INTEGRATING METAGENOMICS AND COMPUTATIONAL BIOLOGY TO UNCOVER MURBURN MECHANISMS

OUR TEAM

- Hemesh Yeteru
- Joel John
- Abhishek S
- Adarsh Pradeep

ABSTRACT

This project integrates metagenomics and computational biology to investigate murburn mechanisms, with a focus on reactive oxygen species (ROS) in microbial communities. We aim to identify ROS-related genes in bacterial genomes by leveraging advanced bioinformatics tools such as BLAST, sequence alignment, and homology detection. Further, we employ graph-based algorithms, network flow models, and random walk simulations to analyze ROS pathways and simulate their interactions within microbial ecosystems. By combining biological insights with efficient computational techniques, we enable pathway analysis, diffusion modeling, and visualization of ROS-driven processes. This interdisciplinary approach provides new perspectives on microbial metabolism, ecological dynamics, and potential applications in biotechnology and medicine.

INTRODUCTION

Metagenomics is the study of the genetic material of entire microbial communities, which gives insights into their functions and interactions. The murburn concept focuses on reactive oxygen species (ROS) and their role in processes like cellular respiration and oxidative stress. By integrating computational biology, this project uses advanced tools and algorithms to identify ROS-related genes, analyze pathways, and simulate microbial interactions. This approach combines biology and algorithmic analysis to uncover new perspectives on microbial ecosystems and their ecological significance.

OBJECTIVE : INTEGRATING BIOLOGY AND ALGORITHM

1. Meta Genomic Data Processing of Bacterial Genes (Using De Bruijn graph)
2. Cross-species analysis of ROS genes (human vs bacteria)
3. Network Flow Algorithms (for Pathway Analysis)
4. Random Walk Model (for ROS Diffusion Simulation)

CHALLENGES IN ANALYZING GENOMIC DATA

1. Data Volume
   Metagenomics data is often massive, posing computational challenges for analysis and storage.

2. Data Complexity
   The diversity of organisms in an environment can make it difficult to identify specific genes and pathways.

3. Data Noise
   Contamination and errors can make it difficult to obtain accurate and reliable results.

What do you find in Metagenomic Data processing ?

Metagenomic data processing is to efficiently assemble, classify, and analyze bacterial genomes from environmental samples. It assembles short DNA sequences (reads) obtained from sequencing technologies to reconstruct the original bacterial genomes.

Why De Bruijn Graphs?

De Bruijn graphs are better suited for assembling short-read sequencing data because they: Traditional methods like overlap graphs require comparing all reads with each other, which is computationally expensive. DBGs convert reads into fixed-length k-mers, reducing the complexity.

DBGs allow error correction by detecting low-frequency k-mers and removing them.

Many bacterial genomes contain repeats, and DBGs efficiently resolve ambiguities in such cases.

The adjacency representation in DBGs avoids storing explicit pairwise read comparisons, making them suitable for large metagenomic datasets.

What Do De Bruijn Graphs Do?

A De Bruijn graph is constructed as follows:

- **Step 1: K-mer Splitting**

Reads are broken into overlapping k-mers (substring of length k).

