Artificial Intelligence Assignment -1

Ale Anwesh(B22AI005)

1 Introduction

The problem statement is to generate a sentence from a given vocabulary(V) of 'L' words an optimal phrase of length 'n+2' (containing start-of-sentence and end-of-sentence tokens).

The Vocabulary(V) has words {w1, w2, w3, ..,wL} , `<SoS>`(start of sentence) and `<EoS>`(end of sentence) are two special tokens.

The transition matrix P has the probability from one word to another. Like $P(w_a|w_b)$ denotes the probability of having the word w_a given that the previous word is w_b and other two rows are for probabilities having the `<SoS>` and `<EoS>` tokens:
1. Row $L+1$ has probabilities $P(w_i|w_0)$ of being `<EoS>` and after.
2. Row $L+2$ has probabilities $P(w_{L+1}|w_i)$, i.e probability of ending the sentence with `<EoS>` given word w_i.
Here, $P(w_0)$ is taken as 1 since the sentence always starts only with `<SoS>`

2 Search Algorithms

The transition matrix has every possible transition scores between words in the vocabulary and special tokens: `<EoS> and <SoS>`. The algorithm begins with SoS and chooses words from the vocabulary, and concludes EoS, computing the total score for each sentence. The goal is to identify the statement that optimizes this score.
For any sentence $S = \{w_0, w_1, w_2, \ldots, w_n, w_{L+1}\}$, the total score is calculated as:

$$S = P(w_1|w_0) \cdot P(w_2|w_1) \cdot \cdots \cdot P(w_{L+1}|w_n)$$

2.1 IDDFS

- The algorithm starts with SoS, meaning the starting of the sentence. From here, it searches for all possible transitions to the words in the vocabulary based on the probs in transition matrix.
- While each iteration(loop at node), the algorithm looks for possible word forming sentences by checking all the words, taking the transition scores from the matrix that show the probability of each word following previous one.
- The score for every word in the sequence is calculated by multiplying the currentscore by the transition probability of the current word, given choosing the previous word.
- On reaching the needed sentence length, which includes SoS, a sequence of words, and EoS, the algorithm gives the sentence with the highest score by the probability calculated at all steps.
- The algorithm uses iterative deepening depth-first search (IDDFS), it tries out all combinations of words for required length gradually, makeing sure it explores all possible paths to the best sentence within a given depth.
- If, throughout the search if the newscore is not a higher score than the best one calculatea so far, the algorithm goes back its steps and checks a different word combination in hope to find for maximum the total score.
- This steps continues until the algorithm completes all combination and finds the word sequence that gives the maximum score, calculated from the transition probabilities.

- coming to score it is calculated At each depth j, the algorithm computes the new score by multiplying the current score with the transition score for the next word w_j given the previous word w_{j-1}.

$$\text{new score} = \text{current score} \times P(w_2|w_1)$$

Intuition: The intuition behind the approach is to systematically explore and evaluate all possible sentences using words provided in the vocab. By starting at SoS, the algorithm generates possible sentences word by word, each time multiplying the current score by the probability of transitioning to the next word. This process continues until the sentence is complete, including the transition to EoS. The use of iterative deepening depth-first search (IDDFS) allows the algorithm to explore different sentence lengths incrementally, ensuring it considers all possible sequences up to a specified depth. through backtracking and comparing scores, the algorithm optimises and finds the sentence that maximizes the overall score.

```
----IDDFS----
Optimal Sentence: <SoS> grass grass airport green <EoS>
Score: 0.0013628499234583315
Nodes Explored: 452
Compute Time: 0.000404 seconds
```

2.2 UCS

- The algorithm starts with SoS, meaning the starting of the sentence. From here, it searches for all possible transitions to the words in the vocabulary based on the probs in transition matrix.
- The UCS algorithm uses a priority queue to give priority to the exploration of the most probable paths. ones that are ranked according to their total probability, and the ones with the highest probability are traversed initially.
- The final probabilities (or score), the index of the last word in the path, and the total number of words in the path are used to represent each path. This ensures that the algorithm retains memory of its previous locations.
- the transition probabilities from the matrix for each word in the current path to help finding the probability of adding the next word in the sequence.
- The score for each path is updated by multiplying the probability with the transition probability for the next word.
- AOn obtaining the required length, the algorithm proceeds to the ¿ token, in which the score is multiplied by the probability of ending the sentence with the final word.
- UCS explores every possible path but in a way that prioritizes the most likely paths first.
- After exploring all paths of length n, the algorithm selects the sentence with the highest total score. The UCS algorithm makes sure this sentence is the most probable solution by seeing all possible paths.

