Evolution strategies : Those are family of algorithms usually special in the way the do the selection.

They do it using truncation selection.

Simplest algorithm is mu lamda algorithm.

Steepest Ascent Hill Climbing is a local search algorithm that iteratively evaluates all neighboring states and moves to the one with the highest value, aiming to find the global maximum. However, it can become stuck in local maxima, plateaus, or ridges, preventing it from reaching the optimal solution.

To mitigate this, the "with replacement" variant introduces a mechanism to escape these pitfalls. When the algorithm encounters a situation where no neighboring state offers a higher value (indicating a local maximum or plateau), it replaces the current state with a new randomly generated state. This allows the search process to explore different regions of the solution space, increasing the likelihood of finding the global maximum.

In summary, while standard Steepest Ascent Hill Climbing strictly moves to the best neighboring state and may get stuck in suboptimal areas, the "with replacement" version incorporates random restarts to overcome these limitations and enhance the search process.

For a practical implementation of Steepest Ascent Hill Climbing with Replacement, you can refer to this Python example:

This code defines a function called `select\_with\_replacement` that selects an individual from a population based on given fitness values, using a method called \*\*weighted random selection with replacement\*\*. Here's a breakdown of the code:

### Function: `select\_with\_replacement(population, fitnesses)`

- \*\*Purpose\*\*: Selects an individual from the `population` based on the provided `fitnesses`.

- \*\*Parameters\*\*:

- `population`: A list of individuals (e.g., solutions, chromosomes, or any other entities).

- `fitnesses`: A list of fitness values corresponding to each individual in the `population`. These values determine the probability of selecting each individual.

- \*\*Return Value\*\*: A single individual from the `population`.

### How It Works:

1. \*\*`random.choices`\*\*:

- This function is part of Python's `random` module.

- It performs a weighted random selection from the `population` based on the `weights` (fitnesses).

- The `weights` parameter assigns a probability to each individual in the `population`. Individuals with higher fitness values have a higher chance of being selected.

- The `k=1` parameter specifies that only \*\*one individual\*\* should be selected.

2. \*\*Selection with Replacement\*\*:

- The term "with replacement" means that the same individual can be selected multiple times across different calls to this function.

- This is in contrast to "without replacement," where an individual can only be selected once.

3. \*\*`[0]`\*\*:

- Since `random.choices` returns a list (even when `k=1`), the `[0]` is used to extract the single selected individual from the list.

### Example:

```python

import random

population = ['A', 'B', 'C', 'D']

fitnesses = [10, 20, 30, 40] # Higher fitness means higher probability of selection

selected\_individual = select\_with\_replacement(population, fitnesses)

print(selected\_individual) # Output: One of 'A', 'B', 'C', or 'D', with 'D' being the most likely

```

### Key Points:

- The function is commonly used in evolutionary algorithms (e.g., genetic algorithms) to select individuals for reproduction or the next generation.

- The fitness values determine the selection probability, so individuals with higher fitness are more likely to be chosen.

- Since selection is "with replacement," the same individual can be selected multiple times.

Let me know if you need further clarification!