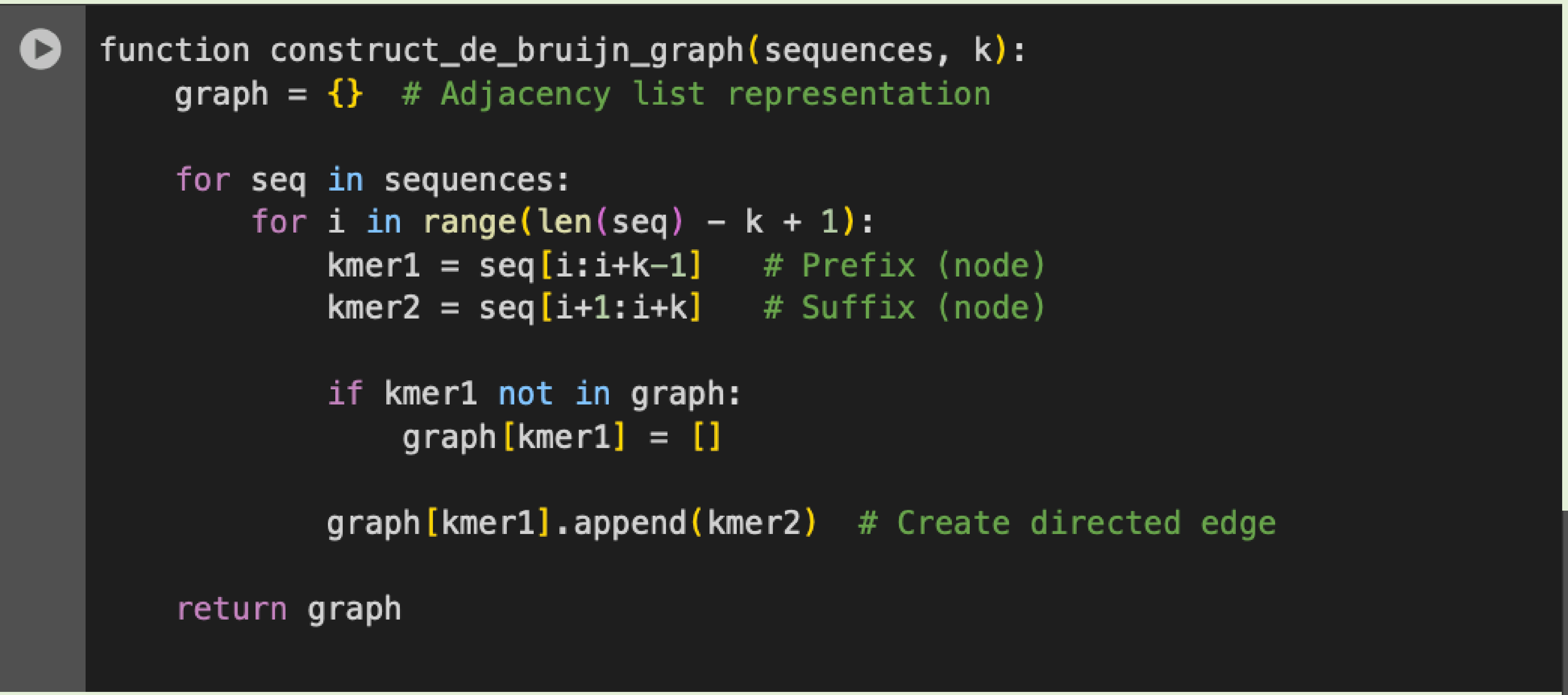
- **Step 2: Graph Construction**

Nodes represent k-1-mers, and directed edges connect consecutive k-mers.

- **Step 3: Path Traversal (Genome Assembly)**

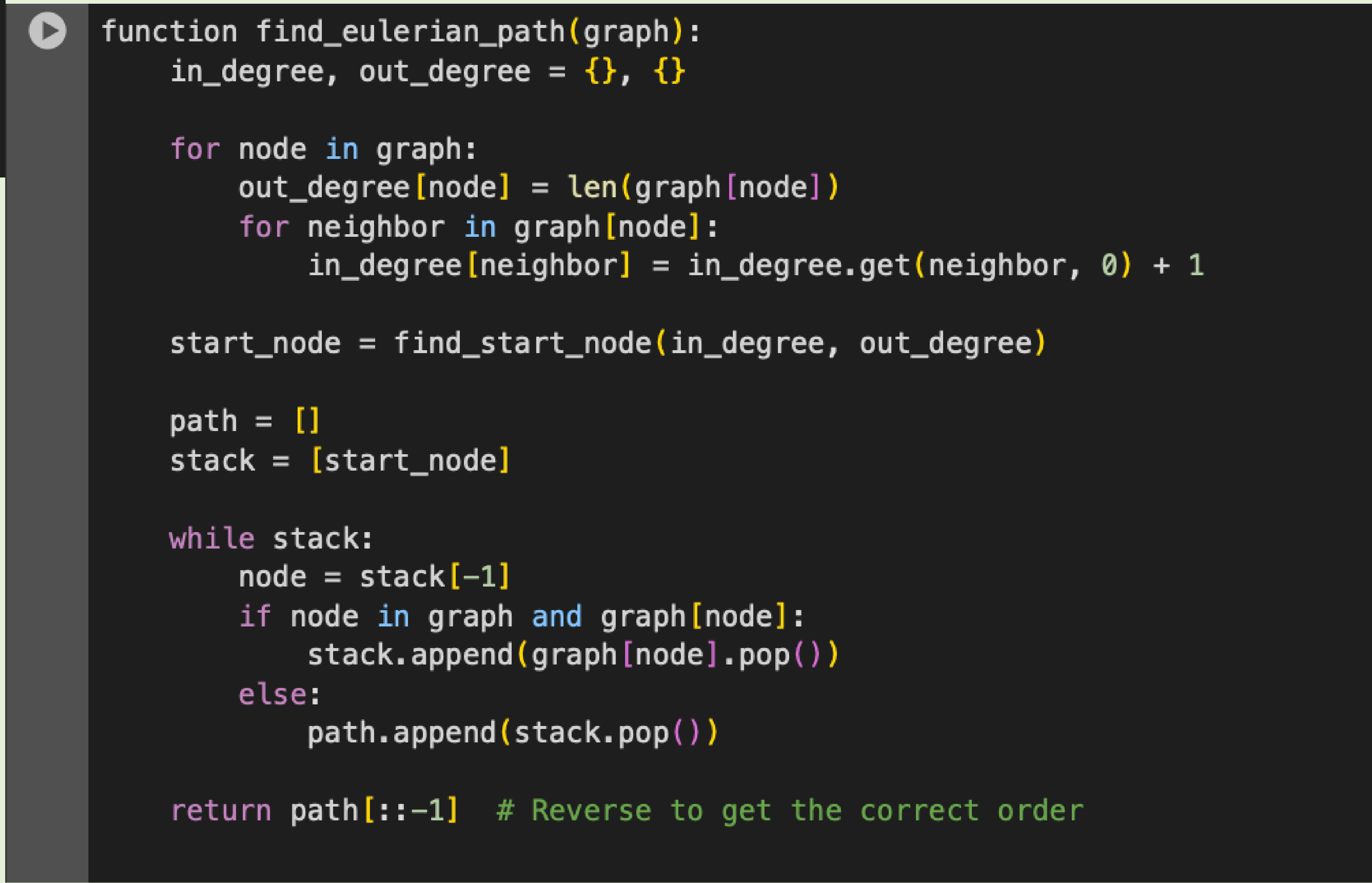
Eulerian paths (paths that visit every edge exactly once) reconstruct the original sequence.