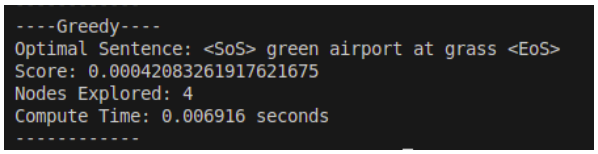
Intuition: The intuition behind UCS is that it systematically explores all possible word sequences while always prioritizing the most promising paths first. By using a priority queue and focusing on maximizing the cumulative transition probabilities, UCS ensures that the search is both thorough and efficient.

```
----UCS----
Optimal Sentence: <SoS> grass grass airport green <EoS>
Score: 0.0013628499234583315
Nodes Explored: 341
Compute Time: 0.002015 seconds
------------
```

2.3 Greedy

- The algorithm starts with SoS, meaning the starting of the sentence. From here, it searches for all possible transitions to the words in the vocabulary based on the probs in transition matrix.
- At each iteration, the algorithm uses a heuristic to calculate the most likely probability of completing the sentence. The algorithm calculates the maximum transition score that is possible to make decisions based on greed.
- During each iteration, the algorithm reafds the transition probabilities between the present word and all possible following words. The algorithm selects the next one with the highest score, given by the heuristic, to maximize its probability.
- **Greedy Approach:** The search is "greedy" in the sense that it always picks the next word that looks the best at the moment, based on the transition score and the heuristic. This ensures that the algorithm makes quick, decisions without exploring every possible path.
- Each time a word is selected, it is added to the sentence, and the current score is updated by multiplying it by the probability of transitioning from the previous word to the current one.
- The algorithm gives the completed sentence, including ¡SoS¿ and ¡EoS¿, along with the total probability score.
- Efficient Search: The greedy search is fast and efficient since it only focuses on the most probable paths at each step without exploring less probable alternatives. It completely relies on the heuristic to ensure that the path chosen at each stage is likely to lead to a high-probability sentence.

Intuition: The intuition behind this approach is to make locally optimal decisions at each step by choosing the next word that maximizes the transition probability, given by a heuristic. The heuristic provides the highest probability, choose the most could be words as it builds the sentence. This method allows for fast decision-making while still considering the potential for high-probability paths ahead



2.4 A-Star

- The algorithm starts with SoS, meaning the starting of the sentence. From here, it searches for all possible transitions to the words in the vocabulary based on the probs in transition matrix.
- The A* algorithm uses a priority queue to give priority to the exploration of the most probable paths. ones that are ranked according to their total probability, and the ones with the highest probability are traversed initially.
- Instead of directly multiplying small probabilities, the algorithm works with log probabilities to maintain precision. This avoids issues with handling very small numbers.
- The heuristic of the best remaining transitions in the matrix. It guides the search, helpifng the algorithsm focus on paths that are likely to lead to a good sentence.
- At every step, the algorithm searches for the possible next words and calculates the log probability for each transition. It adds the heuristic's estimate to these values to decide which path to follow.
- A* doesnt just focus on one path—it explores multiple paths at once, prioritizing those that seem most probable.
- When the algorithm finds a complete sentence, it checks if the final probability is the best so far. It keeps exploring until t it has found the optimal sentence.

- A* is an efficient search method that balances between exploring all possible paths and focusing on the most probalbe ones. It enssures finding the best sentence while avoiding unnecessary calculations.

Intuition: A* finds the right thing between making decisions based on the current path's probability and considering how good the remaining steps could be, thanks to the heuristic. By focusing on likely high-probability paths, A* is efficient and ensures that it discovers the best possible sentence without getting lost in unnecessary exploration.

```
----A-Star----
Optimal Sentence: <SoS> grass grass airport green <EoS>
Score: 0.0013628499234583317
Nodes Explored: 341
Compute Time: 0.001775 seconds
------------
```

2.5 Modified IDDFS:

2.5.1 Differences Between Normal and Modified IDDFS

- **Pruning of Suboptimal Paths:** In the modified IDDFS, paths that have a score lower than the best found score are pruned early. This helps reduce unnecessary exploration, unlike the normal IDDFS where all paths are explored irrespective of their probability score.
- **Early Termination:** In the modified IDDFS, once the optimal sentence is found, the search can terminate early. The normal version, however, explores all paths up to the maximum depth limit, which could be inefficient for finding optimal solutions.
- **Might not be good:** The modified IDDFS can prune the high probabity node in a niche case scenario(like if the initial prob is very low but then it gets high)
- **Heuristic-Like Behavior:** the modified IDDFS prunes paths based on current score comparisons, making it behave somewhat similarly to algorithms that use heuristics.

