Final Report

Report

Work Done: All tasks have been attempted.

Disclosures

1. I thank Prof. Kshitij Gajjar (IIIT-H), who helped me understand the intuition behind the proof of [Theorem 1].
2. I would like to thank my fellow student Manit Roy, who graciously provided me with an OpenAI API key for experimentation.
3. Used GitHub Copilot (with GPT-4o) to assist in implementing functions (but not in defining the architecture).

Assumptions

- For the creation of the dataset, publicly available problems (even with simple heuristics) must be used only to a very limited extent.

Defining the Problem

Given a string `s` and a set of transitions `w -> w'` with exactly one transition of the form `w -> empty`, when a transition is applied, the first occurrence of `w` as a substring of `s` is replaced with `w'`. The goal is to determine the order in which these transitions should be applied (with repetition allowed) so that the final string is empty.

Data generation

Concept of non isomorphic strings

Consider two strings "abca" and "xyzx". While they are distinct, they are very similar to each other structurally. The alphabet used, is not relevant to the problem. Thus, we sample only non isomorphic strings.

Isomorphic Strings: Two strings are said to.....

Methodology:

- Create S(1,1) = {"a"}

- Recursion |S(n,k)| = |S(n-1, k)| x k + |S(n-1, k-1)| (where k is number of unique characters in string)
- Theorem: this is equal to number of partitions
- Proof 1

By taking only non-isomorphic strings, we also reduce the storage space complexity to (n/ln(n))^n from n^n

Methodology of Data Generation

- Select `n` (length) from a Gaussian distribution.
- Select `t` (number of unique transitions) from a Gaussian distribution.
- Select `d` (number of transitions in the puzzle, where the total number of transitions is `d+1`) from a Gaussian distribution.
- Sample `p` and `q` as well as the length of the string in transition.
- For source (`src`) and target (`tgt`):
  - Allow up to 2 new characters from the original string.
    (e.g., if the root string is `abc`, transitions may contain only `a, b, c, d, e`).
- Apply the transition (if not possible, retry).
- Continue until `d` steps are reached.
- Add the transition `final_string -> empty`.

