Class 11 - Future Steps

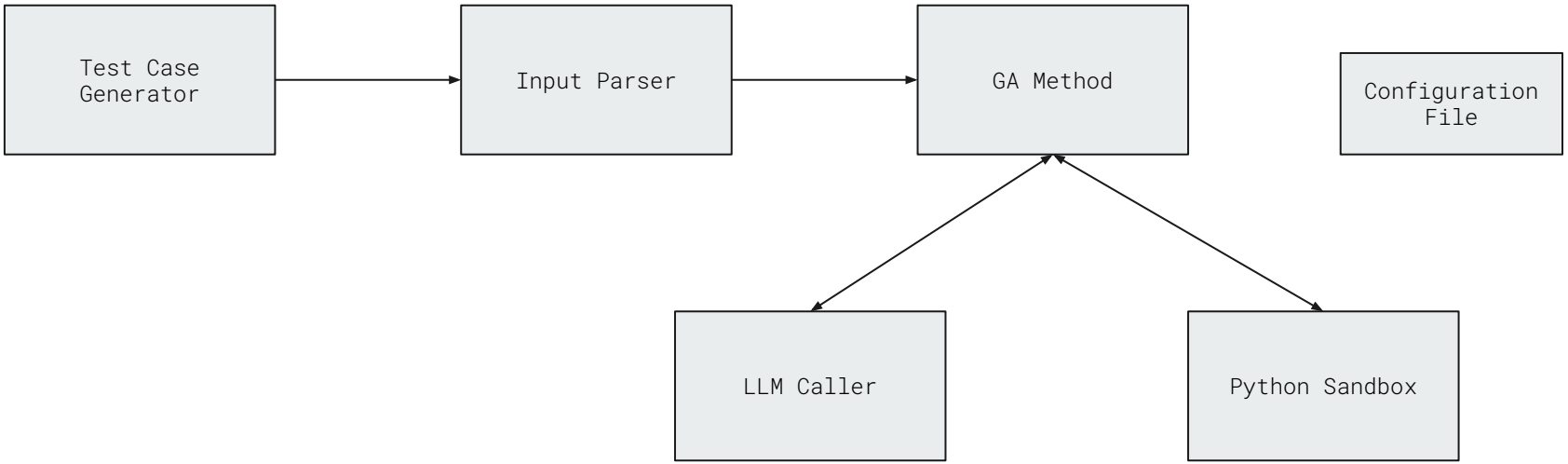
Guiding Genetic Algorithms with Language Models for Combinatorial Optimization

MISE Research Program - July 2025

Pedro Paredes

Components of GA System

System Diagram



Genetic Algorithm Method

Goal: The main method

Description: Runs the genetic algorithm method

Libraries: imports all the other build libraries

Note how we can build our own classes to group together logic components of our implementation, e.g. states or input

```python
class Input:
    def __init__(self, N, K, L, C):
        self.N = N
        self.K = K
        self.L = L
        self.C = C.copy()
```

```python
def genetic_llm_carr(N, K, L, C,
                     P=10,
                     n_gens=20,
                     generations=1,
                     seed=1234):
    random.seed(seed)
    sandbox = PythonSandbox()
    client = OpenAI()
    inp = Input(N, K, L, C)

    initial_state = State(Config.initial_program)
    initial_state.run(sandbox, inp)
    population = [initial_state]

    best = population[0]

    for generation in range(generations):
        offspring = []
        rep_prob = [s.score for s in population]
        rep_prob = [sum(rep_prob) / s for s in rep_prob]

        for _ in range(n_gens):
            s1 = random.choices(population, weights=rep_prob)[0]
            new_s = generate_new_state(s1, client)
            new_s.run(sandbox, inp)
            offspring.append(new_s)

        new_population = population + offspring
        new_population.sort(key=lambda a: a.score)
        population = [s for s in new_population[:P] if s.score < Config.INF]

        if population[0].score < best.score:
            best = population[0]

    return best.code, best.score
```

Input Parser

Goal: Read input files into Python

Description: Implements a function that given a filename, opens it and reads it into your specified format