Pseudo Code for De Bruijn Graph



```
function construct_de_bruijn_graph(sequences, k):
    graph = {}  # Adjacency list representation

    for seq in sequences:
        for i in range(len(seq) - k + 1):
            kmer1 = seq[i:i+k-1]    # Prefix (node)
            kmer2 = seq[i+1:i+k]    # Suffix (node)

            if kmer1 not in graph:
                graph[kmer1] = []

            graph[kmer1].append(kmer2)  # Create directed edge

    return graph
```

Eulerian Path (Genome Assembly)



```
function find_eulerian_path(graph):
    in_degree, out_degree = {}, {}

    for node in graph:
        out_degree[node] = len(graph[node])
        for neighbor in graph[node]:
            in_degree[neighbor] = in_degree.get(neighbor, 0) + 1

    start_node = find_start_node(in_degree, out_degree)

    path = []
    stack = [start_node]

    while stack:
        node = stack[-1]
        if node in graph and graph[node]:
            stack.append(graph[node].pop())
        else:
            path.append(stack.pop())

    return path[::-1]  # Reverse to get the correct order
```

METHODOLOGY

1. Data Collection:

- Obtain metagenomic datasets from public repositories like NCBI SRA and MG-RAST.

2. Data Preprocessing:

- Data retrieval from NCBI using entrez
- Quality filtering (Extracting only valid Nucleotide sequences)

3. K-mer extraction --> Graph Construction (De Bruijn graph)

- The function extract_limited_kmers() extracts short overlapping substrings (k-mers) of size k (default k=10).

The function build_de_bruijn_graph() constructs a directed graph where:

- Nodes represent (k-1)-mers (prefix & suffix of k-mers).
- Edges represent k-mer transitions between nodes.

3. Gene Annotation and Pathway Analysis:

- Bacterial genes similar to human ROS genes, helping in comparative genomics.(Using BLASTp)
- This comparative approach links bacterial ROS defence mechanisms to human systems, aiding in evolutionary and biomedical insights.

4. Network flow algorithm:

- We use Edmonds-Karp algorithm to determine maximum flow between human ROS genes and bacterial homologs.
- The flow capacity represents gene similarity strength, allowing us to quantify gene conservation & functional transfer.

5. Random Walk model:

- Steady-state probabilities (genes most affected by ROS diffusion).
- Mean first passage time (MFPT) (how quickly ROS reaches different bacterial genes).

6. Integration:

- Combine metagenomic analysis with computational modeling to explore murburn mechanisms and their ecological significance.

PROJECT PHASES

STEPS	PROCESS	TECHNIQUES USED
1	Process bacterial genomes for ROS genes	De Bruijn Graphs
2	Cross-Species Analysis of ROS Genes (Human vs. Bacteria)	BLASTp for protein homology search
3	Network Flow Algorithms (for Pathway Analysis)	Max-Flow Algorithms (Edmonds-Karp)
4	Random Walk Model (for ROS Diffusion Simulation)	Random Walk Simulations

Cross-Species Analysis of ROS Genes (Human vs. Bacteria)

Comparing ROS(Reactive Oxygen Species) Defence Mechanisms

- Humans and bacteria both produce ROS-scavenging enzymes (e.g., catalase, superoxide dismutase), but their mechanisms differ.
- This analysis reveals how bacterial ROS defenses help pathogens survive in oxidative environments (e.g., immune system attack).

Identifying Evolutionarily Conserved ROS Genes

- Some ROS-related genes are highly conserved across species.
- Example: Superoxide dismutase (SOD) is found in both humans and bacteria, showing its fundamental role in oxidative stress protection.

We use BLASTp . What is BLASTp ?

Basic Local Alignment Search Tool for Proteins ---> BLASTp

Finds Similar Proteins → Compares a query protein sequence to a protein database (e.g., NCBI, UniProt).

Identifies Evolutionary Relationships → Detects homologous proteins (orthologs, paralogs).

Predicts Function of Unknown Proteins → If a query protein is similar to a known protein, it likely has a similar function.

What does it do ?

Input Query Protein Sequence

Breaks Query into Small Words (k-mers) → Default k=3 (3 amino acids at a time).

Searches for Similar Segments in the Database.

Extends Matches to find high-scoring alignments.

Computes Statistical Significance → Assigns E-value, % Identity, and Bit Score.

Network Flow Algorithms (for Pathway Analysis)

Network Flow Algorithms are used in computational biology, particularly for pathway analysis, to model and analyze the flow of biological entities (such as molecules, proteins, or metabolites)

These algorithms help in understanding how substances move and interact within cellular networks, such as metabolic, signaling, or genetic pathways.

What does it do ?

Network flow algorithms work by modeling biological systems as graphs, where:

Nodes represent biological entities (like genes, proteins, metabolites, or enzymes).

Edges represent the interactions or relationships between those entities (like biochemical reactions, protein-protein interactions, etc.).

