Word Ladder Search in Semantic Embedding Space

Assignment #1 — Introduction to Artificial Intelligence
IBA Karachi · Spring 2026 · Dr. Syed Ali Raza

Task 1: Problem Formulation

1.1 State Representation

Each state in the search problem corresponds to a single word from the GloVe vocabulary (20,000 words). Internally, the word is represented as a 100-dimensional unit-normalised float32 vector. For the purposes of search bookkeeping, the word string itself serves as the state identifier. The state space is therefore a graph where each node is a word and edges connect words that are among each other's k nearest semantic neighbours.

1.2 Initial State and Goal Test

The initial state is the user-provided start word, provided it exists in the vocabulary. The goal test is a simple equality check: the current node equals the goal word. If either word is absent from the vocabulary, the system reports this and terminates gracefully.

1.3 Action Space — Neighbour Definition

From any word w, the available actions move to one of the k = 20 nearest neighbours in cosine-similarity space. Neighbours are computed by dotting the query's unit vector against the entire embedding matrix (a single NumPy operation), then selecting the top-k indices via argpartition. The word itself is excluded. Results are cached per word to avoid repeated computation.

Justification of k = 20: A value of k = 5 produces a graph too sparse for many pairs to have a solution. A value of k = 50 inflates the branching factor dramatically, making BFS and UCS impractical on semantically distant pairs. k = 20 was chosen after empirical testing: it solved all five test pairs while keeping node-expansion counts manageable (BFS expanded at most ~12,000 nodes on the hardest pair).

1.4 Path Cost Function

The edge cost from word a to word b is defined as:

```
cost(a, b) = 1 – cosine_similarity(a, b)
```

This maps high-similarity (semantically close) transitions to low cost and low-similarity transitions to high cost. It is bounded in [0, 2] since cosine similarity lies in [–1, 1]. UCS and A* use this cost to find minimum-cost paths.

1.5 Heuristic Function

The heuristic h(n) estimates the cost from the current word n to the goal g:

```
h(n, g) = 1 – cosine_similarity(n, g)
```

Intuition: if n is already semantically close to g (high cosine similarity), h is small — we are nearly there. Semantically distant words yield h near 1 or 2. The heuristic is fast to evaluate (a single dot product on unit vectors).

Admissibility: The heuristic is NOT guaranteed to be admissible. The true minimum path cost can be less than h(n) when the optimal path takes a shortcut through high-similarity intermediate words that nonetheless lie 'off-axis' from the direct n→g direction. In practice, A* with this heuristic still produces optimal or near-optimal paths much faster than UCS.

Task 2: Search Algorithm Descriptions

Breadth-First Search (BFS)

Uses a FIFO queue. Explores all words at distance d before distance d+1. Guarantees the shortest path in number of hops. Does not use edge costs; all edges are treated as unit cost. Can expand many nodes for distant pairs.

Depth-First Search (DFS)

Uses a LIFO stack with a depth limit of 15 to ensure termination. Explores aggressively down one branch before backtracking. Memory-efficient but not optimal — often returns long paths. The depth limit prevents infinite loops in the implicit graph.

Uniform Cost Search (UCS)

A priority queue ordered by cumulative path cost g(n). Expands the cheapest-so-far node first, guaranteeing optimal cost paths. Uses edge_cost = 1 – cosine_similarity as the step cost. More thorough than BFS; expands more nodes but finds minimum-cost solutions.

Greedy Best-First Search

A priority queue ordered purely by heuristic h(n) = 1 – cosine_similarity(n, goal). Expands the node that looks closest to the goal. Very fast (few nodes expanded) but can get trapped by local minima and produce suboptimal or very long paths. Shown to fail on the 'replace → shoves' pair, producing a 6-step path vs the optimal 4 steps.

A* Search

Combines g(n) (actual cost so far) with h(n) (heuristic estimate). f(n) = g(n) + h(n) drives the priority queue. Balances optimality and efficiency. Expanded far fewer nodes than UCS while still finding optimal paths in all tested pairs.

Graph construction is implicit:

Neighbours are computed on-the-fly during node expansion (never pre-built). All algorithms maintain an 'explored' set to avoid revisiting nodes. All algorithms terminate when the goal is found or the frontier is exhausted.

Task 3: Experimental Evaluation

Five word pairs were tested across all five algorithms. The 'whistler → panah' pair is semantically distant (a proper noun from English paired with a word from another language), representing the hard failure-forcing case.

Start	Goal	Algorithm	Found	Steps	Expanded	Time (s)
network	hate	BFS	Yes	3	178	0.0245
network	hate	DFS	Yes	15	3025	0.0639
network	hate	UCS	Yes	3	581	0.0517
network	hate	Greedy	Yes	3	4	0.0002
network	hate	A*	Yes	3	30	0.0005
leather	soar	BFS	Yes	5	1559	0.1508
leather	soar	DFS	Yes	15	4520	0.0373
leather	soar	UCS	Yes	5	5398	0.4954
leather	soar	Greedy	Yes	6	9	0.0003
leather	soar	A*	Yes	5	159	0.0026
replace	shoves	BFS	Yes	4	247	0.0047
replace	shoves	DFS	Yes	15	2203	0.0147
replace	shoves	UCS	Yes	4	3110	0.1937
replace	shoves	Greedy	Yes	6	8	0.0003
replace	shoves	A*	Yes	4	193	0.0026
knave	brutes	BFS	Yes	4	572	0.0122
knave	brutes	DFS	Yes	15	4826	0.0507
knave	brutes	UCS	Yes	4	5598	0.3569
knave	brutes	Greedy	Yes	8	14	0.0005
knave	brutes	A*	Yes	4	639	0.0071
whistler	panah	BFS	Yes	5	11798	0.7073
whistler	panah	DFS	Yes	14	7793	0.0337
whistler	panah	UCS	Yes	5	18169	0.6158
whistler	panah	Greedy	Yes	26	1158	0.0330
whistler	panah	A*	Yes	5	6348	0.0747