LLM Caller

Goal: Interfaces with some LLM

Description: Implements a function that sends a prompt to an LLM and returns the result

Libraries: openai

Calling an LLM requires an account (OpenAI website)

After creating an account, you need an API key, which is a sort of password for you to call the LLM with your account. Never share your API keys with anyone!

```python
import backoff
import openai
import Config

api_limit = 500
api_calls = 0

@backoff.on_exception(backoff.expo, openai.RateLimitError)
def call_llm(client, instructions, input, max_output_tokens=100):
    global api_calls, api_limit
    if api_calls >= api_limit:
        raise RuntimeError("Exceeded max calls :(")
    response = client.responses.create(
        model="gpt-4.1-mini",
        max_output_tokens=max_output_tokens,
        instructions=instructions,
        input=input
    )
    api_calls += 1
    return response.output_text
```

Choosing an LLM

Making an LLM call costs money, which depends on the number of tokens (i.e. words) your call produces and the type of LLM used. You can see the list of OpenAI models here

For a project like this, you want to make lots of fast LLM calls, so you want a model that is quick and cheap, like the GPT-4.1 mini or the GPT-4.1 nano, which cost $.2 and $ .6 per 1 million tokens

For reference, running the CARR implementation with ~100 API calls uses around 10 thousand tokens, so less than 1 cent for the above models

To avoid going over budget, you want to make sure you control the number of calls by limiting the number of times the method can be called, and by using the max_output_tokens option

Python Sandbox

Goal: Run the Python codes in a safe environment

Description: To score a state we need to run the code associated to it. However, this is LLM generated code, so it could be unsafe (e.g., the LLM could “hallucinate” and try to delete all files on your computer). So we need to prevent it from running dangerous code.

One way of addressing this is by using a “sandbox” software: a special controlled virtual environment where you can run unsafe code. The simplest one out there is called Piston. Setting it up requires installing a few things and running a server that your Python program then connects to (using the requests library). This is not easy at all to setup.

Python Sandbox Alternative

An alternative is to tell the LLM prompt to not produce any unsafe code, and then check if the code contains any of the following potentially unsafe keywords:

```
import, os, subprocess, sys, shutil, socket, ctypes, eval, exec, __import__, open, __globals__, __dict__, __class__
```

You can then run Python code using the exec() function

However, note that this is still potentially unsafe. The LLMs are very unlikely to produce unsafe code, but it could happen

Configuration File

Goal: Store all the parameters of the system

Description: This is a simple file that contains all the parameters that are used in the code, like: the initial code (for the 1st population), the values of size of population, number of offspring per generation, and number of generations, the number of max tokens for LLM calls, the texts in the LLM prompts.

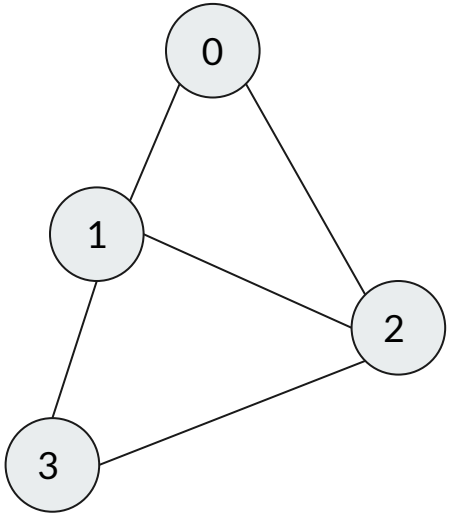
Using a file like this allows you to have control over all of the parameters of your system and makes it easier to tune them. You can think of it as a control center of the whole system

Test Case Generator

Goal: Generates random inputs for testing

Description: Picks N+1 points at random in the plane, one per location, and sets the costs C_{ij} to the distance between the points

Libraries: random, writing to a file



Other Interesting Topics

List of Interesting Topics We Didn't Cover

- Simulated Annealing - a more complex Hill Climbing method
- Island Genetic Algorithms - a method to keep multiple populations to decrease similarity
- Multithreading - running multiple functions at once in different processors

Final Assignment - Report Due Sunday

Details will be on website soon

That’s all!