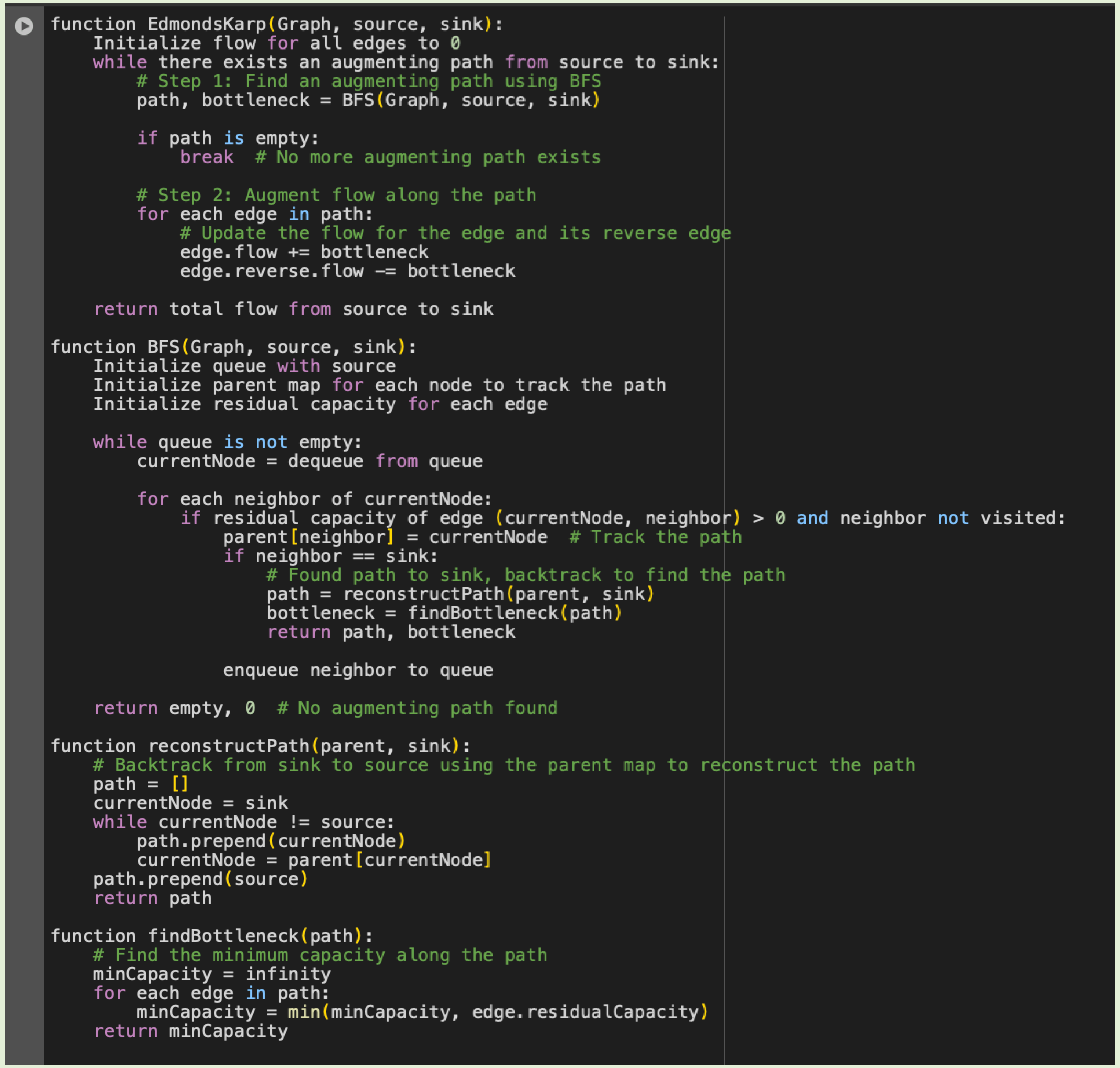
The goal is to analyze how "flow" (e.g., energy, metabolites, signals) travels through the network from one point to another, often to optimize or find bottlenecks, understand dynamics, or identify critical components within a pathway.

Max-Flow Algorithms (Edmonds-Karp)

Edmonds-Karp finds the maximum flow in a flow network. This is the largest amount of flow that can be sent from a source node to a sink node in the network while respecting the capacities of the edges

- A flow network is a directed graph where each edge has a capacity (the maximum amount of flow it can carry).
- The flow must satisfy the constraints:
- The amount of flow entering a node (except for the source and sink) must equal the amount of flow leaving the node (this is the conservation of flow).
- The flow through an edge cannot exceed its capacity.

Pseudo Code for this Algorithm



```
function EdmondsKarp(Graph, source, sink):
    Initialize flow for all edges to 0
    while there exists an augmenting path from source to sink:
        # Step 1: Find an augmenting path using BFS
        path, bottleneck = BFS(Graph, source, sink)

        if path is empty:
            break  # No more augmenting path exists

        # Step 2: Augment flow along the path
        for each edge in path:
            # Update the flow for the edge and its reverse edge
            edge.flow += bottleneck
            edge.reverse.flow -= bottleneck

    return total flow from source to sink

function BFS(Graph, source, sink):
    Initialize queue with source
    Initialize parent map for each node to track the path
    Initialize residual capacity for each edge

    while queue is not empty:
        currentNode = dequeue from queue

        for each neighbor of currentNode:
            if residual capacity of edge (currentNode, neighbor) > 0 and neighbor not visited:
                parent[neighbor] = currentNode  # Track the path
                if neighbor == sink:
                    # Found path to sink, backtrack to find the path
                    path = reconstructPath(parent, sink)
                    bottleneck = findBottleneck(path)
                    return path, bottleneck

                enqueue neighbor to queue

    return empty, 0  # No augmenting path found

function reconstructPath(parent, sink):
    # Backtrack from sink to source using the parent map to reconstruct the path
    path = []
    currentNode = sink
    while currentNode != source:
        path.prepend(currentNode)
        currentNode = parent[currentNode]
    path.prepend(source)
    return path

function findBottleneck(path):
    # Find the minimum capacity along the path
    minCapacity = infinity
    for each edge in path:
        minCapacity = min(minCapacity, edge.residualCapacity)
    return minCapacity
```

Random Walk Model (for ROS Diffusion Simulation)

