# Class 6 - Implementing a Genetic Algorithm

*Guiding Genetic Algorithms with Language Models for Combinatorial Optimization*

MISE Research Program - July 2025


Pedro Paredes

# Hill Climbing

# Warm Up - First-Fit Algorithm for Truck Packing

The ***first-fit algorithm*** is an heuristic algorithm for the Truck Packing

It works in the following way:

- Keep a list of trucks that aren't full yet, which is initially empty
- Iterate through each package one by one, in the order of the input
- For each package find the first truck which the package can fit and place the package in it
- If there is no such truck, add a new truck and place the package in it

This algorithm has a few interesting properties:

- The quality of the output depends on the order of the input
- There is always some order of the input that leads to the optimal solution
- Regardless of the order of the input, the output is always at most twice the optimal solution

# First-Fit Implementation

```
def first_fit(packages, C):
    trucks = []
    for package in packages:
        for t in trucks:
            if sum(t) + package <= C:
                t.append(package)
                break
        else:
            trucks.append([package])
    return trucks
```

# Hill Climbing for Truck Packing

**States**: list of lists, where each internal list contains the packages in one truck

**Score function of a state**: size of the list (i.e., number of trucks used)

**Random state**:

- Copy the list of package lengths and randomly shuffle it
- Use the first-fit algorithm using the shuffled list of packages to produce a state

**Transitions**:

- For each truck, check if its packages can be added to another truck without exceeding the capacity
- In case of ties, pick a random example of the above

Note that the transition method above can only improve the score of a state by 1

# Main Hill Climbing Logic Loop

This is the main logic of the hill climbing algorithm, which could be used in any problem

```
def hill_climb_packing(packages, C, time_limit):
    t0 = time.perf_counter()
    current = random_state(packages, C)
    best = [t.copy() for t in current]
    while time.perf_counter() - t0 < time_limit:
        if not(try_improve_x(current, packages, C)):
            current = random_state(packages, C)
        if len(current) < len(best):
            best = [t.copy() for t in current]
    if not verify(best, packages, C):
        print("Error")
    return best, len(best)
```

# Verify Logic

This is a helper method to help us debug our code, i.e., make sure we are finding valid solutions

```python
def verify(packing, packages, C):
    for truck in packing:
        load = sum(truck)
        if load > C:
            return False
    packed_items = sorted(x for truck in packing for x in truck)
    return packed_items == sorted(packages)
```

# Random State generation

```
def random_state(packages, C):
    items = packages.copy()
    random.shuffle(items)
    return first_fit(items, C)
```

# Find the Best Transition

```
def try_improve_x(state, packages, C):
    trucks_idx = list(range(len(state)))
    random.shuffle(trucks_idx)
    for i in trucks_idx:
        for j in trucks_idx:
            if i == j:
                continue
            if sum(state[j]) + sum(state[i]) <= C:
                for x in state[i]:
                    state[j].append(x)
                del state[i]
                return True
    return False
```

# Genetic Algorithms

# See class 7

This topic was postponed to next class

# What's next?

**Next class tomorrow/thursday (2pm)**

**No assignment sheet today, study the code from the class, which will be on the website**

## Class 7:  Analyzing and Plotting Results