INTRO TO AI ASSIGNMENT 1

REPORT

ALIZA SAMREEN AGHA (31562)

Task 1: Problem Formulation

- **State Representation:**

A state is defined as a single word from the restricted 20,000-word vocabulary of the pretrained GloVe 100-dimensional embeddings

Each word is represented internally as:

- A string
- A 100-dimensional normalized embedding vector

Formally:

$$\text{State} = w \in V$$

where V is the vocabulary of size 20,000.

The embedding vectors are normalized to unit length to enable efficient cosine similarity computation using dot products.

- **Initial State and Goal Test:**

The initial state is the start word entered by the user.

$$s_0 = \text{start\_word}$$

If the word is not present in the vocabulary, the system terminates gracefully.

The goal test checks whether the current state equals the goal word.

$$\text{GoalTest}(s) = (s = \text{goal\_word})$$

The search terminates successfully when the goal word is reached.

- **Action Space:**

Action space is the available actions from a given word that correspond to moving to its top-k nearest neighbors in embedding space, measured using cosine similarity.

Cosine similarity between two normalized vectors $u$and v:

$$\text{cosine}(u, v) = u \cdot v$$

The neighbor set is defined as:

$$\text{Neighbors}(w) = \text{Top-k words with highest cosine similarity to } w$$

The word itself is excluded.

- **How many neighbors (*k*) are considered at each step?**

K = 15

<u>Justification</u>:

If k is too small, the graph becomes sparsely connected and search may fail to find a path. If k is too large, the branching factor increases significantly, causing exponential growth in search complexity. Therefore, k = 15 provides a reasonable balance between connectivity and computational efficiency.

The branching factor is approximately equal to k.This directly affects time complexity of BFS: $O(b^d)$

- **Path Cost Function Uniform cost or similarity-based cost?**

A similarity-based cost function is used:

$$Cost(w_1, w_2) = 1 - cos(w_1, w_2)$$

The total path cost is the sum of these step costs.

cumulative path cost:

$$g(n) = \sum(1 - \text{cosine})$$

Intuition:

- Highly similar words → cosine ≈ 1 → cost ≈ 0
- Dissimilar words → cosine small → cost large

  This ensures that semantically similar transitions have low cost, while larger semantic jumps incur higher cost.

- **Heuristic Function:**

The heuristic used for Greedy Best-First Search and A* is:

$$h(n) = 1 - \text{cosine}(n, \text{goal})$$

If a word is very similar to the goal, the heuristic value approaches zero. If it is semantically distant, the heuristic becomes larger. Thus, the heuristic estimates semantic distance in embedding space.

Although this heuristic works well empirically, cosine similarity does not strictly satisfy all metric properties such as triangle inequality in embedding space. Therefore, it is not guaranteed to be strictly admissible or consistent, though it behaves reasonably well in practice.