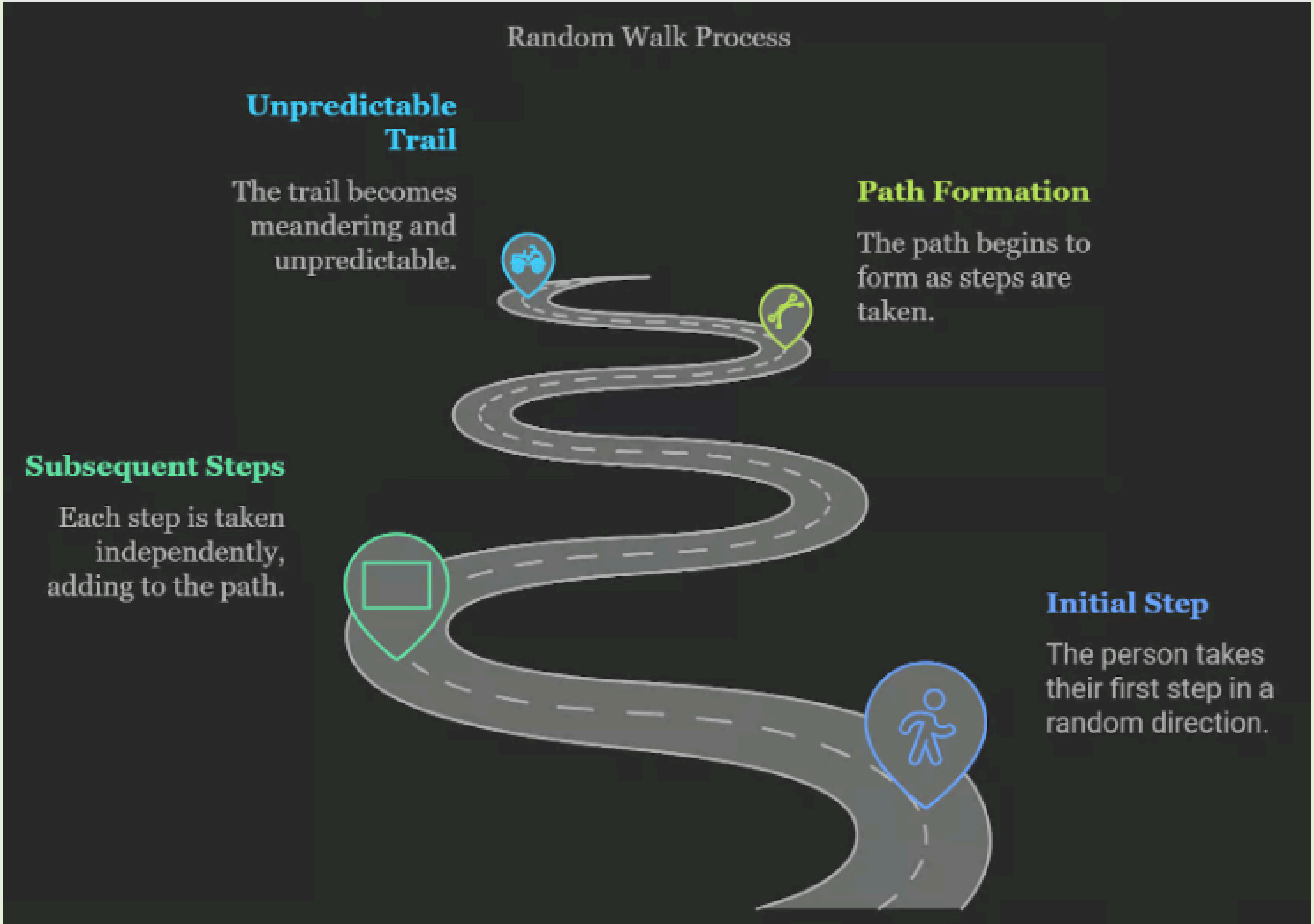
Random Walk Model for ROS (Reactive Oxygen Species) diffusion is used to simulate how ROS molecules, generated either internally (by metabolic processes) or externally (due to environmental stress or immune responses), diffuse through bacterial cells or colonies

Understanding how ROS spreads inside bacterial cells or through bacterial communities (e.g., biofilms) is important for studying the bacterial response to oxidative stress and for designing antimicrobial strategies.

Use if you have a complex system where you want to estimate the behavior of the system using random sampling to account for both random movement and complex interactions (like ROS-induced damage in bacteria or complex biochemical processes).

Random Walk Model (for ROS Diffusion Simulation)

- Initialize the starting position (e.g., starting at the center of a 2D grid).
- Choose a random direction for the first step (e.g., up, down, left, or right).
- Move in that direction by updating the current position.
- Repeat the random movement process for a set number of steps or until a termination condition is met (e.g., after 100 steps or when the ROS reaches the cell boundary).
- Optionally, track the positions over time or collect statistical data about the walk (e.g., average distance traveled).



Random Walk Process

Initial Step

The person takes their first step in a random direction.

Subsequent Steps

Each step is taken independently, adding to the path.