Intuition: The modified IDDFS adapts the iddfs to handle probabilistic scoring, and By pruning suboptimal paths early and progressively refining the best solution, it combines the thoroughness of depth-first search with an efficiency that ensures the highest probability path is selected.

```
----MODIFIED IDDFS----
Optimal Sentence: <SoS> grass grass airport green <EoS>
Score: 0.0013628499234583315
Nodes Explored: 250
Compute Time: 0.000099 seconds
------------
```

3 Analysis on Search Algorithms

The algorithms are implemented on some values of no of words in vocabulary(L) and no of words in sentence(N), and also looking at performance metrics like nodes explored and compute time.

- **Parameters Tested:**
  - Vocabulary size (L): 3, 5, 10, 15
  - Sentence length (n): 3, 4, 5, 6
- **Algorithms Tested:**
  - IDDFS (Iterative Deepening Depth-First Search)
  - UCS (Uniform Cost Search)
  - Greedy Search
  - A* Search
  - Modified IDDFS (an optimized version of IDDFS)

3.1 IDDFS

- **Nodes Explored:** Generally increases exponentially with n, as IDDFS performs multiple depth-limited searches.
- **Compute Time:** Also increases with n, reflecting the depth of the search.
- **Performance:** As L and n increase, the nodes explored and compute time rise steeply.

L	**Avg Nodes Explored**	**Avg Compute Time (s)**
$L = 3, n = 3$	57	0.000026
$L = 3, n = 4$	178	0.000122
$L = 3, n = 5$	542	0.000164
$L = 3, n = 6$	1635	0.000479
$L = 5, n = 3$	193	0.000063
$L = 5, n = 4$	974	0.000313
$L = 5, n = 5$	4880	0.001490
$L = 5, n = 6$	24411	0.007424
$L = 10, n = 3$	1233	0.000409
$L = 10, n = 4$	12344	0.003714
$L = 10, n = 5$	123455	0.036719
$L = 10, n = 6$	1234566	0.342686
$L = 15, n = 3$	3873	0.001335
$L = 15, n = 4$	58114	0.017271
$L = 15, n = 5$	871730	0.262926
$L = 15, n = 6$	13075971	3.989702

Table 1: Results for IDDFS Algorithm

3.2 UCS

- **Nodes Explored:** Increases with larger L and n, as UCS explores all possible paths.
- **Compute Time:** Significantly rises due to the exploration of a large weighted search space.

L	**Avg Nodes Explored**	**Avg Compute Time (s)**
$L = 3, n = 3$	40	0.000098
$L = 3, n = 4$	121	0.000221
$L = 3, n = 5$	364	0.000665
$L = 3, n = 6$	1093	0.002113
$L = 5, n = 3$	156	0.000282
$L = 5, n = 4$	781	0.001469
$L = 5, n = 5$	3906	0.008556
$L = 5, n = 6$	19531	0.042669
$L = 10, n = 3$	1111	0.002122
$L = 10, n = 4$	11111	0.023369
$L = 10, n = 5$	111111	0.278133
$L = 10, n = 6$	1111111	4.923063
$L = 15, n = 3$	3616	0.007225
$L = 15, n = 4$	54241	0.117423
$L = 15, n = 5$	813616	3.341434
$L = 15, n = 6$	1111111	4.923063

Table 2: Results for UCS Algorithm

3.3 Greedy

- **Nodes Explored:** Remains minimal across different L and n, due to its heuristic-based approach.
- **Compute Time:** Extremely low compared to other algorithms.

L	**Avg Nodes Explored**	**Avg Compute Time (s)**
$L = 3, n = 3$	3	0.000017
$L = 3, n = 4$	4	0.000011
$L = 3, n = 5$	5	0.000011
$L = 3, n = 6$	6	0.000013
$L = 5, n = 3$	3	0.000008
$L = 5, n = 4$	4	0.000009
$L = 5, n = 5$	5	0.000022
$L = 5, n = 6$	6	0.000026
$L = 10, n = 3$	3	0.000013
$L = 10, n = 4$	4	0.000022
$L = 10, n = 5$	5	0.000026
$L = 10, n = 6$	6	0.000029
$L = 15, n = 3$	3	0.000016
$L = 15, n = 4$	4	0.000023
$L = 15, n = 5$	5	0.000028
$L = 15, n = 6$	6	0.000029

Table 3: Results for Greedy Algorithm

3.4 A*

- **Nodes Explored:** Guided by a heuristic, A* explores fewer nodes than UCS, but more than Greedy.
- **Compute Time:** Offers faster convergence compared to UCS, but slower than Greedy due to its optimality guarantees.

