Path-Guided Stochastic Gradient Descent for Pangenome Graph Layout

Mathematical Formulation and Algorithm Description

September 25, 2025

1 Introduction

Path-Guided Stochastic Gradient Descent (PG-SGD) is an algorithm for computing a one-dimensional layout of nodes in a pangenome graph such that the distances between nodes in the layout approximates their nucleotide distances along paths through the graph.

2 Problem Formulation

Given:

- A pangenome graph $G = (V, E, P)$ where:
  - V is the set of nodes
  - E is the set of edges
  - P is the set of paths (genomes) through the graph
- Each node $v_i \in V$ has a sequence of length ℓ_i
- Each path $p \in P$ is an ordered sequence of node traversals

Goal: Find a 1D layout $X : V \to \mathbb{R}$ that minimizes the difference between:

- Layout distance: $|X_i - X_j|$ for nodes v_i, v_j
- Nucleotide distance: d_{ij}^{path} along paths

3 Mathematical Formulation

3.1 Objective Function

The algorithm implicitly minimizes the stress function:

$$\text{Stress} = \sum_{(i,j)\in S} w_{ij} \left(|X_i - X_j| - d_{ij}\right)^2 \tag{1}$$

where:

- S is the set of sampled node pairs
- w_{ij} is the weight for pair (i, j)
- d_{ij} is the target distance (nucleotide distance along path)
- X_i, X_j are the 1D positions of nodes i and j

3.2 Weight Calculation

The weight for each pair is inversely proportional to the nucleotide distance:

$$w_{ij} = \frac{1}{d_{ij}} \tag{2}$$

This gives more importance to pairs that are close together along paths.

3.3 Update Rule

For each sampled pair (i, j), the positions are updated using:

$$\Delta = \mu \cdot \frac{|X_i - X_j| - d_{ij}}{2} \tag{3}$$

$$r = \frac{\Delta}{|X_i - X_j|} \tag{4}$$

$$r_x = r \cdot (X_i - X_j) \tag{5}$$

$$X_i \leftarrow X_i - r_x \tag{6}$$

$$X_j \leftarrow X_j + r_x \tag{7}$$

where:

- $\mu = \min(\eta \cdot w_{ij}, 1)$ is the bounded learning rate
- η is the global learning rate (decreases over iterations)
- Δ is the update magnitude
- r is the normalized update factor
- r_x is the actual position update

3.4 Learning Rate Schedule

The learning rate follows an exponential decay:

$$\eta_t = \eta_{\max} \cdot e^{-\lambda t} \tag{8}$$

where:

- $\eta_{\max} = 1/w_{\min}$ is the maximum learning rate
- $\eta_{\min} = \epsilon/w_{\max}$ is the minimum learning rate
- $\lambda = \ln(\eta_{\max}/\eta_{\min})/(t_{\max} - 1)$ is the decay rate
- t is the current iteration
- ϵ is a small constant (default: 0.01)

4 Sampling Strategy

4.1 Path Step Sampling

At each iteration, the algorithm:

1. Randomly samples a step s_a from all path steps
2. Determines the path p and position within path
3. Samples a second step s_b from the same path

4.2 Second Step Selection

The second step s_b is chosen using:

- **Before cooling phase**: Uniform random sampling from the path
- **After cooling phase**: Zipfian distribution sampling

4.2.1 Zipfian Distribution

The probability of selecting a step at distance k follows:

$$P(k) = \frac{k^{-\theta}}{\sum_{i=1}^{n} i^{-\theta}} \tag{9}$$

where:

- θ is the Zipfian parameter (default: 0.99)
- n is the maximum jump distance (capped by path length)
- Higher θ values favor smaller distances

4.3 Distance Calculation

The nucleotide distance between steps is:

$$d_{ij} = |pos_a - pos_b| \tag{10}$$

where pos_a and pos_b are the cumulative nucleotide positions of the steps along the path.

5 Algorithm

Algorithm 1 Path-Guided SGD

```
 1: **Input:** Graph $G$, paths $P$, iterations $t_{\max}$, parameters $\epsilon$, $\delta$, $\theta$
 2: **Output:** Node positions $X$
 3: Initialize $X$ with nodes in topological order
 4: Compute all path step positions
 5: Set cooling iteration $t_{\text{cool}} = 0.5 \cdot t_{\max}$
 6: **for** $t = 1$ to $t_{\max}$ **do**
 7:   Update learning rate: $\eta_t = \eta_{\max} \cdot e^{-\lambda t}$
 8:   **while** term updates < min_updates **do**
 9:     Sample random step $s_a$ from all path steps
10:     Get path $p$ and position of $s_a$
11:     **if** $t > t_{\text{cool}}$ **then**
12:       Sample $s_b$ using Zipfian distribution
13:     **else**
14:       Sample $s_b$ uniformly from path $p$
15:     **end if**
16:     Calculate $d_{ij} = |pos_a - pos_b|$
17:     Calculate weight $w_{ij} = 1/d_{ij}$
18:     Calculate $\mu = \min(\eta_t \cdot w_{ij}, 1)$
19:     Get nodes $i, j$ from steps $s_a, s_b$
20:     Calculate $\Delta = \mu \cdot (|X_i - X_j| - d_{ij})/2$
21:     Update positions:
22:     $r = \Delta/|X_i - X_j|$
23:     $X_i \leftarrow X_i - r \cdot (X_i - X_j)$
24:     $X_j \leftarrow X_j + r \cdot (X_i - X_j)$
25:     Track $\Delta_{\max} = \max(\Delta_{\max}, |\Delta|)$
26:   **end while**
27:   **if** $\Delta_{\max} < \delta$ **then**
28:     **break** // Converged
29:   **end if**
30: **end for**
31: **return** $X$
```

6 Convergence Criteria

The algorithm terminates when either:

1. Maximum iterations $t_{\max}$ is reached (default: 30)
2. Maximum update $\Delta_{\max} < \delta$ (default: $\delta = 0.01$)

7 Parallelization

The algorithm uses HOGWILD! parallelization:

- Multiple threads sample and update pairs concurrently
- Position updates use atomic operations
- No locks except for atomic position updates
- Convergence checked periodically by a monitor thread

8 Key Implementation Details

8.1 Position Initialization

Nodes are initially placed according to their topological order, with position equal to cumulative sequence length:

$$X_i^{(0)} = \sum_{j<i} \ell_j \tag{11}$$

8.2 Handling Zero Distances

When $|X_i - X_j| = 0$, a small value (10^{-9}) is used to avoid division by zero.

8.3 Memory Efficiency

The algorithm avoids storing all pairwise distances by:

- Computing distances on-the-fly during sampling
- Using path index structures for fast position lookups
- Storing only current node positions

Parameter	Default	Description
$t_{\max}$	30	Maximum iterations
ϵ	0.01	Learning rate epsilon
δ	0.01	Convergence threshold
θ	0.99	Zipfian parameter
cooling_start	0.5	Fraction of iterations before cooling
min_term_updates	1000	Updates per iteration
space	100	Maximum jump distance

Table 1: Default PG-SGD parameters in ODGI

9 Parameters

10 Conclusion

Path-Guided SGD leverages the path structure of pangenome graphs to efficiently compute a 1D layout where nodes that are close along genomic paths are placed close together in the layout. The algorithm's use of stochastic sampling and the HOGWILD! parallelization strategy enables it to scale to large pangenome graphs while maintaining biological meaningfulness in the resulting layout.