Include images

1. Full graph and how it becomes very dense on slightly increase in t/d
2. Single path graph

key points

- If we want to see more patterns in solution sequence, keep d>t or d/t high

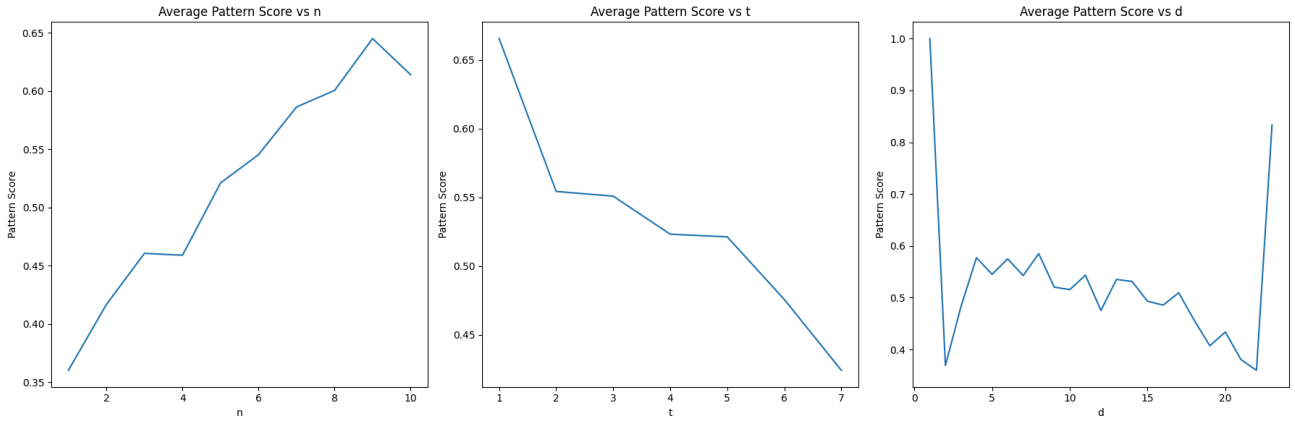
Bias Mitigation

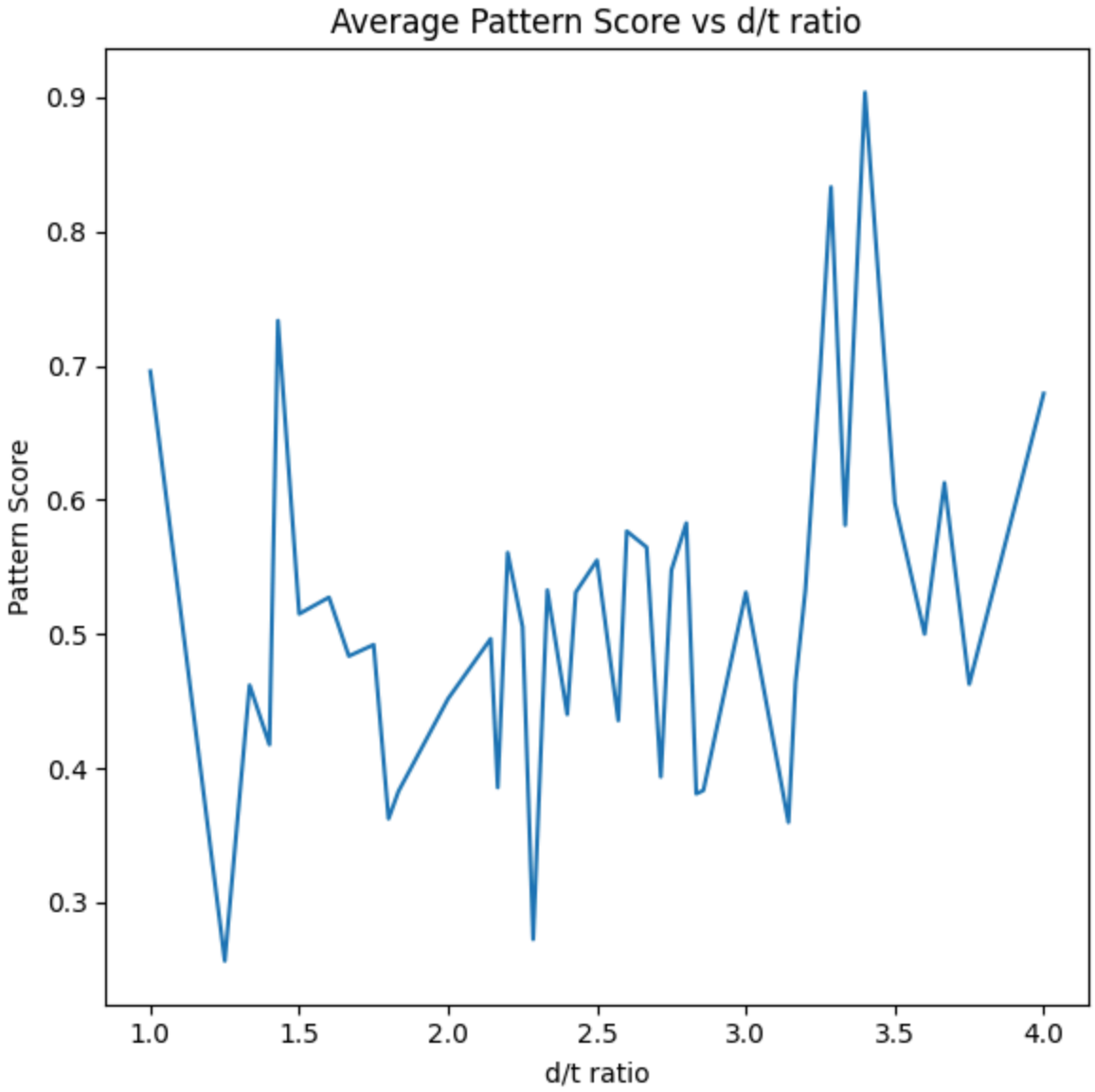
- NIS
- shuffle index of transitions
- letter shuffling

