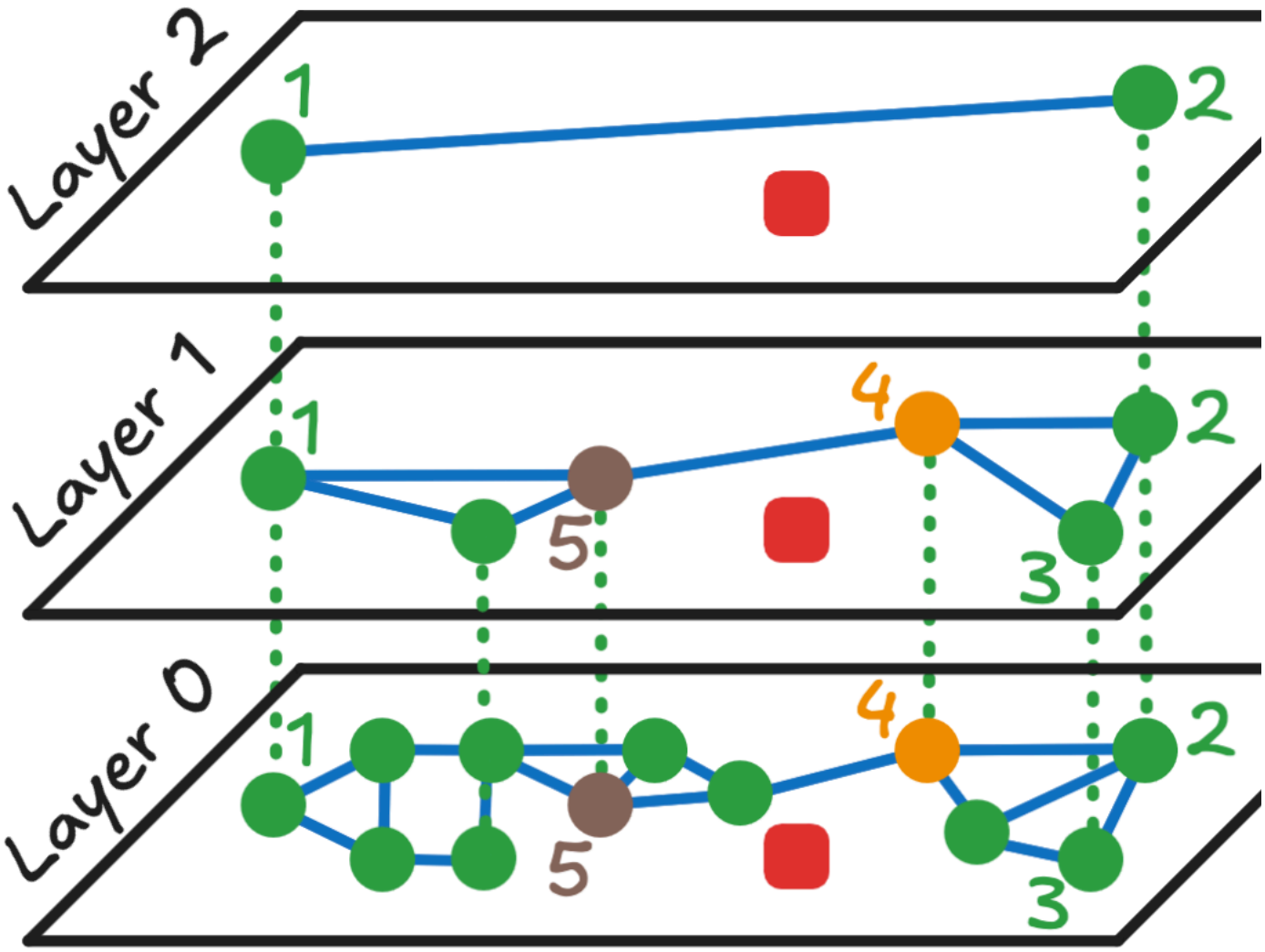
- Start at point `1` in `Layer 2`.
- Compute distances to the query from point `1` and its only neighbor, point `2`.
- Point `2` is closer to the query, so it becomes the new entry point.
- Since point `2` has no unexplored neighbors, proceed to Step 4.

Step 4: Move Down One Layer and Repeat

Now move to `Layer 1`, using point `2` as the entry point.