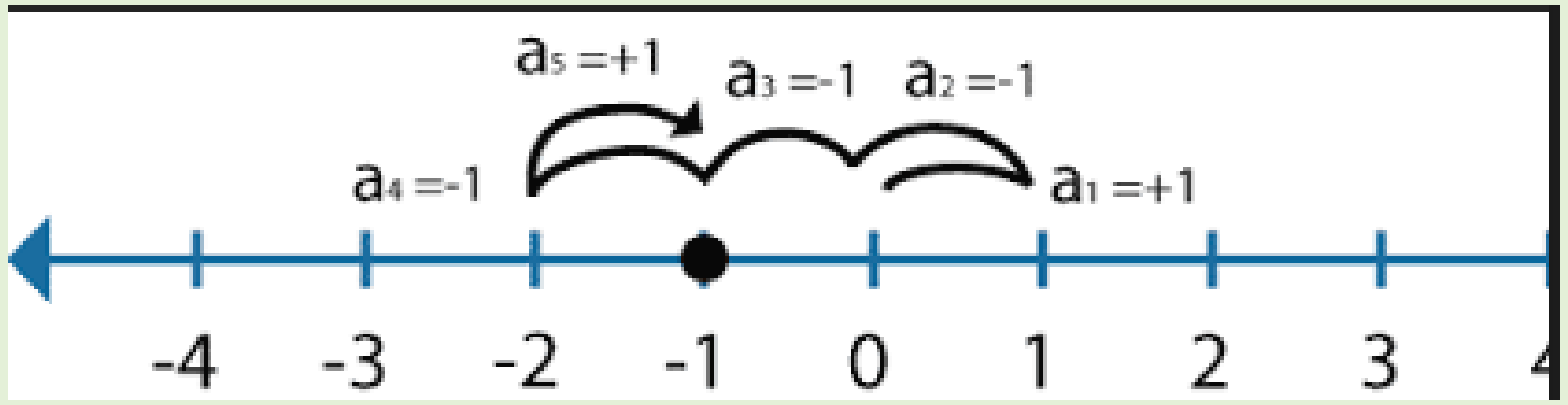
Path Formation

The path begins to form as steps are taken.

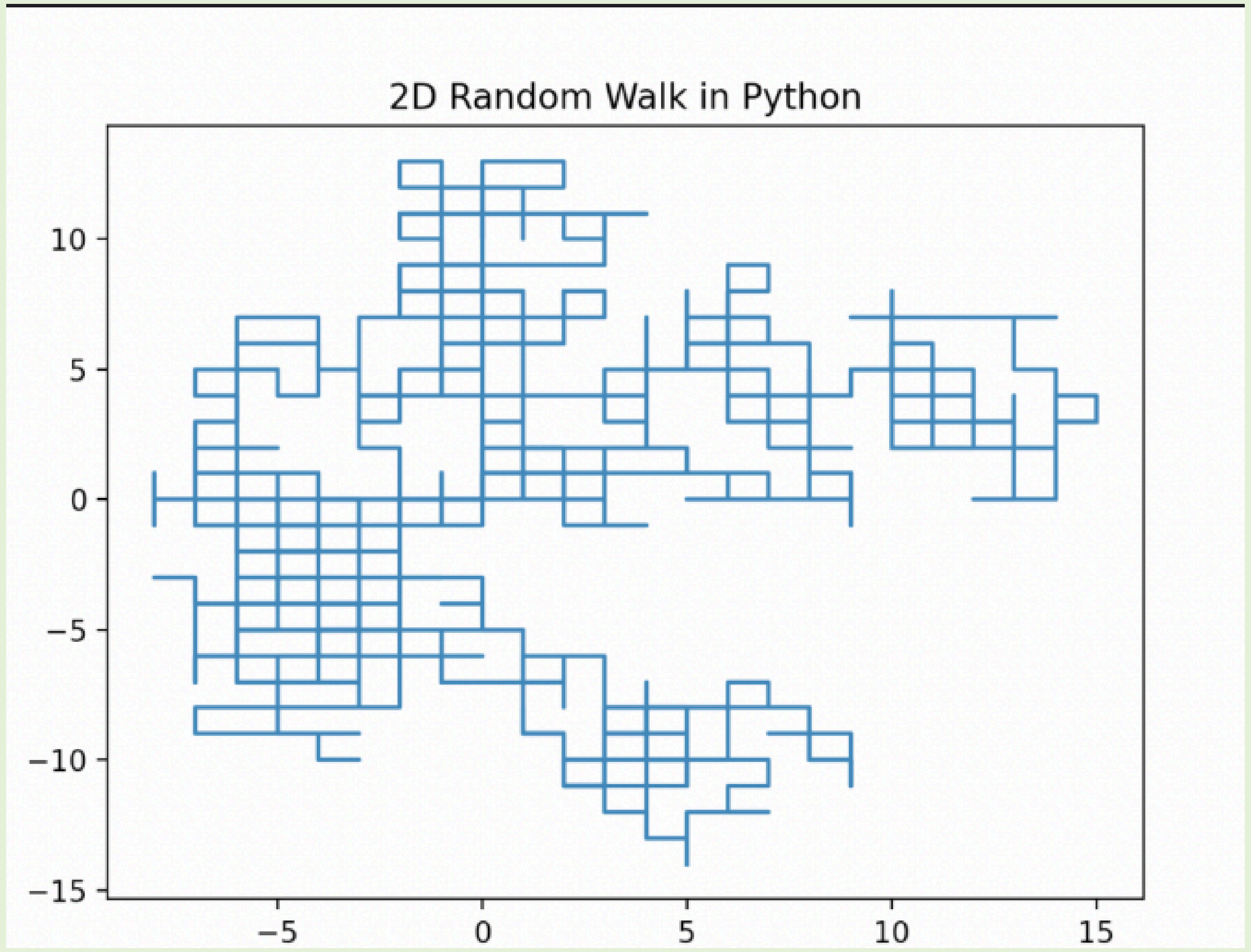
Unpredictable Trail

The trail becomes meandering and unpredictable.

