Genetic Algorithms

Announcements

- Lab4 due Sunday

- HW2 Posted
  - Assertion inference
  - Optionally work with a partner

Outline

- HW2 Overview and Assertion Generation Review
- Review: Randoop
- Genetic algorithms

HW2: Assertion Generation Techniques

Part 1: Neural Approach

a. LLM unconstrained outputs
b. Grammar based

Part 2: IR Based Approach

HW2: Assertion Generation Techniques

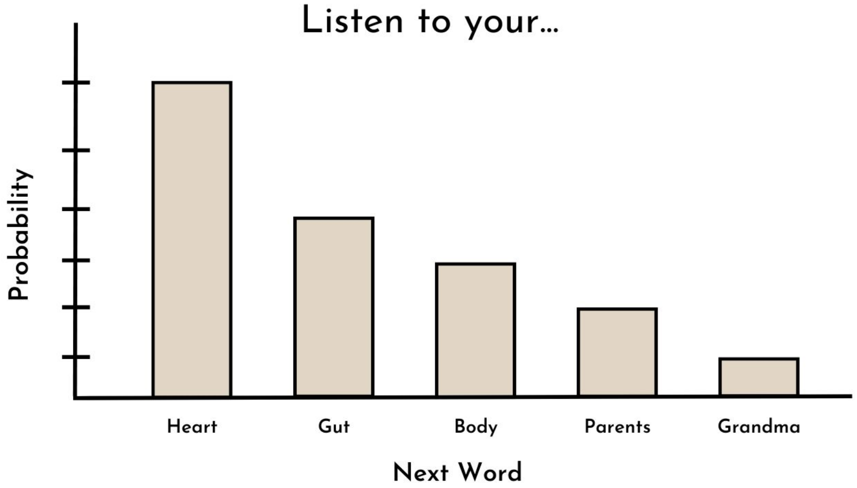
Part 1: Neural Approach

a. **LLM unconstrained outputs**
b. Grammar based

Part 2: IR Based Approach

Task Independent Autocomplete

The core task for most state-of-the-art LLMs is word prediction. Given a sequence of words, what is the probability distribution of the next word?



Prompt Engineering

- The practice of designing inputs (prompts) to effectively communicate with LLMs
  - Usually involves tinkering with the prompts and observing the outputs

- https://platform.openai.com/docs/guides/prompt-engineering
- https://www.promptingguide.ai

Prompting Techniques

Sentiment analysis: determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral.

```
Classify the text into neutral,
negative or positive.

Text: I think the vacation is okay.

Sentiment:
```

→ NEUTRAL

One-shot prompting

```
Classify the text into neutral,
negative or positive.

Text: This is awesome!

Sentiment: Positive


Text: I think the vacation is okay.

Sentiment:
```

NEUTRAL

Few-shot prompting

```
Classify the text into neutral, negative or
positive.

Text: This is awesome!

Sentiment: Positive


Text: Wow that movie was bad...

Sentiment: Negative


Text: I think the vacation is okay.

Sentiment:
```

NEUTRAL

Prompting Techniques

1. **Zero-shot prompting** – No examples, direct query.
2. **One-shot prompting** – Single example provided.
3. **Few-shot prompting** – Multiple examples provided.

Pros and cons?

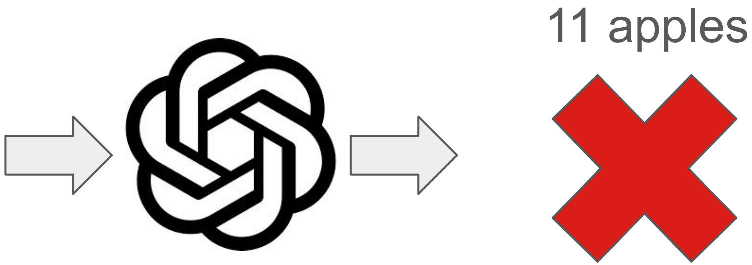
Con: Might overfit to the examples

Con: Might follow implicit patterns in the input

Chain of Thought Prompting

- Enables complex reasoning capabilities through intermediate reasoning steps.

I went to the market and bought 10 apples. I gave 2 apples to the neighbor and 2 to the repairman. I then went and bought 5 more apples and ate 1. How many apples did I remain with?



Chain of Thought Prompting

- Enables complex reasoning capabilities through intermediate reasoning steps.