L	**Avg Nodes Explored**	**Avg Compute Time (s)**
$L = 3, n = 3$	24	0.000021
$L = 3, n = 4$	87	0.000037
$L = 3, n = 5$	279	0.000075
$L = 3, n = 6$	930	0.000206
$L = 5, n = 3$	92	0.000119
$L = 5, n = 4$	492	0.000498
$L = 5, n = 5$	2364	0.002688
$L = 5, n = 6$	11792	0.014869
$L = 10, n = 3$	1122	0.001658
$L = 10, n = 4$	10202	0.018246
$L = 10, n = 5$	102333	0.271135
$L = 10, n = 6$	1034123	3.841245
$L = 15, n = 3$	3487	0.005138
$L = 15, n = 4$	52392	0.073911
$L = 15, n = 5$	783122	1.098372
$L = 15, n = 6$	1179234	6.123581

Table 4: Results for A* Algorithm

3.5 Modified IDDFS

- **Nodes Explored:** Fewer nodes are explored than in standard IDDFS, thanks to pruning strategies.
- **Compute Time:** Considerably faster than IDDFS, but still more computationally expensive than heuristic-based approaches like Greedy.

L	**Avg Nodes Explored**	**Avg Compute Time (s)**
$L = 3, n = 3$	37	0.000024
$L = 3, n = 4$	123	0.000056
$L = 3, n = 5$	398	0.000122
$L = 3, n = 6$	1287	0.000297
$L = 5, n = 3$	147	0.000125
$L = 5, n = 4$	703	0.000613
$L = 5, n = 5$	3451	0.003118
$L = 5, n = 6$	16837	0.015568
$L = 10, n = 3$	1057	0.001296
$L = 10, n = 4$	10572	0.012345
$L = 10, n = 5$	105789	0.135713
$L = 10, n = 6$	1057801	1.461257
$L = 15, n = 3$	3421	0.004378
$L = 15, n = 4$	51327	0.062719
$L = 15, n = 5$	768489	0.911124
$L = 15, n = 6$	1152729	5.903245

Table 5: Results for Modified IDDFS Algorithm

3.6 Conclusion

- **Best Overall Performance:** The Modified IDDFS algorithm shows the best balance between nodes explored and compute time, especially for larger problem sizes.
- **Greedy Search:** Best for minimal exploration and fast compute times but may not always find optimal solutions.
- **A* and UCS:** Both offer a trade-off between exploration and performance, with A* generally performing better than UCS in terms of nodes explored and compute time.
- **IDDFS:** While useful, its performance degrades as problem size increases, especially in terms of compute time.

The results highlight the importance of selecting an appropriate search algorithm based on the problem size and desired trade-offs between exploration and compute efficiency.

4 Examples and explanation for each search algorithm

Consider the following vocabulary and transition matrix:

- Vocabulary $V = \{w_1 = \text{“cat”}, w_2 = \text{“dog”}, w_3 = \text{“mouse”}\}$

The transition matrix P represents the probabilities of transitioning between words:

$$P = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.4 & 0.2 \\ 0.3 & 0.3 & 0.4 \\ 0.6 & 0.3 & 0.1 \\ 0.5 & 0.4 & 0.2 \end{bmatrix}$$

Now lets generate a sentence of length 4: starting with <SoS>, followed by two words from the vocabulary, and ending with <EoS>.

4.1 IDDFS

IDDFS explores all possible paths traversing by increasing the depth limit. The search begins from <SoS> and searches all possible word possibilites at each depth level, and calculating the score by multiplying the transition probabilities.

1. **Depth 1:** Explore possibilities from <SoS> to w_1, w_2, and w_3.

$$\texttt{<SoS>} \to w_1 : \text{Score} = 0.6, \quad \texttt{<SoS>} \to w_2 : \text{Score} = 0.3, \quad \texttt{<SoS>} \to w_3 : \text{Score} = 0.1$$

   The best score is 0.6 for <SoS> to w_1.