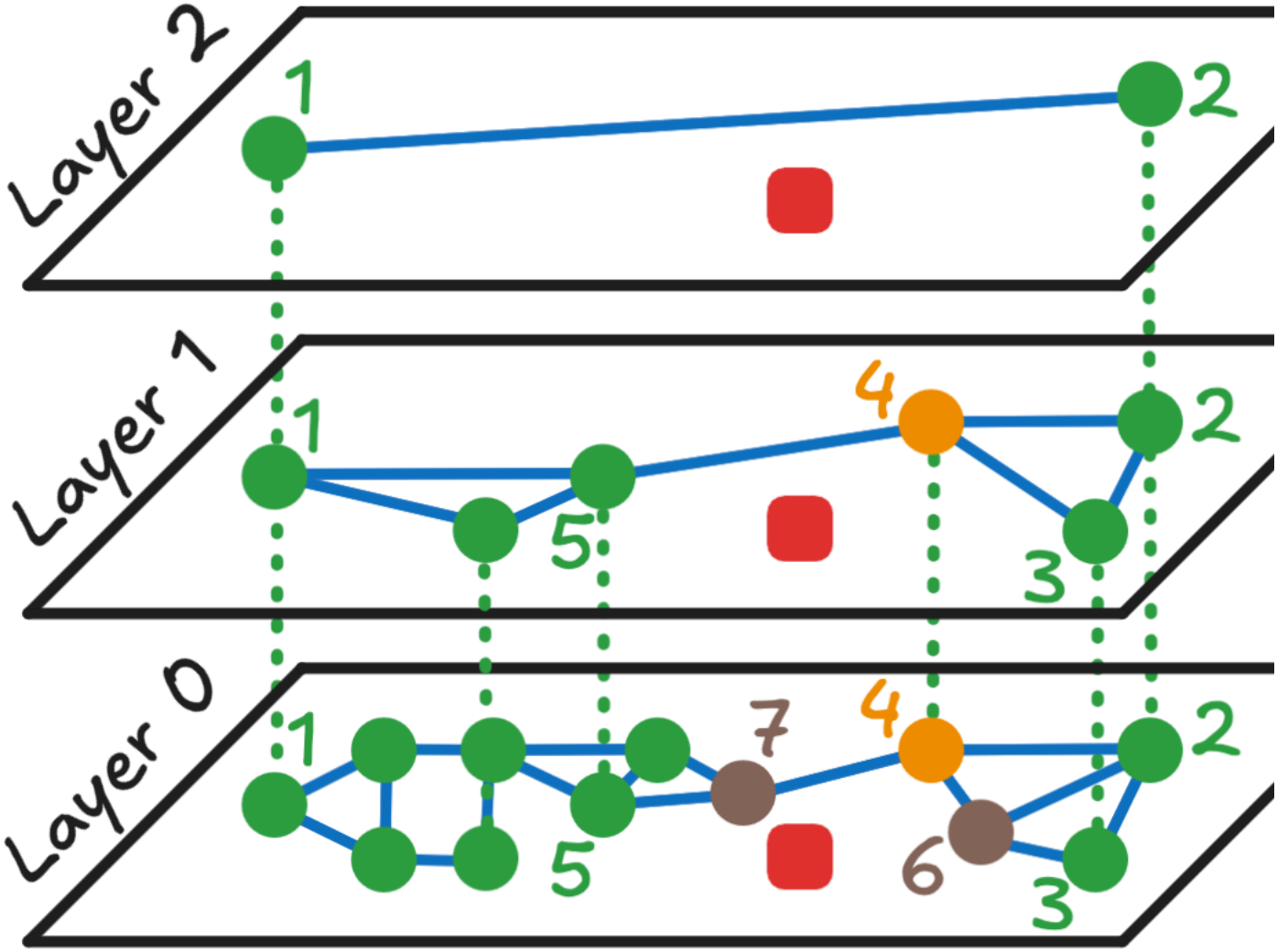


- Reuse the previously computed distance between point 2 and the query.
- Compute distances to point 2's neighbors in `Layer 1` (points 3 and 4).
- Point 4 is closest. However, point 4 was not an entry point and it has unexplored neighbors, so it becomes the new entry point into `Layer 1`.