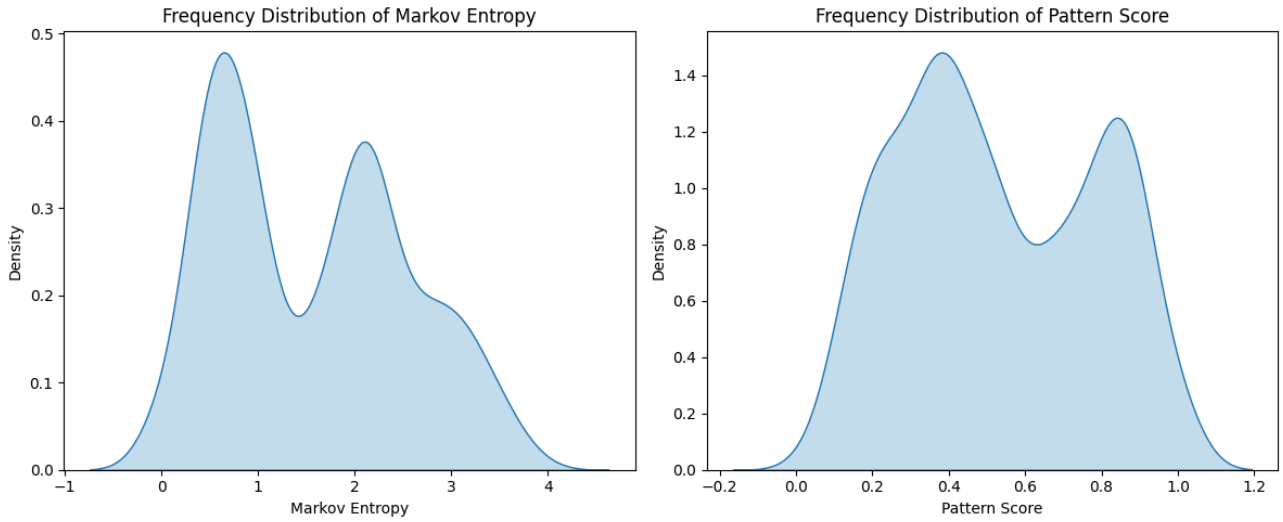
Other possibilities

- Make a graph and sample all puzzles using its leaf nodes (see notebook)
- problem: number of puzzles that we get is uncertain. Logistical issues, not in interest of time

Quantifying patterns

- Throughout data generation and evaluation, the key concept we revolve around is patterns. They make puzzles on sed-puzzle.com distinct from random samples
- to evaluate how "patterny" or "predictable" a sequence is
- introduce concept of markov entropy
  - define markov entropy
- Define pattern score 1- (markov entropy)/log2 (length of string)
- range 0 to 1
- include graphs regarding pattern score from SED_1000 (n: 1 to 7, t 1 to 4, d/t: 1 to 4):
- 





Evaluation metric

- pattern_weighed_accuracy = sum(pattern_score*accuracy)/(sum(pattern_score))
- measures performance with respect to pattern in solution sequence
- Major consideration for being representative of sed-puzzle.com

LLM evaluation

- Exploratory evaluation on very simple datasets (OneShot)
  - high d, lower acc
  - high t, lower acc

Methodology for further testing

- Explain pipeline implementation
  - Zero Shot: train_count =0
  - Few Shot: train_count = few
  - Chain of thought: give_explanation = true
- Interactive prompting (paste figure representing cycle)
- Change in implementation

Results

- Metrics on MIX_3_3_3

Comparison with humans

Humans perform poorly on problems with high number of letters and complex looking transitions, but beat LLMs on problems with high d/t ratio, and high pattern score

Potential

- use concept of pattern_score to sample better root strings

Annexure:

Proof 1:

References