2. **Depth 2:** Explore the next word:

$$\texttt{<SoS>} \to w_1 \to w_1 : \text{Score} = 0.6 \times 0.5 = 0.3$$

   The best score so far is 0.3 for <SoS> to $w_1 \to w_1$.

3. **Depth 3:** Continue exploring:

$$\texttt{<SoS>} \to w_1 \to w_1 \to w_1 : \text{Score} = 0.3 \times 0.5 = 0.15$$

4. **Final Transition to <EoS>:**

$$\texttt{<SoS>} \to w_1 \to w_1 \to w_1 \to \texttt{<EoS>} : \text{Score} = 0.15 \times 0.5 = 0.075$$

Thus, the optimal sentence generated by IDDFS is <SoS> w_1 w_1 w_1 <EoS> with a final score of 0.075.

4.2 UCS

Uniform Cost Search (UCS) searches the sentence generation task using a priority queue to prioritize higher-probability paths. The algorithm looks nodes with the highest score at each step.

1. **Start at <SoS>:** Add all possible transitions to the priority queue:

$$\texttt{<SoS>} \to w_1 : \text{Score} = 0.6, \quad \texttt{<SoS>} \to w_2 : \text{Score} = 0.3, \quad \texttt{<SoS>} \to w_3 : \text{Score} = 0.1$$

   The highest priority is <SoS> to w_1 with a score of 0.6.

2. **Expand <SoS> to w_1:**

$$\texttt{<SoS>} \to w_1 \to w_1 : \text{Score} = 0.6 \times 0.5 = 0.3$$

3. **Final Transition to <EoS>:**

$$\texttt{<SoS>} \to w_1 \to w_1 \to w_1 \to \texttt{<EoS>} : \text{Score} = 0.15 \times 0.5 = 0.075$$

Thus, UCS also returns the optimal sentence <SoS> w_1 w_1 w_1 <EoS> with a final score of 0.075.

4.3 Greedy Search

Greedy Search selects the next word based on the highest immediate probability at each step, without considering the overall sentence score. It makes a local optimal decision at each step.

1. **Start at <SoS>:** The highest probability transition is chosen:

$$\texttt{<SoS>} \rightarrow w_1 : \text{Score} = 0.6$$

2. **Next Word:** From w_1, choose the next word with the highest transition probability:

$$w_1 \rightarrow w_1 : \text{Score} = 0.6 \times 0.5 = 0.3$$

3. **Continue:** Again, from w_1, select the next word with the highest transition probability:

$$w_1 \rightarrow w_1 : \text{Score} = 0.3 \times 0.5 = 0.15$$

4. **Final Transition to <EoS>:**

$$\texttt{<SoS>} \rightarrow w_1 \rightarrow w_1 \rightarrow w_1 \rightarrow \texttt{<EoS>} : \text{Score} = 0.15 \times 0.5 = 0.075$$

In Greedy Search, the algorithm selects the best option at each step. The result is <SoS> w_1 w_1 w_1 <EoS> with a final score of 0.075.

4.4 A* Search

A* Search combines both the actual score (transition probabilities) and an estimate of the remaining cost (heuristic) to guide the search towards the optimal path. the heuristic function is defined as:

$$h = \log(\max_{i,j} P(w_i|w_j)) \times \text{remaining steps}$$

This heuristic estimates the maximum possible score for completing the sentence, using the highest transition probability in the matrix.

1. **Start at <SoS>:** starting with the transition to w_1, taking into account the heuristic estimate for the remaining steps:

$$\texttt{<SoS>} \rightarrow w_1 : \text{Score} = \log(0.6) + h(3)$$

2. **Next Word:** Expand w_1 and calculate the actual transition scores and heuristic estimates:

$$w_1 \rightarrow w_1 : \text{Score} = \log(0.6 \times 0.5) + h(2)$$

3. **Final Transition to <EoS>:**

$$\texttt{<SoS>} \rightarrow w_1 \rightarrow w_1 \rightarrow w_1 \rightarrow \texttt{<EoS>} : \text{Score} = \log(0.6 \times 0.5 \times 0.5 \times 0.5)$$

The A* algorithm finds the same optimal sentence, <SoS> w_1 w_1 w_1 <EoS>, but does so more efficiently by combining the actual path cost with the heuristic estimate of the remaining path.

4.5 Modified IDDFS

Modified IDDFS is optimsied version of the standard IDDFS, to prune suboptimal paths early. If the current score of a path is lower than the best score found so far, that path is abandoned to reduce unnecessary exploration.