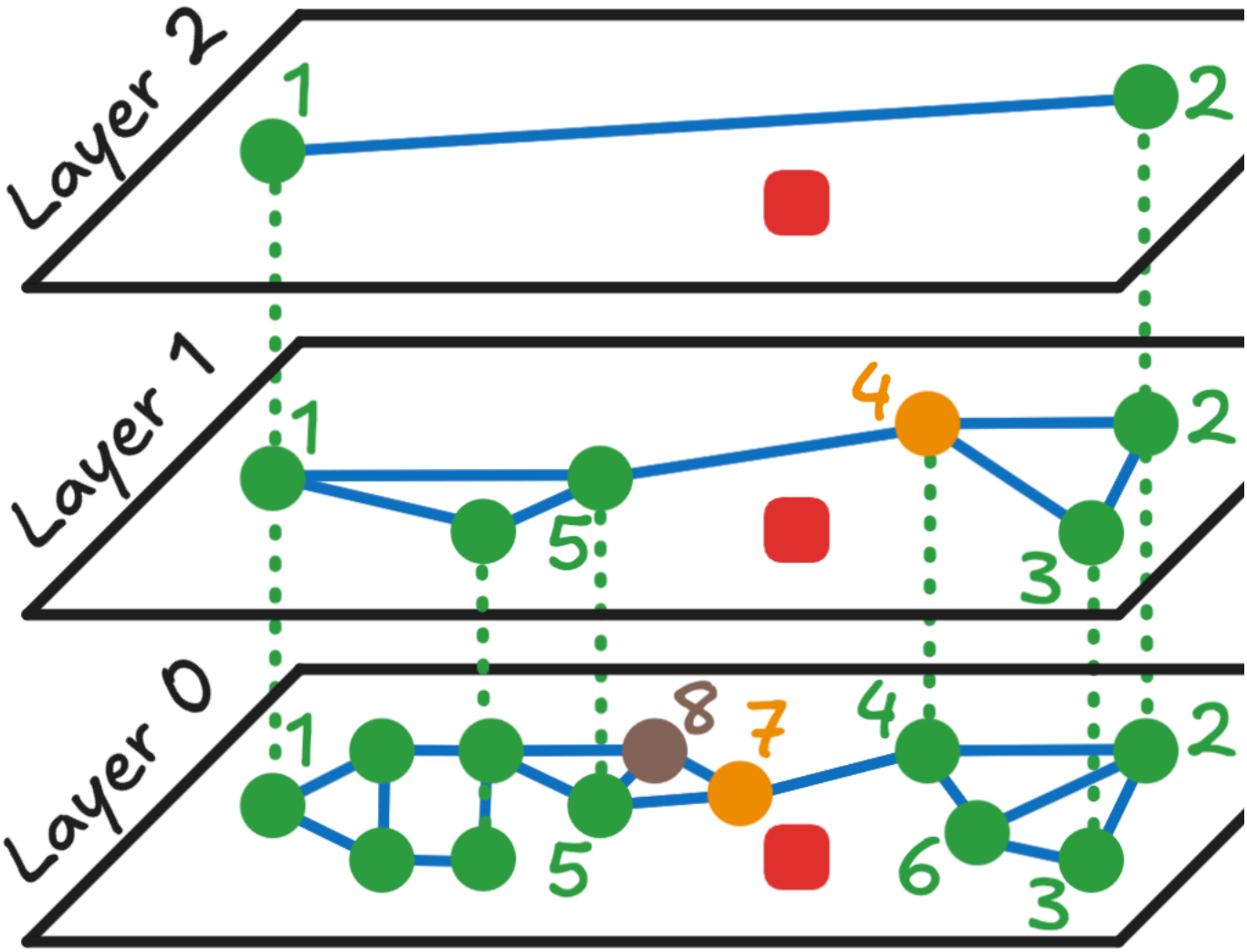


- Compute the distance between point `4` and the query (already known).
- Compute the distance to point `5`, a new neighbor (neighbors `2` and `3` were already considered and can be safely ignored).
- Point `4` remains the closest, so proceed to `Layer 0` with point `4` as the entry.

Step 5: Final Search in the Bottom Layer



- From point `4`, compute distances to its neighbors in `Layer 0` (points `2`, `6`, and `7`). In reality, point `2` is ignored because it was considered before.
- Point `7` is closest and becomes the new entry point.



- Among point 7's neighbors, only point 8 is unexplored.
- Point 7 remains the closest, so the search terminates.