I went to the market and bought 10 apples. I gave 2 apples to the neighbor and 2 to the repairman. I then went and bought 5 more apples and ate 1. How many apples did I remain with?

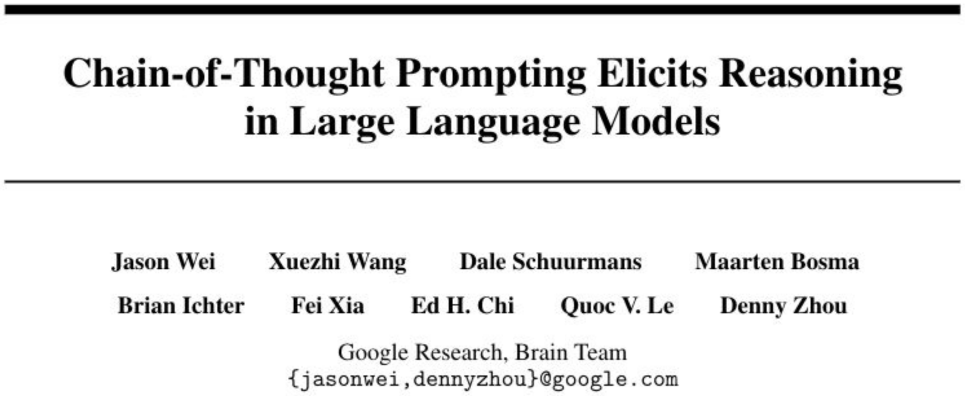
Let's think step by step.

First, you started with 10 apples.
You gave away 2 apples to the neighbor and 2 to the repairman, so you had 6 apples left.
Then you bought 5 more apples, so now you had 11 apples.
Finally, you ate 1 apple, **so you would remain with 10 apples.**

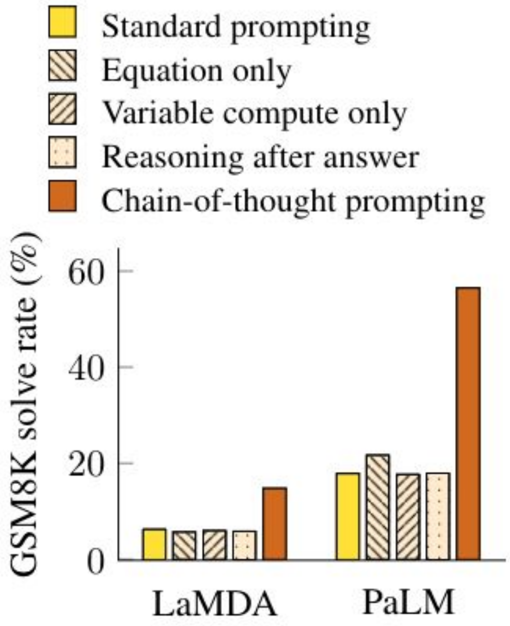
Chain of Thought Prompting

~10k citations

On "grade school math word problems"


Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma
Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou
Google Research, Brain Team
{jasonwei,dennyzhou}@google.com