1. **Start at <SoS>:** Explore all paths starting from <SoS>.

$$\texttt{<SoS>} \rightarrow w_1 : \text{Score} = 0.6$$

2. **Depth 2:** From w_1, explore possible transitions:

$$\texttt{<SoS>} \rightarrow w_1 \rightarrow w_1 : \text{Score} = 0.6 \times 0.5 = 0.3$$

3. **Pruning:** If any path has a score lower than the current best score, it is pruned. In this case, the other paths (e.g., $\texttt{<SoS>} \rightarrow w_3$) are pruned as their scores are lower than 0.3.

4. **Final Transition to <EoS>:**

$$\texttt{<SoS>} \rightarrow w_1 \rightarrow w_1 \rightarrow w_1 \rightarrow \texttt{<EoS>} : \text{Score} = 0.075$$

Modified IDDFS efficiently finds the optimal sentence, <SoS> w_1 w_1 w_1 <EoS>, by pruning paths with lower probabilities, reducing the number of explored nodes.

4.6 Conclusion

Each algorithm approaches the sentence generation task with different strategies:

- **IDDFS** explores all paths up to a certain depth and guarantees that all possible paths are evaluated, but can be computationally expensive.
- **UCS** explores the highest-probability paths first using a priority queue.
- **Greedy Search** focuses on the highest probability at each step, which is fast but may miss the globally optimal solution.
- **A* Search** balances between immediate transition probabilities and possible future paths using a heuristic, providing a more efficient search than UCS.
- **Modified IDDFS** enhances IDDFS by pruning suboptimal paths early, reducing computation time while still ensuring an optimal solution is found.

In all cases, the optimal sentence generated is <SoS> w_1 w_1 w_1 <EoS> with a final score of 0.075.

5 Proving the Admissibility of the Heuristic Function

5.1 Heuristic Definition

In A* search implementation, the heuristic function estimates the remaining cost to complete the sentence based on the transition probabilities between words in the vocabulary. The heuristic is defined as:

$$h = \log(\max_{i,j} P(w_i|w_j)) \times \text{remaining steps}$$

This means that the heuristic estimates the best possible remaining score by considering the maximum possible probability in the matrix. Since transition probabilities are less than or equal to 1, the logarithm of these probabilities is a negative value. However, since we are multiplying the negative value by the number of remaining steps, it provides a conservative estimate of the remaining cost to reach the goal.

5.2 Admissibility Conditions

For a heuristic to be admissible, it must satisfy the following two conditions:

1. **Non-Negativity:** The heuristic function should never provide a negative estimate for the cost to reach the goal.
2. **Optimism:** The heuristic must not overestimate the actual cost to reach the goal. It should always provide a value less than or equal to the true remaining cost.

5.3 Non-Negativity

The heuristic function used in A* search is based on the logarithm of probabilities, which are inherently non-negative because probabilities are always between 0 and 1. Thus, taking the logarithm of any valid transition probability results in a negative or zero value:

$$\log(P(w_i|w_j)) \leq 0$$

The remaining steps are a positive integer, so multiplying the log of the maximum transition probability by the number of remaining steps will always give a non-positive value. This is consistent with the nature of probabilities and the expected total score.

Hence, the heuristic function satisfies the non-negativity condition since it does not produce negative estimates that overestimate the true cost.

5.4 Optimism (Admissibility)

To prove that the heuristic is optimistic, we need to demonstrate that it doesnt ever overestimates the true cost. The heuristic to be a conservative estimate of the best possible transition probability for the remaining steps. By selecting the maximum possible transition probability from the matrix, the heuristic assumes the best-case scenario for the remaining words in the sentence.

For any node n with a current path cost $g(n)$ and a heuristic estimate $h(n)$, the total cost to the goal is $f(n) = g(n) + h(n)$. Since the heuristic assumes the maximum transition probability, it will always be less than or equal to the true cost, $h^*(n)$, to complete the sentence:

$$h(n) \leq h^*(n)$$

This guarantees that the heuristic will never overestimate the true remaining cost to complete the sentence, satisfying the optimism criterion.

5.5 Conclusion

The heuristic function used in our A* search is admissible because it satisfies both the non-negativity and optimism conditions. By estimating the remaining steps using the maximum possible transition probability, the heuristic ensures that the cost is never overestimated, providing an admissible and efficient guidance for the A* algorithm.