Task 2: Search algorithm implementation

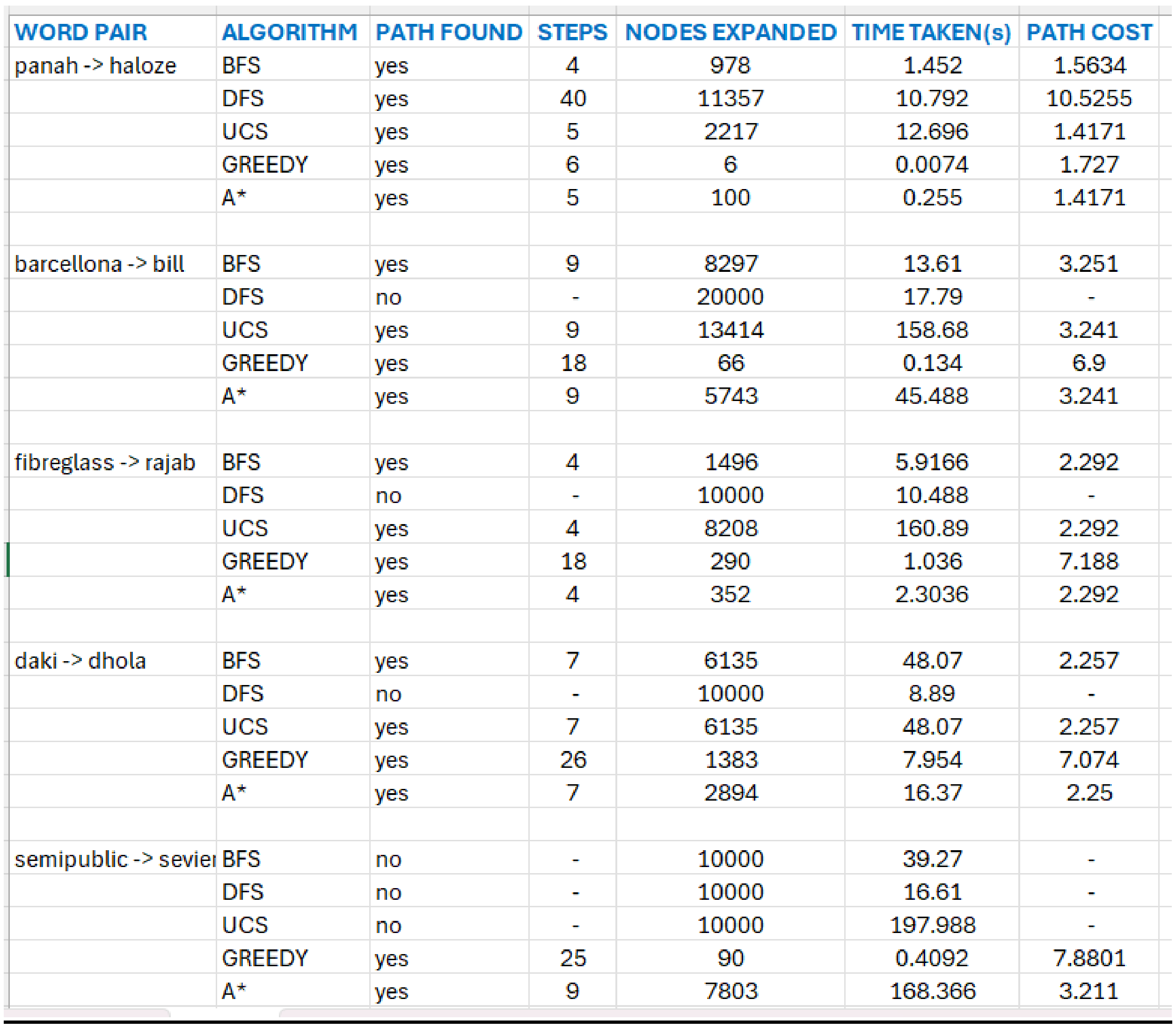
Code is provided

The search graph is constructed implicitly. Neighboring states are generated dynamically using cosine similarity rather than precomputing the entire graph.

The actions() function computes top-k neighbors.
The result() function returns the selected neighbor.
The path_cost() function accumulates similarity-based cost.
The h() function defines the heuristic for informed search.

Repeated states are avoided using graph search implementations provided in the framework.

Task 3: Experimental evaluation



WORD PAIR	ALGORITHM	PATH FOUND	STEPS	NODES EXPANDED	TIME TAKEN(s)	PATH COST
panah -> haloze	BFS	yes	4	978	1.452	1.5634
	DFS	yes	40	11357	10.792	10.5255
	UCS	yes	5	2217	12.696	1.4171
	GREEDY	yes	6	6	0.0074	1.727
	A*	yes	5	100	0.255	1.4171
barcellona -> bill	BFS	yes	9	8297	13.61	3.251
	DFS	no	-	20000	17.79	-
	UCS	yes	9	13414	158.68	3.241
	GREEDY	yes	18	66	0.134	6.9
	A*	yes	9	5743	45.488	3.241
fibreglass -> rajab	BFS	yes	4	1496	5.9166	2.292
	DFS	no	-	10000	10.488	-
	UCS	yes	4	8208	160.89	2.292
	GREEDY	yes	18	290	1.036	7.188
	A*	yes	4	352	2.3036	2.292
daki -> dhola	BFS	yes	7	6135	48.07	2.257
	DFS	no	-	10000	8.89	-
	UCS	yes	7	6135	48.07	2.257
	GREEDY	yes	26	1383	7.954	7.074
	A*	yes	7	2894	16.37	2.25
semipublic -> sevie	BFS	no	-	10000	39.27	-
	DFS	no	-	10000	16.61	-
	UCS	no	-	10000	197.988	-
	GREEDY	yes	25	90	0.4092	7.8801
	A*	yes	9	7803	168.366	3.211

Task 4: Analysis and discussion

- **Comparison of Uninformed and Informed Search**

Across all five word pairs, informed search algorithms (Greedy and A*) consistently required fewer node expansions compared to uninformed algorithms (BFS, DFS, UCS).

Example 1: *panah → haloze*

Algorithm	Nodes Expanded	Time (s)
BFS	978	1.452
DFS	11357	10.792
UCS	2217	12.696
Greedy	6	0.0074
A*	100	0.255

DFS expanded over 11,000 nodes for a 4-step solution, showing deep and inefficient exploration. In contrast, Greedy expanded only 6 nodes, and A* expanded only 100 nodes. This clearly demonstrates the effectiveness of heuristic guidance.