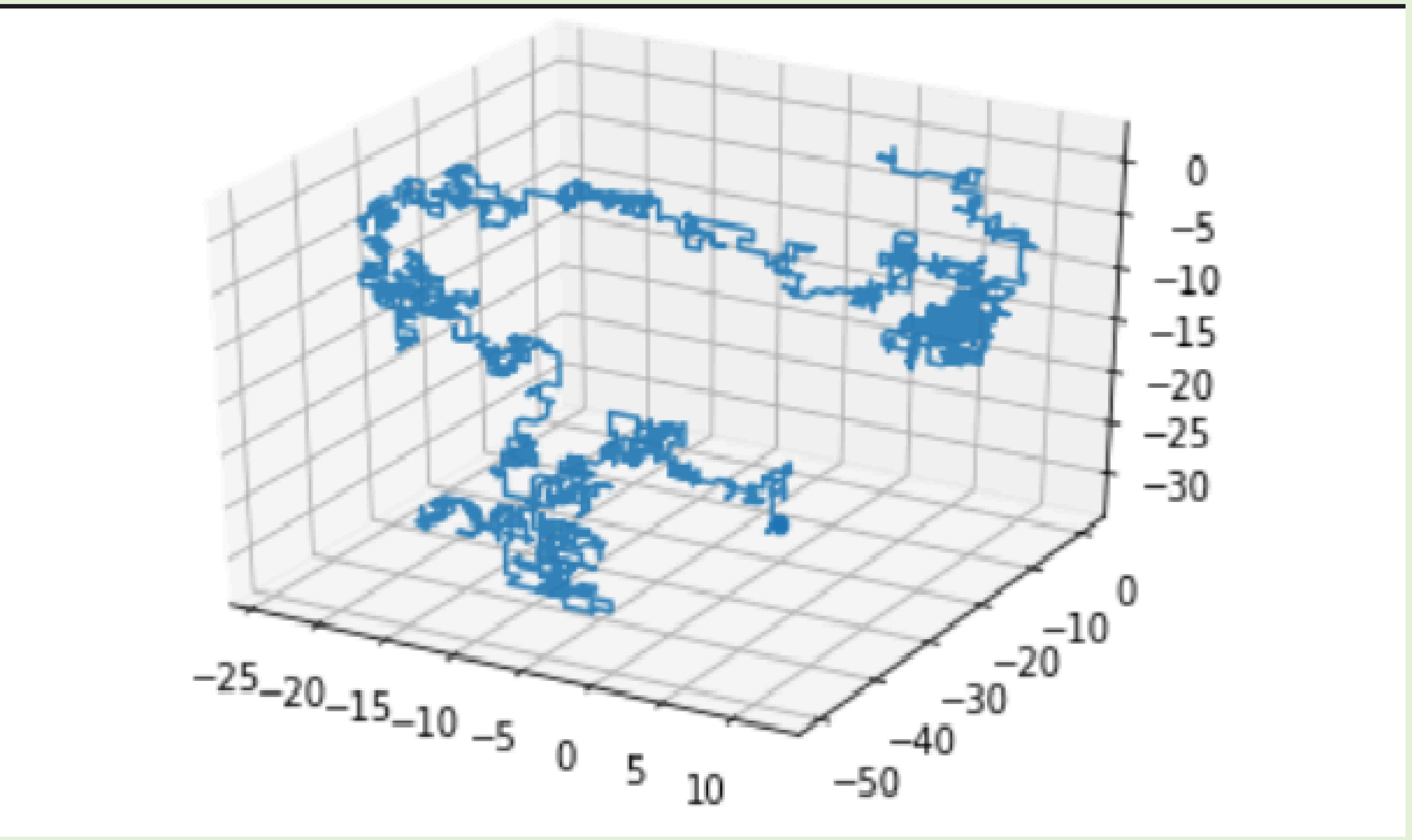
1-D random walk



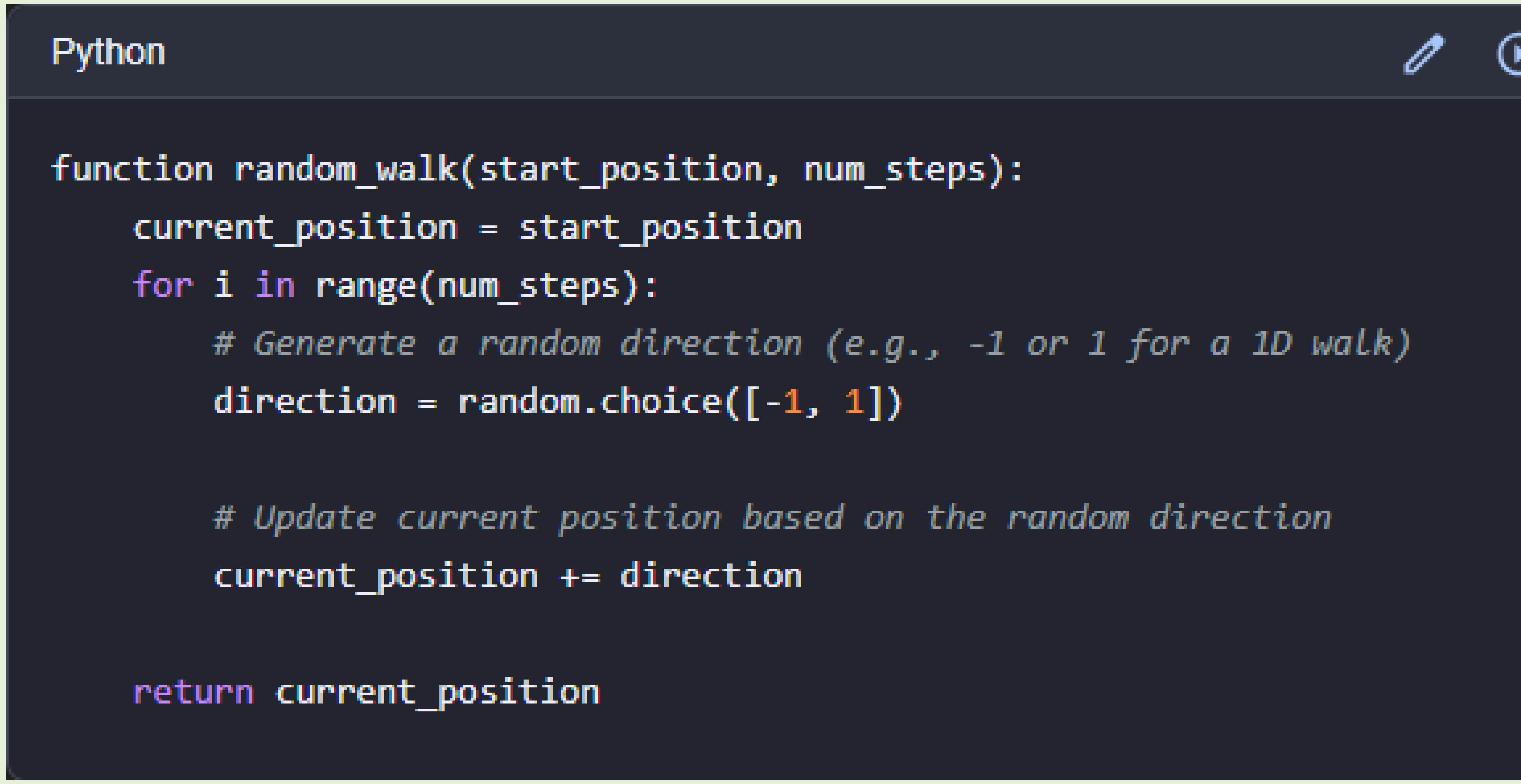
2-D random walk



3-D random walk



pseudocode



```
function random_walk(start_position, num_steps):
    current_position = start_position
    for i in range(num_steps):
        # Generate a random direction (e.g., -1 or 1 for a 1D walk)
        direction = random.choice([-1, 1])

        # Update current position based on the random direction
        current_position += direction

    return current_position
```

Note:

For 1D random walk and 2D random walk there is a high probability of returning to the starting point whereas in case of 3D or higher dimensions the probability decreases.

Relevance of Random Walk in Our Project

Why Use Random Walk for ROS Diffusion?

Unpredictable Movement

ROS molecules do not follow a fixed path; their motion is random due to collisions with cellular structures and other molecules.

This behavior is similar to Brownian motion, where particles move in random directions due to kinetic energy.

Models Diffusion Accurately

Diffusion is a key process in ROS transport, determining how far and fast they spread within bacterial cells or biofilms.

A random walk algorithm mimics this natural diffusion, helping us predict ROS concentration gradients in different regions.

CONCLUSION & RESULTS EXPECTED

This project integrates metagenomics and computational biology to study reactive oxygen species (ROS) in microbial ecosystems. By combining gene identification, pathway analysis, and modeling, it provides insights into microbial functions and ecological roles.

Expected Results:

1. Identification of ROS-related genes.
2. Analysis of ROS pathways in microbial communities.
3. Simulations of ROS interactions.
4. Visualizations like heatmaps and networks.