Step 6: Return the Approximate Nearest Neighbor

The algorithm returns point 7 as the approximate nearest neighbor. While HNSW does not guarantee the exact nearest neighbor due to its greedy nature, it typically finds a very close match if the exact solution is not retrieved. In this example, only 8 distance computations were needed versus 12 for brute-force search. In real-world scenarios with optimized graphs and larger datasets, the efficiency gains are even more significant.

Now that we've explored how HNSW performs a search—navigating from coarse to fine layers using greedy routing—it's equally important to understand how this powerful structure is constructed in the first place. The next section walks through the process of building the HNSW index, layer by layer, revealing how data points are organized to enable such efficient and scalable search performance.

How the HNSW Index is Built — Step by Step

Imagine you're building a smart city map where every building (data point) is connected to others in a way that helps you find the fastest route to any destination. That's what HNSW does — but with data.

Step 1: Start with an Empty Graph

- At the beginning, there's no structure — just an empty space.
- The first data point you insert becomes the starting point (called the entry point) for all future insertions and searches.

Step 2: Assign a Height (Layer) to Each Data Point

- Each data point can be visualized as a building in a city.
- Some buildings are tall (they appear in higher layers), and some are short (they only appear on the ground floor).
- The height is chosen randomly, but taller buildings are rarer. This assignment follows an exponentially decreasing probability distribution:
  - Most points only appear on the ground floor (Layer 0).
  - A few appear on higher floors (Layer 1, 2, 3, and so on).
- This layered structure helps the algorithm "zoom in" from a broad overview to fine details.

Step 3: Insert the Point into the Graph

For each new point:

a. Start at the Top Layer

- Begin at the highest layer where the current entry point exists.
- Use a method called greedy search: look at the neighbors of the current point and move to the one that's closest to the new point.
- Repeat this until you can't get any closer in that layer.

b. Move Down One Layer

- Once you've reached the best spot in the current layer, go down to the next lower layer.
- Repeat the greedy search from the best point found in the previous layer.

c. Connect to Neighbors

- At each layer (from the top down to the ground floor), the new point connects to its M closest neighbors.
- These connections are bidirectional — both points know about each other.
- This creates a small-world network, enabling fast traversal between data points.

Step 4: Repeat for All Points

- Every new point goes through the same process: assign a height, search from the top, and connect to neighbors at each level.
- Over time, this builds a multi-layered graph that's fast to search through.

Key Parameters that Control the Build and Search Process

1. M — Max Connections per Node

- Controls how many neighbors each point connects to.
- Higher M = better accuracy, more memory usage.
- Lower M = faster build, less memory, lower accuracy.

2. efConstruction — Search Breadth During Build

- Controls how many candidates are considered when finding neighbors during insertion.
- Higher efConstruction = better graph quality, slower build.
- Lower efConstruction = faster build, but possibly lower search quality later.

3. efSearch — Search Breadth During Querying

- Used when you're searching the graph (not building it).
- Controls how many candidate nodes are explored during a query.
- Higher efSearch = better accuracy (more likely to find the true nearest neighbors), but slower search.
- Lower efSearch = faster search, but might miss the best matches.
- This is the main knob to tune speed vs. accuracy at query time.

4. ml — Level Multiplier

- Affects how likely a point is to appear in higher layers.
- Controls the shape of the hierarchy — more or fewer tall buildings.

Why This Works

- The top layers help you make big jumps across the data space — such as highways.
- The bottom layer gives you fine-grained detail — such as local streets.
- This combination makes HNSW both fast and accurate, even for huge datasets.

Step 1: Start at the Top

When you want to find something similar to your query, HNSW starts at the top layer. There are only a few points here, but they're strategically positioned and connected by long-range links. The algorithm starts at a predetermined **entry point**, usually the most recently inserted node during index construction.

Step 2: Navigate Toward Your Target

The algorithm performs **greedy routing**: it looks at the current point and asks: "Which of my connected neighbors is closest to what I'm looking for?" It then jumps to that neighbor. This process repeats, getting closer to the target with each jump.

Step 3: Move Down a Layer

Once it can't get any closer in the current layer (reaches a **local minimum**), it moves down to the next layer. This layer has more points and more precise connections, allowing for more refined navigation.

Step 4: Repeat Until You Reach the Bottom

The algorithm continues this process - navigate as close as possible using greedy search, then move down a layer — until it reaches the bottom layer where all data points exist.