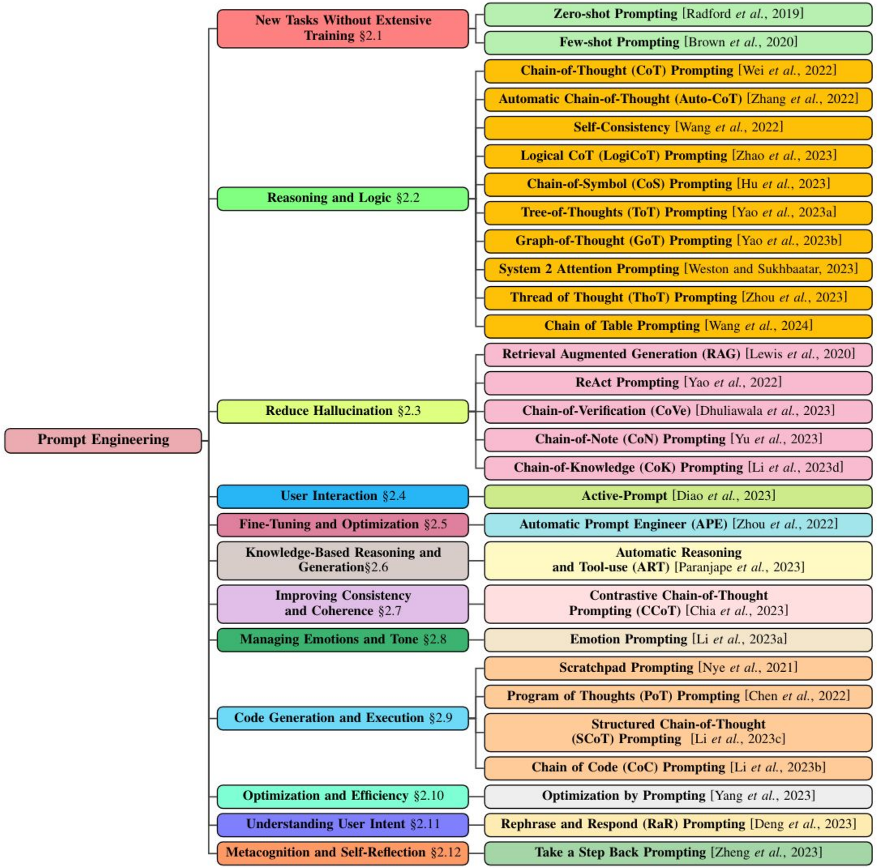


Figure 2: Taxonomy of prompt engineering techniques in LLMs, organized around application domains, providing a nuanced framework for customizing prompts across diverse contexts.

HW2: Assertion Generation Techniques

Part 1: Neural Approach

a. LLM unconstrained outputs
b. Grammar based

Part 2: IR Based Approach

Grammar Based Assertion Generation

Can you suggest a valid assertion to insert at the <AssertPlaceHolder> ?

testStack ( ) { Stack<Integer> s = new Stack<Integer>(); s.push(0); s.push(1); Integer result = s.pop(); <AssertPlaceHolder> ; }

assertTrue (stack.size() == 0);

Grammar Rules

```
1. A -> assertEquals(const, var)
2. A -> assertTrue(var)
3. A -> assertFalse(var)
4. A -> assertNull(var)
5. A -> assertNotNull(var)

6. const -> 0 | 1 | 5
7. var -> type matching var from the prefix
```

```
testStack ( ) {
        Stack<Integer> s = new Stack<Integer>();
        s.push(0);
        s.push(1);
        Integer result = s.pop();
        <AssertPlaceHolder> ;
}
```

Exhaustively enumerate the possible assertions and query LLM to rank rather than generate

```
assertEquals(0, result);     assertNull(s);

assertEquals(1, result);     assertNotNull(s);

assertEquals(5, result);     assertNull(result);

                             assertNotNull(result);
```

Grammar Based Assertion Generation

Can you suggest a valid assertion to insert at the <AssertPlaceHolder> ?

testStack ( ) { Stack<Integer> s = new Stack<Integer>();
s.push(0); s.push(1); Integer result = s.pop();
<AssertPlaceHolder> ; }

Select an assertion from this list:

```
assertEquals(0, result);
assertEquals(1, result);
assertEquals(5, result);
assertNull(s);
assertNotNull(s);
assertNull(result);
assertNotNull(result);
```

assertEquals(1, result);

What if this was our prefix?

```
1. A -> assertEquals(const, var)
2. A -> assertTrue(var)
3. A -> assertFalse(var)
4. A -> assertNull(var)
5. A -> assertNotNull(var)

6. const -> 0 | 1 | 5
7. var -> type matching var from the prefix
```

```
testStack ( ) {
        Stack<Integer> s = new Stack<Integer>();
        s.push(100);
        s.push(42);
        Integer result = s.pop();
        <AssertPlaceHolder> ;
}
```

HW2: Assertion Generation Techniques

Part 1: Neural Approach

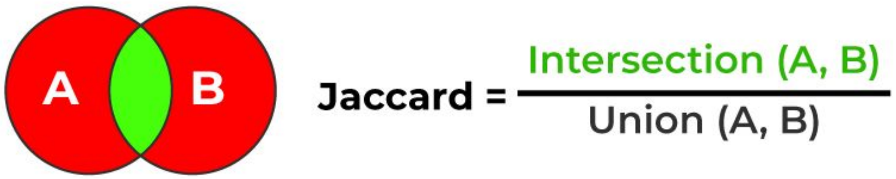
a. LLM unconstrained outputs
b. Grammar based

Part 2: IR Based Approach

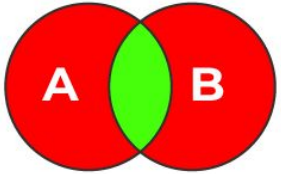
Similarity Metric - Jaccard

- Measures similarity between two sets
- Values range from o (no similarity) to 1 (identical sets)

$$J(X, Y) = |X \cap Y| / |X \cup Y|$$



Jaccard Coefficient



$$\text{Jaccard} = \frac{\text{Intersection (A, B)}}{\text{Union (A, B)}}$$

D1:

"Information Retrieval is useful"

D2:

"Retrieval of information is important"

{ information, retrieval, is useful }

{ retrieval, of, information, is, important }

$(A \cap B) = 3$

$(A \cup B) = 6$

`J(D1, D2)` = 3 / 6 = 0.5

Randoop Review

A randomized algorithm for generating tests

Randomly select a method call and select arguments from components

Iteratively build up *valid* test prefixes

What are used for assertions?

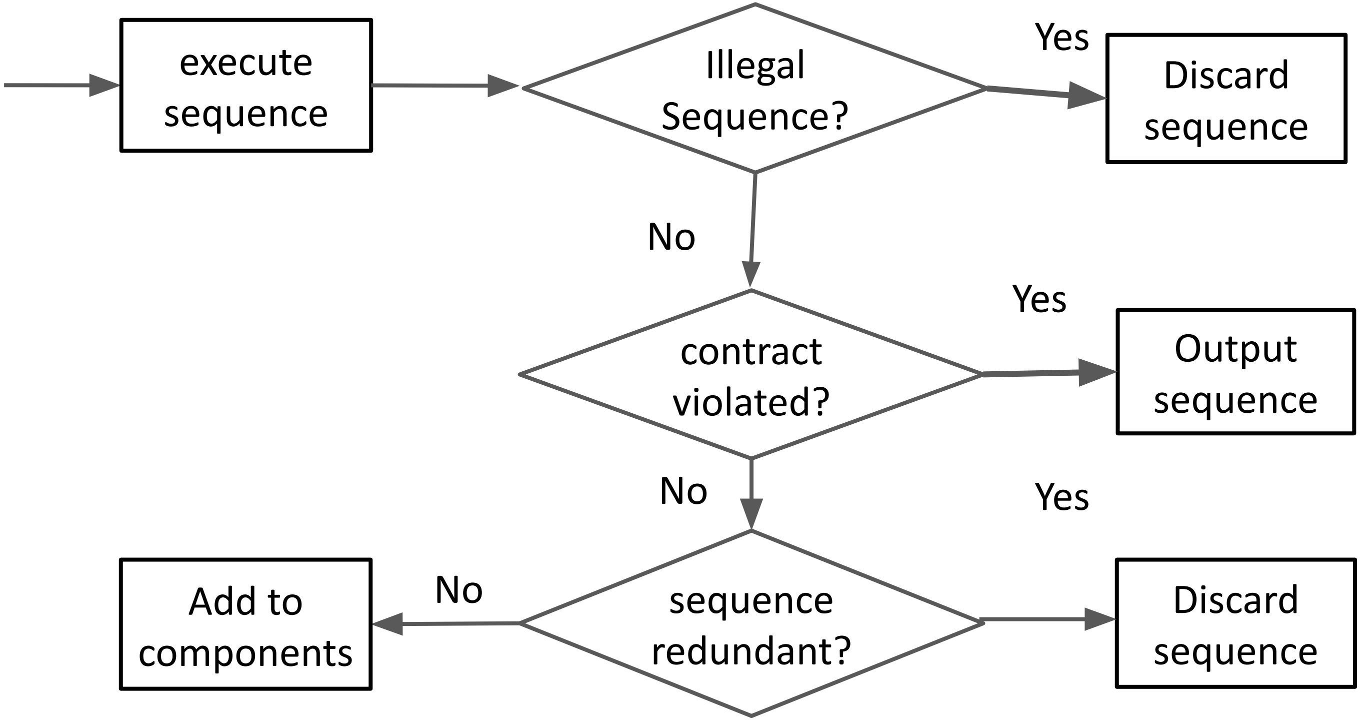
Randoop Algorithm

components = { int i = 0;   boolean b = false;   . . . }

Repeat until time limit expires:

- Create a new sequence
  - Randomly pick a method call $T_{ret}\ m(T_1,...,T_n)$
  - For each argument of type T_i, randomly pick sequence S_i from components that constructs an object v_i of that type
  - Create $S_{new} = S_1;\ ...;\ S_n;\ T_{ret}\ v_{new} = m(v_1,...,v_n);$
- Classify new sequence S_{new}:
  - discard / output as test / add to components

Classifying a Sequence



Randoop Drawbacks

1. Assertions...
2. Poor coverage! Only feedback is that the prefix is valid

Generating Tests From a Different Perspective..