5. Optimized algorithms for metagenomic analysis.

LITERATURE REVIEW

SR.NO	DESCRIPTION	REFERENCE
1	This study discusses the application of De Bruijn graphs in microbiome research, highlighting their role in assembling high-throughput sequencing data and their potential in comparative genomics.	Dufault-Thompson, K., & Jiang, X. (2022). Applications of de Bruijn graphs in microbiome research. iMeta, 1(1)
2	This paper revisits the E-value calculations in BLASTp, aiming to improve the accuracy of protein sequence alignments and the reliability of homology predictions.	Pearson, W. R. (2024). BLAST from the past: revisiting blastp's E-value. Bioinformatics, 40(12), btae729.
3	This study provides an overview of graph models used in DNA sequencing by hybridization, focusing on the application of De Bruijn graphs in genome assembly. It discusses the theoretical framework and compares outcomes using SPAdes and Velvet assemblers.	Sarkar, B. (2024). A Study of Computational Genome Assembly by Graph Theory. Annals of West University of Timisoara: Mathematics and Computer Science, 60(1), 1–24.

LITERATURE REVIEW

SR.NO	DESCRIPTION	REFERENCE
4	This preprint serves as a beginner's guide to microbiome modeling, offering an interdisciplinary overview that starts with fundamental knowledge of microbiomes, metagenomics methods, and modeling approaches.	Lange, E., Kranert, L., Krüger, J., Benndorf, D., & Heyer, R. (2024). Microbiome Modeling: A Beginner's Guide. Preprints, 2024010789
5	This dissertation explores advanced computational methods for analyzing metagenomic data, focusing on the application of De Bruijn graphs in genome assembly and the use of BLASTp for protein sequence alignment.	Smith, J. A. (2023). Advanced Computational Techniques in Metagenomics