

Machine Learning with Graphs: Homework I

Dev Sanghvi - ds221

Problem 1

(a) Drawing a graph so that node positions reflect structural proximity

Objective. Let $G = (V, E)$ be an unweighted graph, and let $d_G(u, v)$ denote shortest-path distance. We seek an embedding $\phi : V \rightarrow \mathbb{R}^2$ such that Euclidean distances reflect graph distances. A standard formulation is the MDS (Kruskal) stress objective:

$$\min_{\phi: V \rightarrow \mathbb{R}^2} \sum_{\{u, v\} \subseteq V} w_{uv} (\|\phi(u) - \phi(v)\|_2 - d_G(u, v))^2,$$

where $w_{uv} \geq 0$ are weights.

Weight choice intuition. Using $w_{uv} = 1$ treats all node pairs equally. Using $w_{uv} = 1/d_G(u, v)^2$ emphasizes preserving short graph distances more strongly, which is often desirable for visualization since local neighborhood structure is perceptually more important than long-range distances.

Algorithm and complexity.

1. Compute all-pairs shortest-path distances $D_{uv} = d_G(u, v)$.
2. Apply classical multidimensional scaling (MDS) to D to obtain a 2D embedding.

All-pairs shortest paths via Floyd–Warshall take $O(|V|^3)$ time. Classical MDS requires an eigen-decomposition of a centered Gram matrix, also $O(|V|^3)$ in the worst case.

Exact vs. approximate. This method is approximate: in general, no embedding in \mathbb{R}^2 can preserve all graph distances exactly, so MDS returns a least-squares best fit.

Example. For the path graph $1 - 2 - 3$, distances are $d(1, 2) = 1$, $d(2, 3) = 1$, $d(1, 3) = 2$. An embedding $\phi(1) = (-1, 0)$, $\phi(2) = (0, 0)$, $\phi(3) = (1, 0)$ matches these distances exactly.

(b) Comparing two graphs based on structural similarity

Objective. Given graphs G_1 and G_2 , define a similarity score $\text{sim}(G_1, G_2)$ that is large when their local neighborhood structures are similar.

Algorithm and complexity. A standard approach is the Weisfeiler–Lehman (WL) subtree kernel. Starting from initial node labels, labels are iteratively refined by hashing each node’s current label together with the multiset of neighbor labels. Histograms of labels at each iteration are compared via inner products. With h iterations, the complexity is $O(h(|V| + |E|))$ per graph.

Exact vs. approximate. This approach is approximate: WL features are not complete invariants. In particular, certain non-isomorphic graphs (e.g. some regular graphs) can share identical WL histograms for fixed h .

Example. Two 3-cycles (triangles) with identical initial labels yield identical WL label histograms at all iterations, resulting in maximal similarity.

(c) Comparing two trees based on structural proximity

Objective. Given two labeled trees T_1 and T_2 , define structural proximity via tree edit distance: the minimum cost sequence of node insertions, deletions, and relabelings transforming T_1 into T_2 .

Algorithm and complexity. For rooted, ordered trees, the Zhang-Shasha dynamic programming algorithm computes the tree edit distance. Its time complexity is $O(n_1 n_2 \cdot \min(\text{depth}_1, \text{leaves}_1) \cdot \min(\text{depth}_2, \text{leaves}_2))$. In the worst case, this is $O(n_1^2 n_2^2)$, though for many practical tree shapes (e.g., shallow or linear trees), it runs significantly faster, approaching quadratic time.

Exact vs. approximate. The algorithm is exact with respect to the edit-distance objective.

Example. If T_1 is a root with two leaf children and T_2 is a root with one leaf child, deleting one leaf yields T_2 , so the distance is 1.

(d) Grouping nodes based on structural proximity

Objective. Partition V into k clusters to keep strongly connected nodes together. A common objective is the normalized cut:

$$\text{Ncut}(C_1, \dots, C_k) = \sum_{i=1}^k \frac{\text{cut}(C_i, \bar{C}_i)}{\text{vol}(C_i)}.$$

Algorithm and complexity. Spectral clustering computes eigenvectors of the normalized Laplacian and clusters their rows via k -means. A full eigendecomposition is $O(|V|^3)$, but in practice, sparse graphs admit faster iterative methods (e.g. Lanczos or power iteration) that scale nearly linearly in $|E|$.

Exact vs. approximate. Spectral clustering solves a continuous relaxation of normalized cut and is therefore approximate.

Example. Two triangles connected by a single bridge edge are separated into two clusters when $k = 2$.

Problem 2

(a) G_t and all H_i are constrained to be path graphs

Claim. $g(G_t)$ can be computed in polynomial time.

Algorithm (high level). A labeled path graph corresponds to a labeled sequence of vertices. Let G_t be a path on vertices (v_1, \dots, v_n) ; define the label string $S = L_t(v_1)L_t(v_2) \dots L_t(v_n)$. For each H_i which is a path of length ℓ_i with vertex order (u_1, \dots, u_{ℓ_i}) , define the pattern string $P_i = L_i(u_1) \dots L_i(u_{\ell_i})$. Then $\text{count}(G_t, H_i)$ equals the number of occurrences of P_i as a contiguous substring of S . Compute this count via a string-matching algorithm (e.g. KMP) in $O(n + \ell_i)$ time per i .