Search Based Software Testing

Idea: A test exists in a finite space of all possible Java tests. We just need to find the one which triggers the bug.

This requires two things

1) Prefix
2) Assertion

A note on this field...

So far we’ve discussed the **problem** of **test oracle generation** and the **tools** to solve it: **neural**, **neural constrained by grammar**, **NL parsing**, and **IR.**

Now we are going to discuss the **problem** of **search based software testing** and the tools to solve it: **feedback directed** (randoop), **genetic algorithms** (EvoSuite), and **coverage guided** (AFL).

Genetic Algorithms

- Search technique used to find true or approximate solutions to search or optimization problems
  - What are some optimization problems you know?

- Use techniques inspired by evolutionary biology:
  - Selection
  - Mutation
  - Crossover

- Motivating Example: Imagine you're trying to create a **meal plan** that meets daily nutritional requirements while **minimizing cost**. This is an **optimization problem** where we search for the best combination of foods.

Genetic Algorithms

Selection (Survival of the Fittest)

- **Biology**: In nature, organisms that are better adapted to their environment are more likely to survive and reproduce. This is known as **natural selection**.

- **Optimization**: In evolutionary algorithms, the best-performing candidate solutions (individuals) are selected from a population based on their "fitness" (how well they solve the problem). These individuals are more likely to contribute to the next generation.

Genetic Algorithms

Crossover (Genetic Recombination)

- **Biology**: During reproduction, genetic material from two parents combines to create offspring with traits inherited from both, leading to increased genetic diversity.

- **Optimization**: In evolutionary algorithms, crossover combines parts of two parent solutions to create a new solution (offspring) that may inherit beneficial traits from both parents, improving the search process.

- **Meal plan example - what would crossover look like? Breakfast from M1 lunch from M2**

Genetic Algorithms

Mutation (Genetic Variation)

- **Biology**: Mutations are random changes in an organism's DNA that introduce genetic diversity. While some mutations may be harmful, others can offer advantages that improve survival.

- **Optimization**: In evolutionary algorithms, mutation introduces small random changes to candidate solutions to maintain diversity and prevent the algorithm from getting stuck in local optima.

- **Meal plan example - what would mutation look like? Maybe swap out breakfast**

Genetic Algorithms

- Start with a population of randomly generated individuals
- Works in multiple “generations”
  - Select individuals for breeding based on “fitness”
  - Mutation + crossover creates individuals for the next generation
  - Used in the next iteration

Genetic Algorithm

Termination: The algorithm terminates when either:

- A max number of iterations have been reached or
- Timeout or
- A satisfactory fitness level has been achieved

NO CONVERGENCE GUARANTEES

Vocabulary

- **Individual** - Any possible solution **(also called chromosome)**
- **Population** - Group of all individuals
- **Fitness** – Target function that we are optimizing (each individual has a fitness)

Genetic Algorithms

To adapt GA to test generation (or any other task) we need to define the following:

1. What is a chromosome / individual in the population?
   a. This is a potential solution
2. How are individuals selected for breeding?
3. What happens during crossover?
4. What happens during mutation?
5. How is fitness evaluated?
   a. How do we measure how “good” a chromosome / individual is?
   b. Called the *fitness function*

Meal Example

1. What is a chromosome?
    a. Meal plan with one option for breakfast and one for dinner
2. How are individuals selected?
    a. Probabilistically based on their fitness function
3. What happens during crossover?
    a. The new individual has breakfast from one plan and lunch from another
4. What happens during mutation?
    a. Breakfast is swapped out
5. Fitness function?
    a. $F(\text{meal}) = \text{nutrition} - (0.5 \times \text{cost})$
    b. $F(\text{mealplan}) = F(\text{breakfast}) + F(\text{lunch})$

Basic Genetic Algorithm

- Start with a large “population” of randomly generated “attempted solutions” to a problem
- Repeatedly do the following:
  - Evaluate each of the attempted solutions
  - (probabilistically) keep a subset of the best solutions
  - Use these solutions to generate a new population
- Quit when you have a satisfactory solution (or you run out of time)

Summary

Genetic Algorithms are search and optimization techniques inspired by natural selection and evolution.

Effective for complex, non-linear problems.

Explores a vast solution space efficiently.

Next week we will see how GAs are used to generate tests