Example 2: *barcellona → bill*

Algorithm	Nodes Expanded	Path Cost
BFS	8297	3.251
UCS	13414	3.241
Greedy	66	6.9
A*	5743	3.241

Here, DFS failed after reaching the 20,000-node limit.

Although Greedy expanded only 66 nodes, it produced a path with cost 6.9, which is more than double the optimal cost (3.241). UCS and A* found the optimal path cost, but UCS required 13,414 expansions compared to A*'s 5,743.

This shows that Greedy is extremely fast but may sacrifice solution quality. A* reduces exploration compared to UCS while maintaining optimality. DFS is unstable in large semantic spaces.

Example 3: *fibreglass → rajab*

Algorithm	Nodes Expanded
BFS	1496
UCS	8208
Greedy	290
A*	352

DFS failed again.

UCS required 8,208 expansions, whereas A* required only 352. Greedy expanded fewer nodes (290) but produced a significantly worse path cost (7.188 vs optimal 2.292).

This clearly distinguishes the trade-off between speed and optimality.

- **Failure Case: Semantically Distant Pair**

For *semipublic → seviel*:

Algorithm	Result	Nodes Expanded
BFS	No	10000
DFS	No	10000
UCS	No	10000
Greedy	Yes	90
A*	Yes	7803

Uninformed algorithms exhausted the maximum expansion limit and failed to find a path. However, Greedy and A* successfully found solutions.

Greedy required only 90 expansions but produced a long path (25 steps) with cost 7.7801. A* found a shorter path (9 steps) with cost 3.211 but required 7,803 expansions.

This demonstrates that:

- Heuristic guidance can enable solutions where uninformed search fails.
- However, Greedy may prioritize heuristic closeness at the expense of path quality.
- A* is more reliable in producing lower-cost solutions.

- Example Where Greedy Fails

The clearest example is *barcellona → bill*:

- Greedy: 18 steps, cost = 6.9
- UCS/A*: 9 steps, cost ≈ 3.241

Greedy focuses solely on minimizing the heuristic (h(n)) and ignores accumulated path cost (g(n)). As a result, it may choose locally promising nodes that lead to globally inefficient paths.

This confirms that Greedy Best-First Search is not optimal.

- Admissibility and Consistency of Cosine Heuristic

The heuristic used is:

$$h(n) = 1 - \cos(n, goal)$$

Cosine similarity operates in high-dimensional embedding space, which is not guaranteed to satisfy strict metric properties such as triangle inequality.

Therefore:

- The heuristic is not theoretically guaranteed to be admissible.
- It is not guaranteed to be consistent.

However, in most successful cases (panah → haloze, barcellona → bill, fibreglass → rajab, daki → dhola), A* produced the same optimal cost as UCS. This suggests that the heuristic behaves well empirically, even if not formally proven optimal.

- Effect of k (Number of Neighbors)

The implementation uses k = 15.

The branching factor is approximately equal to k. As k increases:

- Connectivity improves.
- Probability of finding a path increases.
- Branching factor increases exponentially.
- BFS and UCS become extremely expensive.

This is evident from high node expansions in BFS and UCS across multiple pairs.

When k is moderate (15), the graph remains connected while keeping search manageable for A*. However, uninformed methods still struggle due to exponential growth in exploration.

- **Conclusion**

The experimental results clearly show that:

- DFS is unstable and frequently fails.
- BFS guarantees shortest-step solutions but expands many unnecessary nodes.
- UCS guarantees optimal cost but is computationally expensive.
- Greedy is extremely fast but often produces poor-quality paths.
- A* provides the best balance between efficiency and optimality.

APPLICATION PERSPECTIVE:

Semantic search paths in embedding space have meaningful applications in domains such as medicine and information retrieval. In medicine, such a system could help uncover conceptual relationships between diseases, symptoms, and treatments by identifying intermediate terms that connect two medical concepts. For example, a semantic path between two conditions may reveal related physiological processes or complications, assisting exploratory research and knowledge discovery. This approach goes beyond simple keyword matching by capturing contextual similarity between terms.

In information retrieval, semantic search paths can enhance query expansion and recommendation systems. Instead of relying solely on exact keyword overlap, embedding-based search can connect related concepts through intermediate semantic transitions. This allows search engines and academic databases to retrieve more relevant results, even when the query and documents use different terminology. By transforming embedding space into a navigable structure, semantic search enables more intelligent exploration of large textual datasets.

Although promising, such systems rely on statistical similarity rather than verified domain knowledge, and therefore should complement, not replace, expert judgment.