Step 5: Find the Best Matches

At the bottom layer (Layer 0), it performs a final greedy search to find the most similar items to your query. The algorithm can return multiple approximate nearest neighbors if specified.

Time Complexity: The hierarchical structure enables **logarithmic search complexity** O(log n), compared to linear O(n) for brute-force search.

The Trade-offs: Understanding the Limitations

Approximate Results

HNSW delivers fast and accurate results, though it may occasionally miss the exact nearest neighbor. **Typical recall rates range from 90% to 99%**, meaning it finds the true nearest neighbor 90-99% of the time. For most applications, this trade-off is worthwhile.

Parameter Tuning

Getting the best performance from HNSW requires adjusting various settings:

Key parameters:

- **M (max connections)**: Higher M = better recall, more memory
- **efConstruction**: Build-time parameter affecting quality
- **efSearch**: Query-time parameter affecting speed/accuracy trade-off
- **ml (level multiplier)**: Controls layer probability distribution

Tuning guidelines:

- Start with default values (M=16, efConstruction=200)
- Increase M for higher recall (up to M=64)
- Adjust efSearch based on speed/accuracy requirements
- Use benchmarking tools to find optimal settings

Dynamic Updates

HNSW is best suited for **mostly-static datasets**. Frequent insertions and deletions can:

- **Degrade performance**: Graph becomes less optimal over time
- **Require rebuilding**: Periodic reconstruction for optimal performance
- **Increase complexity**: Dynamic updates require careful synchronization

Distance Metric Limitations

HNSW supports various distance metrics, but performs best with:

- **Euclidean distance (L2)**
- **Cosine similarity**
- **Other metrics**: May require modifications or perform suboptimally

The Science Behind HNSW

The Original Research

HNSW was developed by researchers **Yu. A. Malkov and D. A. Yashunin** in 2016. Their paper, "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs," was published on arXiv (arXiv:1603.09320) and later in IEEE Transactions on Pattern Analysis and Machine Intelligence. It has become one of the most influential works in the field of similarity search, **cited over 2,000 times** in academic literature.

Key Innovation

The breakthrough was combining two existing concepts:

1. **Skip Lists**: A probabilistic data structure invented by William Pugh in 1989 that allows fast searching by maintaining multiple levels of links
2. **Navigable Small World Networks**: Networks where you can reach any point quickly by following smart connections, based on Kleinberg's work on navigable networks

Mathematical Foundation

While the mathematics can be complex, the core insights are elegant:

Search Complexity: The hierarchical structure enables **logarithmic search complexity O(log n)**, compared to linear O(n) for brute-force search.

Connection Strategy: At each layer, nodes connect to their M closest neighbors, where M is a tunable parameter that affects both accuracy and memory usage.

Greedy Search Guarantee: The algorithm is guaranteed to reach a local minimum at each layer, and the hierarchical structure increases the probability that this local minimum is close to the global optimum.

Theoretical Properties

- **Scale Separation**: Upper layers provide "highway" connections for long-distance jumps
- **Polylogarithmic Complexity**: Search time scales as O(log^k n) where k is typically small
- **High Probability Guarantees**: Mathematical proofs show high probability of finding near-optimal results

Practical Considerations

When to Use HNSW

HNSW is ideal when you need:

- Fast similarity search in large datasets.
- Good accuracy with reasonable speed.
- To handle high-dimensional data (data with many features).
- Scalable solutions that work as your data grows.

When Not to Use HNSW

Consider alternatives when:

- You need guaranteed exact results.
- Your dataset is very small (traditional methods might be simpler).
- Memory usage is a critical constraint.
- You need to frequently add or remove data (HNSW is optimized for mostly-static datasets).

Conclusion

Hierarchical Navigable Small World (HNSW) represents a significant breakthrough in how we search for similar information in large datasets. By cleverly organizing data in multiple layers and creating smart connections between similar items, HNSW enables the fast, accurate searches that power many of the digital services we use daily.

While the underlying mathematics can be complex, the core concept - using a hierarchical structure to navigate efficiently through data - is intuitive and powerful. As our digital world continues to generate more data, algorithms such as HNSW become increasingly important for making that data useful and accessible.

Whether you're a student curious about how search works, a professional considering HNSW for a project, or simply someone who wants to understand the technology behind modern digital services, HNSW represents a fascinating example of how clever algorithms can solve real-world problems at incredible scale.

Author(s)

Wojciech "Victor" Fulmyk