Key observation: On all pairs, A* expanded dramatically fewer nodes than BFS or UCS while still finding optimal-length paths. Greedy was fastest by node count on simple pairs but produced suboptimal paths (6 vs 4 steps for 'replace → shoves', 26 vs 5 steps for 'whistler → panah'). DFS always hit the depth limit (15), confirming it rarely finds short paths.

Task 4: Analysis and Discussion

4.1 Uninformed vs Informed Search

Uninformed search (BFS, DFS, UCS) treats all directions equally. BFS expands up to 11,798 nodes on the hardest pair; UCS expands 18,169. In contrast, A* expands only 6,348 nodes on the same pair — a 65% reduction vs BFS and a 3× reduction vs UCS. Greedy is even faster on node count but sacrifices path quality. Informed search is clearly superior when a meaningful heuristic exists.

4.2 Greedy Best-First Failure Example

For 'replace → shoves', Greedy found a 6-step path while BFS and A* found a 4-step path. Greedy got misled: after reaching an intermediate word that is very similar to 'shoves' by cosine similarity, it committed to exploring that neighbourhood greedily, missing a shorter shortcut. This illustrates the classic failure mode: Greedy ignores the actual cost g(n), so it can wander through a longer route.

4.3 Admissibility and Consistency of the Heuristic

Admissibility: h(n) is admissible iff $h(n) \leq h^*(n)$ for all n, where $h^*(n)$ is the true minimum cost to the goal. Our heuristic $h(n) = 1 - \text{sim}(n, \text{goal})$ equals the edge cost of a direct n→goal edge. If goal is a direct neighbour of n, $h(n) = \text{cost}(n, \text{goal}) = h^*(n)$ (tight). But if the optimal path must pass through intermediate nodes, h could overestimate when those intermediate paths happen to be cheaper. Hence admissibility is NOT guaranteed.

Consistency: A heuristic is consistent if $h(n) \leq \text{cost}(n, n') + h(n')$ for all edges (n, n'). Cosine similarity does not satisfy the triangle inequality in general, so consistency is also NOT guaranteed. Despite this, A* performed excellently in practice because the heuristic is a strong directional signal even if not mathematically consistent.

4.4 Misleading Semantic Similarity

A prime example is the pair 'whistler → panah'. 'Whistler' is a Canadian city / painter's name, while 'panah' is a word from Urdu/Persian. Cosine similarity in the GloVe space may place these near culturally or textually co-occurring words (e.g., location names, transliterations) rather than semantically meaningful paths. The Greedy algorithm followed this misleading similarity signal and produced a 26-step path. Another example: words like 'bank' (financial institution vs river bank) are polysemous; their embedding is an average of multiple senses, causing nearest neighbours to mix unrelated concepts.

4.5 Effect of k on Performance

Increasing k raises the branching factor, which increases nodes expanded per step but improves graph connectivity. With k = 5, some pairs have no solution. With k = 50, BFS and UCS become very slow on hard pairs. k = 20 is the empirically chosen sweet spot: all five test pairs were solved, and BFS node counts stayed within reason (<12,000). Informed algorithms (Greedy, A*) are far less sensitive to k than uninformed ones, since they rarely explore the full neighbourhood.

Task 5: GUI Description

The GUI is implemented as a Streamlit application (app.py). To run it:

```
streamlit run app.py
```

The sidebar provides text inputs for start and goal words, a dropdown for algorithm selection (BFS, DFS, UCS, Greedy, A*), a slider for k (5–50), and a depth-limit slider for DFS. Clicking '■ Run Search' executes the chosen algorithm. Results are displayed as four metric cards (found, steps, nodes expanded, time taken), followed by the word ladder as a sequence of highlighted chips, and a per-edge similarity breakdown table. Invalid words trigger a clear error message.

Application Perspective

Semantic search paths have several real-world applications. In **medicine**, a clinician might query 'aspirin → ibuprofen' to find the semantic path of drug terms, revealing intermediate concepts (anti-inflammatory, NSAID, COX-inhibitor) useful for drug repositioning or synonym expansion in medical NLP. In **information retrieval**, query expansion can leverage such paths: a search for 'automobile' could be expanded through the semantic ladder toward 'transportation' or 'vehicle' to capture broader document sets. In **knowledge graph completion**, identifying short semantic paths between entities helps infer missing relations. In **education technology**, a word-ladder path from 'easy' vocabulary to 'advanced' vocabulary could scaffold vocabulary learning, gradually exposing learners to semantically adjacent words.