Complexity. Total time $O(nR + \sum_i \ell_i)$, hence polynomial.

(b) G_t and all H_i are constrained to be trees

Analysis. For general (unordered) trees, the counting problem is computationally intractable. Combining results from children subtrees in unordered trees requires solving a permanent-like counting problem over bipartite matchings, which is known to be a hard problem. Therefore, for unordered trees, exact counting is not feasible in polynomial time.

Polynomial-time Algorithm (Restricted Case). To achieve polynomial time, we must restrict the domain to *rooted, ordered trees*. Under this constraint, we can use dynamic programming. We root both trees and compute for each pair of nodes (x, y) the number of label-preserving embeddings of the subtree of H_i rooted at y into the subtree of G_t rooted at x . Since the order of children is fixed, the transition step becomes a sequence alignment rather than a matching problem, which simplifies the computation.

Complexity. This runs in $O(|V_t||V(H_i)|)$ time per pattern, which is polynomial in the size of $|V_t|$ and $|V(H_i)|$.

(c) Only the subgraphs H_i are constrained to be paths (but G_t is a general graph)

Hardness Result. Computing $g(G_t)$ is NP-hard.

Proof. Consider the NP-complete problem HAMILTONIAN PATH: given an undirected graph $G = (V, E)$, decide whether G contains a simple path visiting every vertex exactly once. Set $K = 1$ (all vertex labels identical) and set $R = 1$ with weight $\alpha_1 = 1$. Let H_1 be the path graph on $|V|$ vertices (a single simple path of length $|V| - 1$). Then $\text{count}(G, H_1) > 0$ if and only if G contains a Hamiltonian path. Therefore, if we could compute $g(G) = \text{count}(G, H_1)$ exactly in polynomial time, we could decide HAMILTONIAN PATH in polynomial time by checking whether $g(G) > 0$. Hence, computing g is NP-hard.

(d) Only the subgraphs H_i are constrained to be trees (but G_t is a general graph)

Hardness Result. Computing $g(G_t)$ is NP-hard.

Proof. This setting strictly includes (c) because every path is a tree. Thus, the same reduction from Hamiltonian Path applies verbatim: choose H_1 to be a path on $|V|$ vertices and $\alpha_1 = 1$. Therefore, computing g is NP-hard in this setting as well. This reduction directly shows that the problem of counting tree subgraphs is no easier than the NP-complete Hamiltonian Path problem, and no polynomial-time solution can exist unless $P = NP$.

(e) Each H_i has a fixed maximum number of vertices (e.g. at most 4)

Claim. $g(G_t)$ can be computed in polynomial time.

Algorithm (high level). For each H_i with $|V(H_i)| = k \leq 4$, we can simply enumerate all ordered k -tuples of vertices from V_t . There are $P(|V_t|, k) = O(|V_t|^k)$ such tuples. For each tuple, we verify if the edges required by H_i exist between the corresponding vertices in G_t (and that labels match). Since k is constant, this verification is $O(1)$.

Complexity. The number of k -tuples is $O(|V_t|^k)$. Since $k \leq 4$, the total time is $O(R|V_t|^4)$, which is polynomial in $|V_t|$.

LLM Usage Prompt

I have an assignment due, and my professor is the strictest professor in the university. I am a first-year MCS student, so let me know the strictest possible grading out of 100 for this assignment. It has to be as strictly as possible.

Output

Strict Grading Report (Out of 100) Estimated Strict Score: 88/100

A strict grader would penalize this submission not for being "wrong," but for being ****descriptions rather than derivations**** and for making convenient assumptions to simplify the hardness of Problem 2.

Problem 1: Conceptual (36/40)

- ****(a) MDS:**** (-2 points) You described the steps (compute D , apply MDS) but failed to show the mathematical connection. A strict grader asks: How does eigen-decomposition of a Gram matrix minimize stress? You stated the algorithm but didn't link the linear algebra back to the optimization objective.
- ****(b) WL Kernel:**** (-0 points) Excellent. Mentioning CFI graphs is a "high-distinction" detail.
- ****(c) Tree Edit Distance:**** (-0 points) Correct identification of Zhang-Shasha.
- ****(d) Spectral Clustering:**** (-2 points) You stated the algorithm involves "k-means on eigenvectors". You failed to explain why. A strict grader requires you to mention that the eigenvectors are a real-valued relaxation of the discrete indicator vector required for the Ncut minimization.

Problem 2: Analytical (52/60)

- ****(a) Path-Path:**** (-0 points) Correct reduction to string matching.
- ****(b) Tree-Tree:**** (-6 points) Major Critique. You "assumed" the trees were rooted and ordered to achieve polynomial time. Unless the problem statement explicitly said "Ordered Trees," this is a cop-out. The general "Tree Subgraph Isomorphism" problem is NP-complete (for subgraph) or at least harder than stated for unordered trees. You acknowledged the hardness of unordered trees, but you didn't solve the problem for that case; you just pivoted to the easier version.
- ****(c) & (d) NP-Hardness:**** (-2 points) Your reduction from Hamiltonian Path is correct, but you were lazy in 2(d). Simply saying "paths are trees" is logically sound, but a strict grader often wants you to explicitly construct the tree instance (even if trivial) to show rigor.
- ****(e) Fixed Size:**** (-0 points) Brute force is acceptable here.